Formal Modelling and Application of Graph Transformations in the Resource Description Framework

vorgelegt von
Diplom-Informatiker

Benjamin Braatz

Von der
Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr. Ing.)

genehmigte Dissertation

Tag der wissenschaftlichen Aussprache: 20. November 2009

Berlin 2009
D 83

This work is typeset using pdfLaTeX, the KOMA-Script scrbook document class, Computer Modern Bright fonts and the hyperref package.
All graphics are generated by Ti*k*Z and PGF from the pgf package.
All of these tools are available under free and open source software licences.

Abstract

In this thesis, a connection between two areas of research is developed. On the one hand, the Resource Desription Framework (RDF) is the basis of the Semantic Web. On the other hand, algebraic graph transformation has a long history of providing formally well-founded modification concepts for various graph and graph-like structures.

By designing an algebraic transformation approach for RDF, the rich theoretical results of algebraic graph transformation are made available to the RDF world. To achieve this goal, the formal abstract syntax and semantics of RDF is first reformulated in the language of category theory which is used heavily in graph transformation. Then, an abstract, categorical transformation framework is developed which is suitable for being afterwards instantiated by RDF structures. This is necessary since the existing frameworks are not applicable in an unmodified form.

The main theoretical results are a sequential composition operation for transformation rules and theorems showing the possibility to analyse and synthesise transformations for these sequentially composed rules. Moreover, these results are also available for transformation rules with negative application conditions.

The applicability of the resulting concept of RDF graph transformations is shown by two application scenarios. One is a classical Semantic Web application managing bibliographical metadata, while the other uses RDF as an abstract syntax for domain-specific modelling languages.

Zusammenfassung

In dieser Arbeit wird eine Verbindung zwischen zwei Forschungsbereichen entwickelt. Auf der einen Seite ist das Resource Description Framework (RDF) die Basis des Semantic Web. Auf der anderen Seite hat die algebraische Graphtransformation eine lange Tradition darin, formal fundierte Modifikationskonzepte für Graphen und graphähnliche Strukturen zur Verfügung zu stellen.

Durch den Entwurf eines algebraischen Transformationskonzepts für RDF werden die reichhaltigen theoretischen Ergebnisse der algebraischen Graphtransformation für die RDF-Welt nutzbar. Um dieses Ziel zu erreichen, wird zunächst die formale abstrakte Syntax und Semantik von RDF in der Sprache der Kategorientheorie, die bei Graphtransformationen intensiv genutzt wird, reformuliert. Dann wird ein abstraktes, kategorielles Transformations-Framework entwickelt, welches geeignet ist, anschließend durch RDF-Strukturen instanziiert zu werden. Dies ist notwendig, da keines der existierenden Frameworks in unmodifizierter Form anwendbar ist.

Die hauptsächlichen theoretischen Ergebnisse sind eine sequenzielle Kompositionsoperation für Transformationsregeln und Theoreme, die die Möglichkeit zeigen, Transformationen entlang dieser sequenziell komponierten Regeln zu analysieren und synthetisieren. Diese Ergebnisse sind weiterhin ebenfalls für Transformationsregeln mit negativen Anwendungsbedingungen verfügbar.

Die Anwendbarkeit des resultierenden Konzeptes für RDF-Graphtransformationen wird durch zwei Anwendungsszenarien gezeigt. Das eine ist eine klassische Semantic-Web-Anwendung, die bibliographische Metadaten verwaltet, während die andere RDF als abstrakte Syntax für domänenspezifische Modellierungssprachen verwendet.

Inhaltsverzeichnis

1. Introduction

1.1. The Semantic Web

The vision of the Semantic Web was first presented in [BHL01]. It comprises the enhancement of the human-readable data on the Web with machine-readable structure such that sophisticated automated investigation on and harvesting of distributed data becomes feasible.

While structured data are already omnipresent, this information is in the majority of cases not preserved when presenting them on the Web. Even if the data stems from a structured database, the presentations of these data mostly contain only layout and very abstract structure information.

In order to facilitate the integration of harvested data from different data sources, they would, however, have to be annotated with semantical information in the sense that a foreign machine can deduce the meaning of an entity even if it uses a totally different database schema.

In the concept of the Semantic Web, this goal shall be achieved by using published schemas for data, sometimes called ontologies. Presenting data with references to these published schemas then allows third parties to interpret them w. r. t. their own schemas.

As a data structure for presenting the schemas as well as the data typed over them, the Resource Description Framework (RDF) was developed and specfied in the set [W3C04] of recommendations. RDF allows the definition of globally usable structures by employing Uniform Resource Identifiers (URIs) as they are already used in various Web-based technologies.

While more sophisticated languages for the definition of ontologies have been developed, e. g., the Web Ontology Language (OWL), presented in [MH04] and related documents, we will only deal with RDF and its schema definition language RDF Schema in this thesis.

Moreover, we will confine ourselves to the abstract syntax representation of RDF graph structures, leaving aside technological issues of representing them using XML or other concrete syntaxes. A detailed formal treatment of RDF will be given in Chapter 2.

1.2. Problem Statement

Two application scenarios for RDF structures will be considered in this thesis. The first one is a classical Semantic Web scenario, where bibliographical metadata shall be managed, while the second one is a novel application in which the abstract syntax of domain-specific modelling languages (DSMLs) is represented in RDF.

More specifically, in the metadata scenario we will consider a simple example application that can relate authors with articles and books. Firstly, the problem arises that such an application will make assumptions about the data that are not expressible using pure RDF and RDF Schema, e. g., that a publication always has a unique title and a person a unique name or that a publication has at least one author.

The second problem is the definition of possible modification steps on the data store which should respect the constraints defined by the solution to the first problem. Most of the work on RDF is concerned with querying and logical inference on RDF structures. As an example, the querying language SPARQL, defined in [PS08], does, in contrast to the SQL for databases, not contain any primitives for modifying the queried RDF graph. An enhancement, called SPARQL/Update, is given in [SM08] and in [EPN09], where also some deficiencies of SPARQL/Update are discussed, an alternative modification protocol is proposed. Both proposals, however, do not contain a mechanism to respect constraints on the created structures as they are given by our first problem.

A third problem arises when data that are represented w. r. t. a different schema shall be imported into the application. This requirement not only demands a possibility to map the elements in the schemas to each other but also to apply structural changes. This problem has been adressed, e. g., by the definition of a mapping language for RDF graph transformation in [Her08], where this language takes a rather informal approach compared to the approach presented in this thesis. Moreover it only aims at ontology integration but not at a general modification framework for RDF.

In summary, our first application scenario demands a general modification framework that can deal with editing modifications as well as schema integrations and at the same time respect structural constraints for the created structures to solve all three problems sufficiently.

The second application scenario is motivated by work in modelling security requirements in banking environments which is partly presented in [BEB$^+$07] and [BHE09], where DSMLs shall not only be used to model small isolated issues but to give a comprehensive overview of the whole organisation and different stakeholders shall be provided with DSMLs tailored to their needs. Thus, the need for a family of interconnected small DSMLs arises which can dynamically evolve according to the users' requirements.

This means that we need a way to define DSMLs and possible modifications, such as, e. g., refactorings on them. Additionally, it should be possible to enhance the language, where, due to the assumption of a large corporate environment combined with frequent requests for small evolution steps of the language, it is not feasible to require a complex migration to a new language definition for each evolution step.

Moreover, due to the requirement that a global overview shall be achieved, the models should be kept in organisation-wide repository servers instead of user workstations. This combination of flexible language definitions and distributed storage of models makes RDF a reasonable choice for the data format, where a solution for the definition of languages and modifications and the evolution of languages is needed in addition.

In [AFR06] an evaluation of existing tools for DSMLs is given which shows that these tools mostly have quite different aims and do, in general, not target the organisation-wide integration which is predominant in our scenario. The popular MetaEdit+ tool, described, e. g., in [Tol06], which is not treated in the above evaluation, also aims at a different purpose emphasising flexible code generation from domain-specific models instead.

1.3. Graph Transformation

The proposal of this thesis is to use algebraic graph transformations to solve the problems identified in the previous section. More specifically, the definition of allowed structures for the metadata application and the definition of the language in the DSML scenario will both be achieved by graph grammars, which employ small graph transformation rules in a similar way to Chomsky grammars for textual languages.

But graph transformation rules can also be used for more complex modifications, where it can be ensured that these complex operations nevertheless stay within the defined language by composing them exclusively from the rules of the grammar.

Regarding the problem statement in the metadata scenario, complex rules can be employed for user-guided editing modifications as well as for the automatic integration of schemas, while in the DSML scenario an additional use case is the automatic evolution of language models when a DSML is modified.

1.4. Organisation of the Thesis

In Chapter 2, the syntax and semantics of RDF is formalised using category theoretical structures. This chapter is structured quite similar to the official RDF documents. While Section 2.1 formalises the abstract syntax given in [KC04], the following sections follow the structure of [Hay04], where the basic semantical structures are given in Section 2.2. RDF Schema, provided by [BG04], and the corresponding semantic extension are treated in Section 2.3. Section 2.4 gives a short overview of semantic extensions for datatypes and typed literals.

In Chapter 3, the MPOC transformation framework is defined and its main results are proven, where Section 3.1 motivates this by some use cases for rule-based transformations and gives a review of existing transformation approaches. Then, Section 3.2 contains the basic definitions which are extended by a notion of sequential composition in Section 3.3. Section 3.4 extends the framework by negative application conditions.

In Chapter 4, this framework is instantiated to RDF. For this purpose, Section 4.1 defines RDF patterns as an extension to RDF graphs and Section 4.2 shows the constructions needed for the instantiation. Section 4.3 then contains proofs for the additional properties that are needed for sequential composition of rules. Section 4.4 shows how inferences for RDF Schema can be implemented by transformation rules.

In Chapter 5, it is shown how the requirements of the application scenarios are met by graph transformations, where Section 5.1 is concerned with the Semantic Web metadata application, while Section 5.2 contains the treatment of the DSML scenario.

Finally, Chapter 6 summarises the thesis and gives pointers to future perspectives, where Section 6.1 summarises the solution for the application scenarios and Section 6.2 highlights the theoretical contributions.

Appendix A gives an overview over the definitions and basic results of category theory that are used throughout the thesis, where Section A.1 contains the basic notions of categories, morphisms and functors and Section A.2 treats limits and colimits, especially initial and final objects, products and coproducts and pushouts and pullbacks.

2. Resource Description Framework

The Resource Description Framework (RDF) is designed to facilitate the modelling and exchange of data and metadata on the Semantic Web. The normative description of RDF is given by the set [W3C04] of World Wide Web Consortium (W3C) recommendations. In this chapter a formalisation of RDF in the framework of category theory (see Appendix A and, e. g., [AHS90]) will be given. In contrast to other formalisations of RDF, most notably [Mar06], which stay close to the original recommendations, we sometimes deviate from them to achieve a cleaner categorical structure.

2.1. Abstract Syntax

The abstract syntax of RDF (defined in [KC04]) is supposed to provide the basic layer of the Semantic Web. In RDF, all data are represented by triples consisting of subjects, predicates and objects. As shown in Figure 2.1, an RDF graph is depicted by using the subjects and objects as nodes and labelling directed edges between these nodes with the predicates. Thus, each edge in the figure corresponds to a triple. Subjects and objects can be Uniform Resource Identifiers, shown as rounded rectangles, literal values, shown as rectangles, and blank nodes, shown as circles.

In the example, some information from the metadata application scenario is presented. More specifically, in the left part of the figure, a book by an author is represented using a blank node with a type and predicates from the imp: namespace. This namespace is supposed to be used for data imported from a flat bibliography without crossreferences like, e. g., a BibTEX file. In the right part of the figure, the bib: namespace is used to represent an article by the same author (at least an author with the same name). This namespace is used for the native vocabulary of our example metadata application, where a more complex structure is used to represent the metadata and URIs in the cont: namespace are employed to uniquely identify authors and publications. Later, in Section 5.1, we will develop transformation rules that integrate the flat imported data to conform to the schema expected by the example application.

In the following, we present a formalisation of RDF using category theoretical terms. This not only prepares our later definition of graph transformations on RDF graphs, but also gives a structured view of the building blocks of RDF itself.

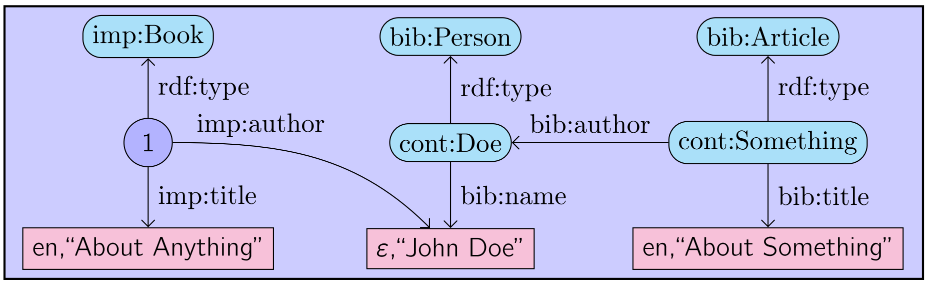


Abbildung 2.1.: RDF graph from the metadata application scenario

2.1.1. Vocabularies

We will use vocabularies to structure the global entities, URIs and literals, that can occur in RDF graphs and that are given a semantics by interpretations in Section 2.2. In this sense, vocabularies constitute an interface between syntax and semantics in RDF.

All strings appearing in RDF graphs will be Unicode strings, where the character set of Unicode is given in [Uni]. Hence, we first define the set of all Unicode strings. While implementations will use some kind of transfer encoding (probably the widely used UTF-8, specified in [Yer03]) for Unicode strings, we will, for the sake of simplicity, base our formal definitions on an alphabet containing the code points of Unicode with no special encoding.

Definition 2.1 (Unicode Strings)
The *Unicode alphabet* $\text{Unicode} := \{0_{16}, \ldots, 10\text{FFFF}_{16}\}$ consists of all Unicode characters as defined by the Unicode Specification in [Uni]. The set $\text{String} := \text{Unicode}^*$ [1] consists of all *strings* over the Unicode alphabet.

Since RDF is supposed to be used in distributed applications, Uniform Resource Identifiers (URIs) as defined in [BFM98] play an important role.[2] In contrast to other application areas like, e. g., the well-known Hypertext Transfer Protocol (HTTP), URIs in RDF are solely used as globally unique identifiers and do not imply that resources are retrievable under these URIs. The RDF Concepts and Abstract Syntax recommendation in [KC04] is a lot more specific about how to handle the encoding of a URI. For our rather theoretical treatment, however, it suffices to assume URIs as a special kind of Unicode strings.

Definition 2.2 (Uniform Resource Identifiers)
The set $\text{URI} \subseteq \text{String}$ consists of all *Uniform Resource Identifiers (URIs)* as defined by RFC 2396 in [BFM98].

[1] As usual, we will use A^* to denote the (infinte) set of all words over the alphabet A.
[2] Newer specifications in the field, as, e. g., [PS08], reference the updated standards in [BFM05] and [DS05] and also allow Internationalised Resource Identifiers (IRIs) containing the whole range of Unicode characters, while URIs may only contain ASCII.

In the subsequent discussions of RDF concepts we will use the XML Namespaces facility, specified in [BHLT06], to abbreviate URIs occuring in examples and definitions. More specifically, we will employ the following namespaces:

- rdf: for http://www.w3.org/1999/02/22-rdf-syntax-ns#
- rdfs: for http://www.w3.org/2000/01/rdf-schema#
- imp: for http://example.org/ImportBib/
- bib: for http://example.org/BibAppl/
- cont: for http://example.org/BibCont/
- dsml: for http://example.org/DSMLDef/
- mod: for http://example.org/DSMLMod/

The namespaces rdf: and rdfs: are used in the RDF specifications to represent entities that are used in all kinds of RDF applications. Some of them are treated in Section 2.3 in more detail. The namespaces imp:, bib: and cont: are used for the metadata application scenario, while dsml: and mod: are used for the domain-specific modelling scenario.

RDF graphs or documents may, moreover, contain literal entities, which do not identify any external resource or concept, but rather represent themselves. The content of a literal is alway given by a Unicode string. For a plain literal this string is supposed to be a name or a text, optionally annotated with the corresponding human language of the text. For typed literals, on the other hand, the string is the representation of a value in a data type, where the type of the literal is given by a URI. Such types may, for example, be the data types of XML Schema, defined in [BM04], or arbitrary XML fragments, for which a type URI `rdf:XMLLiteral` is pre-defined in RDF. Applications that do not understand the data type used by a typed literal are still able to process it as an opaque entity and, e. g., relay it to another application that is able to handle it properly.

Definition 2.3 (Literals)
The set $\mathrm{PLit} := \mathrm{Lang} \times \mathrm{String}$[3] consists of all *plain literals* $l = (l_{\mathrm{Lang}}, l_{\mathrm{String}})$ with $l_{\mathrm{Lang}} \in \mathrm{Lang}$ and $l_{\mathrm{String}} \in \mathrm{String}$, where $\mathrm{Lang} \subseteq \mathrm{String}$ is the set of all *language tags* as defined by RFC 3066 in [Alv01] (normalised to lowercase) and the empty tag $\varepsilon \in \mathrm{Lang}$ is supposed to be included.
The set $\mathrm{TLit} := \mathrm{URI} \times \mathrm{String}$ consists of all *typed literals* $l = (l_{\mathrm{Type}}, l_{\mathrm{String}})$ with a *type* $l_{\mathrm{Type}} \in \mathrm{URI}$ and $l_{\mathrm{String}} \in \mathrm{String}$.

While it would have been possible to use only typed literals and introduce a data type for text with language tags, the choice of plain literals as distinguished first-class

[3] We will use $A \times B$ to denote a categorical product of A and B (see Definition A.6). Here, in the category of sets and functions, this coincides with the usual cartesian product (see Proposition A.9).

elements makes it possible to employ RDF for quite complex, even multi-lingual, applications without the need to implement a data type mechanism. Moreover, a meaningful interaction between different RDF applications is still possible without agreeing on the used data types.

Now, we can define vocabularies to be triples containing a set of URIs, a set of plain literals and a set of typed literals. We extend this to become a rather simple category (see Definition A.1) by inclusions on all three components as morphisms.

Definition 2.4 (Category of Vocabularies)
The category **Voc** of *vocabularies* consists of

objects $V = (V_{\mathsf{URI}}, V_{\mathsf{PLit}}, V_{\mathsf{TLit}})$ with $V_{\mathsf{URI}} \subseteq \mathsf{URI}$, $V_{\mathsf{PLit}} \subseteq \mathsf{PLit}$ and $V_{\mathsf{TLit}} \subseteq \mathsf{TLit}$,

morphisms $V \subseteq V'$ if and only if $V_{\mathsf{URI}} \subseteq V'_{\mathsf{URI}}$, $V_{\mathsf{PLit}} \subseteq V'_{\mathsf{PLit}}$ and $V_{\mathsf{TLit}} \subseteq V'_{\mathsf{TLit}}$ and

compositions and identities given by reflexivity and transitivity of the inclusions.

Vocabularies will be used as an interface between syntax and semantics in Section 2.2. Moreover, Section 2.3 will give special meaning to a certain vocabulary in the rdf: and rdfs: namespaces.

2.1.2. RDF Graphs and Homomorphisms

RDF graphs are sets of statements about resources. These statements are given by subject–predicate–object triples, where subjects and objects can be URIs, literals or blank nodes[4] and predicates are URIs. Blank nodes denote resources with a non-existent or unknown global identifier. Such blank nodes are useful if a global identity for the corresponding resource is not relevant to external users or if they shall not be able to state additional facts about it.

Definition 2.5 (RDF Graph)
An *RDF graph* $G = (G_{\mathsf{Blank}}, G_{\mathsf{Triple}})$ consists of

- a set G_{Blank} of *blank nodes*,
- a set $G_{\mathsf{Triple}} \subseteq G_{\mathsf{Node}} \times \mathsf{URI} \times G_{\mathsf{Node}}$ of *triples* $(s, p, o) \in G_{\mathsf{Triple}}$ with *subject* $s \in G_{\mathsf{Node}}$, *predicate* $p \in \mathsf{URI}$ and *object* $o \in G_{\mathsf{Node}}$, where G_{Node} is the derived set $G_{\mathsf{Node}} := G_{\mathsf{Blank}} + \mathsf{URI} + \mathsf{PLit} + \mathsf{TLit}$[5].

[4] The normative RDF specifications do not allow literals as subjects. To achieve a more symmetrical definition, we choose to abolish this restriction as it is also done in many publications, e. g., in [MPG07], and even in newer W3C recommendations like the SPARQL definition in [PS08].

[5] We will use $A + B$ to denote a categorical coproduct of A and B (see Definition A.6). Here, in the category of sets and functions, this corresponds to a disjoint union (see Proposition A.9), where we will, for the sake of simplicity, assume that URI, PLit, TLit and the blank node sets are already disjoint and, hence, just take the union as coproduct and the inclusions as injections into the coproduct.

The class of all RDF graphs is denoted by RDFGraphs.

In the RDF Abstract Syntax in [KC04] an RDF graph is just a triple set and the blank node identifiers are drawn from a global, infinite set. Hence, two graphs according to the above definition that differ only in the existence of unused blank nodes would be considered equal according to the RDF recommendations. We choose to include the blank nodes in the graph in order to have them accessible in subsequent definitions and constructions enabling us, e. g., to construct limits and colimits by the corresponding contstructions in the category **Set** of sets and functions (see Proposition A.1), which would not be possible if the blank nodes were assumed to originate from a common super set. Moreover, this choice emphasises the locality of blank nodes.

The notion of RDF graphs differs in some aspects from directed graphs as they are considered in the graph transformation literature (cf., e. g., [EEPT06]). These differences lead to consequences for transformations of RDF graphs, as they are developed in this thesis. Firstly, triples do not have an independent identity and their can, thus, only be one triple with the same subject, predicate and object. In this respect, RDF graphs are similar to simple graphs or relations (as in the category **Rel** considered in [AHS90]). Therefore, the addition of a triple which is already present in a graph does not change the graph. Secondly, the URIs and literals are globally given and, thus, implicitly available as nodes in all graphs. Hence, URIs and literals cannot be deleted or added in a transformation.

RDF graph homomorphisms capture the structural relationship between two RDF graphs by a translation of their blank nodes, such that the codomain graph of the homomorphism contains at least the translated triples. Homomorphisms are not considered explicitly in the RDF Abstract Syntax in [KC04] and Semantics in [Hay04], but the notions of graph equivalence and subgraph used in these recommendations will be characterised as special homomorphisms in the following proposition.

Definition 2.6 (RDF Graph Homomorphism)
An *RDF graph homomorphism* $h\colon G \to G'$ for RDF graphs G and G' consists of

- a mapping function $h_{\mathsf{Blank}}\colon G_{\mathsf{Blank}} \to G'_{\mathsf{Blank}}$ for blank nodes,

such that

- there is an inclusion $h_{\mathsf{Triple}}(G_{\mathsf{Triple}}) \subseteq G'_{\mathsf{Triple}}$,

where $h_{\mathsf{Triple}}\colon G_{\mathsf{Node}} \times \mathsf{URI} \times G_{\mathsf{Node}} \to G'_{\mathsf{Node}} \times \mathsf{URI} \times G'_{\mathsf{Node}}$ is a derived translation defined by

$$h_{\mathsf{Triple}}(s, p, o) := (h_{\mathsf{Node}}(s), p, h_{\mathsf{Node}}(o))$$

and $h_{\mathsf{Node}}\colon G_{\mathsf{Node}} \to G'_{\mathsf{Node}}$ is in turn defined by

$$h_{\mathsf{Node}}(x) := \begin{cases} h_{\mathsf{Blank}}(x) & \text{for } x \in G_{\mathsf{Blank}} \\ x & \text{for } x \in \mathsf{URI} + \mathsf{PLit} + \mathsf{TLit} \end{cases}.$$

The following proposition states that RDF graphs and homomorphisms form a category (see Definition A.1) and characterises its special morphisms (see Definition A.2). While the notion of equivalence from [KC04] and [Hay04] is reflected exactly by the isomorphisms of this category, the definition of subgraphs by monomorphisms is more general then the definition by triple set inclusions employed in the recommendations. More specifically, monomorphisms additionally allow the injective renaming of blank nodes highlighting their locality to the specific graph.

Proposition 2.1 (Category **RDFHom**)
RDF graphs and RDF graph homomorphisms constitute a category **RDFHom**, where compositions are compositions of blank node functions and identities are blank node identities.
In this category, the following characterisations of special morphisms hold:

Mono: A homomorphism $m\colon G \to G'$ is a monomorphism if and only if m_{Blank} is injective. In this case, G is called a *subgraph* of G' (via m).

Epi: A homomorphism $e\colon G \to G'$ is an epimorphism if and only if e_{Blank} is surjective.

Iso: A homomorphism $i\colon G \to G'$ is an isomorphism if and only if i_{Blank} is bijective and the equality $i_{\mathsf{Triple}}(G_{\mathsf{Triple}}) = G'_{\mathsf{Triple}}$ is satisfied.
Isomorphic graphs are also called *equivalent*.

Beweis. Associativity of compositions and cancellability of identities follow directly from the corresponding properties of the underlying blank node functions (see Proposition A.1), since the equality of RDF graph homomorphisms is equivalent to the equality of the underlying blank node functions.

Mono: **If:** Suppose m_{Blank} is injective and $m \circ l_1 = m \circ l_2$. Then $m_{\mathsf{Blank}} \circ (l_1)_{\mathsf{Blank}} = m_{\mathsf{Blank}} \circ (l_2)_{\mathsf{Blank}}$ and by injective functions being monomorphisms in **Set** (see Proposition A.3) also $(l_1)_{\mathsf{Blank}} = (l_2)_{\mathsf{Blank}}$. Then we also have $l_1 = l_2$ and, hence, m is a monomorphism. **Only if:** Suppose m_{Blank} is not injective. Then there are blank nodes $b, b' \in G_{\mathsf{Blank}}$ with $b \neq b'$ and $m_{\mathsf{Blank}}(b) = m_{\mathsf{Blank}}(b')$. But then we can define $l_1, l_2\colon (\{*\}, \varnothing) \to G$ with $(l_1)_{\mathsf{Blank}}(*) = b$ and $(l_2)_{\mathsf{Blank}}(*) = b'$ satisfying $m_{\mathsf{Blank}} \circ (l_1)_{\mathsf{Blank}} = m_{\mathsf{Blank}} \circ (l_2)_{\mathsf{Blank}}$, but not $(l_1)_{\mathsf{Blank}} = (l_2)_{\mathsf{Blank}}$. Then we also have $l_1 \neq l_2$ and, hence, m is not a monomorphism.

Epi: **If:** Suppose e_{Blank} is surjective and $f_1 \circ e = f_2 \circ e$. Then $(f_1)_{\mathsf{Blank}} \circ e_{\mathsf{Blank}} = (f_2)_{\mathsf{Blank}} \circ e_{\mathsf{Blank}}$ and by surjective functions being epimorphisms in **Set** (see Proposition A.3) also $(f_1)_{\mathsf{Blank}} = (f_2)_{\mathsf{Blank}}$. Then we also have $f_1 = f_2$ and, hence, e is an epimorphism. **Only if:** Suppose e_{Blank} is not surjective. Then there is a blank node $b \in G'_{\mathsf{Blank}}$ for which no $x \in G_{\mathsf{Blank}}$ with $e_{\mathsf{Blank}}(x) = b$ exists. But then we can define $f_1, f_2\colon G' \to (G'_{\mathsf{Blank}} \cup \{1, 2\}, G'_{\mathsf{Triple}})$ with $(f_1)_{\mathsf{Blank}}(y) = y = (f_2)_{\mathsf{Blank}}(y)$ for all $y \neq b$, $(f_1)_{\mathsf{Blank}}(b) = 1$ and $(f_2)_{\mathsf{Blank}}(b) = 2$ satisfying $(f_1)_{\mathsf{Blank}} \circ e = (f_2)_{\mathsf{Blank}} \circ e$, but not $(f_1)_{\mathsf{Blank}} = (f_2)_{\mathsf{Blank}}$. Then we also have $f_1 \neq f_2$ and, hence, e is not an epimorphism.

Iso: **If:** Suppose i_{Blank} is bijective and $i_{\mathsf{Triple}}(G_{\mathsf{Triple}}) = G'_{\mathsf{Triple}}$. Since i_{Blank} is an isomorphism in **Set** (see Proposition A.3), there is an inverse $j_{\mathsf{Blank}} : G'_{\mathsf{Blank}} \to G_{\mathsf{Blank}}$ with $j_{\mathsf{Blank}} \circ i_{\mathsf{Blank}} = \mathrm{id}_{G_{\mathsf{Blank}}}$ and $i_{\mathsf{Blank}} \circ j_{\mathsf{Blank}} = \mathrm{id}_{G'_{\mathsf{Blank}}}$. Moreover, j_{Blank} constitutes an RDF graph homomorphism j, since $j_{\mathsf{Triple}}(G'_{\mathsf{Triple}}) = j_{\mathsf{Triple}}(i_{\mathsf{Triple}}(G_{\mathsf{Triple}})) = G_{\mathsf{Triple}}$. Hence, i is an isomorphism. **Only if:** Suppose i_{Blank} is not bijective. Then we cannot find an inverse blank node function (see Proposition A.3) and, hence, i cannot be an isomorphism. Suppose i_{Blank} is bijective, but $G'_{\mathsf{Triple}} \nsubseteq i_{\mathsf{Triple}}(G_{\mathsf{Triple}})$. Then there is an inverse blank node function j_{Blank}, but it does not constitute an RDF graph homomorphism, since $j_{\mathsf{Triple}}(G'_{\mathsf{Triple}}) \nsubseteq j_{\mathsf{Triple}}(i_{\mathsf{Triple}}(G_{\mathsf{Triple}})) = G_{\mathsf{Triple}}$. Hence, i is not an isomorphism. □

Figure 2.2 shows an RDF homomorphism, which is neither a mono- nor an epimorphism, while in Figure 2.3 examples for a mono- and an epimorphism are given. Note that the epimorphism does not have to be surjective on triples but just on blank nodes, while the monomorphism is also injective on triples since the triple translation is composed solely of injective functions.

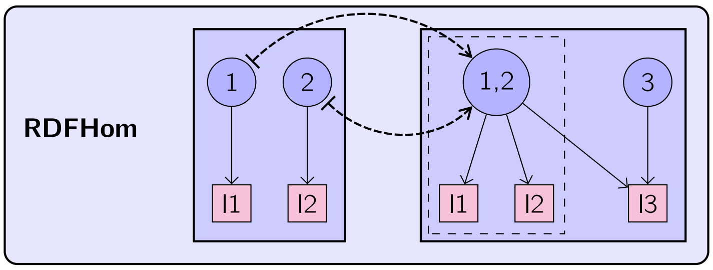
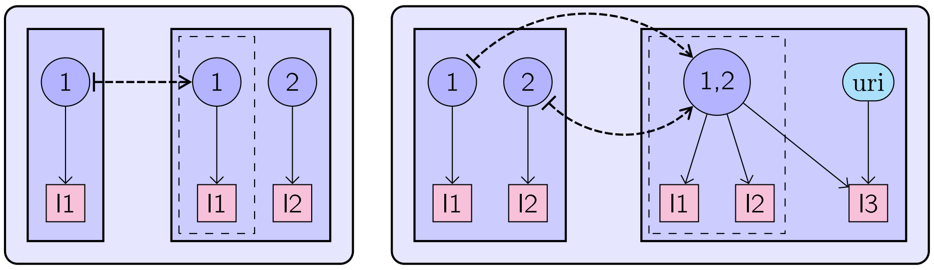


Abbildung 2.2.: Homomorphism in **RDFHom**



(a) Monomorphism (b) Epimorphism

Abbildung 2.3.: Special morphisms in **RDFHom**

In [Hay04] the merge of RDF graphs is defined by standardising apart common blank nodes of the graph. In our category theoretical setting for RDF this corresponds to taking a coproduct (see Definition A.6) of the blank node sets (see Proposition A.9). This leads to a coproduct in the category **RDFHom**.

Proposition 2.2 (Coproducts in **RDFHom**)
Given two RDF graphs G and H, a coproduct $G + H$ can be constructed by the disjoint union $(G + H)_{\text{Blank}} := G_{\text{Blank}} + H_{\text{Blank}}$ with corresponding injections $i_{\text{Blank}} \colon G_{\text{Blank}} \to (G + H)_{\text{Blank}}$ and $j_{\text{Blank}} \colon H_{\text{Blank}} \to (G + H)_{\text{Blank}}$, and the triple set $(G + H)_{\text{Triple}} := i_{\text{Triple}}(G_{\text{Triple}}) \cup j_{\text{Triple}}(H_{\text{Triple}})$.

Beweis. The blank node functions i_{Blank} and j_{Blank} obviously constitute RDF graph homomorphisms i and j since $i_{\text{Triple}}(G_{\text{Triple}}) \subseteq (G + H)_{\text{Triple}}$ and $j_{\text{Triple}}(H_{\text{Triple}}) \subseteq (G + H)_{\text{Triple}}$. For each other RDG graph X with RDF graph homomorphisms $g \colon G \to X$ and $h \colon H \to X$, we have a unique blank node function $x_{\text{Blank}} \colon (G + H)_{\text{Blank}} \to X_{\text{Blank}}$ with $x_{\text{Blank}} \circ i_{\text{Blank}} = g_{\text{Blank}}$ and $x_{\text{Blank}} \circ j_{\text{Blank}} = h_{\text{Blank}}$ because of the coproduct property of $G_{\text{Blank}} + H_{\text{Blank}}$. This also constitutes an RDF graph homomorphism x since $x_{\text{Triple}}((G + H)_{\text{Triple}}) = x_{\text{Triple}}(i_{\text{Triple}}(G_{\text{Triple}}) \cup j_{\text{Triple}}(H_{\text{Triple}})) = x_{\text{Triple}}(i_{\text{Triple}}(G_{\text{Triple}})) \cup x_{\text{Triple}}(j_{\text{Triple}}(H_{\text{Triple}})) = g_{\text{Triple}}(G_{\text{Triple}}) \cup h_{\text{Triple}}(H_{\text{Triple}}) \subseteq X_{\text{Triple}}$ because of the homomorphism properties of g and h. □

We can construct the underlying vocabulary of an RDF graph by just collecting all URIs, plain and typed literals that are used in triples of the graph.

Definition 2.7 (Underlying Vocabulary)
Given an RDF graph G, the *underlying vocabulary* $\text{Voc}(G)$ of G is constructed by

- $\text{Voc}(G)_{\text{URI}} := \{u \in \text{URI} \mid \exists (s, p, o) \in G_{\text{Triple}} \colon s = u, p = u \text{ or } o = u\}$,
- $\text{Voc}(G)_{\text{PLit}} := \{l \in \text{PLit} \mid \exists (s, p, o) \in G_{\text{Triple}} \colon s = l \text{ or } o = l\}$ and
- $\text{Voc}(G)_{\text{TLit}} := \{l \in \text{TLit} \mid \exists (s, p, o) \in G_{\text{Triple}} \colon s = l \text{ or } o = l\}$.

This construction can be extended to become a functor (see Definition A.4) since the preservation of triples by an RDF graph homomorphism implies the necessary inclusions on the underlying vocabularies.

Proposition 2.3 (Underlying Vocabulary Functor)
The construction of underlying vocabularies constitutes a functor $\text{Voc} \colon \textbf{RDFHom} \to \textbf{Voc}$.

Beweis. We have to show that for each RDF graph homomorphism $h \colon G \to G'$ we also have $\text{Voc}(G) \subseteq \text{Voc}(G')$. This is satisfied since for each $u \in \text{Voc}(G)_{\text{URI}}$ we have $\exists (s, p, o) \in G_{\text{Triple}} \colon s = u, p = u$ or $o = u$ by definition. Hence, we also have $\exists (h_{\text{Node}}(s), p, h_{\text{Node}}(o)) \in G'_{\text{Triple}} \colon h_{\text{Node}}(s) = u, p = u$ or $h_{\text{Node}}(o) = u$ by the homomorphism property of h and the fact that h_{Node} maps URIs identically. This finally means that $u \in \text{Voc}(G')_{\text{URI}}$. Similar arguments apply for plain and typed literals. □

The underlying vocabularies of RDF graphs are used in the next section to define their semantics. More specifically, interpretations of vocabularies are defined and an interpretation satisfies an RDF graph if it is consistent with all triples in the graph.

2.2. Semantics

RDF graphs are intended as assertions about the world. The meaning of these assertions will be given by a formal semantics in this section. This model-theoretic semantics, adopted from [Hay04], uses interpretations of vocabularies as models of possible worlds. They map URIs and literals in a vocabulary to resources as their denotations, where plain literals denote themselves and URIs and typed literals denote resources. Properties are special resources with associated relations among resources as extensions.

Definition 2.8 (Interpretation)
An *interpretation* $I = (I_{\mathrm{Voc}}, I_{\mathrm{Res}}, I_{\mathrm{Val}}, I_{\mathrm{Prop}}, \mathrm{pext}_I, \mathrm{uint}_I, \mathrm{tint}_I)$ consists of

- a vocabulary $I_{\mathrm{Voc}} = (I_{\mathrm{URI}}, I_{\mathrm{PLit}}, I_{\mathrm{TLit}}) \in |\mathbf{Voc}|$,
- a set I_{Res} of *resources*, called *universe*, with a subset $I_{\mathrm{Val}} \subseteq I_{\mathrm{Res}}$ of *literal values* containing the plain literals $I_{\mathrm{PLit}} \subseteq I_{\mathrm{Val}}$ and a subset $I_{\mathrm{Prop}} \subseteq I_{\mathrm{Res}}$ of *properties*[6],
- an *extension function* $\mathrm{pext}_I \colon I_{\mathrm{Prop}} \to \mathcal{P}(I_{\mathrm{Res}} \times I_{\mathrm{Res}})$ [7],
- an *interpretation function* $\mathrm{uint}_I \colon I_{\mathrm{URI}} \to I_{\mathrm{Res}}$ for URIs and
- an *interpretation function* $\mathrm{tint}_I \colon I_{\mathrm{TLit}} \to I_{\mathrm{Res}}$ for typed literals.[8]

The class of all interpretations is denoted by Interp.

The relation between the abstract syntax and the semantics is established by the satisfaction relation in the following definition.[9] Informally, an interpretation satisfies a graph if all assertions stated in the triples of the graph are realised by corresponding property extensions in the interpretation.

Definition 2.9 (Satisfaction Relation)
The *satisfaction relation* $\models \subseteq \mathrm{Interp} \times \mathrm{RDFGraphs}$ is defined by the condition that an

[6] In [Hay04], I_{Prop} is not required to be a subset of I_{Res} for simple interpretations, but only for RDF interpretations, defined later. We choose to require $I_{\mathrm{Prop}} \subseteq I_{\mathrm{Res}}$ from the beginning to avoid some unnecessary technical difficulties.

[7] Here, $\mathcal{P}(S)$ denotes the powerset of a set S (see also Proposition A.5).

[8] Note that the codomain for typed literals is the whole set I_{Res} instead of just the set I_{Val} of literal values because ill-formed typed literals are supposed to be interpreted as resources which are *not* literal values.

[9] While the normative RDF semantics in [Hay04] uses an approach, where interpretations are recursively extended to functions assigning a truth value to triples and graphs, we use a satisfaction relation, since it fits better into our category theoretical framework.

interpretation $I \in \text{Interp}$ satisfies an RDF graph $G \in \text{RDFGraphs}$, written as $I \models G$, if and only if $\text{Voc}(G) \subseteq I_{\text{Voc}}$ and there is an assignment $asg\colon G_{\text{Blank}} \to I_{\text{Res}}$ such that for all triples $(s, p, o) \in G_{\text{Triple}}$

- $\text{uint}_I(p) \in I_{\text{Prop}}$ and
- $(\overline{asg}(s), \overline{asg}(o)) \in \text{pext}_I(\text{uint}_I(p))$,

where the function $\overline{asg}\colon G_{\text{Node}} \to I_{\text{Res}}$ is defined by

$$\overline{asg}(x) := \begin{cases} asg(x) & \text{for } x \in G_{\text{Blank}} \\ \text{uint}_I(x) & \text{for } x \in \text{URI} \\ x & \text{for } x \in \text{PLit} \\ \text{tint}_I(x) & \text{for } x \in \text{TLit}. \end{cases}$$

One of the most important notions for the logical view on RDF is the entailment of RDF graphs. An RDF graph entails another graph if the first semantically implies the second, i. e., all interpretations satisfying the first also satisfy the second. This notion formally underpins inferences on RDF graphs.

Definition 2.10 (Entailment)
Given two RDF graphs G and H, G *entails* H, written as $G \Vdash H$, if and only if $I \models G$ implies $I \models H$ for all interpretations $I \in \text{Interp}$.

In order to syntactically characterise entailment, we define a more general class of morphisms on RDF graphs which can not only map blank nodes to blank nodes but also instantiate them to URIs and literals. Hence, the codomain of the blank node functions is the whole set G'_{Node} instead of just the blank nodes G'_{Blank}. This notion can be seen as a combination of the notions of instance and subgraph in [Hay04].

Definition 2.11 (RDF Graph Instantiation)
An *RDF graph instantiation* $i\colon G \to G'$ for RDF graphs G and G' consists of

- an instantiation function $i_{\text{Blank}}\colon G_{\text{Blank}} \to G'_{\text{Node}}$ for blank nodes,

such that

- there is an inclusion $i_{\text{Triple}}(G_{\text{Triple}}) \subseteq G'_{\text{Triple}}$,

where $i_{\text{Triple}}\colon G_{\text{Node}} \times \text{URI} \times G_{\text{Node}} \to G'_{\text{Node}} \times \text{URI} \times G'_{\text{Node}}$ is defined by

$$i_{\text{Triple}}(s, p, o) := (i_{\text{Node}}(s), p, i_{\text{Node}}(o))$$

and $i_{\text{Node}}\colon G_{\text{Node}} \to G'_{\text{Node}}$ by

$$i_{\text{Node}}(x) := \begin{cases} i_{\text{Blank}}(x) & \text{for } x \in G_{\text{Blank}} \\ x & \text{for } x \in \text{URI} + \text{PLit} + \text{TLit} \end{cases}.$$

RDF graph instantiations give rise to a super category of **RDFHom**.

Proposition 2.4 (Category **RDFInst**)
RDF graphs and RDF graph instantiations constitute a category **RDFInst**, where compositions are given by $(j \circ i)_{\mathsf{Blank}} := j_{\mathsf{Node}} \circ i_{\mathsf{Blank}}$ for all RDF graph instantiations $i \colon G \to G'$ and $j \colon G' \to G''$ and identities by $(\mathrm{id}_G)_{\mathsf{Blank}} := \mathrm{incl}_{G_{\mathsf{Blank}}, G_{\mathsf{Node}}} \circ \mathrm{id}_{G_{\mathsf{Blank}}}$ [10] for all RDF graphs G.
There is an inclusion functor $\mathrm{Incl} \colon \mathbf{RDFHom} \to \mathbf{RDFInst}$ with $\mathrm{Incl}(G) := G$ for all RDF graphs G and $\mathrm{Incl}(h)_{\mathsf{Blank}} := \mathrm{incl}_{G'_{\mathsf{Blank}}, G'_{\mathsf{Node}}} \circ h_{\mathsf{Blank}}$ for all RDF graph homomorphisms h.

Beweis. Firstly, we observe that $(k \circ j)_{\mathsf{Node}} = k_{\mathsf{Node}} \circ j_{\mathsf{Node}}$. Associativity of compositions is then obtained by $(k \circ (j \circ i))_{\mathsf{Blank}} = k_{\mathsf{Node}} \circ (j_{\mathsf{Node}} \circ i_{\mathsf{Blank}}) = (k_{\mathsf{Node}} \circ j_{\mathsf{Node}}) \circ i_{\mathsf{Blank}} = (k \circ j)_{\mathsf{Node}} \circ i_{\mathsf{Blank}} = ((k \circ j) \circ i)_{\mathsf{Blank}}$ using associativity of functions. Secondly, since $(\mathrm{id}_{G'})_{\mathsf{Node}} = \mathrm{id}_{G'_{\mathsf{Node}}}$ cancellability of identities follows from $(\mathrm{id}_{G'} \circ i)_{\mathsf{Blank}} = (\mathrm{id}_{G'})_{\mathsf{Node}} \circ i_{\mathsf{Blank}} = \mathrm{id}_{G'_{\mathsf{Node}}} \circ i_{\mathsf{Blank}} = i_{\mathsf{Blank}}$ and $(i \circ \mathrm{id}_G)_{\mathsf{Blank}} = i_{\mathsf{Node}} \circ (\mathrm{id}_G)_{\mathsf{Blank}} = i_{\mathsf{Node}} \circ \mathrm{incl}_{G_{\mathsf{Blank}}, G_{\mathsf{Node}}} \circ \mathrm{id}_{G_{\mathsf{Blank}}} = i_{\mathsf{Blank}}$.
The functor properties of Incl obviously follow from homomorphisms being a special case of instantiations. □

We now have that RDF graph instantiations exactly characterise semantic entailment in the opposite direction. This theorem corresponds to the Interpolation Lemma in [Hay04] stating that "S entails a graph E if and only if a subgraph of S is an instance of E", where the instantiation and the subgraph inclusion are combined in our notion of instantiation from E to S as already alluded to above.

Theorem 2.1 (Characterisation of Entailment by Instantiations)
Given two RDF graphs G and H, $G \Vdash H$ if and only if there exists and RDF graph instantiation $i \colon H \to G$.

Beweis. **If:** Suppose, there is an instantiation $i \colon H \to G$. For all intepretations I with $I \models G$, we have $\mathrm{Voc}(G) \subseteq I_{\mathsf{Voc}}$, and, since $\mathrm{Voc}(H) \subseteq \mathrm{Voc}(G)$ by the underlying vocabulary functor, also $\mathrm{Voc}(H) \subseteq I_{\mathsf{Voc}}$. For all $(s, p, o) \in H_{\mathsf{Triple}}$, we have $(i_{\mathsf{Node}}(s), p, i_{\mathsf{Node}}(o)) \in G_{\mathsf{Triple}}$ by i being an instantiation and, hence, also $\mathrm{uint}_I(p) \in I_{\mathsf{Prop}}$ by $I \models G$. Moreover, there is at least one assignment $asg \colon G_{\mathsf{Blank}} \to I_{\mathsf{Res}}$ with $(\overline{asg}(s'), \overline{asg}(o')) \in \mathrm{pext}_I(\mathrm{uint}_I(p))$ for all $(s', p, o') \in G_{\mathsf{Triple}}$. We obtain an assignment $asg' \colon H_{\mathsf{Blank}} \to I_{\mathsf{Res}}$ by $asg' := \overline{asg} \circ i_{\mathsf{Blank}}$, and, hence, $\overline{asg'} = \overline{asg} \circ i_{\mathsf{Node}}$, with $(\overline{asg'}(s), \overline{asg'}(o)) = (\overline{asg}(i_{\mathsf{Node}}(s)), \overline{asg}(i_{\mathsf{Node}}(o))) \in \mathrm{pext}_I(\mathrm{uint}_I(p))$ for all $(s, p, o) \in H_{\mathsf{Triple}}$. In summary, this means that $I \models H$ and, since this is true for all $I \models G$, also $G \Vdash H$.

Only if: Suppose, that $G \Vdash H$ holds. The interpretation I, constructed by

- $I_{\mathsf{Voc}} := \mathrm{Voc}(G)$,

[10] Here and in the following, $\mathrm{incl}_{S,S'} \colon S \to S'$ denotes the inclusion $S \subseteq S'$ viewed as an injective function mapping elements identically. See also Proposition A.4.

- $I_{\mathrm{Res}} := G_{\mathrm{Blank}} + \mathrm{Voc}(G)_{\mathrm{URI}} + \mathrm{Voc}(G)_{\mathrm{PLit}} + \mathrm{Voc}(G)_{\mathrm{TLit}}$,
- $I_{\mathrm{Val}} := \mathrm{Voc}(G)_{\mathrm{PLit}}$,
- $I_{\mathrm{Prop}} := \{p \in \mathrm{Voc}(G)_{\mathrm{URI}} \mid \exists (s, p, o) \in G_{\mathrm{Triple}}\}$,
- $\mathrm{pext}_I(p) := \{(s, o) \mid \exists (s, p, o) \in G_{\mathrm{Triple}}\}$,
- $\mathrm{uint}_I := \mathrm{incl}_{\mathrm{Voc}(G)_{\mathrm{URI}}, I_{\mathrm{Res}}}$ and
- $\mathrm{tint}_I := \mathrm{incl}_{\mathrm{Voc}(G)_{\mathrm{TLit}}, I_{\mathrm{Res}}}$

obviously satisfies G by the assignment $asg' := \mathrm{incl}_{G_{\mathrm{Blank}}, I_{\mathrm{Res}}}$. Hence, I also satisfies H by an assignment $asg\colon H_{\mathrm{Blank}} \to I_{\mathrm{Res}}$ because of $G \Vdash H$. Since $I_{\mathrm{Res}} \subseteq G_{\mathrm{Node}}$ we can define an instantiation $i\colon H \to G$ by $i_{\mathrm{Blank}} := \mathrm{incl}_{I_{\mathrm{Res}}, G_{\mathrm{Node}}} \circ asg$, and, hence, $i_{\mathrm{Node}} = \overline{asg}$, which constitutes an instantiation because for each $(s, p, o) \in H_{\mathrm{Triple}}$ we have $(\overline{asg}(s), \overline{asg}(o)) \in \mathrm{pext}_I(p)$ by $I \models H$ and, hence, $(i_{\mathrm{Node}}(s), p, i_{\mathrm{Node}}(o)) \in G_{\mathrm{Triple}}$ by the construction of I and the definition of i. $\square$

Most of the results from [Hay04] can be recovered in our category theoretical setting quite easily. More specifically, the Empty Graph Lemma ("The empty set of triples is entailed by any graph, and does not entail any graph except itself.") corresponds to the fact that the empty RDF graph is the only initial object in **RDFInst**, the Subgraph Lemma ("A graph entails all its subgraphs.") is obtained by subgraph inclusions (and also the more general monomorphisms in our setting) being homomorphisms and, hence, also instantiations, the Instance Lemma ("A graph is entailed by any of its instances.") holds because the relation between a graph and an instance in the sense of [Hay04] is a special case of our instantiation and the Merging Lemma ("The merge of a set S of RDF graphs is entailed by S and entails every member of S.") follows from the injections into the coproduct being instantiations.

In [Bag05], another characterisation of RDF entailments as graph homomorphisms is given. Although there are strong similarities to our approach above, the aims are different. While [Bag05] makes available results for directed, labelled multigraphs and directed, labelled hypergraphs by giving a translation of RDF graphs and, in the latter case, also interpretations to these well-known mathematical structures, our aim is to provide a rigorous formal foundation for the RDF abstract syntax and semantics themselves without translating them to another domain.

Our categorical formalisation can be enhanced by defining intepretation morphisms in a straightforward way yielding a category of interpretations. A model functor selecting the subcategory of all interpretations satisfying a given RDF graph would then give rise to a specification frame in the sense of [EG94], where the construction of the instantiation in the "only if" part of Theorem 2.1 can be used as an initial model of a graph. The theory of specification frames is closely related to institutions, treated in [GB92]. While a specification frame considers specifications without substructure, an institution differentiates between signatures and sentences over these signatures. In [LLD06], a treatment of RDF in the framework of institutions is presented, where sets of resource references, without distinction between blank nodes, URIs and literals, are considered as signatures and single triples as sentences. Using the formalisations presented above and considering vocabularies as signatures and whole RDF graphs as sentences could lead to a significant refinement. A more thorough examination is, however, outside the scope of this thesis.

2.3. Schemas

In this section, we will introduce the means for defining schemas in RDF and for typing RDF graphs over these schemas. While edges, i. e., triples, are typed by the predicates in RDF, nodes can be typed using triples with the predefined predicate $\mathrm{rdf{:}type}$ between an instance and its class. In order to allow the definition of classes, predicates and their connections in a vocabulary, RDF Schema, defined in [BG04], provides a set of predicates and classes in the $\mathrm{rdfs{:}}$ namespace to talk about predicates and classes.

The vocabularies in [KC04] and [BG04] are rather extensive, containing some redundancies and utility vocabulary for features like reification (i. e., triples about triples), collections and linked lists. In [MPG07] it is shown that a small subset, called ρdf, is sufficient to argue about the semantics of RDF Schema. Hence, we will also confine ourselves to this subset in the following.

Definition 2.12 (ρdf Vocabulary)
The *ρdf vocabulary* is given by

$$\rho\text{df} := \{\mathrm{rdf{:}type}, \mathrm{rdfs{:}domain}, \mathrm{rdfs{:}range}, \mathrm{rdfs{:}subClassOf}, \mathrm{rdfs{:}subPropertyOf}\}.$$

The URIs in this vocabulary are abbreviated by type, dom, range, sc and sp, respectively.

The semantics of this vocabulary is given by restricting the class of intepretations appropriately. More specifically, all elements of the ρdf vocabulary, the subjects of interpretations of dom and range and the subjects and objects of interpretations of sp are required to be interpreted as properties. The interpretations of sc and sp have to be transitive.[11] The predicate type is interpreted as a property linking instances to classes.[12] The predicates dom and range are intended to determine the classes of subjects and objects of properties, sc is intepreted as a subclass property in the sense that instances of the subject are also instances of the object and sp as a relation between properties such that all pairs in the subject property are also in the object property.

Definition 2.13 (ρdf Interpretation)
A *ρdf interpretation* is an interpretation I with $\rho\text{df} \subseteq I_{\text{URI}}$ satisfying

[11] In [Hay04] and [MPG07], they are also required to be reflexive, but, since it is also shown in [MPG07] that the only effect of reflexivity is the entailment of the corresponding reflexive triples, we choose to drop this requirement from the beginning.

[12] RDF and RDF Schema interpretations in [Hay04] and ρdf interpretations in [MPG07] include an additional subset of I_{Res} and an additional function for the extension of classes, where this extension has to be equivalent to the extension of type. We choose to avoid this structural overhead, leave the definition of interpretations unmodified and use the extension of type directly.

for all ρdf predicates:

- if $u \in \rho\text{df}$ then $\text{uint}_I(u) \in I_{\text{Prop}}$,

for $\mathrm{rdfs{:}domain}$ and $\mathrm{rdfs{:}range}$:

- if $(p, d) \in \text{pext}_I(\text{uint}_I(\mathrm{dom}))$ or $(p, r) \in \text{pext}_I(\text{uint}_I(\mathrm{range}))$ then $p \in I_{\text{Prop}}$,
- if $(p, d) \in \text{pext}_I(\text{uint}_I(\mathrm{dom}))$ and $(x, y) \in \text{pext}_I(p)$ then $(x, d) \in \text{pext}_I(\text{uint}_I(\mathrm{type}))$,
- if $(p, r) \in \text{pext}_I(\text{uint}_I(\mathrm{range}))$ and $(x, y) \in \text{pext}_I(p)$ then $(y, r) \in \text{pext}_I(\text{uint}_I(\mathrm{type}))$,

for $\mathrm{rdfs{:}subClassOf}$:

- if $(c, d) \in \text{pext}_I(\text{uint}_I(\mathrm{sc}))$ and $(d, e) \in \text{pext}_I(\text{uint}_I(\mathrm{sc}))$ then $(c, e) \in \text{pext}_I(\text{uint}_I(\mathrm{sc}))$,
- if $(x, c) \in \text{pext}_I(\text{uint}_I(\mathrm{type}))$ and $(c, d) \in \text{pext}_I(\text{uint}_I(\mathrm{sc}))$ then $(x, d) \in \text{pext}_I(\text{uint}_I(\mathrm{type}))$,

for $\mathrm{rdfs{:}subPropertyOf}$:

- if $(o, p) \in \text{pext}_I(\text{uint}_I(\mathrm{sp}))$ and $(p, q) \in \text{pext}_I(\text{uint}_I(\mathrm{sp}))$ then $(o, q) \in \text{pext}_I(\text{uint}_I(\mathrm{sp}))$ and
- if $(p, q) \in \text{pext}_I(\text{uint}_I(\mathrm{sp}))$ then $p, q \in I_{\text{Prop}}$ and $\text{pext}_I(p) \subseteq \text{pext}_I(q)$.

The class of all ρdf interpretations is denoted by $\text{Interp}_{\rho\text{df}}$.

An example for the usage of this vocabulary is given in Figure 2.4, where a schema for the example application in the metadata scenario is defined. This schema was already used in the RDF graph in Figure 2.1. It defines an abstract class bib:Publication with subclasses bib:Book and bib:Article and a class bib:Person to be used for authors. The predicate bib:author is declared to have subjects of type bib:Publication and objects of type bib:Person. For the predicates bib:title and bib:name only the domains are given as bib:Publication and bib:Person, respectively, while the ranges are not declared since the intended range of plain literals cannot be identified.[13]

The satisfaction relation of Section 2.2 does not have to be modified for ρdf, but the restricted class of interpretations leads to a stronger version of entailment.

Definition 2.14 (ρdf Entailment)
Given two RDF graphs G and H, G *ρdf entails* H, written as $G \Vdash_{\rho\text{df}} H$, if and only if $I \models G$ implies $I \models H$ for all interpretations $I \in \text{Interp}_{\rho\text{df}}$.

[13] In fact, RDF Schema provides a URI rdfs:Literal as the class of all literals, but no identifier for *plain* literals.

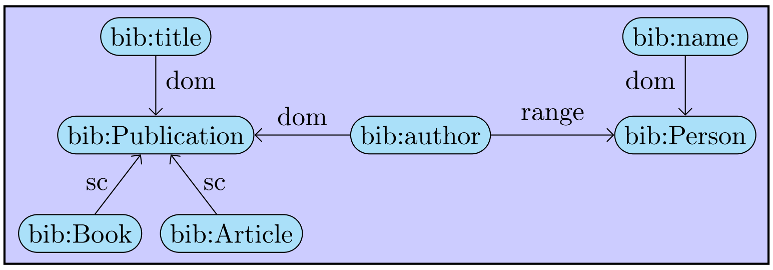


Abbildung 2.4.: Schema for metadata application scenario

A syntactic characterisation of $G \Vdash_{\rho\mathrm{df}} H$ can be obtained by applying inference rules to G before trying to find an instantiation $i\colon H \to G$ as in Theorem 2.1. In Section 4.4 we will give these inference rules as graph transformation rules in the sense of this thesis.

In the theory of algebraic graph transformation as well as in the area of model-driven development a quite different approach to schemas and instances of these schemas is usually taken, where there is a distinct type graph or metamodel over which the graphs or models are typed. The main differences between these approaches and the RDF approach are summarised in Table 2.1. One of the main differences is that in metamodelling and typed graphs the models or graphs have to structurally conform to their schema, while in RDF the usage of types is optional and its consequences are defined on the semantics rather than directly on the syntactical structures. This leads to a more flexible approach, where, e. g., elements in an RDF graph may be typed over several schemas, instances of different schemas can occur mixed in a single RDF graph, or hierarchies of schemas with an arbitrary depth can be created.

Additionally to the intensional semantics given above, an extensional semantics for RDF Schema is defined in [Hay04], where the properties dom, range, sc and sp not only imply their corresponding characterisations, but also reflect them. This means, e. g., that for properties p and p' for which $\mathrm{pext}_I(p) \subseteq \mathrm{pext}_I(p')$ holds in an interpretation I it is also required that $(p, p') \in \mathrm{pext}_I(\mathrm{uint}_I(\mathrm{sp}))$. This leads to a still stronger entailment relation with the drawback that its syntactic characterisation is more complex.

A similar approach to the one used in this section can be employed to define further semantic extensions. The Web Ontology Language (OWL) described in [MH04], e. g., defines a much richer vocabulary on top of RDF Schema to allow the definition of more expressive ontologies. The semantics of OWL is then given by further restricting the class of interpretations to only those who interpret the OWL vocabulary in the intended way. An interesting line of future work would be to examine if such an approach can be feasibly enhanced to achieve a "semantics construction kit", where the semantics of language families (e. g., the DSMLs in our application scenario) can be defined dynamically, where not only structural but also behavioural features are considered.

Tabelle 2.1.: Comparison of metamodelling/typed graphs and RDF/RDF Schema

Metamodelling/Typed Graphs	**RDF/RDF Schema**
There is a *distinct* metamodel/type graph that typically is *not changed* frequently.	The schema definition can be *contained* in the same RDF graph as its instances (possibly imported from elsewhere) and *changed* dynamically.
A typing function assigns *exactly one type* (with possible supertypes) to each element.	The type predicate relates nodes to an *arbitrary number of types* (even none), while the types of edges, i. e., the predicates of triples, are still unique. Predicates can but *do not have to be declared* in a schema.
Models/graphs have to *conform to* a metamodel/type graph.	Typings w. r. t. a schema can be *inferred* from partial information.

2.4. Datatypes

In order to give a semantics to the typed literals used in RDF graphs, we need some way to relate the string representations used in the graphs to actual values of a datatype. Thus, a datatype consists of a lexical space for the string representations, a value space for the actual values and a mapping function between them. Since RDF is used for data structures and not for any operational behaviour this is sufficient to define datatypes in this context.

Definition 2.15 (Datatype)
A *datatype* $DT = (DT_{\text{Lex}}, DT_{\text{Val}}, \text{l2v}_{DT})$ consists of

- a set $DT_{\text{Lex}} \subseteq \text{String}$ of strings, called *lexical space*,
- a set DT_{Val}, called *value space* and
- a function $\text{l2v}_{DT} : DT_{\text{Lex}} \to DT_{\text{Val}}$, called *lexical-to-value mapping*.

The class of all possible datatypes is denoted by Datatype.

The relation to the type URIs used in typed literals is established by assuming that a datatype map is given which assigns datatypes as defined above to a set of URIs.

Definition 2.16 (Datatype Map)
A *datatype map* $D = (D_{\text{URI}}, \text{map}_D)$ consists of

- a set $D_{\text{URI}} \subseteq \text{URI}$ of URIs and

- a function $\mathrm{map}_D \colon D_{\mathrm{URI}} \to \mathrm{Datatype}$.

When such a datatype map is given, the class of semantic interpretations can be further restricted to conform to this datatype map in the sense that typed literals for which the datatype URI is contained in the map are interpreted according to the map, where ill-formed literals, i. e., literals, where the string is not in the lexical space of the corresponding datatype, are required to be interpreted as some resource which is not a literal value. Moreover, the value spaces of datatypes in the map are required to be included in the set of literal values of the interpretation and the values in these spaces are typed by the interpretation of the datatype URI.[14]

Definition 2.17 (D-Interpretation)
Given a datatype map D, a *D-interpretation* is a ρdf interpretation I with $D_{\mathrm{URI}} \subseteq I_{\mathrm{URI}}$ satisfying for each $d \in D_{\mathrm{URI}}$

- $\mathrm{map}_D(d)_{\mathrm{Val}} \subseteq I_{\mathrm{Val}}$,
- $(v, \mathrm{uint}_I(d)) \in \mathrm{pext}_I(\mathrm{uint}_I(\mathrm{type}))$ if and only if $v \in \mathrm{map}_D(d)_{\mathrm{Val}}$,
- if $(d, s) \in I_{\mathrm{TLit}}$ and $s \in \mathrm{map}_D(d)_{\mathrm{Lex}}$ then $\mathrm{tint}_I(d, s) = \mathrm{l2v}_{\mathrm{map}_D(d)}(s)$ and
- if $(d, s) \in I_{\mathrm{TLit}}$ and $s \notin \mathrm{map}_D(d)_{\mathrm{Lex}}$ then $\mathrm{tint}_I(d, s) \notin I_{\mathrm{Val}}$.

The class of all D-interpretations is denoted by Interp_D.

Note that D-interpretations introduce the possibility of contradictory RDF graphs. If, e. g., the range of a predicate p is defined to be a datatype $d \in D_{\mathrm{URI}}$ by a triple (p, range, d) and the object o of a triple (s, p, o) using this predicate is an ill-formed typed literal $o = (d, s)$ with $s \notin \mathrm{map}_D(d)_{\mathrm{Lex}}$ then there can be no D-interpretation satisfying a graph containing both triples since the interpretation of the literal is required to be both inside I_{Val} and outside of I_{Val} by different parts of the semantics which is impossible.

A notion of D-entailment is again achieved by just restricting the considered interpretations to D-interpretations which again leads to a stronger entailment than ρdf entailment. For example, contradictory graphs as the one sketched above entail every other graph.

Definition 2.18 (D-Entailment)
Given a datatype map D and two RDF graphs G and H, G *D-entails* H, written as $G \Vdash_D H$, if and only if $I \models G$ implies $I \models H$ for all interpretations $I \in \mathrm{Interp}_D$.

We will not consider typed literals further in this thesis since our transformations will just leave them unmodified or replace them by a possibly completely different literal at

[14] In [Hay04] the datatypes themselves are used as interpretations of datatype URIs which we have omitted here to reduce complexity. Moreover, the predefined datatype rdf:XMLLiteral is assumed to be contained in every datatype map which is not needed in the context of this thesis.

the user's choice. It would, however, be worthwhile to explore the possibilities of adding a notion of operations and allowing transformations to apply these operations on typed literals.

3. MPOC Transformation Framework

In this chapter the transformation framework which is instantiated for RDF graphs in the next chapter is introduced. First, we will motivate the introduction of rule-based transformations by some use cases and give an overview of existing approaches and frameworks in Section 3.1. Then, the basic notions are introduced in Section 3.2. Our main results regarding the sequential composition of rules are shown in Section 3.3. Finally, negative application conditions are introduced in Section 3.4 to further restrict the applicability of rules under certain circumstances.

3.1. Rule-Based Transformations

Rule-based, algebraic approaches to modifications of graphs and similar structures have been studied for a long time. We will use such an approach in this thesis in order to formalise modifications of RDF graphs. Such a formal concept has some serious advantages over, e. g., a programmatic approach, where some kind of ad-hoc language is used to define transformations.

3.1.1. Use Cases for Transformations

Algebraic, rule-based transformations are a common formal framework for various kinds of transformations. Firstly, a set of rather small transformation rules can be used to define a grammar for a language of the structures that are transformed. This is similar to Chomsky grammars for textual languages and allows us to distinguish between allowed and disallowed structures among those that are in principle possible. We will use such grammars to define the RDF structures expected by the example application in the metadata application scenario in Section 5.1.1 and to define the example DSML in Section 5.2.1.

Secondly, more complex rules can be used to allow complex modification operations to be applied in one step. In order to stay within the language of allowed structures defined by the grammar, it is possible to compose such complex rules from the grammar rules. We will introduce such a composition operation later in this chapter and formally prove that the effect of this rule can also be achieved using the constituent rules and, therefore, do not leave the language of reachable structures. Such complex rules can be used to define refactorings or similar large operations that are meant to be applied as a unit. In Section 5.2.2 we will construct such a rule in the DSML scenario, where an unprotected connection in an IT landscape is replaced by one that is protected by a firewall.

Thirdly, rules can also be meant to be applied automatically as long as possible instead of on demand by the user. Such rules can also be guaranteed to stay within the corresponding language by composing them from grammar rules. We will use them to integrate the imported bibliographical data into the schema expected by the application in the metadata scenario in Section 5.1.2 and to adapt models in the DSML scenario to an evolution of the language in Section 5.2.3.

3.1.2. Approaches for Rule-Based Transformations

In the following, a brief review of existing approaches to rule-based graph transformations is given, where we focus on frameworks using category theory as it is also done by our approach presented in the subsequent sections. Category theory (cf. Appendix A and, e. g., [AHS90]) allows to talk about properties of a large variety of mathematical structures in a common framework. Thus, transformation frameworks that are formulated in categorical terms can also be applied to various structures without the need to redevelop the theoretical underpinnings.

To achieve this goal, category theory does not talk about the internals of objects but just about their relations to other objects, called morphisms in category theory. We have already seen some examples of categories for RDF graphs in the preceding chapter and will introduce another one in the following chapter where variables for transformations are introduced into RDF graphs.

Transformations should be able to add and delete structures. The addition is formalised quite easily by a pushout (cf. Definition A.7) in category theory which intuitively corresponds to a (disjoint) union under some common interface in most categories. The different approaches vary in the solution for the deletion of elements.

The historically first approach is the double pushout (DPO) approach, introduced in [EPS73], for which an early presentation of the complete theory for graphs can be found in [Ehr79]. It uses pushout complements as a means for deletion, i. e., one searches for that structure to which the parts that are to be deleted have to be added (by a pushout) in order to reconstruct the original object. Thus, a DPO rule is given by a left-hand side L, an interface I and a right-hand side R and morphisms from the interface into both sides and it is applied to a host object G by a match from the left-hand side and two pushouts completing the diagram in Figure 3.1a to arrive at the result H with a comatch from the right-hand side.

Pushout complements do not exist in all cases since some of the elements that shall be deleted could be identified by the match or be used in additional structure in the host object. In this case a reconstruction of the host object by a pushout is not possible since information about these identifications and dangling structures is not preserved after deleting them.

In order to overcome this drawback the single pushout (SPO) approach was developed in [LE90], where addition and deletion are done in one step by a pushout along a partial morphism (cf. Figure 3.1b). Since there has to be a morphism from R to H every element from L that has no image due to partiality has to be deleted in H even if this means that structures that were connected or other elements that where identified to that element also have to be deleted. The latter possibility means that the comatch may be partial even if the match was a total morphism in the case that a deleted element is identified to a preserved one which then has no possible image.

This issue is, i. a., solved by the sesqui pushout (SqPO) approach, which was developed in [CHHK06], where deletion is done by a final pullback complement (cf. Figure 3.1c). Intuitively, a pullback (cf. Definition A.7) is an intersection over a common superstructure (dual to a pushout being a union under a common substructure). Thus, the interface *I* being a pullback means that it is the intersection of the object *D* obtained after deletion and the left-hand side over the original host object *G*. There are, however, usually a lot of objects satisfying this. In an extreme case, everything but the image of the interface can be deleted and the result is still a pullback. Therefore, the final pullback complement is chosen from these possibilities, where as much as possible is preserved.

$$\begin{tikzcd}
L \arrow[d] \arrow[dr, phantom, "(PO)"] & I \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(PO)"] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

(a) DPO approach

$$\begin{tikzcd}
L \arrow[r, "\circ" {marking, pos=0}] \arrow[d] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "\circ" {marking, pos=0}] \\
G \arrow[r, "\circ" {marking, pos=0}] & H
\end{tikzcd}$$

(b) SPO approach

$$\begin{tikzcd}
L \arrow[d] \arrow[dr, phantom, "(FPBC)"] & I \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(PO)"] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

(c) SqPO approach

Abbildung 3.1.: Approaches to categorical transformation

While a rule is not applicable in the DPO approach if there are dangling structures or identifications and both are deleted in the SPO approach, the SqPO approach represents an intermediate solution, where dangling structures are deleted, since otherwise the intersection by a pullback would still contain them, but pullback complement does not exist and the rule is, hence, not applicable if deleted elements are identified to preserved ones, since the interface contains only the preserved one but a pullback would contain either both if their image is preserved or none if it is deleted.

From these possible approaches we choose the DPO approach in this thesis since, on the one hand, we do not want the side effect of deleting dangling structures and, on the other hand, the available theory is by far the most comprehensive one for this approach.

The DPO approach was generalised from graphs to more general kinds of categories by the definition of high level replacement (HLR) categories and systems in [EHKP91], where a collection of properties is given that a category has to satisfy in order to obtain the possibility of DPO transformations and the theoretical results for them. This concept was refined in [EEPT06] by replacing most of the required properties by the requirement that the category in which the transformations should happen is (a generalistion of) a so-called adhesive category.

For RDF structures, we, however, do not have unique pushout complements due to the fact that morphisms are inclusions on triple sets. This makes it possible to preserve a triple in a pushout complement even if it is deleted in the interface. In Figure 3.2 we see two pushout complements for the same given situation, where a triple is deleted in the interface. In one pushout complement the corresponding triple is also deleted, while it is preserved in the other one. Since triples do not carry an identity in RDF graphs, they cannot be disjointly unified and, hence, both pushout complements lead to the same pushout.

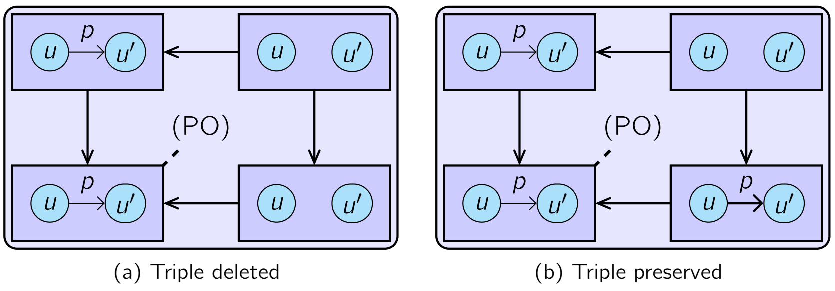


Abbildung 3.2.: Non-unique pushout complements for RDF graphs

Since adhesiveness implies uniqueness of pushout complements this also makes categories for RDF graphs intrinsincally non-adhesive (except for the choice of triple set equalities as morphisms in the rules which would, however, void the purpose of transformations). A subset of the HLR properties in [EHKP91] are still satisfied for RDF categories but non-uniqueness of pushout complements still makes transformations non-deterministic which is not desirable.

Therefore, we will introduce a new variant of the DPO approach, first considered in [BB08], in the following sections, where deletion is achieved by minimal pushout complements instead of arbitrary ones. In a way, this is related to the SqPO approach choosing final pullback complements and, in fact, we will show later that for RDF structures, and presumably other similar categories, a minimal pushout complement is exactly a pushout that is also a pullback.

3.2. MPOC-PO Transformations

In this section we will present the basic notions of our MPOC-PO transformation approach. As already mentioned above, this is a variant of the DPO approach, where the first square is not an arbitrary pushout but a minimal pushout complement.

3.2.1. Minimal Pushout Complements

The need for minimal pushout complements arises because in some categories, namely the ones used for RDF graphs in this thesis, pushout complements are not unique. They may preserve elements that are deleted in the interface. This problem arises because the triples in RDF graphs are uniquely given by their subject, predicate and object, and are, hence, a pushout identifies triples that are identical even if they have no common preimage in the interface. For multigraphs, usually considered in algebraic graph transformation, this problem is avoided by edges having their own identities leading to pushouts disjointly adding them if they have no common preimage.

In order to get a unique result for the left-hand pushout in a DPO transformation, more specifically, the presumably intended result, we require it to delete as much as possible, i. e., to be the smallest possible pushout complement. In terms of category theory, this is expressed by the requirement that the minimal pushout complement is initial among all pushout complements.

Definition 3.1 (Minimal Pushout Complement)
Given two morphisms $l\colon I \to L$ and $m\colon L \to G$ in an arbitrary category $\mathbf{C}$ (cf. Figure 3.3a), the category $\mathbf{POC}(l, m)$ of *pushout complements of l and m* consists of

objects $(D, f\colon D \to G, i\colon I \to D)$ such that (G, f, m) is a pushout of l and i (cf. Figure 3.3b),

morphisms $d\colon D \to D'$ between objects (D, f, i) and (D', f', i') such that $d \circ i = i'$ and $f' \circ d = f$ and

compositions and identities given by compositions and identities in $\mathbf{C}$.

A *minimal pushout complement (MPOC) of l and m* is an initial object in $\mathbf{POC}(l, m)$ (cf. Figure 3.3c).

We now want to characterise when such an MPOC exists. Intuitively an MPOC should delete everything in the left-hand side but not in the interface or equivalently delete the left-hand side and then add again the interface. In most categories, we may, however, not delete all elements of the left-hand side because structures are connected to them or they are identified by the match to the host graph. In order to formalise this, the notion of initial pushout is introduced, which constructs the smallest possible boundary and context, such that a given match can be reconstructed by a pushout of left-hand side and context under the boundary. This can be seen as a category theoretical abstraction

$$\begin{tikzcd}
L \arrow[d, "m"'] & I \arrow[l, "l"'] \\
G &
\end{tikzcd}$$

(a) Given situation

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & I \arrow[l, "l"'] \arrow[d, "i"] \\
G & D \arrow[l, "f"]
\end{tikzcd}$$

(b) POC of l and m

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(MPOC)"] & I \arrow[l, "l"'] \arrow[d, "i"] \arrow[ddr, bend left, "i'"] & \\
G & D \arrow[l, "f"] \arrow[dr, "\exists ! d"] \arrow[r, phantom, "(=)"] \arrow[d, phantom, "(=)"] & \\
 & & D' \arrow[llu, bend left, "f'"]
\end{tikzcd}$$

(c) MPOC of l and m

Abbildung 3.3.: Pushout complements and MPOCs of l and m

of the complement in sets, which, in fact, is the context for an initial pushout in **SetIncl**, where the boundary is always empty since set inclusions do neither identify elements nor glue additional structures to them.

Definition 3.2 (Initial Pushout)
Given a morphism $m\colon L \to G$ in an arbitrary category $\mathbf{C}$ with a distinguished class $\mathcal{M}$ of monomorphisms which is closed under composition, i. e., $f, g \in \mathcal{M}$ implies $g \circ f \in \mathcal{M}$, and decomposition, i. e., $g \circ f \in \mathcal{M}$ implies $f \in \mathcal{M}$, the category $\mathbf{PO}(m)$ of *pushouts over m* consists of

objects $(B, b\colon B \to L, C, c\colon C \to G, m_B\colon B \to C)$ such that $b, c \in \mathcal{M}$ and (G, c, m) is pushout of b and m_B (cf. Figure 3.4a),

morphisms $(b^*\colon B \to B', c^*\colon C \to C')$ between objects (B, b, C, c, m_B) and $(B', b', C', c', m_{B'})$ such that $b' \circ b^* = b$, $c' \circ c^* = c$ and $m_{B'} \circ b^* = c^* \circ m_B$ (where $b^*, c^* \in \mathcal{M}$ due to decomposition) and

compositions and identities given by componentwise compositions and identities in $\mathbf{C}$.

An *initial pushout (IPO) over m* is an initial object (B, b, C, c, m_B) in $\mathbf{PO}(m)$ (cf. Figure 3.4b), where B and b are called *boundary object* and *boundary morphism* and C and c *context object* and *context morphism*, respectively.
An initial pushout is called *strong* if $(C', c^*, m_{B'})$ is a pushout of b^* and m_B for all comparison pushouts, *weak* otherwise.[1]

[1] In [EEPT06] only strong IPOs are considered, while we need weak IPOs in the context of this thesis, since IPOs for RDF graphs are weak. In fact, the existence of strong IPOs would imply unique pushout complements.

$$\begin{tikzcd}
B \arrow[r, hook, "b"] \arrow[d, "m_B"'] \arrow[dr, phantom, "(PO)"] & L \arrow[d, "m"] \\
C \arrow[r, hook, "c"] & G
\end{tikzcd}$$

(a) Pushout over m

$$\begin{tikzcd}
B \arrow[r, hook, "b"] \arrow[d, "m_B"'] \arrow[dr, phantom, "(IPO)"] \arrow[rr, hook, bend left, "\exists ! b^*", "(=)"'] & L \arrow[d, "m"] \arrow[dr, phantom, "(PO)"] & B' \arrow[l, hook', "b'"'] \arrow[d, "m_{B'}"] \\
C \arrow[r, hook, "c"] \arrow[rr, hook, bend right, "\exists ! c^*"', "(=)"] & G & C' \arrow[l, hook', "c'"']
\end{tikzcd}$$

(b) IPO over m

Abbildung 3.4.: Pushouts and initial pushouts over m

A first observation is that initial pushouts are minimal pushout complements of the boundary and the given morphism.

Lemma 3.1 (IPOs are MPOCs)
Given a morphism $m\colon L \to G$ with IPO (B, b, C, c, m_B), then (C, c, m_B) is an MPOC of b and m.

Beweis. For each other pushout complement (D, f, i) of b and m we have that (B, b, D, f, i) is a pushout over m and, hence, there are unique morphisms $b^*\colon B \to B$ and $c^*\colon C \to D$ w. r. t. $b \circ b^* = b$, $f \circ c^* = c$ and $i \circ b^* = c^* \circ m_B$. Obviously, this implies $b^* = \mathrm{id}_B$ and, thus, c^* is unique w. r. t. $f \circ c^* = c$ and $i = c^* \circ m_B$. This is exactly the comparison morphism required for (C, c, m_B) to be an MPOC. □

The construction of an MPOC is only possible if the gluing points in the boundary, i. e., the elements necessary to reconstruct the match by a pushout, are preserved in the interface. Using initial pushouts this can be formalised by the requirement that there is a morphism from the boundary into the interface.

Definition 3.3 (Gluing Condition)
Given a category $\mathbf{C}$, morphisms $l\colon I \to L$ with $l \in \mathcal{M}$ and $m\colon L \to G$ with IPO (B, b, C, c, m_B) over m, l and m satisfy the *gluing condition* if there is a morphism $b^*\colon B \to I$ with $b^* \in \mathcal{M}$ and $l \circ b^* = b$ (cf. Figure 3.5a).

If this condition is met we can construct an MPOC by adding the interface to the context, i. e., building a pushout of interface and context under the boundary. The following theorem also states that, in a category with IPOs, this characterisation is not only sufficient but also necessary for MPOCs in the sense that it can be applied if any pushout complement exists at all.

Theorem 3.1 (Construction of MPOCs by IPOs)
Given a category $\mathbf{C}$ with pushouts preserving $\mathcal{M}$, morphisms $l\colon I \to L$ with $l \in \mathcal{M}$ and $m\colon L \to G$ with IPO (B, b, C, c, m_B) over m,

1. if l and m satisfy the gluing condition then an MPOC (D, f, i) of l and m can be constructed by a pushout (D, c^*, i) of b^* and m_B and the unique morphism $f\colon D \to G$ w. r. t. $f \circ c^* = c$ and $f \circ i = m \circ l$ induced by G with $c \circ m_B = m \circ b = m \circ l \circ b^*$ being a comparison object for this pushout (cf. Figure 3.5b),

2. if there is a pushout complement (D', f', i') of l and m then l and m satisfy the gluing condition and

3. if there is an MPOC (D, f, i) of l and m with morphisms $b^*\colon B \to I$ and $c^*\colon C \to D$ induced by the IPO then (D, c^*, i) is a pushout of b^* and m_B.

Beweis.

1. Firstly, (G, f, m) is a pushout of l and i and, hence, (D, f, i) a pushout complement of l and m due to pushout decomposition (cf. Proposition A.11) and $f \in \mathcal{M}$ due to pushouts preserving $\mathcal{M}$ (cf. Figure 3.5c).
Secondly, (D, f, i) is initial in $\mathbf{POC}(l, m)$ and, hence, an MPOC: For each other pushout complement (D', f', i') of l and m we have that (G, f', m) is a pushout of l and i by definition. The initiality of the IPO then induces unique morphisms $b'\colon B \to I$ and $c'\colon C \to D'$ with $l \circ b' = b$, $f' \circ c' = c$ and $i' \circ b' = c' \circ m_B$ (cf. Figure 3.5d). Since l is a monomorphism and $l \circ b' = b = l \circ b^*$ we have $b' = b^*$. From $i' \circ b^* = i' \circ b' = c' \circ m_B$ we have that D' is a comparison object for the pushout (D, c^*, i) of b^* and m_B and, thus, there is a unique morphism $d\colon D \to D'$ with $d \circ c^* = c'$ and $d \circ i = i'$ (cf. Figure 3.5e). It follows that $f' \circ d \circ i = f' \circ i' = m \circ l = f \circ i$ and $f' \circ d \circ c^* = f' \circ c' = c = f \circ c^*$. Since (D, c^*, i) is a pushout and pushouts are jointly epimorphic (cf. Proposition A.10) it follows that $f' \circ d = f$.
It remains to show that d is unique w. r. t. $f' \circ d = f$ and $d \circ i = i'$. Suppose there is $d'\colon D \to D'$ with $f' \circ d' = f$ and $d' \circ i = i'$. It follows that $f' \circ d' \circ c^* = f \circ c^* = c = f' \circ c'$ and due to f' being a monomorphism also $d' \circ c^* = c'$. Since d is unique w. r. t. $d \circ c^* = c'$ and $d \circ i = i'$ we have $d' = d$.
Thus, (D, f, i) is an MPOC of l and m.

2. For a pushout complement (D', f', i') of l and m initiality of the IPO induces unique morphisms $b'\colon B \to I$ and $c'\colon C \to D'$ with $l \circ b' = b$, $f' \circ c' = c$ and $i' \circ b' = c' \circ m_B$ (cf. Figure 3.5d). The gluing condition for l and m is then satisfied by using $b^* = b'$.

3. Since the construction in 1. applied to the given morphism b^* yields an MPOC of l and m and MPOCs are unique up to isomorphism the result of the construction has

to be isomorphic to D. But since pushouts are also only unique up to isomorphism (D, c^*, i) can be chosen as the pushout in the construction. □

3.2.2. Transformation Rules and Transformations

In order to define transformation rules and transformations in our approach, we first summarise the requirements on categories for which this approach should be applicable, where we will assume all definitions and propositions in the subsequent sections to take place in such a category. In order to apply the construction of MPOCs by IPOs given above, we require the category to have IPOs for all morphisms and pushouts along monomorphisms preserving them. Moreover, we allow the definition of a special class $\mathcal{R} \subseteq \mathcal{M}$ which are later used in the rules. This is useful to further restrict the possibilities of rules, e. g., by disallowing certain elements to be added or deleted as we will do it for variables in the RDF instantiation.

Definition 3.4 (Transformation Category)
A *transformation category* $(\mathbf{C}, \mathcal{M}, \mathcal{R})$ is a category $\mathbf{C}$ with distinguished classes $\mathcal{M}$ and $\mathcal{R} \subseteq \mathcal{M}$ of monomorphisms, such that

- $\mathcal{M}$ is closed under composition, i. e., $f, g \in \mathcal{M}$ implies $g \circ f \in \mathcal{M}$, and decomposition, i. e., $g \circ f \in \mathcal{M}$ implies $f \in \mathcal{M}$,
- $\mathcal{R}$ is closed under composition, i. e., $f, g \in \mathcal{R}$ implies $g \circ f \in \mathcal{R}$,
- pushouts along $\mathcal{M}$ exist, i. e., for all morphisms $r\colon I \to R$ and $i\colon I \to D$, where $r \in \mathcal{M}$, a pushout $(H, g\colon D \to H, n\colon R \to H)$ of r and i exists,
- pushouts preserve $\mathcal{M}$, i. e., for all pushouts (H, g, n) of r and i, $r \in \mathcal{M}$ implies $g \in \mathcal{M}$,
- pushouts preserve $\mathcal{R}$, i. e., for all pushouts (H, g, n) of r and i, $r \in \mathcal{R}$ implies $g \in \mathcal{R}$, and
- IPOs (with boundary and context morphisms in $\mathcal{M}$) exist for all morphisms.

A transformation rule is now given by left-hand side, interface and right-hand side connected by monomorphisms from $\mathcal{M}$.

Definition 3.5 (Transformation Rule)
A *transformation rule* $tr = (\mathrm{L}_{tr}, \mathrm{I}_{tr}, \mathrm{R}_{tr}, tr_{\mathrm{l}}, tr_{\mathrm{r}})$ consists of

- objects $\mathrm{L}_{tr}, \mathrm{I}_{tr}, \mathrm{R}_{tr} \in |\mathbf{C}|$ called *left-hand side* (LHS), *interface* and *right-hand side* (RHS), respectively, and
- morphisms $tr_{\mathrm{l}}\colon \mathrm{I}_{tr} \to \mathrm{L}_{tr}$ and $tr_{\mathrm{r}}\colon \mathrm{I}_{tr} \to \mathrm{R}_{tr}$ with $tr_{\mathrm{l}}, tr_{\mathrm{r}} \in \mathcal{R}$.

$$\begin{tikzcd}
B \arrow[r, hook, "b"] \arrow[rr, hook, bend left=40, "b^*"] \arrow[rr, phantom, bend left=20, "(=)"] \arrow[d, "m_B"'] \arrow[dr, phantom, "(IPO)"] & L \arrow[d, "m"] & I \arrow[l, hook', "l"'] \\
C \arrow[r, hook, "c"] & G &
\end{tikzcd}$$

(a) Gluing condition

$$\begin{tikzcd}
B \arrow[r, hook, "b^*"] \arrow[rr, hook, bend left=40, "b"] \arrow[rr, phantom, bend left=20, "(=)"] \arrow[d, "m_B"'] \arrow[dr, phantom, "(PO)"] & I \arrow[r, hook, "l"] \arrow[d, "i"] \arrow[dr, phantom, "(=)"] & L \arrow[d, "m"] \\
C \arrow[r, hook, "c^*"] \arrow[rr, hook, bend right=40, "c"'] \arrow[rr, phantom, bend right=20, "(=)"] & D \arrow[r, "\exists!f"] & G
\end{tikzcd}$$

(b) Pushout under boundary

$$\begin{tikzcd}
B \arrow[r, hook, "b^*"] \arrow[rr, hook, bend left=40, "b"] \arrow[rr, phantom, bend left=20, "(=)"] \arrow[d, "m_B"'] \arrow[dr, phantom, "(PO)"] & I \arrow[r, hook, "l"] \arrow[d, "i"] \arrow[dr, phantom, "(PO)"] & L \arrow[d, "m"] \\
C \arrow[r, hook, "c^*"] \arrow[rr, hook, bend right=40, "c"'] \arrow[rr, phantom, bend right=20, "(=)"] & D \arrow[r, hook, "f"] & G
\end{tikzcd}$$

(c) Pushout decomposition

$$\begin{tikzcd}
B \arrow[r, hook, "b"] \arrow[rr, hook, bend left=40, "\exists!b'"] \arrow[rr, phantom, bend left=20, "(=)"] \arrow[d, "m_B"'] \arrow[dr, phantom, "(IPO)"] & L \arrow[d, "m"] \arrow[dr, phantom, "(PO)"] & I \arrow[l, hook', "l"'] \arrow[d, "i'"] \\
C \arrow[r, hook, "c"] \arrow[rr, hook, bend right=40, "\exists!c'"'] \arrow[rr, phantom, bend right=20, "(=)"] & G & D' \arrow[l, hook', "f'"']
\end{tikzcd}$$

(d) Comparison complement

$$\begin{tikzcd}
B \arrow[r, hook, "b^* = b'"] \arrow[d, "m_B"'] \arrow[dr, phantom, "(PO)"] & I \arrow[d, "i"] \arrow[rdd, bend left, "i'"] \arrow[rd, phantom, "(=)"] & \\
C \arrow[r, hook, "c^*"] \arrow[rrd, hook, bend right, "c'"'] \arrow[rd, phantom, "(=)"] & D \arrow[rd, "\exists!d"] & \\
 & & D'
\end{tikzcd}$$

(e) D' comparison object for D

Abbildung 3.5.: Construction of MPOCs by IPOs

A transformation by a transformation rule is then constructed by an MPOC for the left-hand side and a pushout for the right-hand side.

Definition 3.6 (Transformation)
A *transformation* from an object G to an object H by a transformation rule tr via a *match* $m\colon \mathrm{L}_{tr} \to G$, written as $G \overset{tr,m}{\Longrightarrow} H$, is given by

- an object D with monomorphisms $f\colon D \to G$ and $g\colon D \to H$ in $\mathcal{M}$,
- a morphism $i\colon \mathrm{I}_{tr} \to D$ and
- a *comatch* $n\colon \mathrm{R}_{tr} \to H$,

such that

- (D, f, i) is an MPOC of tr_l and m and
- (H, g, n) is a pushout of tr_r and i.

(Cf. Figure 3.6.)

$$\begin{tikzcd}
\boxed{\mathrm{L}_{tr}} \arrow[d, "m"'] & \boxed{\mathrm{I}_{tr}} \arrow[l, hook', "tr_\mathrm{l}"'] \arrow[r, hook, "tr_\mathrm{r}"] \arrow[d, "i"] \arrow[dl, phantom, "\text{(MPOC)}"] \arrow[dr, phantom, "\text{(PO)}"] & \boxed{\mathrm{R}_{tr}} \arrow[d, "n"] \\
\boxed{G} & \boxed{D} \arrow[l, hook', "f"'] \arrow[r, hook, "g"] & \boxed{H}
\end{tikzcd}$$

Abbildung 3.6.: Transformation $G \overset{tr,m}{\Longrightarrow} H$

The following theorem summarises the basic properties of transformations. More specifically, transformation results exist if the gluing condition is satisfied and then they are also unique and transformations preserve morphisms in $\mathcal{M}$.

Theorem 3.2 (Properties of Transformations)
Given a transformation rule tr, an object G and a match $m\colon \mathrm{L}_{tr} \to G$, such that tr_l and m satisfy the gluing condition, then

1. the transformation $G \overset{tr,m}{\Longrightarrow} H$ as in Definition 3.6 exists,
2. the result H of the transformation is unique up to isomorphism and
3. f and g are in $\mathcal{R}$.

Beweis.

1. Since the gluing condition is satisfied we can construct an MPOC (D, f, i) of tr_{l} and m by Theorem 3.1 and then build a pushout (H, g, n) of tr_{r} and i since $tr_{\mathrm{r}} \in \mathcal{M}$ and $(\mathbf{C}, \mathcal{M}, \mathcal{R})$ has pushouts along $\mathcal{M}$.
2. Since MPOCs and pushouts are defined as initial objects in appropriate categories and initial objects are unique up to isomorphism (cf. Proposition A.6) (D, f, i) and, hence, also (H, g, n) are unique up to isomorphism.
3. Since tr_{l} and tr_{r} are in $\mathcal{R}$ and $(\mathbf{C}, \mathcal{M}, \mathcal{R})$ is required to have pushouts preserving $\mathcal{R}$ f and g are also in $\mathcal{R}$. □

3.3. Sequential Composition

In order to build larger rules from smaller ones, we introduce the sequential composition of rules in this section, where we show that such sequentially composed rules do not modify the results obtainable by a set of rules and how they can be obtained from transformation sequences.

3.3.1. Composition and Decomposition of MPOCs and IPOs

As preliminaries for composition of rules and synthesis and analysis of transformations we need some results for the composability and deomposability of MPOCs which in turn build on corresponding results for IPOs.

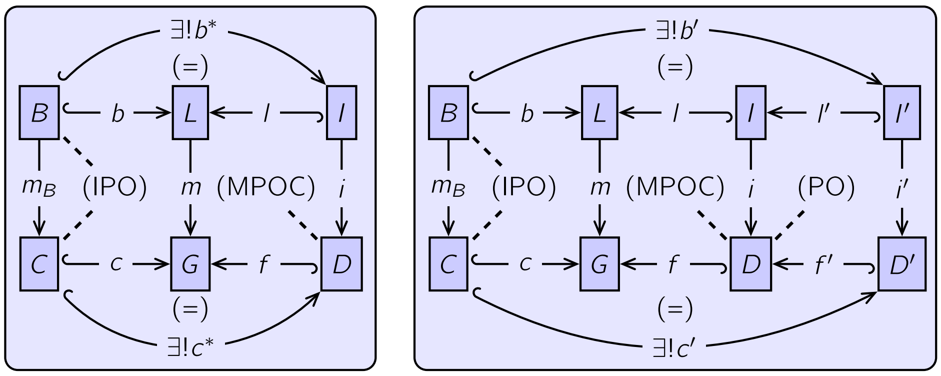
Firstly, we observe that IPOs over a morphism are composable with MPOCs of the same morphism.[2]

Lemma 3.2 (Composition of IPOs with MPOCs)
Given morphisms $l\colon I \to L$ with $l \in \mathcal{M}$ and $m\colon L \to G$ with IPO (B, b, C, c, m_B) over m and an MPOC (D, f, i) of l and m (cf. Figure 3.7a), then (B, b^*, C, c^*, m_B) is an IPO over i, where $b^*\colon B \to I$ and $c^*\colon C \to D$ are the morphisms in $\mathcal{M}$ induced by initiality of the original IPO.

Beweis. Firstly, (D, c^*, i) has to be a pushout of b^* and m_B which is satisfied because of Theorem 3.1 3. applied to the MPOC (D, f, i) of l and m.
Secondly, this pushout is initial in $\mathbf{PO}(i)$: Given another pushout (I', l', D', f', i') over i, then $(I', l \circ l', D', f \circ f', i')$ is a pushout over m by pushout composition (cf. Proposition A.11). Initiality of the original IPO implies the existence of unique $b'\colon B \to I'$ and $c'\colon C \to D'$ w. r. t. $l \circ l' \circ b' = b$ and $f \circ f' \circ c' = c$ (cf. Figure 3.7b). Since $b = l \circ b^*$, $c = f \circ c^*$ and l and f are monomorphisms this implies $l' \circ b' = b^*$ and $f' \circ c' = c^*$. Moreover, b' and c' are also unique w. r. t. these properties since each other morphisms

[2] In [EEPT06] a more general result for the strong initial pushouts in the framework of adhesive HLR categories is given. More specifically, IPOs there are composable with arbitrary pushouts over the same morphism.

b'' and c'' with $l' \circ b'' = b^*$ and $f' \circ c'' = c^*$ would also satisfy $l \circ l' \circ b'' = l \circ b^* = b$ and $f \circ f' \circ c'' = f \circ c^* = c$ by composition with l and f and b' and c' were already unique for the latter equalities. □



(a) Given IPO

(b) Comparison pushout

Abbildung 3.7.: Composition of IPOs with MPOCs

This already allows us to compose and decompose MPOCs horizontically, i. e., along the monomorphisms, in all transformation categories, while vertical composition and decomposition needs additional properties.

Lemma 3.3 (Horizontal Composition and Decomposition of MPOCs)
Given morphisms $l\colon I \to L$, $l'\colon I' \to I$, $f\colon D \to G$ and $f'\colon D' \to D$ with $l, l', f, f' \in \mathcal{M}$, $m\colon L \to G$ with IPO (B, b, C, c, m_B) over m and $i\colon I \to D$ and $i'\colon I' \to D'$, such that (G, f, m) is a pushout of l and i and (D, f', i) is a pushout of l' and i' (cf. Figure 3.8), then $(D', f \circ f', i')$ is an MPOC of $l \circ l'$ and m if and only if (D, f, i) is an MPOC of l and m and (D', f', i') is an MPOC of l' and i.

Beweis. Firstly, initiality of the IPO induces morphisms $b^*\colon B \to I$ and $c^*\colon C \to D$ with $l \circ b^* = b$ and $f \circ c^* = c$ and morphisms $b'\colon B \to I'$ and $c'\colon C \to D'$ with $l \circ l' \circ b' = b$ and $f \circ f' \circ c' = c$. Because l and f are monomorphisms we also have $l' \circ b' = b^*$ and $f' \circ c' = c^*$.

If (Composition): Since (D, f, i) is an MPOC of l and m, Lemma 3.2 implies that (B, b^*, C, c^*, m_B) is an IPO over i. By Theorem 3.1 3. the MPOC (D', f', i') implies that (D', c', i') is a pushout of b' and m_B. Using this pushout in the construction of Theorem 3.1 1. w. r. t. $m\colon L \to G$ yields $(D', f \circ f', i')$ as an MPOC of $l \circ l'$ and m.

Only if (Decomposition): From $(D', f \circ f', i')$ being an MPOC of $l \circ l'$ and m and Theorem 3.1 3. it follows that (D', c', i') is a pushout of b' and m_B. By pushout composition (cf. Proposition A.11) with (D, f', i) we get a pushout $(D, f' \circ c', i)$ of $l' \circ b'$ and m_B which by Theorem 3.1 1. yields (D, f, i) as MPOC of l and m. By Lemma 3.2 this implies again that (B, b^*, C, c^*, m_B) is an IPO over i. Using the pushout (D', c', i') of b' and m_B in Theorem 3.1 1. we get (D', f', i') as MPOC of l' and i. □

$$\begin{tikzcd}
B \arrow[r, hook, "b"] \arrow[d, "m_B"'] \arrow[rr, bend left, "\exists ! b^*"] \arrow[rrr, bend left=40, "\exists ! b'"] \arrow[dr, phantom, "(\mathrm{IPO})"] & L \arrow[d, "m"] \arrow[dr, phantom, "(\mathrm{PO})"] & I \arrow[l, hook', "l"'] \arrow[d, "i"] \arrow[dr, phantom, "(\mathrm{PO})"] & I' \arrow[l, hook', "l'"'] \arrow[d, "i'"] \\
C \arrow[r, hook, "c"'] \arrow[rr, bend right, "\exists ! c^*"'] \arrow[rrr, bend right=40, "\exists ! c'"'] & G & D \arrow[l, hook', "f"] & D' \arrow[l, hook', "f'"]
\end{tikzcd}$$

Abbildung 3.8.: Horizontal composition and decomposition of MPOCs

For the vertical composition and decomposition of IPOs and MPOCs, we need our category to satisfy some additional properties. The first one is a characterisation of MPOCs by pullbacks, i. e., a pushout is an MPOC if and only if it is also a pullback.[3] Intuitively this is a reasonable property since an MPOC should mean that as much as possible is deleted in the MPOC object and this is the case if and only if the interface is the intersection of the MPOC and the left-hand side and not more.

Definition 3.7 (MPOC-Pullback Characterisation)
A category has the *MPOC-pullback characterisation* if, given morphisms $l\colon I \to L$ and $f\colon D \to G$ with $l, f \in \mathcal{M}$, $m\colon L \to G$ and $i\colon I \to D$, such that (G, f, m) is a pushout of l and i, then (D, f, i) is an MPOC of l and m if and only if (I, l, i) is a pullback of f and m.

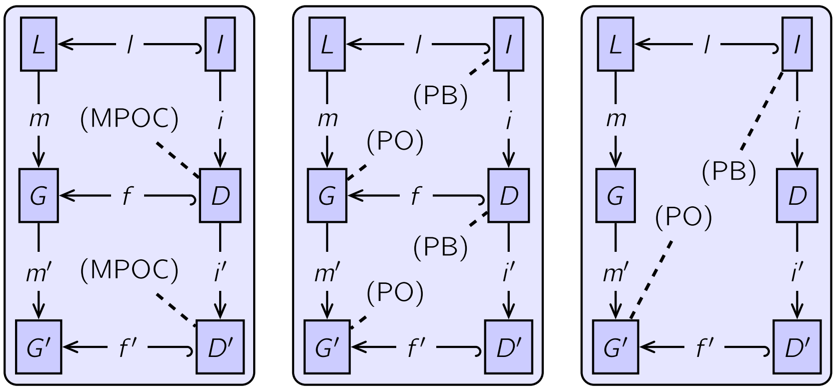
Using the surprisingly strong MPOC-pullback characterisation, vertical composition of MPOCs follows immediately since we can use composition of pushouts and pullbacks.

Lemma 3.4 (Vertical Composition of MPOCs)
Given a category which has the MPOC-pullback characterisation, morphisms $l\colon I \to L$, $f\colon D \to G$ and $f'\colon D' \to G'$ with $l, f, f' \in \mathcal{M}$, $m\colon L \to G$, $m'\colon G \to G'$, $i\colon I \to D$ and

[3] Note that in the categories considered in [EEPT06] and related work pushouts (along $\mathcal{M}$) are pullbacks. Thus, this property then implies that all pushouts are MPOCs which is true because these categories have unique pushout complements. In our framework, and escpecially for the RDF instantiation, pushouts are, however, not necessarily pullbacks.

$i' : D \to D'$, such that (D, f, i) is an MPOC of l and m and (D', f', i') is an MPOC of f and m' (cf. Figure 3.9a), then $(D', f', i' \circ i)$ is an MPOC of l and $m' \circ m$.

Beweis. From the MPOC (D, f, i) of l and m we can conclude by definition that (G, f, m) is a pushout of l and i and by the MPOC-pullback characterisation that (I, l, i) is a pullback of f and m. Analogously, the MPOC (D', f', i') of f and m' implies the pushout (G', f', m') of f and i' and the pullback (D, f, i') of f' and m' (cf. Figure 3.9b). Composition of pushouts and pullbacks (cf. Proposition A.11) yields a pushout $(G', f', m' \circ m)$ of l and $i' \circ i$ and a pullback $(I, l, i' \circ i)$ of f' and $m' \circ m$ (cf. Figure 3.9c). Applying the MPOC-pullback characterisation again we finally obtain the MPOC $(D', f', i' \circ i)$ of l and $m' \circ m$. □



(a) Given situation (b) MPOCs are pullbacks (c) Composistion of pushouts and pullbacks

Abbildung 3.9.: Vertical composition of MPOCs

In order to vertically decompose MPOCs and IPOs we need a decomposition property for pushouts using pullbacks.[4] While the classical pushout decomposition in Proposition A.11 allows us to decompose a pushout if one of the given morphisms is factored by constructing a pushout of this factor, the pushout-pullback decomposition is useful if one of the pushout morphisms is factored such that we can construct a pullback of this factor.

Definition 3.8 (Pushout-Pullback Decomposition)
A category has *pushout-pullback decompositions* if, given morphisms $l : I \to L$, $f : D \to G$ and $f' : D' \to G'$ with $l, f, f' \in \mathcal{M}$, $m : L \to G$, $m' : G \to G'$, $i : I \to D$ and $i' : D \to D'$,

[4] In [EEPT06] and related work another variant of pushout-pullback decomposition is used, where other morphisms are assumed to be monomorphisms (or, more specifically, morphisms in $\mathcal{M}$).

such that $(G', f', m' \circ m)$ is a pushout of l and $i' \circ i$, $m \circ l = f \circ i$ and (D, f, i') is a pullback of f' and m' (cf. Figure 3.10a), then (G, f, m) is a pushout of l and i (cf. Figure 3.10b).

$$\begin{tikzcd}
L \arrow[d, "m"'] & I \arrow[l, hook', "l"'] \arrow[d, "i"] \\
G \arrow[d, "m'"'] \arrow[r, phantom, "(PO)"] & D \arrow[d, "i'"] \\
G' & D' \arrow[l, hook', "f'"']
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[d, "m"'] \arrow[r, phantom, "(=)"] & I \arrow[l, hook', "l"'] \arrow[d, "i"] \\
G \arrow[d, "m'"'] \arrow[r, phantom, "(PB)"] & D \arrow[l, hook', "f"'] \arrow[d, "i'"] \\
G' & D' \arrow[l, hook', "f'"']
\end{tikzcd}$$

(a) Given situation

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[r, phantom, "(PO)"] & I \arrow[l, hook', "l"'] \arrow[d, "i"] \\
G \arrow[d, "m'"'] & D \arrow[l, hook', "f"'] \arrow[d, "i'"] \\
G' & D' \arrow[l, hook', "f'"']
\end{tikzcd}$$

(b) Conclusion

Abbildung 3.10.: Pushout-pullback decomposition

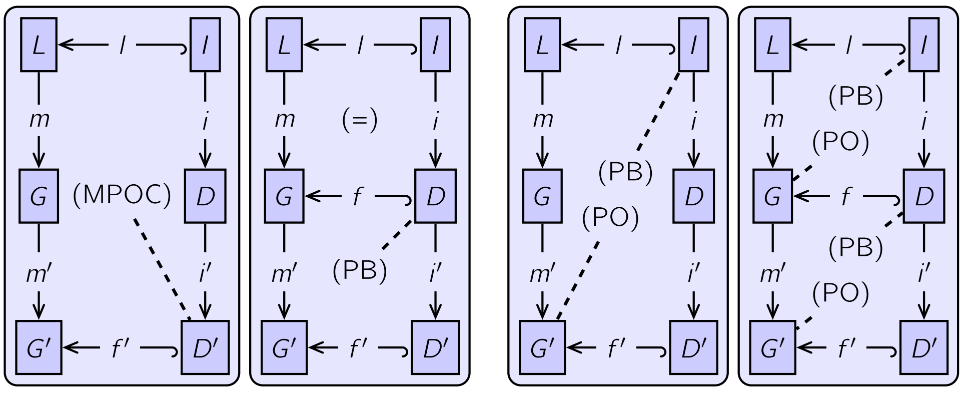
Combining pushout-pullback decompositions and the MPOC-pullback characterisations we obtain a vertical decomposition of MPOCs by pullbacks.

Lemma 3.5 (Vertical Decomposition of MPOCs by Pullbacks)
Given a category which has the MPOC-pullback characterisation and pushout-pullback decompositions, morphisms $l\colon I \to L$, $f\colon D \to G$ and $f'\colon D' \to G'$ with $l, f, f' \in \mathcal{M}$, $m\colon L \to G$, $m'\colon G \to G'$, $i\colon I \to D$ and $i'\colon D \to D'$, such that $(D', f', i' \circ i)$ is an MPOC of l and $m' \circ m$, $m \circ l = f \circ i$ and (D, f, i') is a pullback of f' and m' (cf. Figure 3.11a), then (D, f, i) is an MPOC of l and m and (D', f', i') is an MPOC of f and m'.

Beweis. Using the definition of MPOCs and the MPOC-pullback characterisation, we have that $(G', f', m' \circ m)$ is a pushout of l and $i' \circ i$ and $(I, l, i' \circ i)$ is a pullback of f' and $m' \circ m$ (cf. left side of Figure 3.11b). By pushout-pullback decomposition we obtain (G, f, m) as pushout of l and i. Pushout and pullback decomposition (cf. Proposition A.11) then imply that (G', f', m') is a pushout of f and i' and that (I, l, i) is a pullback of f and m, respectively (cf. right side of Figure 3.11b). Using the MPOC-pullback characterisation again yields (D, f, i) as an MPOC of l and m and (D', f', i') as an MPOC of f and m'. $\square$

It remains to show that MPOCs can be decomposed vertically even if no pullback but only pushouts are involved.

Lemma 3.6 (Vertical Decomposition of MPOCs by Pushouts)
Given a category which has the MPOC-pullback characterisation and pushout-pullback



(a) Given situation (b) MPOC characterisation and pushout and pullback decompositions

Abbildung 3.11.: Vertical decomposition of MPOCs by pullbacks

decompositions, morphisms $l\colon I \to L$, $f\colon D \to G$ and $f'\colon D' \to G'$ with $l, f, f' \in \mathcal{M}$, $m\colon L \to G$, $m'\colon G \to G'$, $i\colon I \to D$ and $i'\colon D \to D'$, such that $(D', f', i' \circ i)$ is an MPOC of l and $m' \circ m$, (G, f, m) is a pushout of l and i and (G', f', m') is a pushout of f and i' (cf. Figure 3.11a), then (D, f, i) is an MPOC of l and m and (D', f', i') is an MPOC of f and m' (cf. **??**).

Beweis. We consider the IPO (B, b, C, c, m_B) over $m' \circ m$. By Lemma 3.2 (B, b^*, C, c^*, m_B) with the unique morphisms $b^*\colon B \to I$ with $l \circ b^* = b$ and $c^*\colon C \to D'$ with $f' \circ c^* = c$, induced by $(G', f', m' \circ m)$ of l and $i' \circ i$ as comparison pushout, is an IPO over $i' \circ i$. We construct pullbacks (C', c', m'_2) of $c = f \circ c^*$ and m' and (C'', c'', m''_2) of c^* and i' with corresponding unique morphisms $m'_1\colon B \to C'$ and $m''_1\colon B \to C''$. Since IPOs are MPOCs by Lemma 3.1 we can use MPOC pullback decomposition to conclude that (C', c', m'_1) is an MPOC of $b = l \circ b^*$ and m, $(C, f \circ c^*, m'_2)$ an MPOC of c' and m', (C'', c'', m''_1) an MPOC of b^* and i and (C, c^*, m''_2) an MPOC of c'' and i'. By composing the MPOC C'' with the pushout (G, g, m) of l and i, C'' is a comparison object for the MPOC C' and there is a unique morphism $C' \to C''$. Vice versa, by composing the pullback C'' with the commuting square $m' \circ f = f' \circ i'$, C'' is also a comparison object for C' as a pullback and there has to be a unique morphism $C'' \to C'$. Hence, C' and C'' are isomorphic. Thus, we can use horizontal MPOC decomposition to conclude that (D, f, i) is an MPOC of l and m and (D', f', i') is an MPOC of f and m'. □

Finally, a last lemma shows the vertical decomposition of IPOs. More specifically, the small IPOs over the components of a composed morphism are also IPOs over the pushout-pullback decomposition of the large IPO over the composed morphism.

Lemma 3.7 (Vertical Decomposition of IPOs)

$$\begin{tikzcd}
L \arrow[d, "m"'] & I \arrow[l, hook', "l"'] \arrow[d, "i"] \\
G \arrow[d, "m'"'] \arrow[r, phantom, "\text{(MPOC)}"] & D \arrow[d, "i'"] \\
G' & D' \arrow[l, hook', "f'"']
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[d, "m"'] \arrow[r, phantom, "\text{(PO)}"'] & I \arrow[l, hook', "l"'] \arrow[d, "i"] \\
G \arrow[d, "m'"'] \arrow[r, phantom, "\text{(PO)}"'] & D \arrow[l, hook', "f"'] \arrow[d, "i'"] \\
G' & D' \arrow[l, hook', "f'"']
\end{tikzcd}$$

(a) Given situation

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[r, phantom, "\text{(MPOC)}"'] & I \arrow[l, hook', "l"'] \arrow[d, "i"] \\
G \arrow[d, "m'"'] \arrow[r, phantom, "\text{(MPOC)}"'] & D \arrow[l, hook', "f"'] \arrow[d, "i'"] \\
G' & D' \arrow[l, hook', "f'"']
\end{tikzcd}$$

(b) Conclusion

Abbildung 3.12.: Vertical decomposition of MPOCs by pushouts

Given a category with pullbacks along $\mathcal{M}$ preserving $\mathcal{M}$, pushout-pullback decompositions and MPOC-pullback characterisation and morphisms $m\colon L \to G$ and $m'\colon G \to G'$ with IPOs $(B_1, b_1, C_1, c_1, m_1)$ over m, $(B_2, b_2, C_2, c_2, m_2)$ over m' and $(B_3, b_3, C_3, c_3, m_3)$ over $m' \circ m$ (cf. Figure 3.13a), then there is an object C'_3 with a monomorphism $c'_3\colon C'_3 \to G$ and morphisms $m_{3,1}\colon B_3 \to C'_3$ and $m_{3,2}\colon C'_3 \to C_3$, such that $m_{3,2} \circ m_{3,1} = m_3$, $(C'_3, c'_3, m_{3,1})$ is an MPOC of b_3 and m, $(C_3, c_3, m_{3,2})$ is an MPOC of c'_3 and m' and $(B_1, b^*_1, C_1, c^*_1, m_1)$ and $(B_2, b^*_2, C_2, c^*_2, m_2)$ are IPOs over $m_{3,1}$ and $m_{3,2}$, respectively (cf. Figure 3.13b).

Beweis. The object C'_3 and the morphisms c'_3 and $m_{3,2}$ can be constructed as a pullback $(C'_3, c'_3, m_{3,2})$ of c_3 and m' inducing $m_{3,1}$ as the unique morphism with $m_{3,2} \circ m_{3,1} = m_3$ and $c'_3 \circ m_{3,1} = m \circ b_3$. The IPO $(B_3, b_3, C_3, c_3, m_3)$ over $m' \circ m$ implies that (C_3, c_3, m_3) is an MPOC of b_3 and $m' \circ m$ by Lemma 3.1. We can now apply Lemma 3.5 leading to MPOCs $(C'_3, c'_3, m_{3,1})$ of b_3 and m and $(C_3, c_3, m_{3,2})$ of c'_3 and m'. Finally, Lemma 3.2 can be used to obtain that $(B_1, b^*_1, C_1, c^*_1, m_1)$ and $(B_2, b^*_2, C_2, c^*_2, m_2)$ are IPOs over $m_{3,1}$ and $m_{3,2}$, respectively, where b^*_1, c^*_1, b^*_2 and c^*_2 are the induced morphisms from the IPO into the MPOCs as comparison pushouts. □

3.3.2. Sequentially Composed Rules

We will now develop a sequential composition of transformation rules. First, some properties that are needed for a sensible sequential composition are collected. The first property in the following definition is a variant of one of the HLR properties in [EHKP91] which is, on the one hand, slightly weaker since we do not need the full strength for analysis and synthesis of sequential composition, on the other hand, we additionally require a second

$$\begin{tikzcd}
B_3 \arrow[rr, hook, "b_3"] \arrow[dddddd, "m_3"'] \arrow[dddddd, phantom, shift left=6, "\text{(IPO)}"] & & L \arrow[ddd, "m"] \\
& B_1 \arrow[ur, hook', "b_1"'] \arrow[d, "m_1"] \arrow[d, phantom, shift left=6, "\text{(IPO)}"] & \\
& C_1 \arrow[dr, hook, "c_1"] & \\
& & G \arrow[ddd, "m'"] \\
& B_2 \arrow[ur, hook', "b_2"'] \arrow[d, "m_2"] \arrow[d, phantom, shift left=6, "\text{(IPO)}"] & \\
& C_2 \arrow[dr, hook, "c_2"] & \\
C_3 \arrow[rr, hook, "c_3"] & & G'
\end{tikzcd}$$

(a) Given situation

$$\begin{tikzcd}
& B_3 \arrow[r, hook, "b_3"] \arrow[ddd, "m_{3,1}"] \arrow[ddd, phantom, shift left=8, "\text{(MPOC)}"] & L \arrow[ddd, "m"] \\
B_1 \arrow[ur, hook', "\exists! b_1^*"] \arrow[d, "m_1"] \arrow[d, phantom, shift left=6, "\text{(IPO)}"] & & \\
C_1 \arrow[dr, hook, "\exists! c_1^*"] & & \\
& C'_3 \arrow[r, hook, "c'_3"] \arrow[ddd, "m_{3,2}"] \arrow[ddd, phantom, shift left=8, "\text{(MPOC)}"] & G \arrow[ddd, "m'"] \\
B_2 \arrow[ur, hook', "\exists! b_2^*"] \arrow[d, "m_2"] \arrow[d, phantom, shift left=6, "\text{(IPO)}"] & & \\
C_2 \arrow[dr, hook, "\exists! c_2^*"] & & \\
& C_3 \arrow[r, hook, "c_3"] & G'
\end{tikzcd}$$

(b) Conclusion

Abbildung 3.13.: Vertical IPO decomposition

cube property for MPOCs which is not considered in the original version.

Definition 3.9 (Cube Properties)
A transformation category satisfies the *cube properties* if, given the commutative cube in Figure 3.14,

- the front faces (the ones containing I and D) are pushouts if the back faces (the ones containing E and G) are pushouts and top and bottom are pullbacks (pushouts are preserved by pullbacks) and
- if all vertical faces are pushouts and the top is a pullback then the back right face is an MPOC if and only if the front left face is an MPOC (MPOCs are inherited over pushouts).

$$\begin{tikzcd}
 & & E \arrow[ddd] & & \\
J_1 \arrow[urr, hook] \arrow[ddd] & & & & J_2 \arrow[ull, hook'] \arrow[ddd] \\
 & & & I \arrow[ulll, hook'] \arrow[ur, hook] \arrow[ddd] & \\
 & & G & & \\
D_1 \arrow[urr, hook] & & & & D_2 \arrow[ull, hook'] \\
 & & & D \arrow[ulll, hook'] \arrow[ur, hook] &
\end{tikzcd}$$

Abbildung 3.14.: Cube properties

Now, we define an extended notion of transformation categories which collects all necessary properties for the results in this section.

Definition 3.10 (Tranformation Category with Sequential Composition)
A *transformation category with sequential composition* $(\mathbf{C}, \mathcal{M}, \mathcal{R})$ is a transformation category as in Definition 3.4, such that

- pullbacks along $\mathcal{M}$ exist,
- pullbacks preserve $\mathcal{M}$,
- the MPOC-pullback characterisation in Definition 3.7 holds and
- pushout-pullback decompositions as in Definition 3.8 exist,
- the cube properties as in Definition 3.9 hold.

A sequentially composed rule is, intuitively, a large rule combining the effects of two smaller rules. Since this process can be arbitrarily iterated the construction of arbitrarily complex rules is possible this way. Formally, the relation between the large rule and the smaller rule is established by sequential transformation steps from the left-hand side of the composed rule to the right-hand side of the composed rule, where the interface is obtained as a pullback of the interfaces of the component rules.[5]

Definition 3.11 (Sequentially Composed Rule)
Given transformation rules tr_1, tr_2 and tr_3, then tr_3 is a *sequentially composed rule* of tr_2 after tr_1 if there are

- objects J_1, E and J_2 with morphisms $l_1\colon J_1 \to \mathsf{L}_{tr_3}$, $r_1\colon J_1 \to E$, $r_1^*\colon \mathsf{I}_{tr_3} \to J_2$, $l_2\colon J_2 \to E$, $l_2^*\colon \mathsf{I}_{tr_3} \to J_1$ and $r_2\colon J_2 \to \mathsf{R}_{tr_3}$ in $\mathcal{R}$ and
- morphisms $p_1\colon \mathsf{L}_{tr_1} \to \mathsf{L}_{tr_3}$, $q_1\colon \mathsf{I}_{tr_1} \to J_1$, $e_1\colon \mathsf{R}_{tr_1} \to E$, $e_2\colon \mathsf{L}_{tr_2} \to E$, $q_2\colon \mathsf{I}_{tr_2} \to J_2$ and $p_2\colon \mathsf{R}_{tr_2} \to \mathsf{R}_{tr_3}$,

such that

- $\mathsf{L}_{tr_3} \xRightarrow{tr_1,p_1} E$ by (J_1, l_1, q_1) being an MPOC of $(tr_1)_\mathsf{l}$ and p_1 and (E, r_1, e_1) being a pushout of $(tr_1)_\mathsf{r}$ and q_1,
- $E \xRightarrow{tr_2,e_2} \mathsf{R}_{tr_3}$ by (J_2, l_2, q_2) being an MPOC of $(tr_2)_\mathsf{l}$ and e_2 and $(\mathsf{R}_{tr_3}, r_2, p_2)$ being a pushout of $(tr_2)_\mathsf{r}$ and q_2,
- $(\mathsf{I}_{tr_3}, l_2^*, r_1^*)$ is a pullback of l_2 and r_1,
- $(tr_3)_\mathsf{l} = l_1 \circ l_2^*$ and $(tr_3)_\mathsf{r} = r_2 \circ r_1^*$.

(Cf. Figure 3.15.)

The first of the two main results regarding sequential composition is the analysis of transformations by a sequentially composed rule into transformations by the component which allows to conclude that, whenever a result is reachable by a sequentially composed rule, it is also reachable using the component rules.

Theorem 3.3 (Analysis of Sequentially Composed Rule Transformations)
Given a sequentially composed rule tr_3 of tr_2 after tr_1 and a transformation $G_1 \xRightarrow{tr_3,m_3} G_3$, there are an object G_2 and transformations $G_1 \xRightarrow{tr_1,m_1} G_2$ and $G_2 \xRightarrow{tr_2,m_2} G_3$.

[5] Sequentially composed rules are called E-concurrent rules in [EEPT06] and related work, where they are uniquely constructed from the comatch of the first rule and the match of the second rule into the object E. Since the pushout complement J_1 of the right-hand side of the first rule is not unique in our framework and there are several choices of additional data that would uniquely determine the composition, from which none is clearly preferrable, we have chosen to give a descriptive definition instead.

$$\begin{tikzcd}
\mathsf{L}_{tr_1} \arrow[ddd, "p_1"'] & \mathsf{I}_{tr_1} \arrow[l, hook', "(tr_1)_\mathsf{l}"'] \arrow[r, hook, "(tr_1)_\mathsf{r}"] \arrow[dd, "q_1"] & \mathsf{R}_{tr_1} \arrow[dr, "e_1"] \arrow[d, phantom, "(\mathrm{PO})"] & & \mathsf{L}_{tr_2} \arrow[dl, "e_2"'] & \mathsf{I}_{tr_2} \arrow[l, hook', "(tr_2)_\mathsf{l}"'] \arrow[r, hook, "(tr_2)_\mathsf{r}"] \arrow[dd, "q_2"] \arrow[dl, phantom, "(\mathrm{MPOC})"] & \mathsf{R}_{tr_2} \arrow[ddd, "p_2"] \arrow[ddl, phantom, "(\mathrm{PO})"] \\
& (\mathrm{MPOC}) & & E & & & \\
& J_1 \arrow[urr, hook, "r_1"] \arrow[dl, hook', "l_1"'] \arrow[d, phantom, "(=)"] & & (\mathrm{PB}) & & J_2 \arrow[ull, hook', "l_2"'] \arrow[dr, hook, "r_2"] \arrow[d, phantom, "(=)"] & \\
\mathsf{L}_{tr_3} & & & \mathsf{I}_{tr_3} \arrow[lll, hook', "(tr_3)_\mathsf{l}"'] \arrow[rrr, hook, "(tr_3)_\mathsf{r}"] \arrow[ull, hook', "l_2^*"'] \arrow[urr, hook, "r_1^*"] & & & \mathsf{R}_{tr_3}
\end{tikzcd}$$

Abbildung 3.15.: Sequentially composed rule

Beweis. From the given transformation $G_1 \overset{tr_3,m_3}{\Longrightarrow} G_3$ shown in Figure 3.16a and the structure of the sequentially composed rule in Figure 3.15 we can construct pushouts of D_3 and J_1 and of D_3 and J_2 which induce unique morphisms from these pushouts into G_1 and G_3, respectively. Pushout decomposition (cf. Proposition A.11) and horizontal MPOC decomposition (cf. Lemma 3.3) then lead to the situation in Figure 3.16b.
Then we can construct a cube by a pushout G_2 of E and D_1. The pushout composition with the pushout D_1 of J_1 and D_3, combined with the fact the top is a pullback and, thus, commutes, makes G_2 a comparison object for the pushout D_2 of J_2 and D_3 inducing a unique morphism from D_2 to G_2. Pushout decomposition then leads to G_2 being a pushout of E and D_2 as shown in Figure 3.16c.
By the second cube property D_2 is then also an MPOC of G_2 under J_2. When we combine this with the information from the composed rule in Figure 3.15, the pushout and MPOC decompositions in Figure 3.16b and the cube in Figure 3.16c we arrive at the situation in Figure 3.16d.
Pushout composition and vertical MPOC composition then lead to the existence of transformations $G_1 \overset{tr_1,m_1}{\Longrightarrow} G_2$ with $m_1 := m_3 \circ p_1$ and $G_2 \overset{tr_2,m_2}{\Longrightarrow} G_3$ with $m_2 := i_3' \circ e_2$. $\square$

The second main result is the synthesis of sequentially composed rules from transformation sequences. This can be used to define a complex rule by the execution of a transformation sequence on example data and then sequentially composing this sequence, thus, providing a formal foundation for a macro recording facility.

Theorem 3.4 (Synthesis of Sequentially Composed Rule Transformations)
Given transformation rules tr_1 and tr_2 and transformations $G_1 \overset{tr_1,m_1}{\Longrightarrow} G_2$ and $G_2 \overset{tr_2,m_2}{\Longrightarrow} G_3$, there are a sequentially composed rule tr_3 of tr_2 after tr_1 and a transformation $G_1 \overset{tr_3,m_3}{\Longrightarrow} G_3$.

$$\begin{tikzcd}
\mathrm{L}_{tr_3} \arrow[d, "m_3"'] \arrow[dr, phantom, "\text{(MPOC)}"] & \mathrm{I}_{tr_3} \arrow[l, hook', "(tr_3)_\mathrm{l}"'] \arrow[r, hook, "(tr_3)_\mathrm{r}"] \arrow[d, "i_3"] \arrow[dr, phantom, "\text{(PO)}"] & \mathrm{R}_{tr_3} \arrow[d, "n_3"] \\
G_1 & D_3 \arrow[l, hook', "f_3"] \arrow[r, hook, "g_3"'] & G_3
\end{tikzcd}$$

(a) Given transformation

$$\begin{tikzcd}
\mathrm{L}_{tr_3} \arrow[d, "m_3"'] \arrow[dr, phantom, "\text{(MPOC)}"] & J_1 \arrow[l, hook', "l_1"'] \arrow[d] \arrow[dr, phantom, "\text{(MPOC)}\ \text{(PO)}"] & \mathrm{I}_{tr_3} \arrow[l, hook', "l_2^*"'] \arrow[r, hook, "r_1^*"] \arrow[d, "i_3"] \arrow[dr, phantom, "\text{(PO)}"] & J_2 \arrow[r, hook, "r_2"] \arrow[d] \arrow[dr, phantom, "\text{(PO)}"] & \mathrm{R}_{tr_3} \arrow[d, "n_3"] \\
G_1 & D_1 \arrow[l, hook', "\exists!"'] & D_3 \arrow[l, hook'] \arrow[r, hook] & D_2 \arrow[r, hook, "\exists!"] & G_3
\end{tikzcd}$$

(b) Interface construction

$$\begin{tikzcd}
 & & E \arrow[ddd, "i_3'"] & & \\
J_1 \arrow[urr, hook, "r_1"] \arrow[ddd] & & & & J_2 \arrow[ulll, hook', "l_2"'] \arrow[ddd] \\
 & \text{(PO)} & & \mathrm{I}_{tr_3} \arrow[ulll, hook', "l_2^*"'] \arrow[ur, hook, "r_1^*"] \arrow[ddd, "i_3"] & \text{(PO)} \\
 & & G_2 & & \\
D_1 \arrow[urr, hook] & & & & D_2 \arrow[ulll, hook', "\exists!"'] \\
 & & & D_3 \arrow[ulll, hook'] \arrow[ur, hook] &
\end{tikzcd}$$

(c) Interface cube

$$\begin{tikzcd}
\mathrm{L}_{tr_1} \arrow[d, "p_1"'] \arrow[dr, phantom, "\text{(MPOC)}"] & \mathrm{I}_{tr_1} \arrow[l, hook', "(tr_1)_\mathrm{l}"'] \arrow[r, hook, "(tr_1)_\mathrm{r}"] \arrow[d, "q_1"] \arrow[drr, phantom, "\text{(PO)}"] & \mathrm{R}_{tr_1} \arrow[dr, "e_1"] & & \mathrm{L}_{tr_2} \arrow[dl, "e_2"'] \arrow[dr, phantom, "\text{(MPOC)}"] & \mathrm{I}_{tr_2} \arrow[l, hook', "(tr_2)_\mathrm{l}"'] \arrow[r, hook, "(tr_2)_\mathrm{r}"] \arrow[d, "q_2"] \arrow[dr, phantom, "\text{(PO)}"] & \mathrm{R}_{tr_2} \arrow[d, "p_2"] \\
\mathrm{L}_{tr_3} \arrow[d, "m_3"'] \arrow[dr, phantom, "\text{(MPOC)}"] & J_1 \arrow[l, hook', "l_1"'] \arrow[rr, hook, "r_1"] \arrow[d] \arrow[drr, phantom, "\text{(PO)}"] & & E \arrow[d, "i_3'"] \arrow[drr, phantom, "\text{(MPOC)}"] & & J_2 \arrow[ll, hook', "l_2"'] \arrow[r, hook, "r_2"] \arrow[d] \arrow[dr, phantom, "\text{(PO)}"] & \mathrm{R}_{tr_3} \arrow[d, "n_3"] \\
G_1 & D_1 \arrow[l, hook'] \arrow[rr, hook] & & G_2 & & D_2 \arrow[ll, hook'] \arrow[r, hook] & G_3
\end{tikzcd}$$

(d) Component transformations

Abbildung 3.16.: Analysis of sequentially composed rule transformations

Beweis. The given transformations $G_1 \xRightarrow{tr_1,m_1} G_2$ and $G_2 \xRightarrow{tr_2,m_2} G_3$ are shown in Figure 3.17a.
We can now choose an arbitrary factorisation of n_1 and m_2 via an object E and construct pullbacks J_1 of E and D_1 and J_2 of E and D_2 inducing unique morphisms from I_{tr_1} to J_1 and from I_{tr_2} to J_2. By pushout-pullback decomposition and vertical MPOC-pullback decomposition (cf. Lemma 3.5) we obtain E as pushout of J_1 and I_{tr_1} and J_2 as MPOC of E under I_{tr_2}.
Now, L_{tr_3} can be constructed as pushout of L_{tr_1} and J_1 and R_{tr_3} as pushout of R_{tr_2} and J_2 inducing unique morphisms m_3 into G_1 and n_3 into G_3, respectively. Pushout decomposition and vertical MPOC-pushout decomposition (cf. Lemma 3.6) complete the situation shown in Figure 3.17b.
Now I_{tr_3} is constructed as a pullback of J_1 and J_2 and D_3 as a pullback of D_1 and D_2 resulting in the cube in Figure 3.17c, where the front faces are pushouts due to the first cube property and the front left face is an MPOC due to the second.
In Figure 3.17d the relevant properties from the previous figures are summarised, where the resulting transformation $G_1 \xRightarrow{tr_3,m_3} G_3$ is obtained by pushout composition and horizontal MPOC composition (cf. Lemma 3.3) and using $(tr_3)_\mathsf{l} := l_1 \circ l_2^*$ and $(tr_3)_\mathsf{r} := r_2 \circ r_1^*$. □

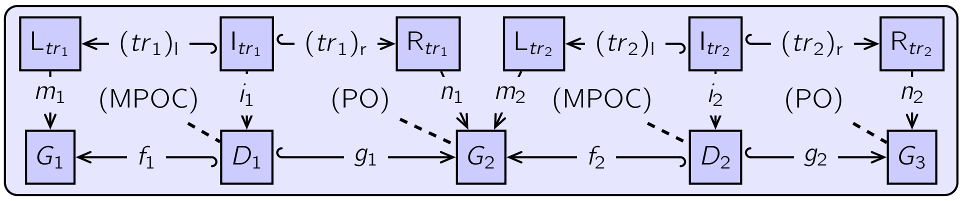
3.4. Negative Application Conditions

In this section, we will enhance the MPOC transformation framework by negative application conditions (NACs). NACs are used to prohibit the application of a rule even if the gluing condition is satisfied. NACs for the adhesive high level replacement categories and systems of [EEPT06] are extensively studied in [Lam07] which is used as a basis of this adaption to MPOC transformations.

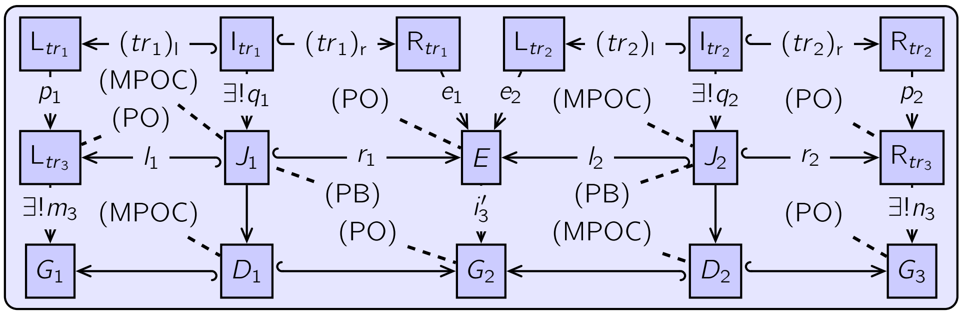
3.4.1. MPOC-PO Transformations with NACs

A negative application condition for an object, typically the left-hand side object of a rule, is given by another object with a morphism from the original object. The NAC is satisfied for a morphism, typically the match for a transformation, if there is no occurrence morphism from the NAC object such that it commutes with the tested morphism. The NAC object can contain additional structures which shall be forbidden to exist but it is also possible to use it to prevent the match from identifying certain elements.

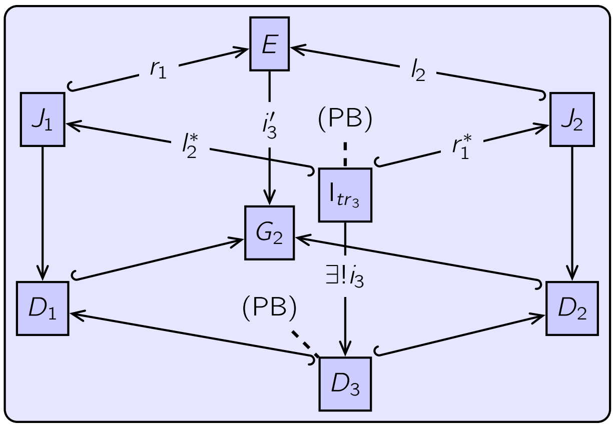
Definition 3.12 (Negative Application Condition)
Given a category $\mathbf{C}$ with a distinguished class $\mathcal{O}$ of morphisms, called *occurrence morphisms*, and an object $L \in |\mathbf{C}|$, a *negative application condition* (NAC) (N, c) on L consists of an object N and a morphism $c\colon L \to N$.


L_{tr_1}, $(tr_1)_l$, I_{tr_1}, $(tr_1)_r$, R_{tr_1}, L_{tr_2}, $(tr_2)_l$, I_{tr_2}, $(tr_2)_r$, R_{tr_2}
m_1, (MPOC), i_1, (PO), n_1, m_2, (MPOC), i_2, (PO), n_2
G_1, f_1, D_1, g_1, G_2, f_2, D_2, g_2, G_3

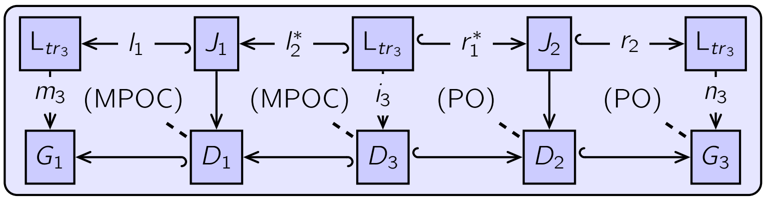
(a) Given transformation sequence


L_{tr_1}, $(tr_1)_l$, I_{tr_1}, $(tr_1)_r$, R_{tr_1}, L_{tr_2}, $(tr_2)_l$, I_{tr_2}, $(tr_2)_r$, R_{tr_2}
p_1, (MPOC), (PO), $\exists! q_1$, (PO), e_1, e_2, (MPOC), $\exists! q_2$, (PO), p_2
L_{tr_3}, l_1, J_1, r_1, E, l_2, J_2, r_2, R_{tr_3}
$\exists! m_3$, (MPOC), (PB), (PO), i_3', (PB), (MPOC), (PO), $\exists! n_3$
G_1, D_1, G_2, D_2, G_3

(b) Decompositions


E, r_1, l_2, J_1, J_2, i_3', (PB), l_2^*, r_1^*, I_{tr_3}, G_2, $\exists! i_3$, D_1, (PB), D_2, D_3

(c) Interface cube


L_{tr_3}, l_1, J_1, l_2^*, L_{tr_3}, r_1^*, J_2, r_2, L_{tr_3}
m_3, (MPOC), (MPOC), i_3, (PO), (PO), n_3
G_1, D_1, D_3, D_2, G_3

(d) Composed transformation

Abbildung 3.17.: Synthesis of sequentially composed rule transformations

A morphism $m\colon L \to G$ *satisfies* (N, c), written as $m \vDash (N, c)$, if there is no morphism $o\colon N \to G$ with $o \in \mathcal{O}$ and $o \circ c = m$ (cf. Figure 3.18).

$$\begin{tikzcd}
N \arrow[dr, "\nexists o"'] & L \arrow[l, "c"'] \arrow[d, "m"] \arrow[dl, phantom, "(=)"] \\
 & G
\end{tikzcd}$$

Abbildung 3.18.: Negative application condition

A new morphism class is introduced for occurrence morphisms since they need to be restricted to some class of non-injective morphisms in order for the NAC to "count" structures, e. g., if a rule shall only be applied to an element if it is the only one with certain properties. If occurrences can be non-injective then the target structure itself matches the NAC and inhibits the transformation. With injective occurrences the target structure and the prohibitted additional structure cannot be identified and the NAC is satisfied.

We do not require this class to be a class of monomorphisms since the appropriate kind of injectiveness not necessarily coincides with a subclass of the monomorphisms. More specifically, in our RDF instantiation a class of occurrences whis is more general than monos will be needed.

In order to just apply transformations no further requirements need to be satisfied and, hence, a NAC transformation category is just a transformation category with the additional class for occurrence morphisms. As for transformation categories before we will from now on silently assume that all of the definitions and lemmas occur within a NAC transformation category.

Definition 3.13 (NAC Transformation Category)
A *NAC transformation category* $(\mathbf{C}, \mathcal{M}, \mathcal{R}, \mathcal{O})$ is a transformation category $(\mathbf{C}, \mathcal{M}, \mathcal{R})$ as in Definition 3.4 with an additional class $\mathcal{O}$ of morphisms.

A NAC transformation rule is then a transformation rule with a set of NACs on the left-hand side and a NAC transformation is a transformation such that the match satisfies all of the NACs.

Definition 3.14 (NAC Transformation Rules and NAC Transformations)
A *NAC transformation rule* (tr, NAC) consists of

- a transformation rule tr as in Definition 3.5 and
- a set NAC of NACs on L_{tr}.

A *NAC transformation* $G \xRightarrow{(tr,NAC),m} H$ is given by

- a transformation $G \xRightarrow{tr,m} H$ as in Definition 3.6,

such that

- $m \models (N, c)$ for all NACs $(N, c) \in NAC$.

3.4.2. Translation of NACs

In order to shift NACs through the structure that defines a sequentially composed rule, we now examine the translation of NACs morphisms and rules as it is examined in detail in [Lam07] for the case of adhesive HLR categories. For this purpose we need a collection of additional properties that is summarised in the following definition.

Definition 3.15 (NAC Transformation Category with Sequential Composition)
A *NAC transformation category with Sequential Compostion* $(\mathbf{C}, \mathcal{M}, \mathcal{R}, \mathcal{O})$ is a NAC transformation category $(\mathbf{C}, \mathcal{M}, \mathcal{R}, \mathcal{O})$ as in Definition 3.13, such that $(\mathbf{C}, \mathcal{M}, \mathcal{R})$ is a transformation category with sequential composition as in Definition 3.10 and

- $\mathcal{M} \subseteq \mathcal{O}$,
- $\mathcal{O}$ is closed under composition,
- pushouts along $\mathcal{M}$ preserve $\mathcal{O}$,
- pullbacks along $\mathcal{M}$ preserve $\mathcal{O}$ and
- $\mathbf{C}$ has epi-$\mathcal{M}$-factorisations.

We first define a downward translation for sets of NACs, where we have to consider all possible overlappings of the NACs and the codomain object.

Definition 3.16 (Downward Translation of NAC)
Given a morphism $t\colon L \to L'$ and a set NAC of NACs on L, then the *downward translation* $\mathrm{Dn}_t(NAC)$ is a set of NACs on L' defined by

$$\mathrm{Dn}_t(NAC) := \bigcup_{(N,c)\in NAC} \{(N', c'\colon L' \to N') \mid \exists o'\colon N \to N' \text{ with } o' \in \mathcal{O}, \\ o' \circ c = c' \circ t \text{ and } c' \text{ and } o' \text{ jointly epi}\}.$$

For this translation we obtain an equivalence in the following lemma, i. e., a morphism satisfies the translated set of NACs if and only the composition with the translation morphism satisfies the original set of NACs.

Lemma 3.8 (Correctness of Downward Translation)
Given a morphism $t\colon L \to L'$, a set NAC of NACs on L and a morphism $m\colon L' \to G$,

then we have that $m \circ t \models (N, c)$ for all $(N, c) \in \mathit{NAC}$ if and only if $m \models (N', c')$ for all $(N', c') \in \mathrm{Dn}_t(\mathit{NAC})$.

Beweis. **If:** Suppose there is a NAC $(N, c) \in \mathit{NAC}$ with $m \circ t \not\models (N, c)$. Then there is an $o\colon N \to G$ with $o \in \mathcal{O}$ and $o \circ c = m \circ t$. We take an epi-$\mathcal{M}$-factorisation of m consiting of $e\colon L' \to m(L')$ and $i\colon m(L') \to G$. Since $i \in \mathcal{M}$ we can construct a pullback (P, i^*, o^*) of i and o, where $o^* \in \mathcal{O}$ due to preservation of $\mathcal{O}$ by pullbacks. There is a unique morphism $c^*\colon L \to P$ with $i^* \circ c^* = c$ and $o^* \circ c^* = e \circ t$ due to the pullback property. Then we can take a pushout (N', i', o') of i^* and o^* with a unique morphism $o^+\colon N' \to G$ with $o^+ \circ o' = o$ and $o^+ \circ i' = i$, where $o' \in \mathcal{O}$ due to pushouts preserving $\mathcal{O}$ and o' and o' and i' are jointly epi. Since jointly epis are composable with epis o' and $i' \circ e$ are also jointly epi and, moreover, $o' \circ c = o' \circ i^* \circ c^* = i' \circ o^* \circ c^* = i' \circ e \circ t$. Hence, $(N', i' \circ e) \in \mathrm{Dn}_t(\mathit{NAC})$. Finally, we also have $o^+\colon N' \to G$ with $o^+ \circ i' \circ e = i \circ e = m$ and, hence, $m \not\models (N', i' \circ e)$.

Only if: Suppose there is a NAC $(N', c') \in \mathrm{Dn}_t(\mathit{NAC})$ with $m \not\models (N', c')$. Then there is an $o\colon N' \to G$ with $o \in \mathcal{O}$ and $o \circ c' = m$. Moreover, by defintion of $\mathrm{Dn}_t(\mathit{NAC})$, there is a NAC $(N, c) \in \mathit{NAC}$ and a morphism $o'\colon N \to N'$ with $o' \in \mathcal{O}$ and $o' \circ c = c' \circ t$. By composition with t we have $o \circ c' \circ t = m \circ t$ and, hence also $o \circ o' \circ c = m \circ t$. Since $\mathcal{O}$ is closed under composition and, hence, $o \circ o' \in \mathcal{O}$ this means $m \circ t \not\models (N, c)$. □

We also need a translation of NACs from the right-hand side of a rule to the left-hand side, where the translation is achieved by double MPOCs, i. e., the translation of a NAC only exists if an MPOC under the interface and a pushout under the left-hand side exists, such that the NAC under the interface is also an MPOC w. r. t. the left-hand side. If no MPOC under the interface exists then the NAC forbids structures that are glued to elements that are created by the right-hand side. Therefore, the NAC cannot match a transformation result which just added these elements without the forbidden structure. If the pushout under the left-hand side is not an MPOC then this means that the NAC and the left-hand side overlap on something that is not in the interface. But such structures are deleted by the MPOC of a transformation and, hence, the NAC can also not match.

Definition 3.17 (Right-to-Left Translation of NACs)
Given a transformation rule tr and a set NAC of NACs on R_{tr}, then the *right-to-left translation* $\mathrm{R2L}_{tr}(\mathit{NAC})$ is a set of NACs on L_{tr} defined by

$$\mathrm{R2L}_{tr}(\mathit{NAC}) := \bigcup_{(N,c)\in \mathit{NAC}} \{(N'', c''\colon \mathrm{L}_{tr} \to N'') \mid \exists (N', c'\colon \mathrm{I}_{tr} \to N'), l'\colon N' \to N'' \text{ and } r'\colon N' \to N \text{ with } (N', r', c') \text{ MPOC of } tr_{\mathrm{r}} \text{ and } c \text{ and } (N', l', c') \text{ MPOC of } tr_{\mathrm{l}} \text{ and } c''\}.$$

(Cf. Figure 3.19.)

$$\begin{tikzcd}
\mathrm{L}_{tr} \arrow[d, "c''"] \arrow[dr, phantom, "\text{(MPOC)}"] & \mathrm{I}_{tr} \arrow[l, hook', "tr_{\mathrm{l}}"'] \arrow[r, hook, "tr_{\mathrm{r}}"] \arrow[d, "c'"] \arrow[dr, phantom, "\text{(MPOC)}"] & \mathrm{R}_{tr} \arrow[d, "c"] \\
N'' & N' \arrow[l, hook', "l'"'] \arrow[r, hook, "r'"] & N
\end{tikzcd}$$

Abbildung 3.19.: Right-to-left translation of NACs

We now show that this right-to-left translation is correct in the sense that for a given transformation the match satisfies the right-to-left translation if and only if the comatch satisfies the original set of NACs.

Lemma 3.9 (Correctness of Right-to-Left Translation)
Given a transformation rule tr, a set NAC of NACs on R_{tr} and a transformation $G \overset{tr,m}{\Longrightarrow} H$ with comatch $n\colon \mathrm{R}_{tr} \to H$, then we have that $n \vDash (N, c)$ for all $(N, c) \in NAC$ if and only if $m \vDash (N'', c'')$ for all $(N'', c'') \in \mathrm{R2L}_{tr}(NAC)$.

Beweis. **If:** Suppose there is a NAC $(N, c) \in NAC$ with $n \nvDash (N, c)$. Then we have an $o\colon N \to H$ with $o \in \mathcal{O}$ and $o \circ c = n$. But then we can take a pullback (N^*, r^*, o^*) of g and o and obtain a unique morphism $c^*\colon I \to N^*$ with $o^* \circ c^* = i$ and $r^* \circ c^* = c \circ tr_{\mathrm{r}}$. Moreover, $o^* \in \mathcal{O}$ because pullbacks preserve $\mathcal{O}$. Then pushout-pullback decomposition leads to (N^*, r^*, c^*) being a pushout complement of tr_{r} and c. Hence, there is also a minimal pushout complement (N', r', c') by Theorem 3.1 2. which induces a unique morphism $n\colon N' \to N^*$ with $n \in \mathcal{M}$, $n \circ c' = n^*$ and $r' = r^* \circ n$. Since $\mathcal{M} \subseteq \mathcal{O}$ and $\mathcal{O}$ is closed under composition we have that $o' := o^* \circ n \in \mathcal{O}$. Now we take the pushout (N'', l', c'') of tr_{l} and c' which induces a unique morphism $o''\colon N'' \to G$ with $o'' \circ c'' = m$ and $o'' \circ l' = f \circ o'$. By pushout decomposition (G, f, o'') is a pushout of l' and o' and $o'' \in \mathcal{O}$ because pushouts preserve $\mathcal{O}$. Moreover, by MPOC pushout decomposition (N', l', c') is an MPOC of tr_{l} and c''. Hence, $(N'', c'') \in \mathrm{R2L}_{tr}(NAC)$, whence, $o'' \circ c'' = m$ implies $m \nvDash (N'', c'')$.

Only if: Suppose there is a NAC $(N'', c'') \in \mathrm{R2L}_{tr}(NAC)$ with $m \nvDash (N'', c'')$. Then we have an $o''\colon N'' \to G$ with $o'' \in \mathcal{O}$ and $o'' \circ c'' = m$. Moreover, due to definition of $\mathrm{R2L}_{tr}(NAC)$ we have $(N, c\colon \mathrm{R}_{tr} \to N) \in NAC$, $(N', c'\colon \mathrm{I}_{tr} \to N')$, $l'\colon N' \to N''$ and $r'\colon N' \to N$ with (N', l', c') MPOC of tr_{l} and c'' and (N', r', c') MPOC of tr_{r} and c. Then we can take a pullback (N^*, l^*, o^*) of f and o'' and obtain a unique morphism $c^*\colon \mathrm{I}_{tr} \to N^*$ with (N^*, l^*, c^*) being also an MPOC of tr_{l} and c'' due to MPOC pullback decomposition. Hence, $N^* \cong N'$ and we also have $o'\colon N' \to D$ such that (N', l', o') is a pullback of f and o'' with $o' \in \mathcal{O}$ due to pullbacks preserving $\mathcal{O}$. Then because of $n \circ tr_{\mathrm{r}} = g \circ i = g \circ o' \circ c'$ there is a unique $o\colon N \to H$ with $o \circ r' = g \circ o'$ and $o \circ c = n$. Due to pushout decomposition (H, g, o) is a pushout of r' and o' with $o \in \mathcal{O}$ due to pushouts preserving $\mathcal{O}$. This means that $m \not\models (N, c)$. □

3.4.3. Composition with NACs

We now show how sequential composition can be enhanced to rules with NACs. Therefore, we first define sequentially composed rules with NACs and then prove that analysis and synthesis is still possible with these notions.

Definition 3.18 (Sequentially Composed Rule with NACs)
Given NAC transformation rules (tr_1, NAC_1), (tr_2, NAC_2) and (tr_3, NAC_3), then (tr_3, NAC_3) is a *sequentially composed rule with NACs* of (tr_2, NAC_2) after (tr_1, NAC_1) if

- tr_3 is a sequentially composed rule of tr_2 after tr_1 as in Definition 3.11 and
- $NAC_3 = \mathrm{Dn}_{p_1}(NAC_1) \cup \mathrm{R2L}_{l_1,r1}(\mathrm{Dn}_{e_2}(NAC_2))$, where the morphisms are defined as in Definition 3.11 and the span consisting of l_1 and r_1 is interpreted as a rule.

The following theorem shows that the analysis and synthesis results can be lifted to sequential composition with NACs. Using the lemmas from the previous sections this is fairly obvious since they have already shown equivalence of the right-to-left and downward translations used in the definition of sequentially composed rules with NACs.

Theorem 3.5 (Analysis and Synthesis of Composed Rule Transformations with NACs)
Given a sequentially composed rule with NACs (tr_3, NAC_3) of (tr_2, NAC_2) after (tr_1, NAC_1) and a transformation $G_1 \xRightarrow{(tr_3,NAC_3),m_3} G_3$, there are an object G_2 and transformations $G_1 \xRightarrow{(tr_1,NAC_1),m_1} G_2$ and $G_2 \xRightarrow{(tr_2,NAC_2),m_2} G_3$.
Given rules (tr_1, NAC_1) and (tr_2, NAC_2) and transformations $G_1 \xRightarrow{(tr_1,NAC_1),m_1} G_2$ and $G_2 \xRightarrow{(tr_2,NAC_2),m_2} G_3$, there are a sequentially composed rule with NACs (tr_3, NAC_3) of (tr_2, NAC_2) after (tr_1, NAC_1) and a transformation $G_1 \xRightarrow{(tr_3,NAC_3),m_3} G_3$.

Beweis. **Analysis:** Since tr_3 is a sequentially composed rule of tr_2 after tr_1 Theorem 3.3 leads to transformations $G_1 \xRightarrow{tr_1,m_1} G_2$ and $G_2 \xRightarrow{tr_2,m_2} G_3$. It remains to show that $m_3 \models (N_3, c_3)$ for all NACs $(N_3, c_3) \in NAC_3$ implies $m_1 \models (N_1, c_1)$ for all NACs $(N_1, c_1) \in NAC_1$ and $m_2 \models (N_2, c_2)$ for all NACs $(N_2, c_2) \in NAC_2$. Firstly, suppose there is a NAC $(N_1, c_1) \in NAC_1$ with $m_1 \not\models (N_1, c_1)$, since $m_1 = m_3 \circ p_1$ Lemma 3.8 then implies that there is a NAC $(N', c') \in \mathrm{Dn}_{p_1}(NAC_1)$ with $m_3 \not\models (N', c')$ but since $\mathrm{Dn}_{p_1}(NAC_1) \subseteq NAC_3$ this is a contradiction to $m_3 \models (N_3, c_3)$ for all NACs $(N_3, c_3) \in NAC_3$. Secondly, suppose there is a NAC $(N_2, c_2) \in NAC_2$ with $m_2 \not\models (N_2, c_2)$, since $m_2 = i_3' \circ e_2$ Lemma 3.8 then implies that there is a NAC $(N', c') \in \mathrm{Dn}_{e_2}(NAC_2)$ with $i_3' \not\models (N', c')$ but then Lemma 3.9 combined with the corresponding MPOC and pushout under l_1 and r_1 in Theorem 3.3 implies that there is a NAC $(N', c') \in \mathrm{R2L}_{l_1,r_1}(\mathrm{Dn}_{e_2}(NAC_2))$ with $m_3 \nvDash (N', c')$ which again is a contradiction since $\mathrm{R2L}_{l_1,r_1}(\mathrm{Dn}_{e_2}(NAC_2)) \subseteq NAC_3$.

Synthesis: Theorem 3.4 allows us to build a sequentially composed rule tr_3 of tr_2 after tr_1 with a transformation $G_1 \xRightarrow{tr_3,m_3} G_3$. It remains to show that $m_1 \vDash (N_1, c_1)$ for all NACs $(N_1, c_1) \in NAC_1$ and $m_2 \vDash (N_2, c_2)$ for all NACs $(N_2, c_2) \in NAC_2$ implies $m_3 \vDash (N_3, c_3)$ for all NACs $(N_3, c_3) \in NAC_3$. Suppose that there is a NAC $(N_3, c_3) \in NAC_3$ with $m_3 \nvDash (N_3, c_3)$. By the construction of the sequentially composed rule with NACs we have $(N_3, c_3) \in \mathrm{Dn}_{p_1}(\mathrm{NAC}_1)$ or $(N_3, c_3) \in \mathrm{R2L}_{l_1,r_1}(\mathrm{Dn}_{e_2}(NAC_2))$. In the first case Lemma 3.8 implies that there is $(N_1, c_1) \in \mathrm{NAC}_1$ with $m_3 \circ p_1 = m_1 \nvDash (N_1, c_1)$ which is a contradiction. In the second case Lemma 3.9 implies that there is a NAC $(N', c') \in \mathrm{Dn}_{e_2}(NAC_2)$ with $i_3' \nvDash (N', c')$ which by Lemma 3.8 in turn implies that there is $(N_2, c_2) \in NAC_2$ with $i_3' \circ e_2 = m_2 \nvDash (N_2, c_2)$ which completes the contradiction.

□

This theorem concludes our treatment of the MPOC transformation framework. In the next chapter it is instantiated to RDF structures by proving requirements that where summarised in Definition 3.4, Definition 3.10, Definition 3.13 and Definition 3.15.

4. RDF Graph Transformations

In this chapter we will instantiate the MPOC transformation framework developed in the previous chapter to RDF graphs. For this purpose, the notion of RDF graph is enhanced to RDF patterns by also including variables in Section 4.1. Pushouts, initial pushouts and MPOCs for these patterns are presented in Section 4.2. Then, the additional properties that are necessary for sequential composition and independence are shown in Section 4.3. As a first application of transformation rules in the context of RDF, inference rules for the ρdf semantics, presented in Section 2.3, are given in Section 4.4.

4.1. RDF Patterns and RDF Graph Transformations

In algebraic graph transformation, the structures of the graphs themselves are usually used as variables in the transformation rules and instantiated to different structures when applying the rules to host graphs. This approach is, however, not possible for RDF graphs since most of the nodes and the predicates are given by globally unique URIs and literals. Therefore, we enhance the notion of RDF graph to RDF patterns which also contain variables that can be instantiated to URIs, literals and blank nodes.[1] A distinguished subset of the variables is restricted to be instantiated to URIs and can be used as predicates (which are not allowed to be blank or literal).

Definition 4.1 (RDF Pattern)
An *RDF pattern* $P = (P_{\mathsf{Blank}}, P_{\mathsf{Var}}, P_{\mathsf{UVar}}, P_{\mathsf{Triple}})$ consists of

- a set P_{Blank} of blank nodes,
- a set P_{Var} of *variables* with a subset $P_{\mathsf{UVar}} \subseteq P_{\mathsf{Var}}$ of *URI variables* and
- a set $P_{\mathsf{Triple}} \subseteq (P_{\mathsf{Var}} + P_{\mathsf{Node}}) \times (P_{\mathsf{UVar}} + \mathsf{URI}) \times (P_{\mathsf{Var}} + P_{\mathsf{Node}})$ of triples, where $P_{\mathsf{Node}} := P_{\mathsf{Blank}} + \mathsf{URI} + \mathsf{PLit} + \mathsf{TLit}$.

Homomorphisms of RDF patterns are now corresponding enhancements of RDF graph homomorphisms, where variables can be (partially) instantiated to URIs, literals and blank nodes, while they may also be mapped to variables in the codomain pattern. URI variables can only be instantiated to URIs or mapped to URI variables since they may appear as

[1] In [BB08], blank nodes were used for this purpose with the drawback of intermingling the roles of blank nodes as existential variables in the logical interpretation of RDF and as transformation variables in structural modifications. Moreover, blank nodes cannot be used as predicates and, therefore, the approach in [BB08] is restricted to transformation rules with fixed predicates.

predicates in triples and an instantiation to blank nodes or literals (and, thus, indirectly also a mapping to general variables) could lead to inconsistent RDF patterns.

Definition 4.2 (RDF Pattern Homomorphism)
An *RDF pattern homomorphism* $h\colon P \to P'$ consists of

- a mapping function $h_{\mathrm{Blank}}\colon P_{\mathrm{Blank}} \to P'_{\mathrm{Blank}}$ for blank nodes and
- an assignment function $h_{\mathrm{Var}}\colon P_{\mathrm{Var}} \to P'_{\mathrm{Var}} + P'_{\mathrm{Node}}$ for variables,

such that

- there is an inclusion $h_{\mathrm{Var}}(P_{\mathrm{UVar}}) \subseteq P'_{\mathrm{UVar}} + \mathrm{URI}$ and
- there is an inclusion $h_{\mathrm{Triple}}(P_{\mathrm{Triple}}) \subseteq P'_{\mathrm{Triple}}$,

where $h_{\mathrm{Triple}}\colon (P_{\mathrm{Var}} + P_{\mathrm{Node}}) \times (P_{\mathrm{UVar}} + \mathrm{URI}) \times (P_{\mathrm{Var}} + P_{\mathrm{Node}}) \to (P'_{\mathrm{Var}} + P'_{\mathrm{Node}}) \times (P'_{\mathrm{UVar}} + \mathrm{URI}) \times (P'_{\mathrm{Var}} + P'_{\mathrm{Node}})$ is defined by

$$h_{\mathrm{Triple}}(s, p, o) := (h_{\mathrm{VN}}(s), h_{\mathrm{VU}}(p), h_{\mathrm{VN}}(o))$$

and $h_{\mathrm{VN}}\colon P_{\mathrm{Var}} + P_{\mathrm{Node}} \to P'_{\mathrm{Var}} + P'_{\mathrm{Node}}$ and $h_{\mathrm{VU}}\colon P_{\mathrm{UVar}} + \mathrm{URI} \to P'_{\mathrm{UVar}} + \mathrm{URI}$ in turn by

$$h_{\mathrm{VN}}(x) := \begin{cases} h_{\mathrm{Var}}(x) & \text{for } x \in P_{\mathrm{Var}} \\ h_{\mathrm{Blank}}(x) & \text{for } x \in P_{\mathrm{Blank}} \\ x & \text{for } x \in \mathrm{URI} + \mathrm{PLit} + \mathrm{TLit} \end{cases} \qquad \text{and}$$

$$h_{\mathrm{VU}}(x) := \begin{cases} h_{\mathrm{Var}}(x) & \text{for } x \in P_{\mathrm{UVar}} \\ x & \text{for } x \in \mathrm{URI} \end{cases} .$$

Note that RDF patterns are very similar to the basic graph patterns used in the SPARQL W3C recommendation in [PS08]. RDF pattern homomorphisms then correspond to the pattern instance mappings defined there, with the difference that in our setting blank nodes may only be mapped to blank nodes, not instantiated to URIs or literals. This variation is motivated by the difference in viewpoints. While SPARQL takes the logical viewpoint and, hence, uses blank nodes as existential variables that can be instantiated in the host graph, we take an algebraic viewpoint, where we want to be able to add and delete blank nodes to and from the host graph. The latter would not be possible if blank nodes could be instantiated since URIs and literals are global and cannot be added or deleted.

RDF patterns and RDF pattern homomorphisms again constitute a category which will be the basis for the transformation categories for RDF, developed in the remainder of this chapter.

Proposition 4.1 (Category **RDFPat**)
RDF patterns and RDF pattern homomorphisms constitute a category **RDFPat**, where compositions are given by $(h \circ g)_{\mathsf{Blank}} := h_{\mathsf{Blank}} \circ g_{\mathsf{Blank}}$ and $(h \circ g)_{\mathsf{Var}} := h_{\mathsf{VN}} \circ g_{\mathsf{Var}}$ for all RDF pattern homomorphisms $g\colon P \to P'$ and $h\colon P' \to P''$ and identities by $(\mathrm{id}_P)_{\mathsf{Blank}} := \mathrm{id}_{P_{\mathsf{Blank}}}$ and $(\mathrm{id}_P)_{\mathsf{Var}} := \mathrm{incl}_{P_{\mathsf{Var}}, P_{\mathsf{Var}}+P_{\mathsf{Node}}} \circ \mathrm{id}_{P_{\mathsf{Var}}}$ for all RDF patterns P.
In this category, the following characterisations of special morphisms hold:

Mono: A homomorphism $m\colon P \to P'$ is a monomorphism if and only if m_{Blank} and m_{Var} are injective and $m_{\mathsf{Var}}(P_{\mathsf{Var}}) \subseteq P'_{\mathsf{Var}} + (P'_{\mathsf{Blank}} \setminus m_{\mathsf{Blank}}(P_{\mathsf{Blank}}))$, i. e., m_{Var} only maps to variables and blank nodes not reached by m_{Blank}, not to URIs, literals and blank nodes reached by m_{Blank}.

Epi: A homomorphism $e\colon P \to P'$ is an epimorphism if and only if e_{Blank} and e_{Var} are jointly surjective on P'_{Blank} and e_{Var} is surjective on P'_{Var}.

Iso: A homomorphism $i\colon P \to P'$ is an isomorphism if and only if i_{Blank} is bijective, i_{Var} is bijective between P_{Var} and P'_{Var} and the equalities $P'_{\mathsf{UVar}} = i_{\mathsf{Var}}(P_{\mathsf{UVar}})$ and $P'_{\mathsf{Triple}} = i_{\mathsf{Triple}}(P_{\mathsf{Triple}})$ are satisfied.

Beweis. For blank nodes, associativity and cancellability of identities follow directly from the corresponding properties in **Set**. For variables, we first observe that $(h \circ g)_{\mathsf{VN}} = h_{\mathsf{VN}} \circ g_{\mathsf{VN}}$, $f_{\mathsf{VN}} \circ \mathrm{incl}_{P_{\mathsf{Var}}, P_{\mathsf{Var}}+P_{\mathsf{Node}}} = f_{\mathsf{Var}}$ and $(\mathrm{id}_P)_{\mathsf{VN}} = \mathrm{id}_{P_{\mathsf{Var}}+P_{\mathsf{Node}}}$. Associativity then follows from $(h \circ (g \circ f))_{\mathsf{Var}} = h_{\mathsf{VN}} \circ (g \circ f)_{\mathsf{Var}} = h_{\mathsf{VN}} \circ g_{\mathsf{VN}} \circ f_{\mathsf{Var}} = (h \circ g)_{\mathsf{VN}} \circ f_{\mathsf{Var}} = ((h \circ g) \circ f)_{\mathsf{Var}}$ and cancellability of identities from $(f \circ \mathrm{id}_P)_{\mathsf{Var}} = f_{\mathsf{VN}} \circ (\mathrm{id}_P)_{\mathsf{Var}} = f_{\mathsf{VN}} \circ \mathrm{incl}_{P_{\mathsf{Var}}, P_{\mathsf{Var}}+P_{\mathsf{Node}}} \circ \mathrm{id}_{P_{\mathsf{Var}}} = f_{\mathsf{Var}} = \mathrm{id}_{P'_{\mathsf{Var}}+P'_{\mathsf{Node}}} \circ f_{\mathsf{Var}} = (\mathrm{id}_{P'})_{\mathsf{VN}} \circ f_{\mathsf{Var}} = (\mathrm{id}_{P'} \circ f)_{\mathsf{Var}}$.

Mono: If: Suppose m_{Blank} and m_{Var} are injective, then $m_{\mathsf{Var}}(P_{\mathsf{Var}}) \subseteq P'_{\mathsf{Var}} + (P'_{\mathsf{Blank}} \setminus m_{\mathsf{Blank}}(P_{\mathsf{Blank}}))$ and $m \circ l_1 = m \circ l_2$. Then $m_{\mathsf{Blank}} \circ (l_1)_{\mathsf{Blank}} = m_{\mathsf{Blank}} \circ (l_2)_{\mathsf{Blank}}$ and, because m_{Blank} is injective, also $(l_1)_{\mathsf{Blank}} = (l_2)_{\mathsf{Blank}}$. Moreover, $m_{\mathsf{VN}} \circ (l_1)_{\mathsf{Var}} = m_{\mathsf{VN}} \circ (l_2)_{\mathsf{Var}}$ and m_{VN} is injective (since m_{Blank} and m_{Var} are injective and m_{Var} does not map variables to URIs, literals or blank nodes already reached by m_{Blank}) and, hence, also $(l_1)_{\mathsf{Var}} = (l_2)_{\mathsf{Var}}$. We obtain $l_1 = l_2$ and, therefore, m is a monomorphism. **Only if:** Suppose either m_{Blank} or m_{Var} is not injective. Then we can construct a counterexample as in Proposition 2.1 and m is not a monomorphism. Now, suppose m_{Blank} and m_{Var} are injective, but $m_{\mathsf{Var}}(P_{\mathsf{Var}}) \nsubseteq P'_{\mathsf{Var}} + (P'_{\mathsf{Blank}} \setminus m_{\mathsf{Blank}}(P_{\mathsf{Blank}}))$. There has to be a variable $x \in P_{\mathsf{Var}}$ and a blank node, URI or literal $y \in P'_{\mathsf{Node}}$ with $m_{\mathsf{Var}}(x) = y$. Then we can define $l_1, l_2\colon (\varnothing, \{*\}, \varnothing, \varnothing) \to P$ with $(l_1)_{\mathsf{Var}}(*) = x$ and $(l_2)_{\mathsf{Var}}(*) = y$ if y is a URI or literal and $(l_2)_{\mathsf{Var}}(*) = z$ if y is a blank node in P'_{Blank} and z a corresponding blank node in P_{Blank} with $m_{\mathsf{Blank}}(z) = y$. We have $m_{\mathsf{VN}}((l_1)_{\mathsf{Var}}(*)) = m_{\mathsf{VN}}((l_2)_{\mathsf{Var}}(*))$, but $(l_1)_{\mathsf{Var}}(*) \neq (l_2)_{\mathsf{Var}}(*)$. Hence, m is not a monomorphism.

Epi: If: Suppose e_{Blank} and e_{Var} are jointly surjective on P'_{Blank}, e_{Var} is surjective on P'_{Var} and $f_1 \neq f_2$. Then there has to be either a blank node $b \in P'_{\mathsf{Blank}}$ with $(f_1)_{\mathsf{Blank}}(b) \neq (f_2)_{\mathsf{Blank}}(b)$ or a variable $x \in P'_{\mathsf{Var}}$ with $(f_1)_{\mathsf{Var}}(x) \neq (f_2)_{\mathsf{Var}}(x)$. In the first case, we have either a blank node $c \in P_{\mathrm{Blank}}$ with $e_{\mathrm{Blank}}(c) = b$ or a variable $z \in P_{\mathrm{Var}}$ with $e_{\mathrm{Var}}(z) = b$ (since e_{Blank} and e_{Var} are jointly surjective) and, hence, $(f_1)_{\mathrm{Blank}}(e_{\mathrm{Blank}}(c)) \neq (f_2)_{\mathrm{Blank}}(e_{\mathrm{Blank}}(c))$ or $(f_1)_{\mathrm{VN}}(e_{\mathrm{Var}}(z)) \neq (f_2)_{\mathrm{VN}}(e_{\mathrm{Var}}(z))$. In the second case, we have a variable $y \in P_{\mathrm{Var}}$ with $e_{\mathrm{Var}}(y) = x$ (since e_{Var} is surjective) and, hence, $(f_1)_{\mathrm{VN}}(e_{\mathrm{Var}}(y)) \neq (f_2)_{\mathrm{VN}}(e_{\mathrm{Var}}(y))$. In all cases, we obtain $f_1 \circ e \neq f_2 \circ e$. Hence, $f_1 \circ e = f_2 \circ e$ already implies $f_1 = f_2$ and e is an epimorphism. **Only if:** Suppose either m_{Blank} and m_{Var} are not jointly surjective on P'_{Blank} or m_{Var} is not surjective on P'_{Var}. Then we can select an unreached blank node $b \in P'_{\mathrm{Blank}}$ in the first case or an unreached variable $x \in P'_{\mathrm{Var}}$ in the second case and construct a counterexample as in Proposition 2.1. Hence, e is not an epimorphism.

Iso: **If:** Suppose i_{Blank} is bijective, i_{Var} is bijective between P_{Var} and P'_{Var}, $P'_{\mathrm{UVar}} = i_{\mathrm{Var}}(P_{\mathrm{UVar}})$ and $P'_{\mathrm{Triple}} = i_{\mathrm{Triple}}(P_{\mathrm{Triple}})$. Then, we have inverses $j_{\mathrm{Blank}} \colon P'_{\mathrm{Blank}} \to P_{\mathrm{Blank}}$ with $j_{\mathrm{Blank}} \circ i_{\mathrm{Blank}} = \mathrm{id}_{P_{\mathrm{Blank}}}$ and $i_{\mathrm{Blank}} \circ j_{\mathrm{Blank}} = \mathrm{id}_{P'_{\mathrm{Blank}}}$ and $j_{\mathrm{Var}} \colon P'_{\mathrm{Var}} \to P_{\mathrm{Var}} + P_{\mathrm{Node}}$ with $j_{\mathrm{VN}} \circ i_{\mathrm{Var}} = \mathrm{incl}_{P_{\mathrm{Var}}, P_{\mathrm{Var}}+P_{\mathrm{Node}}}$ and $i_{\mathrm{VN}} \circ j_{\mathrm{Var}} = \mathrm{incl}_{P'_{\mathrm{Var}}, P'_{\mathrm{Var}}+P'_{\mathrm{Node}}}$. These inverses constitute an RDF pattern homomorphism, since $j_{\mathrm{Var}}(P'_{\mathrm{UVar}}) = j_{\mathrm{Var}}(i_{\mathrm{Var}}(P_{\mathrm{UVar}})) \subseteq P_{\mathrm{UVar}} + \mathrm{URI}$ and $j_{\mathrm{Triple}}(P'_{\mathrm{Triple}}) = j_{\mathrm{Triple}}(i_{\mathrm{Triple}}(P_{\mathrm{Triple}}) = P_{\mathrm{Triple}}$. Hence, i is an isomorphism. **Only if:** Suppose i_{Blank} or i_{Var} are not bijective. Then we cannot find an inverse and i is not an isomorphism. Suppose they are bijective and we have inverses j_{Blank} and j_{Var}, but $P'_{\mathrm{UVar}} \nsubseteq i_{\mathrm{Var}}(P_{\mathrm{UVar}})$. Then $j_{\mathrm{Var}}(P'_{\mathrm{UVar}}) \nsubseteq j_{\mathrm{Var}}(i_{\mathrm{Var}}(P_{\mathrm{UVar}})) = P_{\mathrm{UVar}}$ and, hence, j_{Blank} and j_{Var} do not constitute an RDF pattern homomorphism and i is not an isomorphism. Suppose i_{Blank} and i_{Var} are bijective with inverses j_{Blank} and j_{Var}, but $P'_{\mathrm{Triple}} \nsubseteq i_{\mathrm{Triple}}(P_{\mathrm{Triple}})$. Then $j_{\mathrm{Triple}}(P'_{\mathrm{Triple}}) \nsubseteq j_{\mathrm{Triple}}(i_{\mathrm{Triple}}(P_{\mathrm{Triple}})) = P_{\mathrm{Triple}}$ and, hence, j_{Blank} and j_{Var} do not constitute an RDF pattern homomorphism and i is not an isomorphism. □

Figure 4.1 shows an example of an RDF pattern homomorphism in the metadata application scenario, where a part of the example RDF graph from Figure 2.1 in Section 2.1 is matched by an RDF pattern which is supposed to match a book with title and author expressed in the imp: vocabulary. This is achieved by mapping the blank node α of the pattern to the blank node 1 in the graph, the variable x to the title literal and y to the author's name.

Obviously the definition of underlying vocabularies for RDF graphs can be generalised to underlying vocabularies for RDF patterns and also constitute a functor in this case.

Proposition 4.2 (Underlying Vocabulary Functor)
There is an underlying vocabulary functor $\mathrm{Voc} \colon \mathbf{RDFPat} \to \mathbf{Voc}$ with

- $\mathrm{Voc}(P)_{\mathrm{URI}} := \{u \in \mathrm{URI} \mid \exists (s, p, o) \in P_{\mathrm{Triple}} \colon s = u, p = u \text{ or } o = u\}$,
- $\mathrm{Voc}(P)_{\mathrm{PLit}} := \{l \in \mathrm{PLit} \mid \exists (s, p, o) \in P_{\mathrm{Triple}} \colon s = l \text{ or } o = l\}$ and
- $\mathrm{Voc}(P)_{\mathrm{TLit}} := \{l \in \mathrm{TLit} \mid \exists (s, p, o) \in P_{\mathrm{Triple}} \colon s = l \text{ or } o = l\}$

for all RDF patterns P and $\mathrm{Voc}(h)$ is the inclusion $\mathrm{Voc}(P) \subseteq \mathrm{Voc}(P')$ for all RDF pattern homomorphisms $h \colon P \to P'$.

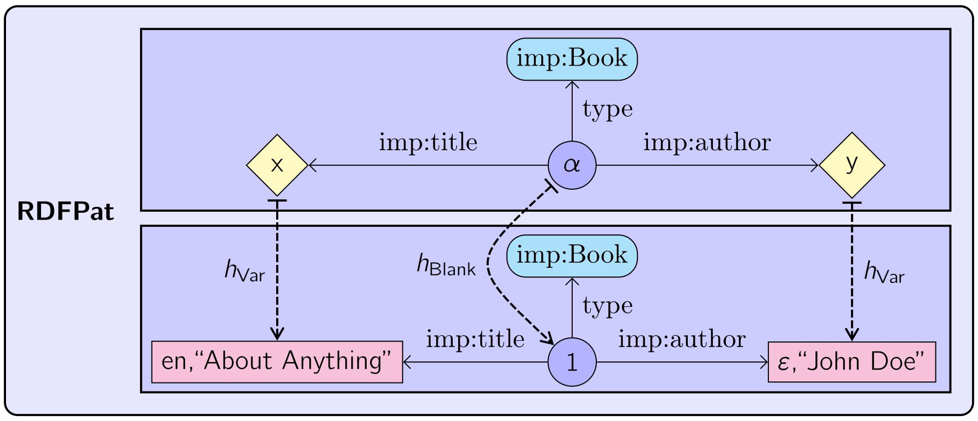


Abbildung 4.1.: Homomorphism in **RDFPat**

Beweis. The functor properties follow directly from the triple inclusion requirement for homomorphisms. □

In order to instantiate the MPOC transformation framework we need to select the classes $\mathcal{M}$ (monomorphisms, for which pushouts and pullbacks exist), $\mathcal{R} \subseteq \mathcal{M}$ (rule morphisms, which are used in transformation rules and in the spans between host patterns and transformation results) and $\mathcal{O} \supseteq \mathcal{M}$ (occurrence morphisms to be used for testing negative application conditions).

For $\mathcal{M}$ we choose to restrict monomorphisms to those, which do not instantiate variables to blank nodes (which is possible for monomorphisms if the blank node is not reached by the blank node function), since such an instantiation could conflict with an instantiation to a URI or literal rendering the construction of a pushout impossible. Moreover, non-URI variables cannot be mapped to URI variables, because this could also lead to a conflict with an instantiation to literals or blank nodes in the match.

For the rule morphisms, we choose to preserve the variables isomorphically since URIs and literals are globally given and, hence, only blank nodes can be added and deleted. But for this purpose, rules can use blank nodes directly and do not need to delete or add variables. Moreover, this choice ensures that transformations applied to RDF graphs (interpreted as RDF patterns) also result in RDF graphs, while the addition of variables would impede this.

For the occurrence morphisms, we choose morphisms o which are injective on blank nodes and are only allowed to instantiate variables to nodes which are not already used in the pattern, i. e., the extension o_{VN} of the variable mapping is injective w. r. t. the combination of variables, blank nodes and URIs and literals from the vocabulary of the domain pattern, but not necessarily w. r. t. all URIs and literals. Hence, occurrence morphisms are more general than monomorphisms which is necessary since otherwise a NAC could never match if a variable is instantiated to a URI or literal in the match of the left-hand side (which will usually be the case).

Definition 4.3 ($\mathcal{M}_{\mathsf{RDF}}$, $\mathcal{R}_{\mathsf{RDF}}$ and $\mathcal{O}_{\mathsf{RDF}}$)
The class $\mathcal{M}_{\mathsf{RDF}}$ is given by all monomorphisms $m\colon P \to P'$ for which $m_{\mathsf{Var}}(P_{\mathsf{Var}}) \subseteq P'_{\mathsf{Var}}$ and $m_{\mathsf{Var}}(P_{\mathsf{UVar}}) = P'_{\mathsf{UVar}} \cap m_{\mathsf{Var}}(P_{\mathsf{Var}})$.
The class $\mathcal{R}_{\mathsf{RDF}}$ is given by all monomorphisms $r\colon P \to P'$ for which r_{Var} is bijective between P_{Var} and P'_{Var} with $P'_{\mathsf{UVar}} = r_{\mathsf{Var}}(P_{\mathsf{UVar}})$.
The class $\mathcal{O}_{\mathsf{RDF}}$ is given by morphisms $o\colon P \to P'$ for which o_{Blank} is injective and for all variables $x \in P_{\mathsf{Var}}$ and all nodes $n \in P_{\mathsf{Var}} + P_{\mathsf{Blank}} + \mathsf{Voc}(P)_{\mathsf{URI}} + \mathsf{Voc}(P)_{\mathsf{PLit}} + \mathsf{Voc}(P)_{\mathsf{TLit}}$ the equality $o_{\mathsf{Var}}(x) = o_{\mathsf{VN}}(n)$ implies $x = n$.

In the following proposition we show the required compositionality and decompositionality properties for $\mathcal{M}$, rule and occurrence morphisms.

Proposition 4.3 (Compositionality of $\mathcal{M}_{\mathsf{RDF}}$, $\mathcal{R}_{\mathsf{RDF}}$ and $\mathcal{O}_{\mathsf{RDF}}$)
$\mathcal{M}_{\mathsf{RDF}}$, $\mathcal{R}_{\mathsf{RDF}}$ and $\mathcal{O}_{\mathsf{RDF}}$ are closed under homomorphism compositions. Moreover, $\mathcal{R}_{\mathsf{RDF}} \subseteq \mathcal{M}_{\mathsf{RDF}}$ and $\mathcal{M}_{\mathsf{RDF}} \subseteq \mathcal{O}_{\mathsf{RDF}}$.

Beweis. Compositionality of $\mathcal{M}_{\mathsf{RDF}}$ and $\mathcal{R}_{\mathsf{RDF}}$ follows immediately from compositionality of inclusions, bijections and equalities.
Compositionality of $\mathcal{O}_{\mathsf{RDF}}$ is satisfied because the underlying vocabulary inclusions ensure that the second occurrence is even more restricted than the first one and, hence, the composition also respects the injectiveness requirement.
$\mathcal{R}_{\mathsf{RDF}} \subseteq \mathcal{M}_{\mathsf{RDF}}$ since bijectiveness of r_{Var} between P_{Var} and P'_{Var} especially implies $r_{\mathsf{Var}}(P_{\mathsf{Var}}) \subseteq P'_{\mathsf{Var}}$.
$\mathcal{M}_{\mathsf{RDF}} \subseteq \mathcal{O}_{\mathsf{RDF}}$ since the requirement for the variable assignment is trivially satisfied if o_{Var} is injective and only maps to variables. □

Since we want to apply transformations to plain RDF graphs, not only to RDF patterns, we observe (as already mentioned in the example above) that all RDF graphs can be interpreted as RDF patterns with empty variable sets. This is formalised as a functor from the category **RDFHom** to the category **RDFPat**. Later, we will show that transformations preserve RDF graphs in the sense that for transformations of images of RDF graphs the results are also images of RDF graphs.

Proposition 4.4 (Functor Lift: **RDFHom** $\to$ **RDFPat**)
There is a functor Lift: **RDFHom** $\to$ **RDFPat** with $\mathsf{Lift}(G) := (G_{\mathsf{Blank}}, \varnothing, \varnothing, G_{\mathsf{Triple}})$ for all RDF graphs G, $\mathsf{Lift}(h)_{\mathsf{Blank}} := h_{\mathsf{Blank}}$ and $\mathsf{Lift}(h)_{\mathsf{Var}} := \mathrm{id}_{\varnothing}$ for all RDF graph homomorphisms h.

Beweis. The functor properties follow obviously since the definitions are equivalent for blank nodes and triples without variables and the empty variable set and the empty variable assignments trivially satisfy all requirements. □

4.2. Pushouts, IPOs and MPOCs for RDF Patterns

In this section, we will give the basic constructions needed for RDF graph transformations, i. e., pushouts and initial pushouts. Moreover, we will show how MPOCs can be constructed directly as a corollary.

Pushouts along $\mathcal{M}_{\mathrm{RDF}}$ can be constructed by importing the host graph of the match and disjointly adding all additional structures from the right-hand side graph.

Proposition 4.5 (Pushouts along $\mathcal{M}_{\mathrm{RDF}}$ preserving $\mathcal{M}_{\mathrm{RDF}}$)
Given RDF pattern homomorphisms $r\colon I \to R$ and $i\colon I \to D$ with $r \in \mathcal{M}_{\mathrm{RDF}}$, a pushout (H, g, n) of r and i with $g \in \mathcal{M}_{\mathrm{RDF}}$ can be constructed by

- $H_{\mathrm{Blank}} := D_{\mathrm{Blank}} + (R_{\mathrm{Blank}} \setminus r_{\mathrm{Blank}}(I_{\mathrm{Blank}}))$ with injections $g_{\mathrm{Blank}}\colon D_{\mathrm{Blank}} \to H_{\mathrm{Blank}}$ and $j_{\mathrm{Blank}}\colon (R_{\mathrm{Blank}} \setminus r_{\mathrm{Blank}}(I_{\mathrm{Blank}})) \to H_{\mathrm{Blank}}$,
- $H_{\mathrm{Var}} := D_{\mathrm{Var}} + (R_{\mathrm{Var}} \setminus r_{\mathrm{Var}}(I_{\mathrm{Var}}))$ with injections $g_{\mathrm{Var}}\colon D_{\mathrm{Var}} \to H_{\mathrm{Var}}$ and $j_{\mathrm{Var}}\colon (R_{\mathrm{Var}} \setminus r_{\mathrm{Var}}(I_{\mathrm{Var}})) \to H_{\mathrm{Var}}$,
- $H_{\mathrm{UVar}} := \{h \in H_{\mathrm{Var}} \mid \exists d \in D_{\mathrm{UVar}}\colon g_{\mathrm{Var}}(d) = h \text{ or } \exists r \in R_{\mathrm{UVar}} \setminus r_{\mathrm{Var}}(I_{\mathrm{Var}})\colon j_{\mathrm{Var}}(r) = h\}$,
- $n_{\mathrm{Blank}}(r) := \begin{cases} g_{\mathrm{Blank}}(i_{\mathrm{Blank}}(i)) & \text{for } r_{\mathrm{Blank}}(i) = r \\ j_{\mathrm{Blank}}(r) & \text{for } r \in (R_{\mathrm{Blank}} \setminus r_{\mathrm{Blank}}(I_{\mathrm{Blank}})) \end{cases}$,
- $n_{\mathrm{Var}}(r) := \begin{cases} g_{\mathrm{VN}}(i_{\mathrm{Var}}(i)) & \text{for } r_{\mathrm{Var}}(i) = r \\ j_{\mathrm{Var}}(r) & \text{for } r \in (R_{\mathrm{Var}} \setminus r_{\mathrm{Var}}(I_{\mathrm{Var}})) \end{cases}$ and
- $H_{\mathrm{Triple}} := g_{\mathrm{Triple}}(D_{\mathrm{Triple}}) \cup n_{\mathrm{Triple}}(R_{\mathrm{Triple}})$.

Beweis.

Well-Definedness: We first have to show that g and n are well-defined RDF pattern homomorphisms. Firstly, $g_{\mathrm{Var}}(D_{\mathrm{UVar}}) \subseteq H_{\mathrm{UVar}} + \mathrm{URI}$, since H_{UVar} is explicitly defined to contain all images of URI variables from D_{UVar}, and $n_{\mathrm{Var}}(R_{\mathrm{UVar}}) \subseteq H_{\mathrm{UVar}} + \mathrm{URI}$, since H_{UVar} explicitly contains all images of additional URI variables from $R_{\mathrm{UVar}} \setminus r_{\mathrm{Var}}(I_{\mathrm{Var}})$ and for all URI variables $x \in R_{\mathrm{UVar}} \cap r_{\mathrm{Var}}(I_{\mathrm{Var}})$ we have a URI variable $i \in I_{\mathrm{UVar}}$ with $r_{\mathrm{Var}}(i) = x$ (since $R_{\mathrm{UVar}} \cap r_{\mathrm{Var}}(I_{\mathrm{Var}}) \subseteq r_{\mathrm{Var}}(I_{\mathrm{UVar}})$), $n_{\mathrm{Var}}(x) = g_{\mathrm{VN}}(i_{\mathrm{Var}}(i))$ (by definition of n_{Var}) and, hence, $n_{\mathrm{Var}}(x) \in H_{\mathrm{UVar}} + \mathrm{URI}$ (via $i_{\mathrm{Var}}(i) \in D_{\mathrm{UVar}} + \mathrm{URI}$). Secondly, $g_{\mathrm{Triple}}(D_{\mathrm{Triple}}) \subseteq H_{\mathrm{Triple}}$ and $n_{\mathrm{Triple}}(R_{\mathrm{Triple}}) \subseteq H_{\mathrm{Triple}}$, since H_{Triple} is explicitly defined to contain both subsets.

Commutativity: We have to show that $g \circ i = n \circ r$. For all $i \in I_{\mathrm{Blank}}$ we have $n_{\mathrm{Blank}}(r_{\mathrm{Blank}}(i)) = g_{\mathrm{Blank}}(i_{\mathrm{Blank}}(i))$ by defintion of n_{Blank} and for all $i \in I_{\mathrm{Var}}$ we have $n_{\mathrm{VN}}(r_{\mathrm{Var}}(i)) = g_{\mathrm{VN}}(i_{\mathrm{Var}}(i))$ by definition of n_{VN}, n_{Var} and n_{Blank}. Thus, commutativity is satisfied.

Existence of Comparison Morphism: Suppose another RDF pattern H' with homomorphisms $g' \colon D \to H'$ and $n' \colon R \to H'$ satisfying $g' \circ i = n' \circ r$. We have to construct a homomorphism $h \colon H \to H'$ with $h \circ g = g'$ and $h \circ n = n'$. For blank nodes, this is achieved by $h_{\mathrm{Blank}}(h) := g'_{\mathrm{Blank}}(d)$ for $d \in D_{\mathrm{Blank}}$ with $g_{\mathrm{Blank}}(d) = h$ and $h_{\mathrm{Blank}}(h) := n'_{\mathrm{Blank}}(r)$ for $r \in R_{\mathrm{Blank}} \setminus r_{\mathrm{Blank}}(I_{\mathrm{Blank}})$ with $j_{\mathrm{Blank}}(r) = h$, which satisfies $h_{\mathrm{Blank}} \circ g_{\mathrm{Blank}} = g'_{\mathrm{Blank}}$ by construction and $h_{\mathrm{Blank}} \circ n_{\mathrm{Blank}} = n'_{\mathrm{Blank}}$ by construction for blank nodes from $R_{\mathrm{Blank}} \setminus r_{\mathrm{Blank}}(I_{\mathrm{Blank}})$ and by $h_{\mathrm{Blank}}(n_{\mathrm{Blank}}(r)) = h_{\mathrm{Blank}}(g_{\mathrm{Blank}}(i_{\mathrm{Blank}}(i))) = g'_{\mathrm{Blank}}(i_{\mathrm{Blank}}(i)) = n'_{\mathrm{Blank}}(r_{\mathrm{Blank}}(i)) = n'_{\mathrm{Blank}}(r)$ for $r \in r_{\mathrm{Blank}}(I_{\mathrm{Blank}})$, $i \in I_{\mathrm{Blank}}$ and $r_{\mathrm{Blank}}(i) = r$. For variables, h is defined by $h_{\mathrm{Var}}(h) := g'_{\mathrm{Var}}(d)$ for $d \in D_{\mathrm{Var}}$ with $g_{\mathrm{Var}}(d) = h$ and $h_{\mathrm{Var}}(h) := n'_{\mathrm{Var}}(r)$ for $r \in R_{\mathrm{Var}} \setminus r_{\mathrm{Var}}(i_{\mathrm{Var}})$ with $j_{\mathrm{Var}}(r) = h$, which satisfies $h_{\mathrm{VN}} \circ g_{\mathrm{Var}} = g'_{\mathrm{Var}}$ by construction and $h_{\mathrm{VN}} \circ n_{\mathrm{Vat}} = n'_{\mathrm{Var}}$ by construction for variables from $R_{\mathrm{Var}} \setminus r_{\mathrm{Var}}(I_{\mathrm{Var}})$ and by $h_{\mathrm{VN}}(n_{\mathrm{Var}}(r)) = h_{\mathrm{VN}}(g_{\mathrm{VN}}(i_{\mathrm{Var}}(i))) = g'_{\mathrm{VN}}(i_{\mathrm{Var}}(i)) = n'_{\mathrm{VN}}(r_{\mathrm{Var}}(i)) = n'_{\mathrm{Var}}(r)$ for $r \in r_{\mathrm{Var}}(I_{\mathrm{Var}})$, $i \in I_{\mathrm{Var}}$ and $r_{\mathrm{Var}}(i) = r$. The homomorphism property $h_{\mathrm{Var}}(H_{\mathrm{UVar}}) \subseteq H'_{\mathrm{UVar}} + \mathrm{URI}$ holds, since $h \in H_{\mathrm{UVar}}$ implies either the existence of $d \in D_{\mathrm{UVar}}$ with $g_{\mathrm{Var}}(d) = h$ or the existence of $r \in R_{\mathrm{UVar}}$ with $n_{\mathrm{Var}}(r) = h$ (or both) by construction of H_{UVar} and, hence, either $h_{\mathrm{Var}}(h) = g'_{\mathrm{Var}}(d)$ or $h_{\mathrm{Var}}(h) = n'_{\mathrm{Var}}(r)$, but this already implies $h_{\mathrm{Var}}(h) \in H'_{\mathrm{UVar}} + \mathrm{URI}$ by the homomorphism properties of g' and n'. Finally, the homomorphism property $h_{\mathrm{Triple}}(H_{\mathrm{Triple}}) \subseteq H'_{\mathrm{Triple}}$ follows from $h_{\mathrm{Triple}}(H_{\mathrm{Triple}}) = h_{\mathrm{Triple}}(g_{\mathrm{Triple}}(D_{\mathrm{Triple}}) \cup n_{\mathrm{Triple}}(R_{\mathrm{Triple}})) = g'_{\mathrm{Triple}}(D_{\mathrm{Triple}}) \cup n'_{\mathrm{Triple}}(R_{\mathrm{Triple}}) \subseteq H'_{\mathrm{Triple}}$ by the homomorphism properties of g' and n'.

Uniqueness of Comparison Morphism: Suppose another morphism $h' \colon H \to H'$ with $h' \circ g = g'$ and $h' \circ n = n'$. We have to show $h' = h$. For all blank nodes $d \in D_{\mathrm{Blank}}$ we have $h'_{\mathrm{Blank}}(g_{\mathrm{Blank}}(d)) = g'_{\mathrm{Blank}}(d) = h_{\mathrm{Blank}}(g_{\mathrm{Blank}}(d))$ and for all blank nodes $r \in R_{\mathrm{Blank}} \setminus r_{\mathrm{Blank}}(I_{\mathrm{Blank}})$ we have $h'_{\mathrm{Blank}}(j_{\mathrm{Blank}}(r)) = h'_{\mathrm{Blank}}(n_{\mathrm{Blank}}(r)) = n'_{\mathrm{Blank}}(r) = h_{\mathrm{Blank}}(n_{\mathrm{Blank}}(r)) = h_{\mathrm{Blank}}(j_{\mathrm{Blank}}(r))$. Hence, $h'_{\mathrm{Blank}} = h_{\mathrm{Blank}}$. For all variables $d \in D_{\mathrm{Var}}$ we have $h'_{\mathrm{VN}}(g_{\mathrm{Var}}(d)) = g'_{\mathrm{Var}}(d) = h_{\mathrm{VN}}(g_{\mathrm{Var}}(d))$ and for all variables $r \in R_{\mathrm{Var}} \setminus r_{\mathrm{Var}}(I_{\mathrm{Var}})$ we have $h'_{\mathrm{VN}}(j_{\mathrm{Var}}(r)) = h'_{\mathrm{VN}}(g_{\mathrm{Var}}(r)) = g'_{\mathrm{Var}}(r) = h_{\mathrm{VN}}(g_{\mathrm{Var}}(r)) = h_{\mathrm{VN}}(j_{\mathrm{Var}}(r))$ and, hence, $h'_{\mathrm{Var}} = h_{\mathrm{Var}}$. Since both functions are identical, we have $h' = h$.

$g \in \mathcal{M}_{\mathrm{RDF}}$**:** Since g_{Blank} and g_{Var} are constructed as injections into a coproduct of sets, they are obviously injective. Moreover, the injection g_{Var} does not instantiate any variable and, hence, $g_{\mathrm{Var}}(D_{\mathrm{Var}}) \subseteq H_{\mathrm{Var}}$. Lastly, $H_{\mathrm{UVar}} \cap g_{\mathrm{Var}}(D_{\mathrm{Var}}) = g_{\mathrm{Var}}(D_{\mathrm{UVar}})$, since all additional URI variables in H_{UVar} have to be from $R_{\mathrm{Var}} \setminus r_{\mathrm{Var}}(I_{\mathrm{Var}})$ which is disjoint from $g_{\mathrm{Var}}(D_{\mathrm{Var}})$ by the coproduct construction in **Set**. $\square$

This pushout construction also preserves the rule and occurrence morphisms.

Proposition 4.6 (Pushouts preserve $\mathcal{R}_{\mathrm{RDF}}$ and $\mathcal{O}_{\mathrm{RDF}}$)
Given RDF pattern homomorphisms $r \colon I \to R$ and $i \colon I \to D$ with $r \in \mathcal{M}_{\mathrm{RDF}}$ and a pushout (H, g, n) of r and i, then $r \in \mathcal{R}_{\mathrm{RDF}}$ implies $g \in \mathcal{R}_{\mathrm{RDF}}$ and $i \in \mathcal{O}_{\mathrm{RDF}}$ implies $n \in \mathcal{O}_{\mathrm{RDF}}$.

Beweis. The preservation of $\mathcal{R}_{\mathsf{RDF}}$ follows immediately from the fact that r_{Var} being a bijection on variables implies that $R_{\mathsf{Var}} \setminus r_{\mathsf{Var}}(l_{\mathsf{Var}})$ is empty and, thus, nothing is added to the variable set and the injection g_{Var} becomes a bijection.
The preservation of $\mathcal{O}_{\mathsf{RDF}}$ follows from n_{Var} being defined like i_{Var} for preserved variables, while new variables are injectively mapped and not instantiated. □

We now present IPOs for RDF pattern homomorphisms, where the definition intuitively amounts to the fact that the boundary contains all blank nodes and variables that are used in context triples, identified to other blank nodes or variables or instantiated to a different kind of element, while the context contains the triples of the host graph not matched by the LHS and all blank nodes and variables needed to represent these triples and the identifications and instantiations of the given morphism.

Proposition 4.7 (IPOs in **RDFPat**)
Given an RDF pattern homomorphism $m\colon L \to G$, an IPO (B, b, C, c, m_B) over m with $b, c \in \mathcal{M}_{\mathsf{RDF}}$ can be constructed by

- $B_{\mathsf{Triple}} := \varnothing$,
- $C_{\mathsf{Triple}} := G_{\mathsf{Triple}} \setminus m_{\mathsf{Triple}}(L_{\mathsf{Triple}})$,
- $\begin{aligned} B_{\mathsf{Blank}} :=& \{b \in L_{\mathsf{Blank}} \mid \exists (s,p,o) \in C_{\mathsf{Triple}}\colon m_{\mathsf{Blank}}(b) = s \text{ or } m_{\mathsf{Blank}}(b) = o\} \cup \\ & \{b \in L_{\mathsf{Blank}} \mid \exists b' \in L_{\mathsf{Blank}}\colon b \neq b' \text{ and } m_{\mathsf{Blank}}(b) = m_{\mathsf{Blank}}(b')\} \cup \\ & \{b \in L_{\mathsf{Blank}} \mid \exists x \in L_{\mathsf{Var}}\colon m_{\mathsf{Blank}}(b) = m_{\mathsf{Var}}(x)\}\end{aligned}$,
- $b_{\mathsf{Blank}} := \mathsf{incl}_{B_{\mathsf{Blank}}, L_{\mathsf{Blank}}}$,
- $C_{\mathsf{Blank}} := (G_{\mathsf{Blank}} \setminus m_{\mathsf{Blank}}(L_{\mathsf{Blank}})) \cup m_{\mathsf{Blank}}(B_{\mathsf{Blank}})$,
- $c_{\mathsf{Blank}} := \mathsf{incl}_{C_{\mathsf{Blank}}, G_{\mathsf{Blank}}}$,
- $(m_B)_{\mathsf{Blank}} := m_{\mathsf{Blank}} \circ \mathsf{incl}_{B_{\mathsf{Blank}}, L_{\mathsf{Blank}}}$,
- $\begin{aligned} B_{\mathsf{Var}} :=& \{x \in L_{\mathsf{Var}} \mid \exists (s,p,o) \in C_{\mathsf{Triple}}\colon m_{\mathsf{Var}}(x) = s, m_{\mathsf{Var}}(x) = p \text{ or} \\ & m_{\mathsf{Var}}(x) = o\} \cup \\ & \{x \in L_{\mathsf{Var}} \mid \exists x' \in L_{\mathsf{Var}}\colon x \neq x' \text{ and } m_{\mathsf{Var}}(x) = m_{\mathsf{Var}}(x')\} \cup \\ & \{x \in L_{\mathsf{Var}} \mid \exists n \in G_{\mathsf{Node}}\colon m_{\mathsf{Var}}(x) = n\} \cup \\ & \{x \in L_{\mathsf{Var}} \setminus L_{\mathsf{UVar}} \mid m_{\mathsf{Var}}(x) \in G_{\mathsf{UVar}}\}\end{aligned}$,
- $B_{\mathsf{UVar}} := B_{\mathsf{Var}} \cap L_{\mathsf{UVar}}$,
- $b_{\mathsf{Var}} := \mathsf{incl}_{B_{\mathsf{Var}}, L_{\mathsf{Var}}}$,
- $C_{\mathsf{Var}} := (G_{\mathsf{Var}} \setminus m_{\mathsf{Var}}(L_{\mathsf{Var}})) \cup \{x \in G_{\mathsf{Var}} \mid \exists b \in B_{\mathsf{Var}}\colon x = m_{\mathsf{Var}}(b)\}$,
- $C_{\mathsf{UVar}} := C_{\mathsf{Var}} \cap G_{\mathsf{UVar}}$,

- $c_{\mathrm{Var}} := \mathrm{incl}_{C_{\mathrm{Var}},G_{\mathrm{Var}}}$ and
- $(m_B)_{\mathrm{Var}} := m_{\mathrm{Var}} \circ \mathrm{incl}_{B_{\mathrm{Var}},L_{\mathrm{Var}}}$.

Beweis.

Well-Definedness: The boundary morphism b and the context morphism c are well-defined, since they just consist of inclusions and the homomorphism properties $b_{\mathrm{UVar}}(B_{\mathrm{UVar}}) = B_{\mathrm{UVar}} \subseteq L_{\mathrm{UVar}}$, $b_{\mathrm{Triple}}(B_{\mathrm{Triple}}) = \varnothing \subseteq L_{\mathrm{Triple}}$, $c_{\mathrm{UVar}}(C_{\mathrm{UVar}}) = C_{\mathrm{UVar}} \subseteq G_{\mathrm{UVar}}$ and $c_{\mathrm{Triple}}(C_{\mathrm{Triple}}) = C_{\mathrm{Triple}} \subseteq G_{\mathrm{Triple}}$ all hold by construction. The blank node function $(m_B)_{\mathrm{Blank}} = m_{\mathrm{Blank}} \circ \mathrm{incl}_{B_{\mathrm{Blank}},L_{\mathrm{Blank}}}$ is well-defined, since $m_{\mathrm{Blank}}(B_{\mathrm{Blank}})$ is explicitly included in C_{Blank} by definition. The assignment function $(m_B)_{\mathrm{Var}} = m_{\mathrm{Var}} \circ \mathrm{incl}_{B_{\mathrm{Var}},L_{\mathrm{Var}}}$ is well-defined, since all variables reachable from B_{Var} are explicitly included in C_{Var} by definition, while blank nodes reachable from B_{Var} are either in $G_{\mathrm{Blank}} \setminus m_{\mathrm{Blank}}(L_{\mathrm{Blank}})$ (if they are not reached by m_{Blank}) or in $m_{\mathrm{Blank}}(B_{\mathrm{Blank}})$ (if they are reached by both, m_{Blank} and m_{Var}). The homomorphism property $(m_B)_{\mathrm{Var}}(B_{\mathrm{UVar}}) \subseteq C_{\mathrm{UVar}} + \mathrm{URI}$ holds, since for all $x \in B_{\mathrm{UVar}}$ we have $x \in B_{\mathrm{Var}}$, $x \in L_{\mathrm{UVar}}$ and $(m_B)_{\mathrm{Var}}(x) = m_{\mathrm{Var}}(x)$. By the homomorphism property of m we obtain $m_{\mathrm{Var}}(x) \in G_{\mathrm{UVar}} + \mathrm{URI}$. While $m_{\mathrm{Var}}(x) \in \mathrm{URI}$ obviously satisfies $m_{\mathrm{Var}}(x) \in C_{\mathrm{UVar}} + \mathrm{URI}$, $m_{\mathrm{Var}}(x) \in G_{\mathrm{UVar}} \subseteq G_{\mathrm{Var}}$, together with $x \in B_{\mathrm{Var}}$, implies $m_{\mathrm{Var}}(x) \in C_{\mathrm{Var}}$ (by construction of C_{Var}) and, hence, also $m_{\mathrm{Var}}(x) \in C_{\mathrm{Var}} \cap G_{\mathrm{UVar}} = C_{\mathrm{UVar}} \subseteq C_{\mathrm{UVar}} + \mathrm{URI}$. Finally, the homomorphism property $(m_B)_{\mathrm{Triple}}(B_{\mathrm{Triple}}) \subseteq C_{\mathrm{Triple}}$ is obviously satisfied by $B_{\mathrm{Triple}} = \varnothing$.

Pushout property: First, $c \circ m_B = m \circ b$ holds by definition of m_B as restriction of m (and b and c consisting solely of inclusions). Now, suppose an RDF pattern G' and homomorphisms $c' \colon C \to G'$ and $m' \colon L \to G'$ with $c' \circ m_B = m' \circ b$. We have to construct a homomorphism $g \colon G \to G'$ with $g \circ c = c'$ and $g \circ m = m'$. For blank nodes, we are forced to choose $g_{\mathrm{Blank}}(g) := c'_{\mathrm{Blank}}(g)$ for all $g \in C_{\mathrm{Blank}}$ and $g_{\mathrm{Blank}}(g) := m'_{\mathrm{Blank}}(l)$ for all $l \in L_{\mathrm{Blank}}$ with $m_{\mathrm{Blank}}(l) = g$ in order to satisfy the required equalities. This completely determines g_{Blank}, since all blank nodes of G_{Blank} are either reached by $G_{\mathrm{Blank}} \setminus m_{\mathrm{Blank}}(L_{\mathrm{Blank}}) \subseteq C_{\mathrm{Blank}}$ or by $m_{\mathrm{Blank}}(L_{\mathrm{Blank}})$. Moreover, it is well-defined, since conflicts on blank nodes $g \in m_{\mathrm{Blank}}(B_{\mathrm{Blank}}) = C_{\mathrm{Blank}} \cap m_{\mathrm{Blank}}(L_{\mathrm{Blank}})$ are impeded by the commutativity $c'_{\mathrm{Blank}}(m_{\mathrm{Blank}}(b)) = m'_{\mathrm{Blank}}(b_{\mathrm{Blank}}(b))$ and conflicts on blank nodes $l, l' \in L_{\mathrm{Blank}}$ with $l \neq l'$ and $m_{\mathrm{Blank}}(l) = m_{\mathrm{Blank}}(l')$ are prevented, since these nodes have to be in B_{Blank} by construction. For variables, we have to choose $g_{\mathrm{Var}}(g) := c'_{\mathrm{Var}}(g)$ for all $g \in C_{\mathrm{Var}}$ and $g_{\mathrm{Var}}(g) := m'_{\mathrm{Var}}(l)$ for all $l \in L_{\mathrm{Var}}$ with $m_{\mathrm{Var}}(l) = g$. This again completely determines g_{Var}, since all variables of G_{Var} are either in $G_{\mathrm{Var}} \setminus m_{\mathrm{Var}}(L_{\mathrm{Var}}) \subseteq C_{\mathrm{Var}}$ or in $m_{\mathrm{Var}}(L_{\mathrm{Var}})$. Analogously to blank nodes, conflicts are avoided, since all possible conflicting cases are already present in B_{Var} and therefore prohibited by $c'_{\mathrm{VN}} \circ (m_B)_{\mathrm{Var}} = m'_{\mathrm{VN}} \circ b_{\mathrm{Var}}$. The homomorphism property $g_{\mathrm{Var}}(G_{\mathrm{UVar}}) \subseteq G'_{\mathrm{UVar}} + \mathrm{URI}$ is ensured, since all $g \in G_{\mathrm{UVar}}$ are either images of $l \in L_{\mathrm{UVar}}$ with $m_{\mathrm{Var}}(l) = g$ and $g_{\mathrm{Var}}(g) = m'_{\mathrm{Var}}(l) \in G'_{\mathrm{UVar}} + \mathrm{URI}$ holds due to the homomorphism property of m' or $g \in C_{\mathrm{UVar}} = C_{\mathrm{Var}} \cap G_{\mathrm{UVar}}$ and

$g_{\text{Var}}(g) = c'_{\text{Var}}(g) \in G'_{\text{UVar}} + \text{URI}$ due to the corresponding property of c'. This is complete, since $G_{\text{UVar}} = C_{\text{UVar}} \cup m_{\text{Var}}(L_{\text{UVar}})$ is ensured by all $l \in L_{\text{Var}} \setminus L_{\text{UVar}}$ with $m_{\text{Var}}(l) \in G_{\text{UVar}}$ being in B_{Var} and, therefore $m_{\text{Var}}(l) \in C_{\text{UVar}}$. Finally, the homomorphism property $g_{\text{Triple}}(G_{\text{Triple}}) \subseteq G'_{\text{Triple}}$ holds, since each triple $(s, p, o) \in G_{\text{Triple}}$ is either from $C_{\text{Triple}} = G_{\text{Triple}} \setminus m_{\text{Triple}}(L_{\text{Triple}})$ or from $m_{\text{Triple}}(L_{\text{Triple}})$ and the homomorphism properties of c' and m' ensure $g_{\text{Triple}}(s, p, o) \in G'_{\text{Triple}}$, respectively.

Initiality: Without loss of generality, we will assume that b' and c' consist of inclusions and the pushout (G, c', m) of b' and m'_B is constructed by Proposition 4.5. We have to show that $B_{\text{Blank}} \subseteq B'_{\text{Blank}}$, $B_{\text{Var}} \subseteq B'_{\text{Var}}$, $B_{\text{UVar}} \subseteq B'_{\text{UVar}}$, $B_{\text{Triple}} \subseteq B'_{\text{Triple}}$, $C_{\text{Blank}} \subseteq C'_{\text{Blank}}$, $C_{\text{Var}} \subseteq C'_{\text{Var}}$, $C_{\text{UVar}} \subseteq C'_{\text{UVar}}$, $C_{\text{Triple}} \subseteq C'_{\text{Triple}}$ and $\text{incl}_{C,C'} \circ m_B = m'_B \circ \text{incl}_{B,B'}$. Since $B_{\text{Triple}} = \varnothing$, $B_{\text{Triple}} \subseteq B'_{\text{Triple}}$ is obviously satisfied. For $C_{\text{Triple}} = G_{\text{Triple}} \setminus m_{\text{Triple}}(L_{\text{Triple}})$ we assume a triple $t \in C_{\text{Triple}}$ ($t \in G_{\text{Triple}}$ and $t \notin m_{\text{Triple}}(L_{\text{Triple}})$). According to Proposition 4.5, we have $G_{\text{Triple}} = C'_{\text{Triple}} \cup m_{\text{Triple}}(L_{\text{Triple}})$ and especially $t \in G_{\text{Triple}}$ implies $t \in C'_{\text{Triple}}$ or $t \in m_{\text{Triple}}(L_{\text{Triple}})$, but the second case is prohibited by the assumption and, hence, we can conclude $t \in C'_{\text{Triple}}$.

$b, c \in \mathcal{M}_{\text{RDF}}$**:** The blank node functions b_{Blank} and c_{Blank} and the assignment functions b_{Var} and c_{Var} are inclusions and, therefore, injective. Moreover, the assignment functions do not instantiate any variable and satisfy $b_{\text{Var}}(B_{\text{UVar}}) = B_{\text{UVar}} = B_{\text{Var}} \cap L_{\text{UVar}} = L_{\text{UVar}} \cap b_{\text{Var}}(B_{\text{Var}})$ and $c_{\text{Var}}(C_{\text{UVar}}) = C_{\text{UVar}} = C_{\text{Var}} \cap G_{\text{UVar}} = G_{\text{UVar}} \cap c_{\text{Var}}(C_{\text{Var}})$ by definition. □

Since it is much more feasible to implement MPOCs directly than to employ the theoretical construction via IPOs, we give the direct construction of MPOCs as a corollary.

Corollary 4.1 (MPOCs in **RDFPat**)
Given RDF pattern homomorphisms $l\colon I \to L$ and $m\colon L \to G$ with $l \in \mathcal{M}_{\text{RDF}}$, such that the gluing condition is satisfied, an MPOC (D, f, i) of l and m with $f \in \mathcal{M}_{\text{RDF}}$ can be constructed by

- $D_{\text{Blank}} := G_{\text{Blank}} \setminus m_{\text{Blank}}(L_{\text{Blank}} \setminus l_{\text{Blank}}(I_{\text{Blank}}))$,
- $D_{\text{Var}} := G_{\text{Var}} \setminus m_{\text{Var}}(L_{\text{Var}} \setminus l_{\text{Var}}(I_{\text{Var}}))$,
- $D_{\text{UVar}} := D_{\text{Var}} \cap G_{\text{UVar}}$
- $f_{\text{Blank}} := \text{incl}_{D_{\text{Blank}}, G_{\text{Blank}}}$,
- $f_{\text{Var}} := \text{incl}_{D_{\text{Var}}, G_{\text{Var}}}$,
- $i_{\text{Blank}} := m_{\text{Blank}} \circ l_{\text{Blank}}$,
- $i_{\text{Var}} := m_{\text{Var}} \circ l_{\text{Var}}$ and
- $D_{\text{Triple}} := G_{\text{Triple}} \setminus m_{\text{Triple}}(L_{\text{Triple}} \setminus l_{\text{Triple}}(I_{\text{Triple}}))$.

Beweis. This follows immediately from combining the constructions in Proposition 4.7 and Proposition 4.5 due to Theorem 3.1. □

With these structures we are able to instantiate the MPOC transformation framework for RDF patterns which is summarised in the following corollary.

Corollary 4.2 (NAC Transformation Category for RDF Patterns)
($\mathbf{RDFPat}, \mathcal{M}_{\mathsf{RDF}}, \mathcal{R}_{\mathsf{RDF}}, \mathcal{O}_{\mathsf{RDF}}$) is a NAC transformation category as given in Definition 3.4 and Definition 3.13.

Beweis. In particular, $\mathcal{M}_{\mathsf{RDF}}$ and $\mathcal{R}_{\mathsf{RDF}}$ are closed under composition as shown in Proposition 4.3, pushouts along $\mathcal{M}_{\mathsf{RDF}}$ preserving $\mathcal{M}_{\mathsf{RDF}}$ exist due to Proposition 4.5, pushouts preserve $\mathcal{R}_{\mathsf{RDF}}$ according to Proposition 4.6 and IPOs exist for all morphisms according to Proposition 4.7. □

Because we want to apply the transformations to plain RDF graphs without variables it is an important result that the transformation of RDF graphs interpreted as RDF patterns again yields RDF graphs.

Proposition 4.8 (Transformations in RDFPat preserve RDF Graphs)
Given an RDF graph $G \in |\mathbf{RDFHom}|$ and a RDF pattern transformation $\mathrm{Lift}(G) \overset{tr,m}{\Longrightarrow} P'$, then there exists an RDF graph $G' \in |\mathbf{RDFHom}|$ with $\mathrm{Lift}(G') = P'$.

Beweis. This follows immediately from the fact that morphisms in $\mathcal{R}_{\mathsf{RDF}}$ are bijective on the variable sets. Since the variable sets of images of the Lift functor are empty sets, the variable sets of the transformation result are also empty and an RDF graph which has this pattern as image is obtained by just omitting these empty variable sets. □

4.3. Composition and Independence for RDF Patterns

In this section, we present the additional constructions and properties that are necessary to instantiate the sequential composition operation of the MPOC transformation framework.

Proposition 4.9 (Pullbacks along Monos preserving Monos in **RDFPat**)
Given RDF pattern homomorphisms $g\colon D \to H$ and $n\colon R \to H$ with $g \in \mathcal{M}_{\mathsf{RDF}}$, a pullback (I, r, i) of g and n with $r \in \mathcal{M}_{\mathsf{RDF}}$ can be constructed by

- $I_{\mathsf{Blank}} := \{r \in R_{\mathsf{Blank}} \mid \exists d \in D_{\mathsf{Blank}} \colon n_{\mathsf{Blank}}(r) = g_{\mathsf{Blank}}(d)\}$,
- $I_{\mathsf{Var}} := \{r \in R_{\mathsf{Var}} \mid \exists d \in D_{\mathsf{Var}} + D_{\mathsf{Node}} \colon n_{\mathsf{Var}}(r) = g_{\mathsf{VN}}(d)\}$,
- $I_{\mathsf{UVar}} := I_{\mathsf{Var}} \cap R_{\mathsf{UVar}}$,
- $r_{\mathsf{Blank}} := \mathrm{incl}_{I_{\mathsf{Blank}}, R_{\mathsf{Blank}}}$,

- $r_{\text{Var}} := \text{incl}_{I_{\text{Var}},R_{\text{Var}}}$,
- $i_{\text{Blank}}(i) := d$ for $d \in D_{\text{Blank}}$ with $n_{\text{Blank}}(i) = g_{\text{Blank}}(d)$,
- $i_{\text{Var}}(i) := d$ for $d \in D_{\text{Var}} + D_{\text{Node}}$ with $n_{\text{Var}}(i) = g_{\text{VN}}(d)$ and
- $I_{\text{Triple}} := \{(s, p, o) \in R_{\text{Triple}} \mid \exists (s', p', o') \in D_{\text{Triple}} \colon n_{\text{Triple}}(s, p, o) = g_{\text{Triple}}(s', p', o')\}$.

Beweis.

Well-Definedness: The homomorphism r is well-defined, since r_{Blank} and r_{Var} are inclusions and I_{UVar} and I_{Triple} are constructed as subsets of R_{UVar} and R_{Triple}, respectively. The functions i_{Blank} and i_{Var} are well-defined, since the existence of the funtion results is ensured by the construction of I_{Blank} and I_{Var}, respectively, and the uniqueness by the injectivity of g_{Blank} and g_{VN}. The homomorphism property $i_{\text{Var}}(I_{\text{UVar}}) \subseteq D_{\text{UVar}} + \text{URI}$ follows from $I_{\text{UVar}} \subseteq R_{\text{UVar}}$ (by construction of I_{UVar}), $n_{\text{Var}}(R_{\text{UVar}}) \subseteq H_{\text{UVar}} + \text{URI}$ (by the homomorphism property of n) and the fact that $g_{\text{VN}}(d) = n_{\text{Var}}(i) \in H_{\text{UVar}} + \text{URI}$ implies $d \in D_{\text{UVar}} + \text{URI}$ (by $g \in \mathcal{M}_{\text{RDF}}$). Finally, the homomorphism property $i_{\text{Triple}}(I_{\text{Triple}}) \subseteq D_{\text{Triple}}$ is ensured by the construction of I_{Triple}.

Commutativity: For the commutativity $g \circ i = n \circ r$ we have $g_{\text{Blank}}(i_{\text{Blank}}(i)) = n_{\text{Blank}}(i)$ for all $i \in I_{\text{Blank}}$ and $g_{\text{VN}}(i_{\text{Var}}(i)) = n_{\text{Var}}(i)$ for all $i \in I_{\text{Var}}$, since the construction of i chooses exactly those (unique) elements of D_{Blank} and $D_{\text{Var}} + D_{\text{Node}}$, respectively, satisfying these equations.

Existence of Comparison Morphism: Suppose another RDF pattern I' with homomorphisms $r' \colon I' \to R$ and $i' \colon I' \to D$ satisfying $g \circ i' = n \circ r'$. We have to construct a homomorphism $x \colon I' \to I$ with $r \circ x = r'$ and $i \circ x = i'$. For blank nodes, this is achieved by $x_{\text{Blank}}(i') := r'_{\text{Blank}}(i') = r$ which is well-defined, since $n_{\text{Blank}}(r'_{\text{Blank}}(i')) = g_{\text{Blank}}(i'_{\text{Blank}}(i'))$ implies that $d = i'_{\text{Blank}}(i') \in D_{\text{Blank}}$ with $n_{\text{Blank}}(r) = g_{\text{Blank}}(d)$ exists and, hence, $r \in I_{\text{Blank}}$ holds by construction of I_{Blank}. This definition satisfies $r_{\text{Blank}} \circ x_{\text{Blank}} = r'_{\text{Blank}}$ by definition and $i_{\text{Blank}} \circ x_{\text{Blank}} = i'_{\text{Blank}}$ by $g_{\text{Blank}} \circ i_{\text{Blank}} \circ x_{\text{Blank}} = n_{\text{Blank}} \circ r_{\text{Blank}} \circ x_{\text{Blank}} = n_{\text{Blank}} \circ r'_{\text{Blank}} = g_{\text{Blank}} \circ i'_{\text{Blank}}$ and g_{Blank} being a monomorphism. For variables, we analogously define $x_{\text{Var}}(i') := r'_{\text{Var}}(i') = r$ which is again well-defined, since $n_{\text{VN}}(r'_{\text{Var}}(i')) = g_{\text{VN}}(i'_{\text{Var}}(i'))$ implies that $d = i'_{\text{Var}}(i') \in D_{\text{Var}} + D_{\text{Node}}$ with $n_{\text{VN}}(r) = g_{\text{VN}}(d)$ exists and, hence, $r \in I_{\text{Var}}$ holds for $r \in R_{\text{Var}}$ by construction of I_{Var} and $r \in I_{\text{Blank}}$ holds for $r \in R_{\text{Blank}}$ by construction of I_{Blank}. This definition also satisfies $r_{\text{VN}} \circ x_{\text{Var}} = r'_{\text{Var}}$ by definition and $i_{\text{VN}} \circ x_{\text{Var}} = i'_{\text{Var}}$ by $g_{\text{VN}} \circ i_{\text{VN}} \circ x_{\text{VN}} = n_{\text{VN}} \circ r_{\text{VN}} \circ x_{\text{VN}} = n_{\text{VN}} \circ r'_{\text{Var}} = g_{\text{VN}} \circ i'_{\text{VN}}$ and g_{VN} being a injective and, hence, a monomorphism. The homomorphism property $x_{\text{Var}}(I'_{\text{UVar}}) \subseteq I_{\text{UVar}} + \text{URI}$ follows from $i' \in I'_{\text{UVar}}$ implying $r'_{\text{Var}}(i') = r_{\text{VN}}(x_{\text{Var}}(i')) = x_{\text{Var}}(i') \in R_{\text{UVar}}$ and $i'_{\text{Var}}(i') = i_{\text{VN}}(x_{\text{Var}}(i')) \in D_{\text{UVar}}$ (by the homomorphism properties of r' and i') and, together with $g_{\text{VN}}(i_{\text{VN}}(x_{\text{Var}}(i))) = n_{\text{VN}}(r_{\text{VN}}(x_{\text{Var}}(i'))) = n_{\text{VN}}(x_{\text{Var}}(i'))$ and the definitions of I_{Var} and I_{UVar} also $x_{\mathsf{Var}}(i') \in I_{\mathsf{UVar}} + \mathsf{URI}$. Finally, the homomorphism property $x_{\mathsf{Triple}}(I'_{\mathsf{Triple}}) \subseteq I_{\mathsf{Triple}}$ holds by the fact, that for all $(s, p, o) \in I'_{\mathsf{Triple}}$ we have $r'_{\mathsf{Triple}}(s, p, o) = r_{\mathsf{Triple}}(x_{\mathsf{Triple}}(s, p, o)) = x_{\mathsf{Triple}}(s, p, o) \in R_{\mathsf{Triple}}$ and $i'_{\mathsf{Triple}}(s, p, o) = i_{\mathsf{Triple}}(x_{\mathsf{Triple}}(s, p, o)) \in D_{\mathsf{Triple}}$ by the homomorphism properties of r' and i' which, together with $g_{\mathsf{Triple}}(i_{\mathsf{Triple}}(x_{\mathsf{Triple}}(s, p, o))) = n_{\mathsf{Triple}}(r_{\mathsf{Triple}}(x_{\mathsf{Triple}}(s, p, o))) = n_{\mathsf{Triple}}(x_{\mathsf{Triple}}(s, p, o))$ and the definition of I_{Triple}, implies $x_{\mathsf{Triple}}(s, p, o) \in I_{\mathsf{Triple}}$.

Uniqueness of Comparison Morphism: Suppose another morphism $x' \colon I' \to I$ with $r \circ x' = r'$ and $i \circ x' = i'$. We have to show $x' = x$. Since r is a monomorphism and $r \circ x' = r' = r \circ x$, we can immediately conclude $x' = x$.

$r \in \mathcal{M}_{\mathsf{RDF}}$: The functions r_{Blank} and r_{Var} are constructed as inclusions, which are obviously injective. Moreover, the inclusion r_{Var} does not instantiate any variable and $r_{\mathsf{Var}}(I_{\mathsf{UVar}}) = I_{\mathsf{UVar}} = R_{\mathsf{UVar}} \cap I_{\mathsf{Var}} = R_{\mathsf{UVar}} \cap r_{\mathsf{Var}}(I_{\mathsf{Var}})$ holds by construction of I_{UVar}. □

In the MPOC transformation framework, we additionally need the preservation of the occurrence morphisms in $\mathcal{O}_{\mathsf{RDF}}$ by pullbacks.

Proposition 4.10 (Pullbacks preserve $\mathcal{O}_{\mathsf{RDF}}$)
Given RDF pattern homomorphisms $g \colon D \to H$ and $n \colon R \to H$ with $g \in \mathcal{M}_{\mathsf{RDF}}$ and a pullback (I, r, i) of g and n, then $n \in \mathcal{O}_{\mathsf{RDF}}$ implies $i \in \mathcal{O}_{\mathsf{RDF}}$.

Beweis. Suppose $n \in \mathcal{O}_{\mathsf{RDF}}$. Firstly, i_{Blank} is injective by decomposition of $g_{\mathsf{Blank}} \circ i_{\mathsf{Blank}} = n_{\mathsf{Blank}} \circ r_{\mathsf{Blank}}$ with $n_{\mathsf{Blank}} \circ r_{\mathsf{Blank}}$ being injective by composition. Secondly, suppose we have $x \in I_{\mathsf{Var}}$ and $n \in I_{\mathsf{Var}} + I_{\mathsf{Blank}} + \mathsf{Voc}(I)_{\mathsf{URI}} + \mathsf{Voc}(I)_{\mathsf{PLit}} + \mathsf{Voc}(I)_{\mathsf{TLit}}$ with $x \neq n$. Since r_{Var} is injective and only maps to variables, we have $r_{\mathsf{Var}}(x) \in R_{\mathsf{Var}}$, $r_{\mathsf{VN}}(n) \in R_{\mathsf{Var}} + R_{\mathsf{Blank}} + \mathsf{Voc}(R)_{\mathsf{URI}} + \mathsf{Voc}(R)_{\mathsf{PLit}} + \mathsf{Voc}(R)_{\mathsf{TLit}}$ and $r_{\mathsf{Var}}(x) \neq r_{\mathsf{VN}}(n)$. Because of $n \in \mathcal{O}_{\mathsf{RDF}}$ $n_{\mathsf{Var}}(r_{\mathsf{Var}}(x)) \neq o_{\mathsf{VN}}(r_{\mathsf{VN}}(n))$ and by commutativity we obtain $g_{\mathsf{VN}}(i_{\mathsf{Var}}(x)) \neq g_{\mathsf{VN}}(i_{\mathsf{VN}}(n))$. Since g_{VN} is injective this implies $i_{\mathsf{Var}}(x) \neq i_{\mathsf{VN}}(n)$. Hence, $i_{\mathsf{Var}}(x) = i_{\mathsf{VN}}(n)$ already implies $x = n$ and $i \in \mathcal{O}_{\mathsf{RDF}}$. □

Proposition 4.11 (MPOC-Pullback Characterisation in **RDFPat**)
The MPOC-pullback characterisation given in Definition 3.7 is satisfied for **RDFPat**.

Proof Sketch. Pushout complements for the same given situation only differ in their triple sets. For the minimal pushout complement all triples that are in the left-hand side but not in the interface have to be removed which implies that the triple set of the interface is the intersection of the triple set of the left-hand side and the triple-set of the MPOC and, hence, makes the interface a pullback. On the other hand, a pushout complement for which the interface is a pullback is already a minimal pushout complement since no further triples can be removed from the triple set without losing the pushout property. □

Proposition 4.12 (Pushout-Pullback Decompositions in **RDFPat**)
RDFPat has pushout-pullback decompositions as defined in Definition 3.8.

Proof Sketch. The pullback D as constructed above is a subgraph of G which contains exactly those elements for which the instances are also found in the subgraph D' of the codomain G'. If this subgraph is now an instantiation of a pushout interface I for the original codomain G' then this means by the pushout construction above that G' can be exactly reconstructed by disjointly and injectively adding the additional elements of L to the instantiation D' of I. But since G is an intermediate object in this instantiation it also obtained by disjointly adding the additional elements of L to the intermediate instantiation D of I and is, hence, a pushout of D and L under I. $\square$

Proposition 4.13 (Cube Properties in **RDFPat**)
The cube properties given in Definition 3.9 are satisfied in **RDFPat**.

Proof Sketch. Since all horizontal morphisms are monos we will without loss of generality assume them to be inclusions. In both properties I is then the intersection of J_1 and J_2. For the first property, D is additionally the intersection of D_1 and D_2 and the back faces being pushouts means that G can be obtained by either disjointly and injectively adding the additional elements of E w. r. t. J_1 to J_1's instance D_1 or by adding the additional elements of E w. r. t. J_2 to its instance D_2. But then the instances D_1 and D_2 must already contain the elements that do not stem from their intersection D in a compatible way and, hence, be pushouts of J_1 and D and J_2 and D over the intersection I of J_1 and J_2, respectively.
For the second property we already know that all vertical faces are pushouts. We have to show that D is a minimal complement of D_1 under I if and only if D_2 is a minimal complement of G under J_2.
First, we assume that D_2 is minimal, but D is not. Then there has to be a triple in J_1 which is not in I, but for which the corresponding instance is nevertheless in D. Since the instance is in D it has to be in all graphs in the bottom by the inclusions. Moreover, since the triple is in J_1 it has to be in E, but it cannot be in J_2 because otherwise it would also be in the intersection I. Hence, D_2 cannot be minimal since it contains an instance of a triple which is in E but not in J_2.
Second, we assume that D is minimal, but D_2 is not. Then, there has to be a triple in E but not in J_2 for which the corresponding instance is nevertheless in D_2. Since D_2 is a pushout of J_2 and D and the triple is not from J_2, the instance has to be in D. Since I is the intersection of J_1 and J_2 the triple cannot be in I. Hence, D is not minimal. $\square$

As an epi-$\mathcal{M}_{\mathrm{RDF}}$ factorisation we choose the image of the homomorphism as the most easily realisable variant. Note that epi-mono factorisations are not unique in **RDFPat** since we are free to include more triples into the factorisation object, while e still is an epimorphism.

Proposition 4.14 (Epi-$\mathcal{M}_{\mathrm{RDF}}$ Factorisations in **RDFPat**)
Given an RDF pattern homomorphism $m\colon L \to G$, an epi-$\mathcal{M}_{\mathrm{RDF}}$ factorisation of m can be obtained by the RDF pattern $m(L)$, the epimorphism $e\colon L \to m(L)$ and the

monomorphism $i\colon m(L) \to G$ with $i \in \mathcal{M}_{\mathrm{RDF}}$ constructed by

- $m(L)_{\mathrm{Blank}} := \{b \in G_{\mathrm{Blank}} \mid \exists l \in L_{\mathrm{Blank}}\colon m_{\mathrm{Blank}}(l) = b \text{ or } \exists l \in L_{\mathrm{Var}}\colon m_{\mathrm{Var}}(l) = b\}$,
- $m(L)_{\mathrm{Var}} := \{x \in G_{\mathrm{Var}} \mid \exists l \in L_{\mathrm{Var}}\colon m_{\mathrm{Var}}(l) = x\}$,
- $m(L)_{\mathrm{UVar}} := m(L)_{\mathrm{Var}} \cap G_{\mathrm{UVar}}$,
- $m(L)_{\mathrm{Triple}} := \{(s, p, o) \in G_{\mathrm{Triple}} \mid \exists (s', p', o') \in L_{\mathrm{Triple}}\colon m_{\mathrm{Triple}}(s', p', o') = (s, p, o)\}$
- $e_{\mathrm{Blank}}(b) := m_{\mathrm{Blank}}(b)$ for all $b \in L_{\mathrm{Blank}}$,
- $e_{\mathrm{Var}}(x) := m_{\mathrm{Var}}(x)$ for all $x \in L_{\mathrm{Var}}$,
- $i_{\mathrm{Blank}} := \mathrm{incl}_{m(L)_{\mathrm{Blank}}, G_{\mathrm{Blank}}}$ and
- $i_{\mathrm{Var}} := \mathrm{incl}_{m(L)_{\mathrm{Var}}, G_{\mathrm{Var}}}$.

Beweis. The homomorphism e is obviously surjective on blank nodes and variables since $m(L)$ is explicitly defined to only contain elements that are reached by m. On the other hand i is defined as an inclusion of $m(L)$ into G, which is injective, does not instantiate variables and does not map non-URI to URI variables. Hence, e is an epimorphism and $i \in \mathcal{M}_{\mathrm{RDF}}$. □

Now we have all constructions and results that are necessary for sequential composition which is summarised in the following corollary.

Corollary 4.3 (NAC Transformation Category with Sequential Composition for RDF Patterns)
$(\mathbf{RDFPat}, \mathcal{M}_{\mathrm{RDF}}, \mathcal{R}_{\mathrm{RDF}}, \mathcal{O}_{\mathrm{RDF}})$ is a NAC transformation category with sequential composition as given in Definition 3.10 and Definition 3.15.

Beweis. For sequential composition, pullbacks along $\mathcal{M}_{\mathrm{RDF}}$ preserving $\mathcal{M}_{\mathrm{RDF}}$ are shown in Proposition 4.9, the MPOC-pullback characterisation is proven in Proposition 4.11, pushout-pullback decomposition is given in Proposition 4.12 and the cube properties are shown in Proposition 4.13. For NACs, the facts that $\mathcal{M}_{\mathrm{RDF}} \subseteq \mathcal{O}_{\mathrm{RDF}}$ and $\mathcal{O}_{\mathrm{RDF}}$ is closed under composition are given in Proposition 4.3, pushouts and pullbacks along $\mathcal{M}_{\mathrm{RDF}}$ preserving $\mathcal{O}_{\mathrm{RDF}}$ are shown in Proposition 4.6 and Proposition 4.10, respectively, and epi-$\mathcal{M}_{\mathrm{RDF}}$ factorisations are given in Proposition 4.14. □

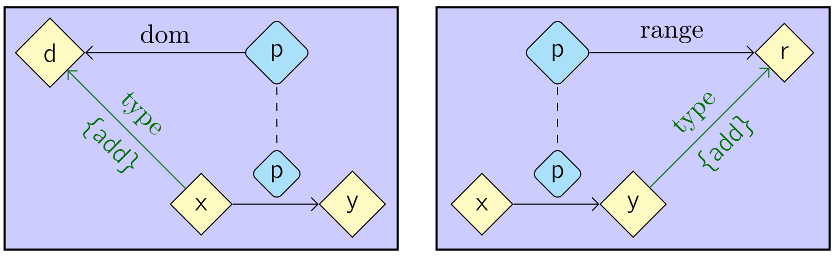
4.4. Inference Rules for RDF and RDF Schema

In this section, we will give transformation rules which implement inference rules for ρdf. These rules are algebraic transformation rule versions of a set of inference rules given in

[MPG07]. Since we confined ourselves to a ρdf variant without reflexivity in Section 2.3 we will also omit corresponding inference rules for reflexivity in this section.

The purpose of inference is that semantic ρdf entailment shall be characterised syntacitically in order to validate if a given RDF graph G ρdf entails another graph H. With the help of the rules given in the following this can be characterised by the fact that G ρdf entails H if and only if there is a transformation sequence using the following rules from G to a graph G' such that there is an RDF graph instantiation $i\colon H \to G'$.

The first rules, given in Figure 4.2, realise the inferences implied by dom and range triples.[2] Since ρdf interpretations require that the interpretations of dom, range and type are compatible in a reasonable way we can infer the types of subjects and objects of a predicate from triples defining their domain and range in the schema of the predicate.



(a) Domain (b) Range

Abbildung 4.2.: Typing rules for dom and range

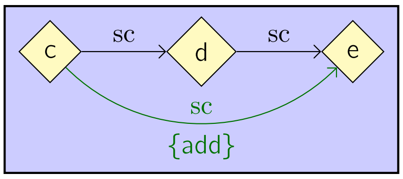
For the subclass predicate sc, the transitivity that is required by the definition of ρdf interpretations is syntactically realised by the rule in Figure 4.3a, while the inheritance of type properties is realised by the rule in Figure 4.3b.

For the subproperty predicate sp, transitivity is realised by the transformation rule in Figure 4.4a, while the inheritance of superproperties is achieved by the rule in Figure 4.4b.

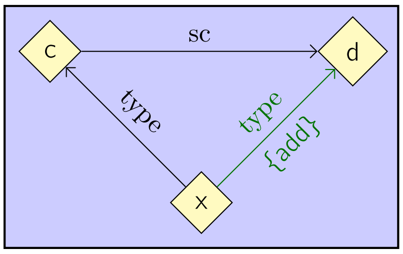
While the rules given so far are quite straightforward and are, in fact, already given in [Hay04], it was shown in [Mar06] and [MPG07] that these are not complete. The reason is that blank nodes (and, although not really sensible, also literals[3]) may appear in subproperty hierarchies, but the rules in Figure 4.2 and Figure 4.4 together may only infer the corresponding type triples if the superproperty is represented by a URI. This can be readily seen in our inference transformation rules since the rules in Figure 4.2 and

[2] Here and in the following, rules are depicted using a compact notation, where LHS, interface and RHS are given in one figure and the deleted parts only included in the LHS are tagged with "{del}", while the added parts only included in the RHS are tagged with "{add}". Moreover, we will use variable nodes with rounded corners and the same background color as URIs for URI variables.

[3] The problem that is sketched here does not arise for literals w. r. t. the original RDF specification since they are only allowed as objects there but for this problem they have to be objects of an sp triple and *subjects* of a dom or range triple. In our setting, where the restriction of subjects is omitted, both, blank nodes and literals, can be the culprit.

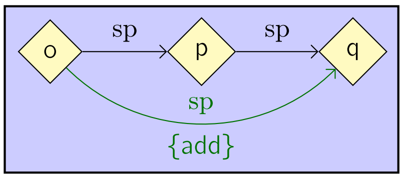


(a) Transitivity

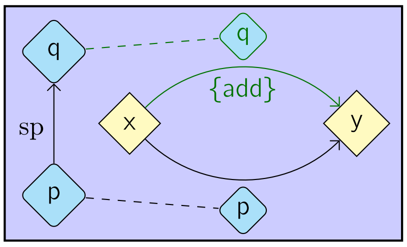


(b) Inheritance

Abbildung 4.3.: Inference rules for sc



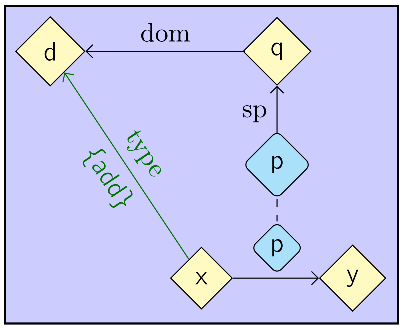
(a) Transitivity



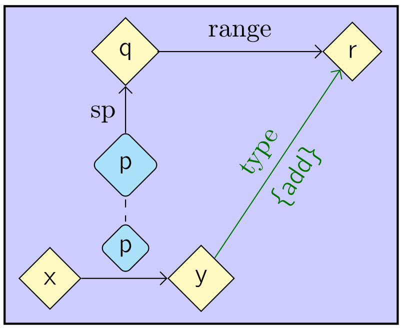
(b) Inheritance

Abbildung 4.4.: Inference rules for sp

Figure 4.4b use URI variables and are, hence, not applicable to blank nodes and literals. The rules in Figure 4.5 solve this issue by also allowing the implicit inference of type triples for domains and ranges of superproperties.



(a) Domain



(b) Range

Abbildung 4.5.: Implicit typing rules for dom and range

These rules constitute a sound and complete calculus for ρdf entailment which will be shown in the following theorems. The soundness is rather easy to show, since the inference rules exactly reflect the semantic conditions of ρdf interpretations.

Theorem 4.1 (Soundness of ρdf Inference Rules)
Given a transformation $G \overset{tr,m}{\Longrightarrow} H$ by one of the inference rules in Figure 4.2–4.5 and a ρdf interpretation I, $I \models G$ implies $I \models H$.

Beweis. We distinguish by the given transformation rules:

Domain typing: Because of the match m, the graph G has to contain the triples $(m_{\mathsf{Var}}(x), m_{\mathsf{Var}}(p), m_{\mathsf{Var}}(y))$ and $(m_{\mathsf{Var}}(p), \mathrm{dom}, m_{\mathsf{Var}}(d))$. Since $I \models G$ holds, Definition 2.9 implies that there is an assignment $asg\colon G_{\mathsf{Blank}} \to I_{\mathsf{Res}}$, such that we have $(\overline{asg}(m_{\mathsf{Var}}(x)), \overline{asg}(m_{\mathsf{Var}}(y))) \in \mathrm{pext}_I(\mathrm{uint}_I(m_{\mathsf{Var}}(p)))$ and $(\overline{asg}(m_{\mathsf{Var}}(p)), \overline{asg}(m_{\mathsf{Var}}(d))) \in \mathrm{pext}_I(\mathrm{uint}_I(\mathrm{dom}))$. The semantic condition for dom in Definition 2.13 implies that $(\overline{asg}(m_{\mathsf{Var}}(x)), \overline{asg}(m_{\mathsf{Var}}(d))) \in \mathrm{pext}_I(\mathrm{uint}_I(\mathrm{type}))$. Then I also satisfies the triple $(m_{\mathsf{Var}}(x), \mathrm{type}, m_{\mathsf{Var}}(d))$ and, since this is the only triple added by the transformation, we also have $I \models H$.

Range typing: Analogously to domain typing with the semantic condition for range in Definition 2.13

Subclass transitivity: Analogously to domain typing with the transitivity condition for sc in Definition 2.13

Subclass inheritance: Analogously to domain typing with the inheritance condition for sc in Definition 2.13

Subproperty transitivity: Analogously to domain typing with the transitivity condition for sp in Definition 2.13

Subproperty inheritance: G_{Triple} has to contain the triples $(m_{\mathsf{Var}}(p), \mathrm{sp}, m_{\mathsf{Var}}(q))$ and $(m_{\mathsf{Var}}(x), m_{\mathsf{Var}}(p), m_{\mathsf{Var}}(y))$. Then we also have $(\overline{asg}(m_{\mathsf{Var}}(p)), \overline{asg}(m_{\mathsf{Var}}(q))) \in \mathrm{pext}_I(\mathrm{uint}_I(\mathrm{sp}))$ and $(\overline{asg}(m_{\mathsf{Var}}(x)), \overline{asg}(m_{\mathsf{Var}}(y))) \in \mathrm{pext}_I(\mathrm{uint}_I(m_{\mathsf{Var}}(p)))$ by $I \models G$ and Definition 2.9. Moreover, we have $(\overline{asg}(m_{\mathsf{Var}}(x)), \overline{asg}(m_{\mathsf{Var}}(y))) \in \mathrm{pext}_I(\mathrm{uint}_I(m_{\mathsf{Var}}(q)))$ by $\mathrm{pext}_I(\mathrm{uint}_I(m_{\mathsf{Var}}(p))) \subseteq \mathrm{pext}_I(\mathrm{uint}_I(m_{\mathsf{Var}}(q)))$ whis is implied by the semantic condition for subproperty inheritance in Definition 2.13. Again, this corresponds to the only triple added by the rule and, hence, $I \models H$.

Implicit domain typing: G_{Triple} has to contain the triples $(m_{\mathsf{Var}}(x), m_{\mathsf{Var}}(p), m_{\mathsf{Var}}(y))$, $(m_{\mathsf{Var}}(p), \mathrm{sp}, m_{\mathsf{Var}}(q))$ and $(m_{\mathsf{Var}}(q), \mathrm{dom}, m_{\mathsf{Var}}(d))$. By $I \models G$ and Definition 2.9, we have $(\overline{asg}(m_{\mathsf{Var}}(x)), \overline{asg}(m_{\mathsf{Var}}(y))) \in \mathrm{pext}_I(\mathrm{uint}_I(m_{\mathsf{Var}}(p)))$, $(\overline{asg}(m_{\mathsf{Var}}(p)), \overline{asg}(m_{\mathsf{Var}}(q))) \in \mathrm{pext}_I(\mathrm{uint}_I(\mathrm{sp}))$ and, finally, $(\overline{asg}(m_{\mathsf{Var}}(p)), \overline{asg}(m_{\mathsf{Var}}(d))) \in \mathrm{pext}_I(\mathrm{uint}_I(\mathrm{dom}))$. From subproperty inheritance in Definition 2.13 we obtain $(\overline{asg}(m_{\mathsf{Var}}(x)), \overline{asg}(m_{\mathsf{Var}}(y))) \in \mathrm{pext}_I(\mathrm{uint}_I(m_{\mathsf{Var}}(q)))$ and the semantic condition for dom implies $(\overline{asg}(m_{\mathsf{Var}}(x)), \overline{asg}(m_{\mathsf{Var}}(d))) \in \mathrm{pext}_I(\mathrm{uint}_I(\mathrm{type}))$, which again lets us conclude $I \models H$.

Implicit range typing: Analogously to implicit domain typing with the semantic condition for range instead of dom □

The completeness of this calculus is not as easy to show as soundness. We only give a sketch here, since a complete proof would require some considerable extensions to the formal framework of Chapter 2.

Theorem 4.2 (Completeness of ρdf Inference Rules)
Given RDF graphs G and H with $G \Vdash_{\rho\mathrm{df}} H$, there is a transformation sequence $G \Rightarrow^* G'$ by the inference rules in Figure 4.2–4.5 and an instantiation $i\colon H \to G'$.

Proof Sketch. Similar to the "only if" part in the proof of Theorem 2.1, we construct a minimal (simple) interpretation I of G and afterwards close it under the semantic conditions for ρdf interpretations leading to an initial ρdf interpretation I' of G. On the other hand, we apply the inference rules of this section as long as possible leading to a graph G'. Since the inference rules exactly reflect the semantic conditions of ρdf interpretations, I' is the initial (simple) interpretation of G'. Moreover, $I' \models G$ implies $I' \models H$ and we can construct an instantiation $i\colon H \to G'$ analogously to Theorem 2.1. □

We have now seen that transformation rules can be used to formalise inference rules but a more thorough examination of the relationships between inference and transformations is outside the scope of this thesis. More specifically, it is not really sensible to infer all possible triples since such inferences significantly increase the storage space needed for RDF graphs. Therefore, a lot of RDF engines employ inference implicitly during querying of a graph instead of explicitly adding the inferred triples to the graph. The relationships of this kind of inference to our transformation approach are non-trivial since, e. g., the question has to be answered what the result of a transformation deleting an inferred triple is.

Nevertheless, the application of RDF graph transformations to RDF datastores without inference is not critical and can be already used before solving these further reasearch topics.

5. Application Scenarios

In this chapter we will show how transformation rules can be used in the two application scenarios described in Section 1.2. The first scenario, presented in Section 5.1, is a Semantic Web application for bibliographical metadata. While structured metadata are a classical use case for Semantic Web structures, the second scenario, treated in Section 5.2, is a novel application area for RDF. There, RDF graphs are used as an abstract syntax for a domain-specific modelling language (DSML).

5.1. A Semantic Web Metadata Application

The small application that is considered in this section manages two types of publications, articles and books, and their relations to authors. We will first give a schema for the data used by the application and grammar rules that describe the allowed structures in Section 5.1.1. In addition we also assume that metadata shall be imported from external sources, where a different schema is used which is presented in Section 5.1.2 together with transformation rules that integrate this foreign data into the application.

5.1.1. Schema and Grammar for Bibliographies

The schema for our example application is given in Figure 5.1[1] We have two types of publications, $\mathrm{bib{:}Book}$ and $\mathrm{bib{:}Article}$, with the common super class $\mathrm{bib{:}Publication}$. Publications can have a title given by the predicate $\mathrm{bib{:}title}$ and an author given by the predicate $\mathrm{bib{:}author}$. The range of $\mathrm{bib{:}author}$ is the class $\mathrm{bib{:}Person}$ which in turn can have a name given by the predicate $\mathrm{bib{:}name}$. Titles and names are supposed to be plain literal content which, however, is not represented in this schema.

Semantic Web applications often have more or less implicit constraints on the data that are not expressible using RDF Schema. For our example, we assume the following constraints:

- A person shall have exactly one name.
- A publication shall have exactly one title.
- A publication shall have at least one author.

We use the grammar in Figure 5.2 to express these constraints. Only graphs that can be built using the rules in the grammar are allowed.

[1] This figure was already presented in Section 2.3 and is only repeated here.

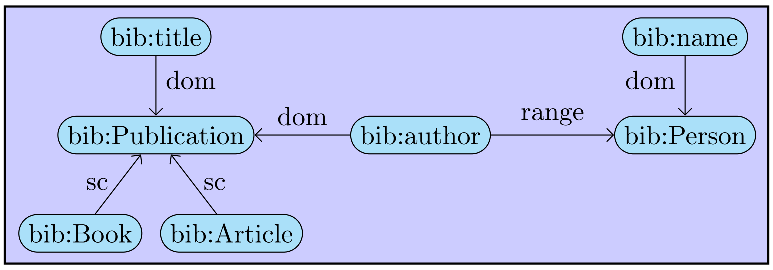


Abbildung 5.1.: Schema for bibliographies

The rule addPerson in Figure 5.2a is the only possibility to introduce new nodes with type bib:Person and new triples with predicate bib:name into the system. This ensures that every author has exactly one name. The NACs require that the name is not already used for another person and the person does not already have another name. Note that this, on the one hand, ensures to have a unique node for each person but, on the other hand, does not take into account that different persons may in reality have the same name.

The rules addBook in Figure 5.2b and addArticle in Figure 5.2c are used to add publications and their relations to authors to the system, where the node of type bib:Person has to exist in advance. The NACs again ensure that there is a one-to-one correspondence between titles and publications with the side effect that there cannot be a book and an article with the same title.

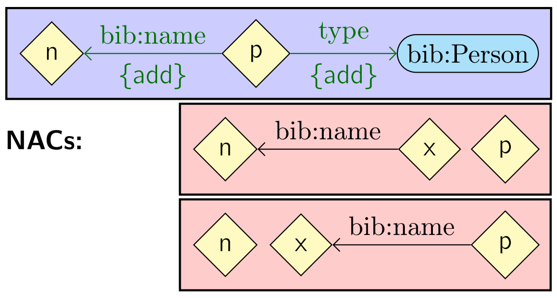
Since type triples to the types bib:Book and bib:Article can only be introduced by this rule which at the same time introduces a bib:author triple it is guaranteed that each publication has at least one author. Note that the same rules can be used to add additional authors to a publication since pushouts in RDF do not add triples again which are already contained in the host graph. If a publication with the same title already exists the NACs restrict the possible matches such that the corresponding variable (b or a) has to be mapped to this publication.

Observe that there is no rule involving bib:Publication from the schema. Hence, this class can be seen as an abstract class which has no direct instances that can be created by the grammar.

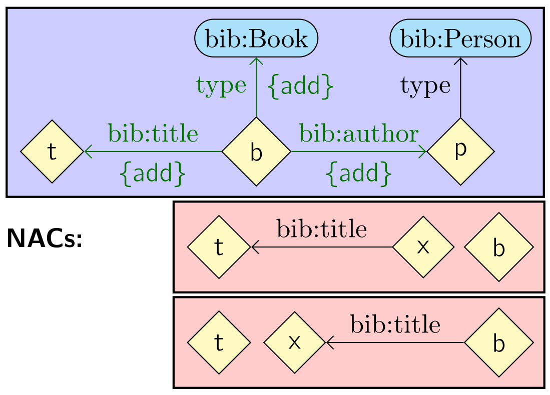
In summary, the informal constraints of our language are ensured on the one hand by the NACs guaranteeing a one-to-one correspondence between persons and their names and publications and their titles, on the other hand by the fact that publications can only be created together with their first author.

5.1.2. Integration of Schemas

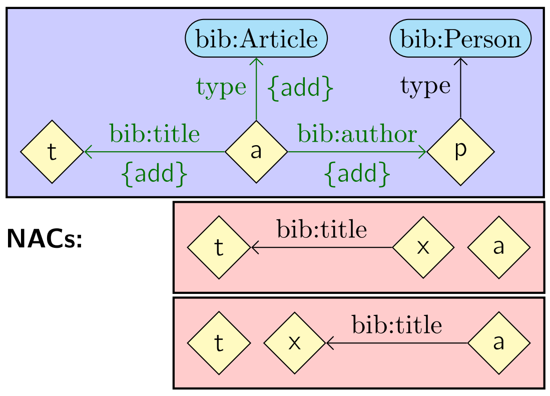
Another use case for graph transformations on RDF is the automatic integration or adaption between data w. r. t. different schemas. For example, Figure 5.3 shows a schema



(a) Rule addPerson



(b) Rule addBook



(c) Rule addArticle

Abbildung 5.2.: Grammar for bibliographies

for a record-based bibliography.

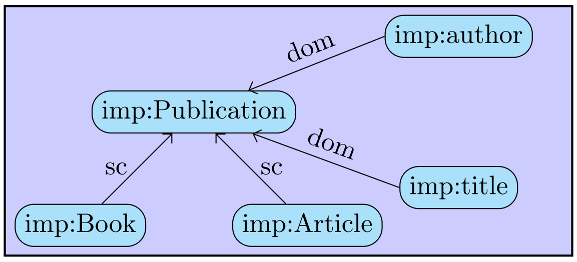


Abbildung 5.3.: Schema for record entries

Such a vocabulary can be used for straightforward parsing of flat bibliographies without cross-references into RDF data. For each record of the source a blank node with corresponding triples is created. Therefore, we introduce a simple grammar for record-based bibliographies in Figure 5.4.

The RDF graph in Figure 5.5[2] contains an imported publication created by this grammar together with a publication that was created by the grammar from the previous section. Since RDF graphs are supposed to represent whole data stores, such situations, where data w. r. t. different schemas and grammars are contained in the same graph, will arise quite often.

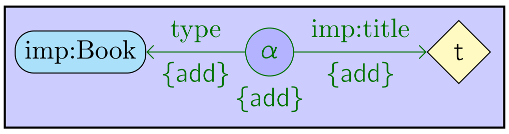
In order to make the imported data usable for our example application we have to integrate it into the schema used by the application. The rules in Figure 5.6 facilitate this. Firstly, the rules $\mathrm{intBook}$ in Figure 5.6a and $\mathrm{intArticle}$ in Figure 5.6b translate books and articles with an arbitrary number of authors into the native schema. The NACs (inherited from $\mathrm{addPerson}$ and $\mathrm{addBook}$) ensure that names and titles exactly correspond to persons and publications. If the book (for the second and further authors) or the author or both already have a corresponding URI we can still use the same rule since the NACs only forbid that another node with the same type and name exists. In this case some of the structures added by the rule are already contained in the graph and not added again.

Finally, the rules $\mathrm{delBook}$ in Figure 5.6c and $\mathrm{delArticle}$ in Figure 5.6d remove the blank node and the triples for type and title. Because of the dangling condition this rule can only be applied after all authors have been translated and deleted. If a publication does not have any authors from the beginning it can also be deleted by this rule which is intended since the native schema does not allow publications without authors and such publications can, thus, not be integrated.

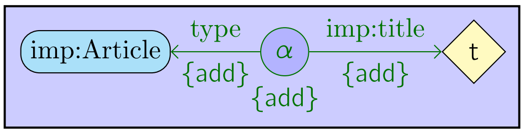
The fact that these integration rules are (for the parts regarding bib: vocabulary) composed of the grammar rules is responsible for ensuring that they only produce structures which are compatible with the grammar.

The example graph can be integrated by first applying $\mathrm{intBook}$ and then $\mathrm{delBook}$. The

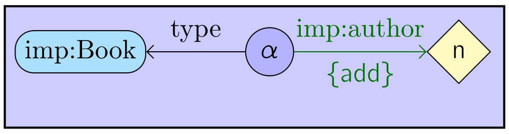
[2] This figure was already used as the first example of an RDF graph in Section 2.1.



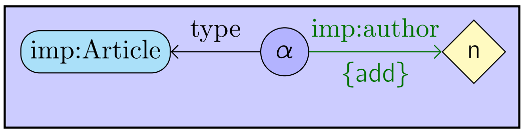
(a) Rule addImpBook



(b) Rule addImpArticle



(c) Rule addImpBookAuth



(d) Rule addImpArtAuth

Abbildung 5.4.: Grammar for record entries

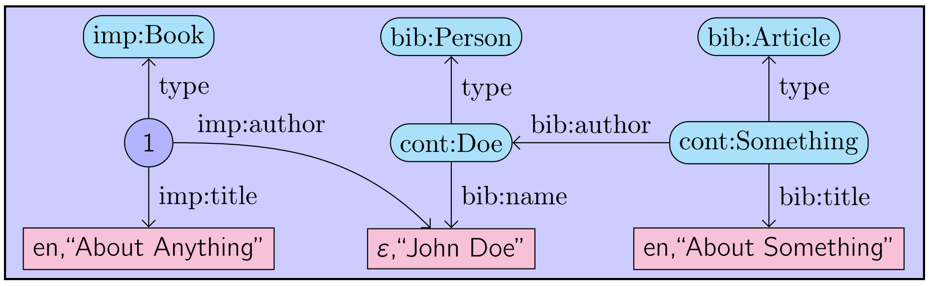
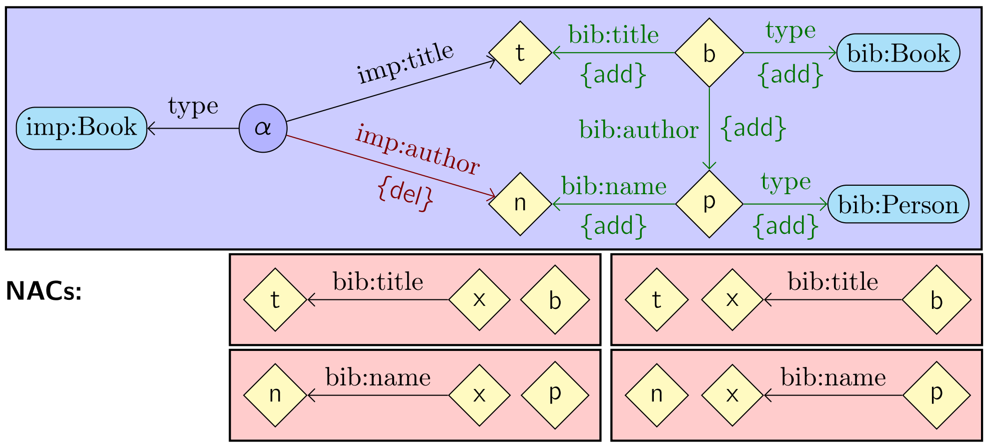
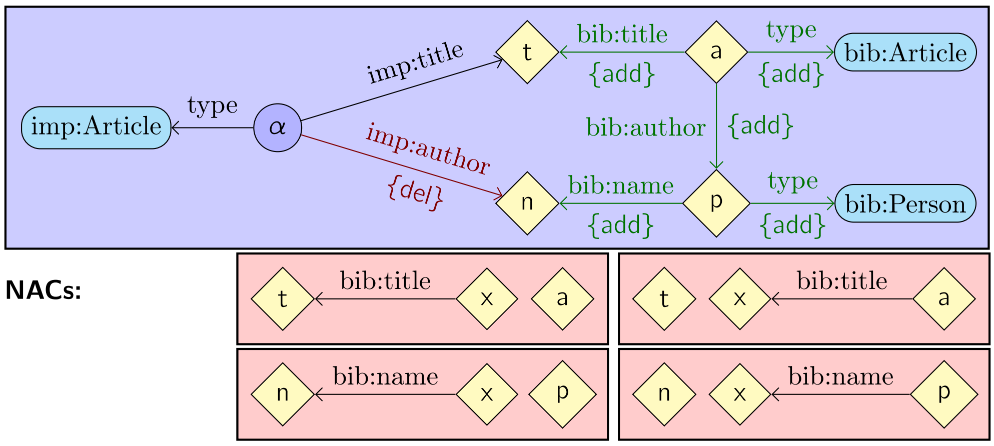


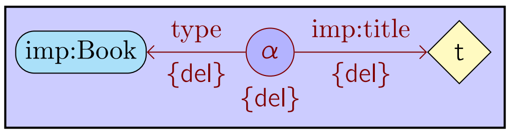
Abbildung 5.5.: RDF graph with imported and native metadata



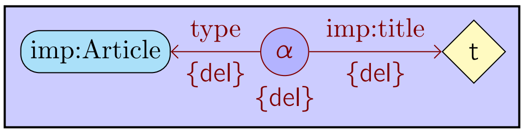
(a) Rule intBook



(b) Rule intArticle



(c) Rule delBook



(d) Rule delArticle

Abbildung 5.6.: Schema integration rules

result of this integration is given in Figure 5.7, where the new URI $\mathrm{cont{:}New}$ for the book was chosen by the match of $\mathrm{intBook}$ and the type and $\mathrm{bib{:}name}$ triples for the author were not added again, since they were already contained in the graph.

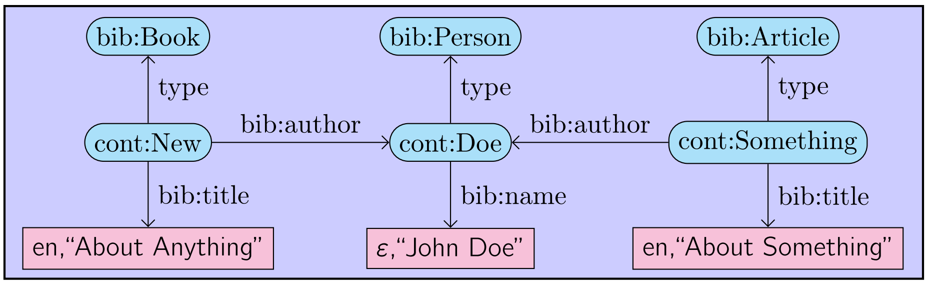


Abbildung 5.7.: Result of integration

5.2. Domain-Specific Modelling Languages

For the DSML application scenario we consider a simple language for IT landscapes which is intended to easily depict the network layout of an organisation. As an example, Figure 5.8 shows an IT landscape, where two local networks, named “Data Center 1” and “Data Center 2”, are both connected to a demilitarised zone (DMZ) for public servers via firewall protected connections. The DMZ itself is connected to the Internet via another firewall protected connection. There is a remote cluster for backups which also has a protected connection to the Internet. In order to reach this backup network, the data centers also have direct connections to the Internet that (as a deliberately introduced design flaw which will be repaired later) are not protected.

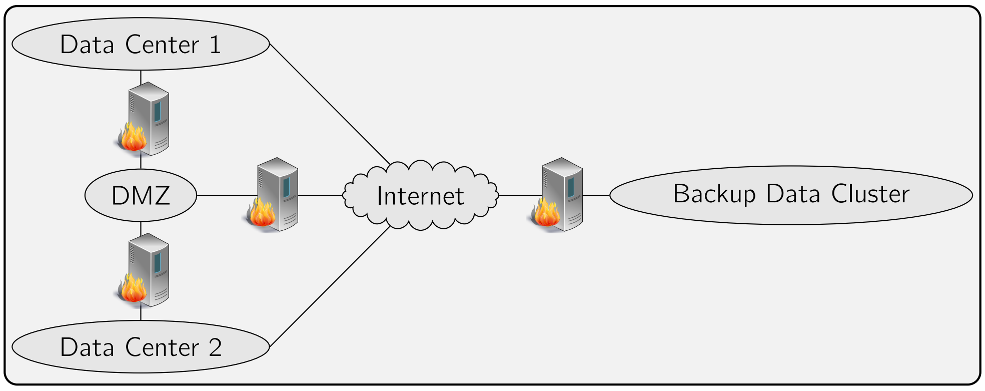


Abbildung 5.8.: Example IT landscape

We now want to give the abstract syntax for models in this language as RDF graphs. Since the whole model would be too large, only one connection is presented with concrete and abstract syntax in Figure 5.9, where one of the data centers is represented by a URI in the mod: namespace with triples connecting it to the type $\mathrm{dsml{:}LAN}$ and the inscription as a plain literal. Correspondingly the Internet entitity in the model is represented by a URI $\mathrm{mod{:}INet}$ of type $\mathrm{dsml{:}WAN}$ with the inscription “Internet” as a plain literal. Moreover, the connection itself is given by a blank node of type $\mathrm{dsml{:}Connection}$ with $\mathrm{dsml{:}connect}$ triples to both URIs.

5.2.1. Schema and Grammar for IT Landscapes

The schema for the example DSML is given in Figure 5.10. Connections are declared such that they can connect all instances of the abstract class $\mathrm{dsml{:}Connectable}$ which is the common superclass for applicances, such as the firewalls in the example, and networks. Since appliances are supposed to be identified solely by an icon depicting their type, only nets are declared to be the domain of the $\mathrm{dsml{:}name}$ predicate for attaching inscriptions.

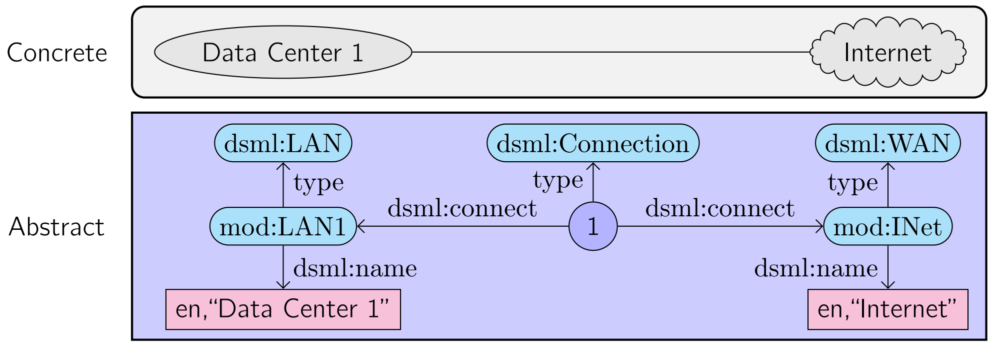


Abbildung 5.9.: Concrete and abstract syntax of a connection

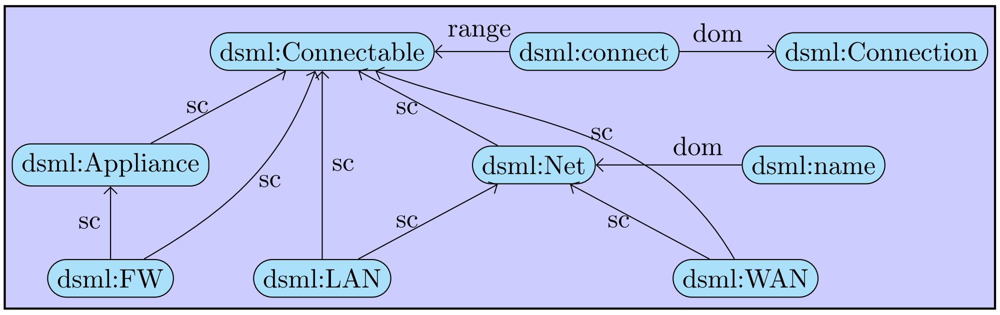
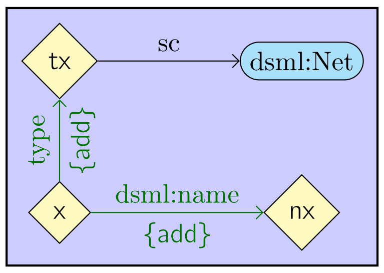


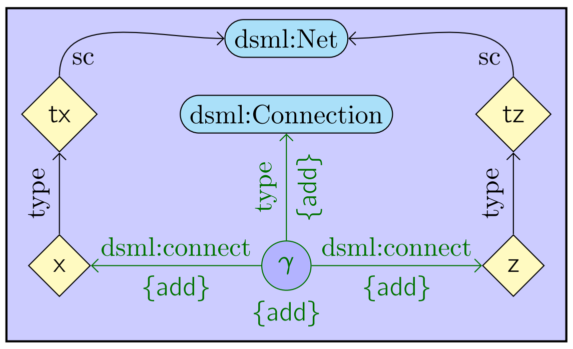
Abbildung 5.10.: Schema for IT landscape DSML

Again, the structural constraints assumed for the language are expressed by a grammar which is given in Figure 5.11. The grammar allows to add local networks as well as WANs by the same rule addnet which uses a variable for the class and just restricts it to be a subclass of $\mathrm{dsml{:}Net}$. The rule also ensures that a name has to be given for each instance of $\mathrm{dsml{:}Net}$. The rule $\mathrm{addConnection}$ is meant to introduce connections between networks, where a blank node is used to represent the connection. Since the blank node is created by the rule itself it is ensured that each connection connects exactly two networks. The most complex rule $\mathrm{addFirewall}$ creates a connection protected by a firewall, where the firewall itself and the connections on both sides are added at the same time. NACs are used to ensure that a firewall is only introduced between distinct nets, i. e., the variables x and z are not assigned to the same net, that the node that is assigned to the variable y is not already used in any connection and that it does not have a type already.

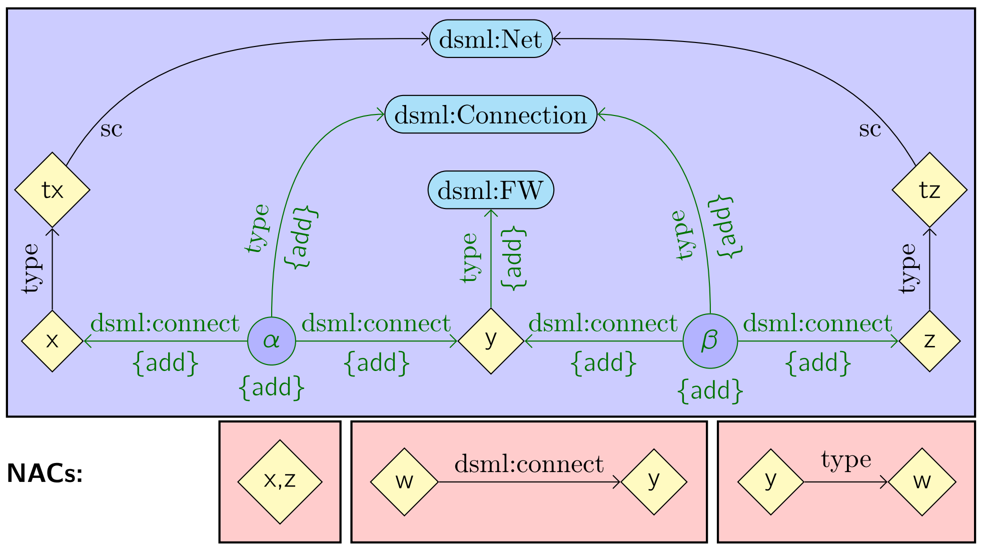
In order to derive some more complex editing rules in the next section, we also need rules that allow us to delete structures, which are shown in Figure 5.12. While it would be useful to be able to automatically derive such deleting rules from a constructive grammar, this task is non-trivial and no formal support is available up to now. A small example of the problems arising during the derivation of deletion rules can be already seen in this simple grammar. While the main body of the rules is just reverted to delete the structures that were added by the constructive grammar, the treatment of NACs needs special attention. Since nodes of type $\mathrm{dsml{:}Net}$ are required in the left-hand side of the rules $\mathrm{addConnection}$ and $\mathrm{addFirewall}$ we cannot delete these types at will without changing the language. Therefore, the rule delNet uses a NAC that ensures that the deleted network is not used in any connection. On the other hand, the rule $\mathrm{delFirewall}$ does, in contrast to its constructive counterpart, not need any NACs since the NACs only forbid to add the structure if certain conditions are not met, while the deletion of a firewall cannot violate these structural constraints.



(a) Rule addNet

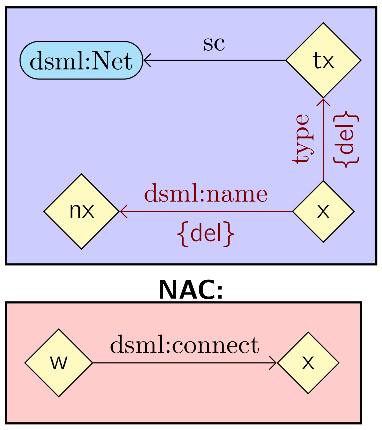


(b) Rule addConnection

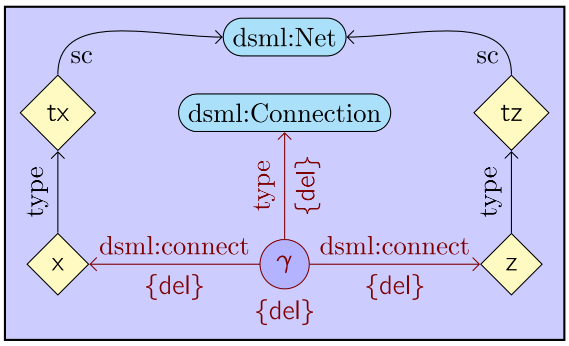


(c) Rule addFirewall

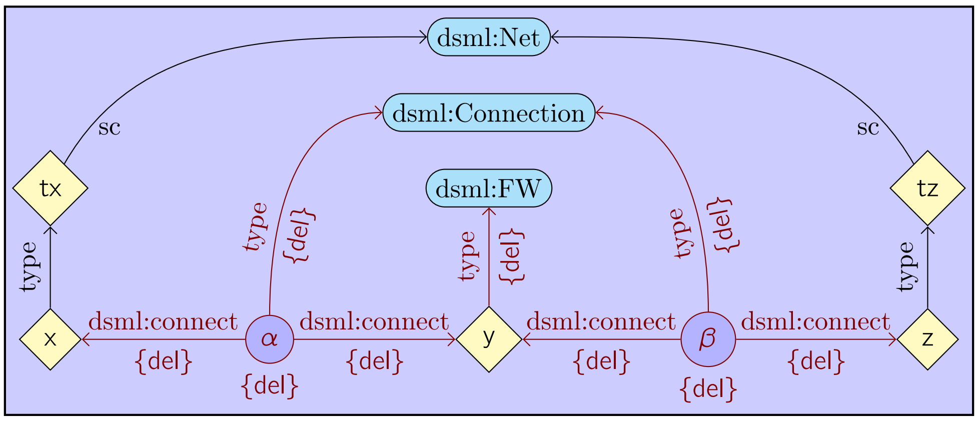
Abbildung 5.11.: Grammar rules for IT landscape DSML



(a) Rule delNet



(b) Rule delConnection



(c) Rule delFirewall

Abbildung 5.12.: Deleting rules for IT landscape DSML

5.2.2. Modification Rules for IT Landscapes

In this section we will give examples of more complex modification rules that are composed of the grammar rules from the previous section. Such rules are intended for end users to achieve elaborate transformations in one step without the risk to leave the language defined by the grammar (and deleting rules). This is ensured by Theorem 3.3 which ensures that an application of a composed rule can be substituted by applications of its constituents.

The first example rule, editNet in Figure 5.13, combines the rules delNet and addNet to achieve the modification of an existing net in a landscape model. Note that during rule composition the NACs are inherited from the component rules and, therefore, this rule is only applicable to unconnected networks.

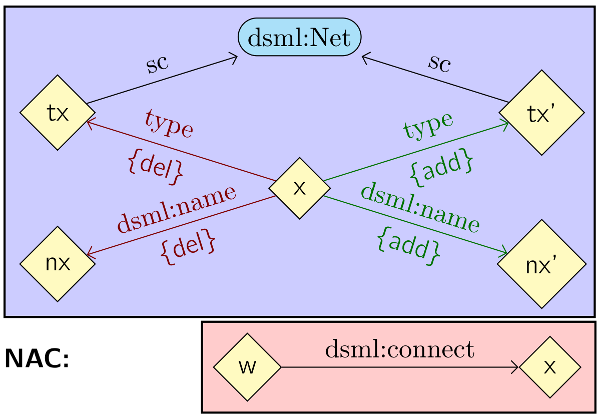
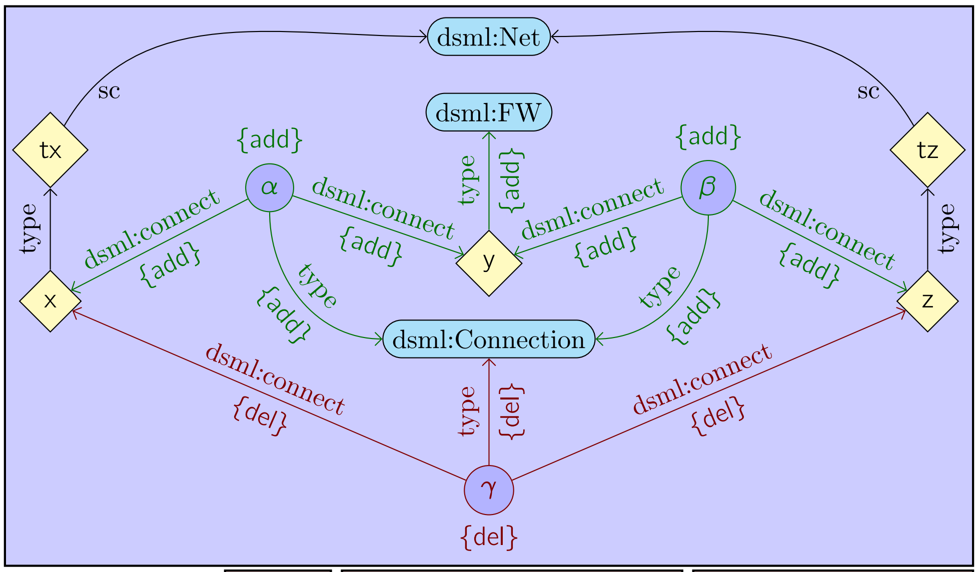


Abbildung 5.13.: Rule editNet as composition of delNet and addNet

The second modification rule, protect in Figure 5.14, combines the rules delConnection and addFirewall in order to protect an existing connection in a landscape by a firewall. Again, the NACs are inherited from the component rule addFirewall.

The protect rule can be used to repair the design flaw in the example model from Figure 5.9, where the unprotected connection is replaced by a connection with a firewall.



NACs:

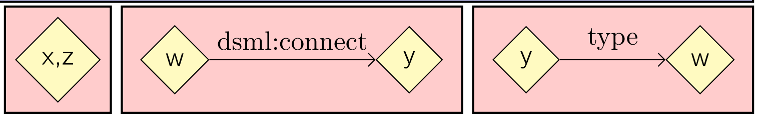


Abbildung 5.14.: Rule protect as composition of delConnection and addFirewall

5.2.3. Evolution of Domain-Specific Modelling Languages

In this last section we will shortly sketch how the flexibility provided by RDF and graph transformation can be used to support the evolution of DSMLs. We assume that local networks shall be refined by introducing a distinction between public and restricted networks. This can be achieved by adding the schema elements in Figure 5.15 to the schema. Note that the evolved structures are just added to the schema but the original class dsml:LAN is not removed. This is sensible for the continued support of legacy models since it is not feasible to migrate all models in a large organisation at the same time.

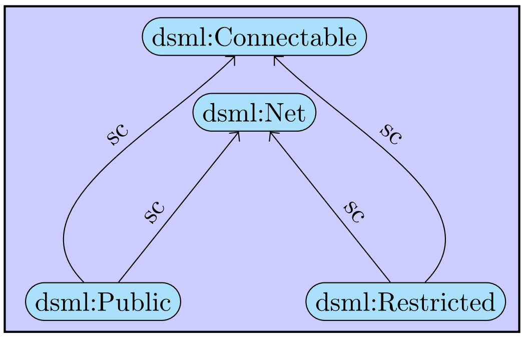


Abbildung 5.15.: Additional schema elements for evolved DSML

In order to adapt such legacy models to the evolved language graph transformation rules can again be used. Figure 5.16 shows a rather simple one which just adapts all legacy local area networks to become public networks w. r. t. the evolved schema. Such adaption rules could be applied manually when the need arises or automatically to whole models or even repositories.

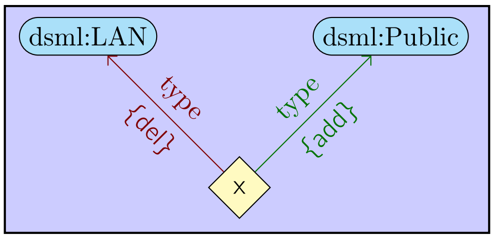


Abbildung 5.16.: Adaption rule for evolved DSML

6. Conclusion and Future Perspectives

In this thesis, a formally founded transformation concept for RDF has been developed and applied in two application scenarios. In Section 6.1, we will shortly summarise how the requirements of the scenarios are solved by graph transformations. Then, in Section 6.2, the main theoretical contributions and perspectives for future work are recapitulated.

6.1. Solution for Application Scenarios

In Chapter 5 it has been shown that graph transformations provide a solution for the problem statement in Section 1.2. Graph transformations have been presented as a versatile tool to deal with all kinds of modifications of RDF data.

The requirements w. r. t. the example metadata application habe been met by providing a grammar that defines all allowed structures for the application and then showing how metadata that are imported using another schema can be integrated into the language defined by this grammar.

With respect to the domain-specific language scenario, the abstract syntax has also been defined by a grammar. Then, it has been shown that complex editing operations on models in the language can be built from the grammar rules. Moreover, a small example for the evolution of the domain-specific modelling language has lead to an adaption rule that allows to migrate models in the original language to models in the evolved language.

In future work, the formal transformation concept should be implemented in an RDF graph transformation engine which would be useful for both application scenarios. Especially for the domain-specific language scenario, an extension of the example to a case study with a whole language family would be useful to show how graph transformations can be used to integrate models for different aspects of a system.

6.2. Theoretical Contributions

A formalisation of RDF in category theory has been given in Chapter 2, where it is already shown that semantic entailment corresponds to morphisms on the syntax. Since this type of result is the basis for category theoretical specification formalism, represented by specifications frames (cf. [EG94]) and institutions (cf. [GB92]), it seems to be a valuable line of future research to use these to give a new structured overview for the abstract syntax and semantics of RDF.

For graph transformation theory, a new abstract transformation framework, built around the notion of minimal pushout complements, has been developed in Chapter 3 and instantiated to RDF in Chapter 4. This framework is not only applicable to RDF structures but may also be valuable for other kinds of graphical structures that do not fit into the existing frameworks because of non-unique pushout complements.

In the framework, a notion of sequential composition of transformation rules is given for which analysis and synthesis results habe been proven. These results already allow the definition of complex transformation rules by recording and sequentially composing an example transformation sequence which gives a formal foundation for a macro recording feature in a possible implementation. Moreover, the analysis result for sequentially composed rules proves that these rules do not enhance the class of reachable structures and, thus, the exclusive application of rules that are composed from the rules of a grammar ensures that the modifications stay within the language of the grammar.

A fairly complete collection of compositionality results for initial pushouts and minimal pushout complements has been proven in this thesis which should make it possible to also prove more of the theoretical results available in other categorical transformation frameworks, especially adhesive HLR categories (cf. [EEPT06]). More specifically, independence analysis for sequential and parallel rule applications would be a valuable addition to the theory since it would allow, on the one hand, to implement possibilities for the navigation in modification histories, making it possible to undo and redo independent modifications irrespective of their temporal order, on the other hand, to reconcile concurrent modifications automatically if they are independent.

Altogether, while giving a complete and self-contained solution for the goals formulated in the problem statement in Section 1.2, the concepts and theoretical results developed in this thesis also provide a sound basis for future work w. r. t. more ambitious goals.

A. Category Theory

In this appendix, we will summarise the definitions and results of category theory that are used throughout the thesis. For an introduction to category theory, see, e. g., [AHS90].

A.1. Categories and Functors

The notion of category is used to give an abstract container for various kinds of mathematical structures, where the internals of objects are not considered but rather their external relationships are represented by morphisms.

Definition A.1 (Category)
A *category* $\mathbf{C}$ consists of

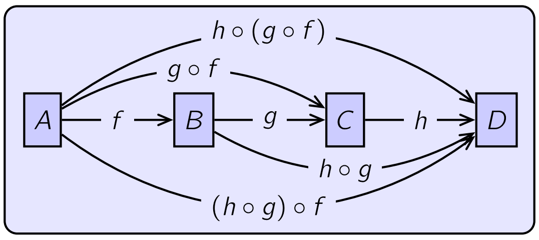
- a class $|\mathbf{C}|$ of *objects*,
- a set $\mathbf{C}(A, B)$ of *morphisms* for each pair of objects $A, B \in |\mathbf{C}|$,
- an *identity* $\mathrm{id}_A \in \mathbf{C}(A, A)$ for each object $A \in |\mathbf{C}|$ and
- a *composition* $g \circ f \in \mathbf{C}(A, C)$ for all objects $A, B, C \in |\mathbf{C}|$ and morphisms $f \in \mathbf{C}(A, B)$ and $g \in \mathbf{C}(B, C)$

such that

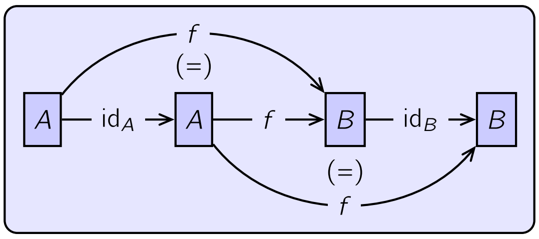
- the compositions are *associative*, i. e., $h \circ (g \circ f) = (h \circ g) \circ f$ for all objects $A, B, C, D \in |\mathbf{C}|$ and morphisms $f \in \mathbf{C}(A, B)$, $g \in \mathbf{C}(B, C)$ and $h \in \mathbf{C}(C, D)$ (cf. Figure A.1a), and
- the identities are *cancellable*, i. e., $f \circ \mathrm{id}_A = f = \mathrm{id}_B \circ f$ for all objects $A, B \in |\mathbf{C}|$ and morphisms $f \in \mathbf{C}(A, B)$ (cf. Figure A.1b).

A morphism $f \in \mathbf{C}(A, B)$ is also denoted as "$f \colon A \to B$ in $\mathbf{C}$". The explicit mentioning of the category is often omitted if it is understood from the context.

The most prominent example of a category is the category **Set** of sets and functions. While other, more general categories of sets with partial functions or general relations as morphism can also be defined we will confine ourselves to total functions (and inclusions as a special case in the following proposition) since these are the only kind of relations needed in this thesis.


$$\begin{tikzcd} A \arrow[r, "f"] \arrow[rr, bend left, "g \circ f"] \arrow[rrr, bend left=40, "h \circ (g \circ f)"] \arrow[rrr, bend right=40, "(h \circ g) \circ f"'] & B \arrow[r, "g"] \arrow[rr, bend right, "h \circ g"'] & C \arrow[r, "h"] & D \end{tikzcd}$$

(a) Associativity of compositions


$$\begin{tikzcd} A \arrow[r, "\mathrm{id}_A"] \arrow[rr, bend left, "f"] \arrow[rr, phantom, bend left=20, "(=)"] & A \arrow[r, "f"] \arrow[rr, bend right, "f"'] \arrow[rr, phantom, bend right=20, "(=)"'] & B \arrow[r, "\mathrm{id}_B"] & B \end{tikzcd}$$

(b) Cancellability of identities

Abbildung A.1.: Axioms for categories

Proposition A.1 (Category of Sets and Functions)
The category **Set** of sets and functions consists of

objects being all possible sets A,

morphisms being (total) functions $f\colon A \to B$ which map an element $a \in A$ to an element $f(a) \in B$,

compositions given by $g \circ f(a) := g(f(a))$ for each $a \in A$ and

identities given by $\mathrm{id}_A(a) := a$ for each $a \in A$.

Beweis.

Associativity: For all $a \in A$ we have $h \circ (g \circ f)(a) = h(g \circ f(a)) = h(g(f(a))) = h \circ g(f(a)) = (h \circ g) \circ f(a)$.

Cancellability: For all $a \in A$ we have $f \circ \mathrm{id}_A(a) = f(\mathrm{id}_A(a)) = f(a) = \mathrm{id}_B(f(a)) = \mathrm{id}_B \circ f(a)$. □

Another interesting category for the same class of all sets as objects is the category **SetIncl** of sets and inclusions, where we have a unique morphism between two sets if and only if the first is a subset of the second.

Proposition A.2 (Category of Sets and Inclusions)
The category **SetIncl** of sets and inclusions consists of

objects being all possible sets A,

morphisms $A \subseteq B$ if and only if for all $a \in A$ we also have $a \in B$,

compositions $B \subseteq C \circ A \subseteq B = A \subseteq C$ because of transitivity and

identities $A \subseteq A$ because of reflexivity.

Beweis. Since morphisms are unique associativity and cancellability are obviously satisfied. □

As already alluded to above, category theory does not refer to the internals of objects or morphisms. Therefore notions like injectivity, surjectivity and bijectivity, which are given by the effect on internal elements of the objects, are replaced by notions referring to the external properties of morphisms.[1] The most important notion is that of an isomorphism. Since isomorphisms are cancellable to identities in both directions, isomorphic objects are absolutely equivalent from a categorical point of view, where only the external relations by morphisms and not the internal structure of objects are relevant.

Definition A.2 (Special Morphisms)
Given a category **C**, the following types of special morphisms are defined:

Isomorphism: A morphism $i\colon A \to B$ is an *isomorphism*, or *iso* for short, if there is a morphism $j\colon B \to A$ such that $j \circ i = \mathrm{id}_A$ and $i \circ j = \mathrm{id}_B$ (cf. Figure A.2a). In this case A and B are also called *isomorphic*, written as $A \cong B$.

Monomorphism: A morphism $m\colon B \to C$ is a *monomorphism*, or *mono* for short, if for all morphisms $l_1, l_2\colon A \to B$ the equality $m \circ l_1 = m \circ l_2$ implies the equality $l_1 = l_2$ (cf. Figure A.2b).

Epimorphism: A morphism $e\colon A \to B$ is an *epimorphism*, or *epi* for short, if for all morphisms $f_1, f_2\colon B \to C$ the equality $f_1 \circ e = f_2 \circ e$ implies the equality $f_1 = f_2$ (cf. Figure A.2c).

We now characterise these special morphism notions in our two example categories, where the case of **SetIncl** is rather simple since there is at most one morphism between two objects and in the case of **Set** we get exactly the bijective, injective and surjective functions. For other, more complex categories this may, however, not be the case. Note that the isomorphisms are different in both categories. While in **Set** all sets with the same cardinality, i. e., bijective funtions between them, are isomorphic, in **SetIncl** only absolutely identical sets are isomorphic.

[1] We omit the notions of sections and retractions, sometimes called split monos and split epis, for the sake of brevity and because they are not needed in this thesis. They are defined as “half isomorphisms”, where there is an inverse such that only one of the compositions is required to yield the identity. In Set they coincide with the monos and epis to be injective and surjective functions, respectively, while in SetIncl only the identities are sections and retractions.

$$\begin{tikzcd}
B \arrow[r, "j"] \arrow[rr, bend left, "\mathrm{id}_B\ (=)"] & A \arrow[r, "i"] \arrow[rr, bend right, "(=)\ \mathrm{id}_A"'] & B \arrow[r, "j"] & A
\end{tikzcd}$$

(a) Isomorphism

$$\begin{tikzcd}
A \arrow[r, shift left, "l_1"] \arrow[r, shift right, "l_2"'] \arrow[rr, bend left, "m \circ l_1"] \arrow[rr, bend right, "m \circ l_2"'] & B \arrow[r, "m"] & C
\end{tikzcd}$$

(b) Monomorphism

$$\begin{tikzcd}
A \arrow[r, "e"] \arrow[rr, bend left, "f_1 \circ e"] \arrow[rr, bend right, "f_2 \circ e"'] & B \arrow[r, shift left, "f_1"] \arrow[r, shift right, "f_2"'] & C
\end{tikzcd}$$

(c) Epimorphism

Abbildung A.2.: Special morphisms

Proposition A.3 (Special Morphisms in **Set** and **SetIncl**)
For the category **Set** we have the following characterisation of special morphisms:

Mono: A morphism $m\colon A \to B$ is mono if and only if it is injective, i. e., $f(x) = f(y)$ implies $x = y$ for all $x, y \in A$.

Epi: A morphism $e\colon A \to B$ is epi if and only if it is surjective, i. e., there exists an $x \in B$ with $f(x) = y$ for all $y \in B$.

Iso: A morphism $i\colon A \to B$ is iso if and only if it is bijective, i. e., injective and surjective.

For the category **SetIncl**, on the other hand, the following holds:

Mono and Epi: All morphisms $A \subseteq B$ are monos and epis.

Iso: Only the identities $A \subseteq A$ are isos.

Beweis.

Mono in Set: If: If for all $x \in A$ we have that $m \circ l_1(x) = m \circ l_2(x)$ then injectivity implies $l_1(x) = l_2(x)$ which is the required property for a mono. **Only if:** Suppose a mono which is not injective. Then there have to be $a \neq a'$ with $m(a) = m(a')$. But then there are also functions $l_1, l_2\colon \{*\} \to B$ with $l_1(*) = a$ and $l_2(*) = a'$ satisfying $m \circ l_1(*) = m(a) = m(a') = m \circ l_2(*)$ but not $l_1(*) = l_2(*)$. Hence, m is not a mono.

Epi in Set: If: If for all $x \in A$ we have that $f_1 \circ e(x) = f_2 \circ e(x)$ then surjectivity also implies that for all $y \in B$ we have $f_1(y) = f_2(y)$ which is the required property for an epi. **Only if:** Suppose an epi which is not surjective. Then there has to be $b \in B$ for which no $x \in A$ with $e(x) = b$ exists. But then we can define two functions $f_1, f_2 \colon B \to B \cup \{1, 2\}$ with $f_1(y) = f_2(y) = y$ for all $y \neq b$, $f_1(b) = 1$ and $f_2(b) = 2$ satisfying $f_1 \circ e(x) = e(x) = f_2 \circ e(x)$ for all $x \in A$ but not $f_1(y) = f_2(y)$ for all $y \in B$ (since $f_1(b) \neq f_2(b)$). Hence, e is not an epi.

Iso in Set: If: Define j by $j(b) = a$ if and only if $i(a) = b$ which is functional because of injectivity and total because of surjectivity. Then $j \circ i(a) = a$ for all $a \in A$ and $i \circ j(b) = b$ for all $b \in B$. Hence, i is an iso. **Only if:** Suppose an iso which is not bijective. Then it is either not injective or not surjective. If it is not injective for $a \neq a'$ with $i(a) = i(a')$ then $j(i(a)) = j(i(a'))$ but $\mathrm{id}_A(a) = a \neq a' = \mathrm{id}_A(a')$. Hence, i is not an iso. If it is not surjective for $b \in B$ for which there is no $a \in A$ with $i(a) = b$ then we cannot find $j(b)$ with $i(j(b)) = b = \mathrm{id}_B(b)$. Hence, i is also not an iso.

Mono and Epi in SetIncl: Since the defining property of monos and epis only requires the equality of certain morphisms and morphisms are unique in **SetIncl** these properties are trivially satisfied for all existing morphisms.

Iso in SetIncl: Since an iso is a morphism with an inverse and $A \subseteq B$ and $B \subseteq A$ implies that $A = B$ we have only the identities as isos in **SetIncl**. □

A generalistion of epimorphism from single morphisms to pairs of morphisms is needed at various points in the thesis. It represents a categorical abstraction of joint surjectiveness, i. e., everything in the codomain is in the image of at least one of the two morphisms.

Definition A.3 (Jointly Epimorphic Morphisms)
Given a category **C**, two morphisms $f : A \to C$ and $g : B \to C$ are *jointly epimorphic*, or *jointly epi* for short, if for all morphisms $h_1, h_2 : C \to D$ the equalities $h_1 \circ f = h_2 \circ f$ and $h_1 \circ g = h_2 \circ g$ together imply the equality $h_1 = h_2$ (cf. Figure A.3).

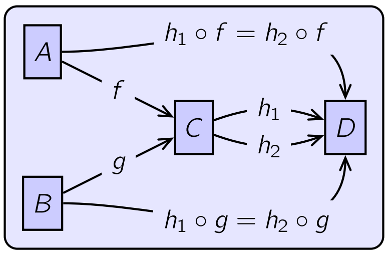


Abbildung A.3.: Jointly epimorphic morphisms

In order to relate different categories the notion of a functor is introduced which can be seen as a morphism between categories. It consists of a mapping of objects and a mapping of morphisms such that the relevant structure of a category, i. e., the identities and the compositions are preserved.

Definition A.4 (Functor)
Given two categories **C** and **D**, a *functor* $F\colon \mathbf{C} \to \mathbf{D}$ is given by

- an object $F(A) \in |\mathbf{D}|$ for each object $A \in |\mathbf{C}|$ and
- a morphism $F(f) \in \mathbf{D}(F(A), F(B))$ for all objects $A, B \in |\mathbf{C}|$ and morphisms $f \in \mathbf{C}(A, B)$ (cf. Figure A.4),

such that

- the identities are preserved, i. e., $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$ for each object $A \in |\mathbf{C}|$, and
- the compositions are preserved, i. e., $F(g \circ f) = F(g) \circ F(f)$ for all objects $A, B, C \in |\mathbf{C}|$ and morphisms $f \in \mathbf{C}(A, B)$ and $g \in \mathbf{C}(B, C)$.

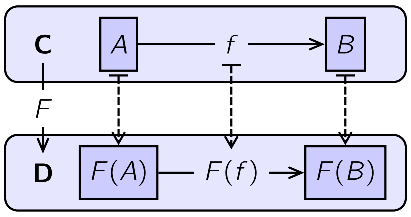


Abbildung A.4.: Functor

As a first, simple example of a functor we consider the relation between the categories **SetIncl** and **Set**. Obviously, inclusions are somehow a special kind of functions and the functor just has to define this „somehow“, i. e., show how we construct the inclusion function for all sets with $A \subseteq B$.

Proposition A.4 (Inclusion Functor $\mathrm{Incl}\colon \mathbf{SetIncl} \to \mathbf{Set}$)
The functor $\mathrm{Incl}\colon \mathbf{SetIncl} \to \mathbf{Set}$ is defined by $\mathrm{Incl}(A) := A$ for all sets $A \in |\mathbf{SetIncl}| = |\mathbf{Set}|$ and $\mathrm{Incl}(A \subseteq B) := \mathrm{incl}_{A,B}$ for all morphisms $A \subseteq B$ in **SetIncl**, where $\mathrm{incl}_{A,B}\colon A \to B$ is a function defined by $\mathrm{incl}_{A,B}(a) := a$ for all $a \in A$.

Beweis. The inclusion functions $\mathrm{incl}_{A,B}$ used as functor images for the morphisms $A \subseteq B$ of **SetIncl** are well-defined since for all $a \in A$ we have by definition of **SetIncl** also $a \in B$. The functor preserves identities because $\mathrm{incl}_{A,A}(a) = a = \mathrm{id}_A(a)$ for all $a \in A$ and each set $A \in |\mathrm{SetIncl}|$ and it also preserves compositions since $\mathrm{incl}_{A,C}(a) = a = \mathrm{incl}_{B,C}(a) = \mathrm{incl}_{B,C}(\mathrm{incl}_{A,B}(a))$ for all $a \in A$. ☐

A more complex example of an endofunctor, i. e., a functor from a category into itself, is the powerset functor which assigns the set of all its subsets to a given set. For functions we will use the extension of a function to subsets of its domain as the functor image.

Proposition A.5 (Powerset Functor)
The functor $\mathcal{P}\colon \mathbf{Set} \to \mathbf{Set}$ is defined by $\mathcal{P}(A) := \{S \mid S \subseteq A\}$ for all sets $A \in |\mathbf{Set}|$ and $\mathcal{P}(f)(S) := \{f(a) \mid a \in S\} \subseteq B$ for all morphisms $f\colon A \to B$ in **Set** and $S \in \mathcal{P}(A)$.

Beweis. The functor preserves identities because for all $A \in |\mathbf{Set}|$ and $S \in \mathcal{P}(A)$, i. e., $S \subseteq A$, we have $\mathcal{P}(\mathrm{id}_A)(S) = \{\mathrm{id}_A(a) \mid a \in S\} = S = \mathrm{id}_{\mathcal{P}(A)}(S)$ and it also preserves compositions since $\mathcal{P}(g \circ f)(S) = \{g(f(a)) \mid a \in S\} = \mathcal{P}(g)(\{f(a) \mid a \in S\}) = \mathcal{P}(g)(\mathcal{P}(f)(S))$ for all $f\colon A \to B$ and $g\colon B \to C$ and all $S \subseteq A$. □

A.2. Limits and Colimits

Definitions by universal properties are very important in category theory. Such properties are given by the requirement that an object is the smallest or largest object among a class of similar objects, where the order is given by the morphisms. The simplest form of such objects are initial and final objects which are the smallest and largest objects of a whole category, respectively.

Definition A.5 (Initial and Final Objects)
Given a category **C**, an *initial object* $I \in |\mathbf{C}|$ is an object satisfying the *universal property* that there exists a unique morphism $i\colon I \to A$ for each object $A \in |\mathbf{C}|$. Dually, for a *final object* $F \in |\mathbf{C}|$ there exists a unique morphism $f\colon A \to F$ for each object $A \in |\mathbf{C}|$.

An important property of such objects is that they are unique up to isomorphism, i. e., there may be other objects satisfying the universal property but there are isomorphisms between all of them which makes them absolutely equivalent from a categorical point of view.

Proposition A.6 (Uniqueness of Initial and Final Objects)
Initial and final objects are unique up to isomorphisms, i. e., given two initial objects I and I', there is an isomorphism $i\colon I \to I'$, and given two final objects F and F', there is an isomorphism $f\colon F \to F'$.

Beweis. Since I is initial w. r. t. I' as comparison object there is a unique morphism $i\colon I \to I'$ and, vice versa, since I' is initial w. r. t. I as comparison object there is a unique morphism $j\colon I' \to I$. Hence, we have the compositions $j \circ i\colon I \to I$ and $i \circ j\colon I' \to I'$. But since I and I' are also comparison objects for themselves, respectively, $\mathrm{id}_I\colon I \to I$ and $\mathrm{id}_{I'}\colon I' \to I'$ are unique, $j \circ i = \mathrm{id}_I$ and $i \circ j = \mathrm{id}_{I'}$ and, hence, i (and j) are isomorphisms. A similar argument applies for final objects. □

The similarity, alluded to in the last sentence of the previous proof, is called duality in category theory. In general, most categorical constructions have a dual construction, where all morphisms are reversed in direction and all other used constructions are also replaced with their duals. It is usually sufficient to just prove one property and the dual one follows immediately and we will do so for the remainder of this appendix. If, however, constructions involve the inner structure of the objects of a special category duality cannot be used anymore.

In **Set** the initial object is the empty set and any object with a single element is a final object, while in **SetIncl** the empty set is also initial but there is no final object since we do not assume a universe set of all possible elements.

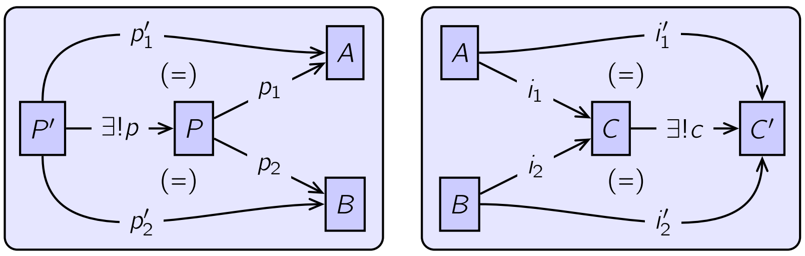
Proposition A.7 (Initial and Final Objects in **Set** and **SetIncl**)
In **Set** the empty set $\varnothing$ is the only initial object and the singleton set $\{*\}$ with one element $*$ is a final object.
In **SetIncl** the empty set $\varnothing$ is the only initial object and there does not exist a final object.

Beweis. The empty set $\varnothing$ is initial in **Set** since for each other set $A \in |\mathbf{Set}|$ there is the unique (likewise empty) function $\mathrm{incl}_{\varnothing,A} \colon \varnothing \to A$. There can be no other initial objects since there can be no total function from a set containing elements to the empty set (and no other set is isomorphic to the empty set because isomorphisms correspond to bijections between sets with the same cardinality).
The singleton set $\{*\}$ is final in **Set** since for each other set $A \in |\mathbf{Set}|$ a function $f \colon A \to \{*\}$ is given by $f(a) := *$ for all $a \in A$ which is unique since the function has to be total and there are no other possible image elements. Other singleton sets are also final, where the isomorphisms between them are given by the bijections mapping the corresponding single elements to each other.
The empty set $\varnothing$ is initial in **SetIncl** since $\varnothing \subseteq A$ for all sets $A \in |\mathbf{SetIncl}|$.
There is no final object in **SetIncl** since we assume to work in the class of all sets (as opposed to the set of all subsets of a given universe) and the union of all possible sets is not a set itself in order to avoid the well-known paradoxes of set theory. □

The next kinds of objects defined by universal properties are products and coproducts, where two objects are given and the product is the final object among those who have morphisms into both given objects, while the coproduct is the initial object among those who have morphisms from both objects. Such initial and final objects w. r. t. given diagrams are called limits and colimits in category theory, where limits are final objects with morphisms into all objects in the diagram and colimits are initial objects with morphisms from all objects in the diagram. Initial and final objects themselves may be considered as limits and colimits of the empty diagram.

Definition A.6 (Products and Coproducts)
Given a category $\mathbf{C}$ and two objects $A, B \in |\mathbf{C}|$, a *product* of A and B is an object P with *projection morphisms* $p_1 \colon P \to A$ and $p_2 \colon P \to A$, such that for each other object

P' with morphisms $p'_1\colon P' \to A$ and $p'_2\colon P' \to B$ there is a unique morphism $p\colon P' \to P$ with $p'_1 = p_1 \circ p$ and $p'_2 = p_2 \circ p$ (cf. Figure A.5a).
Vice versa, a *coproduct* of A and B is an object C with *injection morphisms*[2] $i_1\colon A \to C$ and $i_2\colon B \to C$, such that for each other object C' with morphisms $i'_1\colon A \to C'$ and $i'_2\colon B \to C'$ there is a unique morphism $c\colon C \to C'$ with $i'_1 = c \circ i_1$ and $i'_2 = c \circ i_2$ (cf. Figure A.5b).



(a) Product (b) Coproduct

Abbildung A.5.: Products and coproducts

We now show that the injections into the coproduct are jointly epimorphic in all categories.

Proposition A.8 (Coproducts are Jointly Epimorphic)
Given a category **C**, two objects A and B and a coproduct C of A and B with injections $i_1\colon A \to C$ and $i_2\colon B \to C$, then i_1 and i_2 are jointly epimorphic.

Beweis. Each object D with morphisms $f_1, f_2\colon C \to D$ satisfying $f_1 \circ i_1 = f_2 \circ i_1$ and $f_1 \circ i_2 = f_2 \circ i_2$ is a comparison object for the coproduct by the morphisms $g\colon A \to D := f_1 \circ i_1 = f_2 \circ i_1$ and $h\colon B \to D := f_1 \circ i_2 = f_2 \circ i_2$. Then there has to be a unique morphism $f : C \to D$ with $f \circ i_1 = g$ and $f \circ i_2 = h$ by the coproduct definition. But since both, f_1 and f_2, satisfy these equalities uniqueness implies $f_1 = f = f_2$. Hence, i_1 and i_2 are jointly epi. $\square$

For **Set** and **SetIncl** products and coproducts are categorical characterisations for some well-known constructions. More specifically, in **Set** the usual cartesian product is a product and the disjoint union is a coproduct, while in **SetIncl** the intersection is the only product and the (non-disjoint) union is the only coproduct.

[2] Note that despite the intuitive name given to them these morphisms do not need to be monomorphisms or injective in all categories. However, they are in all categories we deal with in this thesis.

Proposition A.9 (Products and Coproducts in **Set** and **SetIncl**)
Given sets $A, B \in |\mathbf{Set}|$, a product of A and B can be constructed by $A \times B := \{(a, b) \mid a \in A, b \in B\}$ with projections $p_1(a, b) := a$ and $p_2(a, b) := b$ and a coproduct by $A + B := \{(1, a) \mid a \in A\} \cup \{(2, b) \mid b \in B\}$ with injections $i_1(a) := (1, a)$ and $i_2(b) := (2, b)$.
Given sets $A, B \in |\mathbf{SetIncl}|$, the product of A and B is given by $A \cap B$ with projections $A \cap B \subseteq A$ and $A \cap B \subseteq B$ and the coproduct by $A \cup B$ with injections $A \subseteq A \cup B$ and $B \subseteq A \cup B$.

Beweis. For the product in **Set** and a comparison object P' with projections p'_1 and p'_2 the requirements $p_1(p(x)) = p'_1(x)$ and $p_2(p(x)) = p'_2(x)$ uniquely determine the morphism p to be defined by $p(x) := (p'_1(x), p'_2(x))$.
For the coproduct in **Set** and a comparison object C' with injections i'_1 and i'_2 the requirements $c(i_1(a)) = i'_1(a)$ and $c(i_2(a)) = i'_2(a)$ uniquely determine the morphism c to be defined by $c(1, a) := i'_1(a)$ for elements from A and $c(2, b) := i'_2(b)$ for elements from B.
For **SetIncl** the required inclusions for comparison objects exist since all sets that are contained in both sets are also contained in the intersection and, vice versa, all sets containing both sets also contain their union. These inclusions are obviously unique since all inclusions in **SetIncl** are unique. □

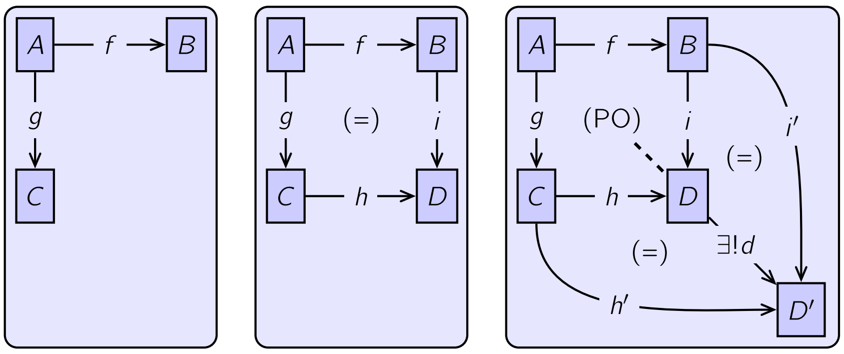
The most important limits and colimits for the purposes of this thesis are pushout and pullbacks, where not only objects but also morphisms are in the given diagram. More specifically, pushouts are colimits under[3] two objects that share a common third object, while pullbacks are limits over two objects that are contained in a common third object by corresponding morphisms. The resulting morphisms are an intrinsic part of the pushouts and pullbacks and can usually not be understood from the context (as it was the case for products and coproducts). Therefore, we will define a category, called cone or cocone category, respectively, of comparison objects built from the original category that consists of objects containing not only the comparison objects themselves but also the corresponding morphisms. The pushouts and pullbacks are then initial and final objects in these categories, respectively. This style of definition is also used for the minimal pushout complements and the initial pushouts in Section 3.2.

Definition A.7 (Pushouts and Pullbacks)
Given a category $\mathbf{C}$ and morphisms $f\colon A \to B$ and $g\colon A \to C$ (cf. Figure A.6a), then the category $\mathbf{Cocone}(f, g)$ of *cocones under f and g* consists of

objects $(D, h\colon C \to D, i\colon B \to D)$ such that $h \circ g = i \circ f$ (cf. Figure A.6b),

morphisms $d\colon D \to D'$ between objects (D, h, i) and (D', h', i') such that $d \circ h = h'$ and $d \circ i = i'$ and

[3] Note that the words "under" and "over" are usually used in category theory with the intuition that morphisms are drawn from top to bottom, i. e., a colimit *under* a diagram means that the morphisms go into the colimit, while a limit *over* a diagram means that the morphisms go from the limit into the diagram.



(a) Given situation (b) Cocone of f and g (c) Pushout of f and g

Abbildung A.6.: Pushouts of f and g

compositions and identities given by compositions and identities in **C**.

A *pushout of* f *and* g is then an initial object (D, h, i) in **Cocone**(f, g) (cf. Figure A.6c). Given a category **C** and morphisms $f: C \to D$ and $g: B \to D$ (cf. Figure A.7a), then the category **Cone**(f, g) of *cones over* f *and* g consists of

objects $(A, d: A \to B, e: A \to C)$ such that $g \circ d = f \circ e$ (cf. Figure A.7b),

morphisms $a: A \to A'$ between objects (A, d, e) and (A', d', e') such that $d \circ a = d'$ and $e \circ a = e'$ and

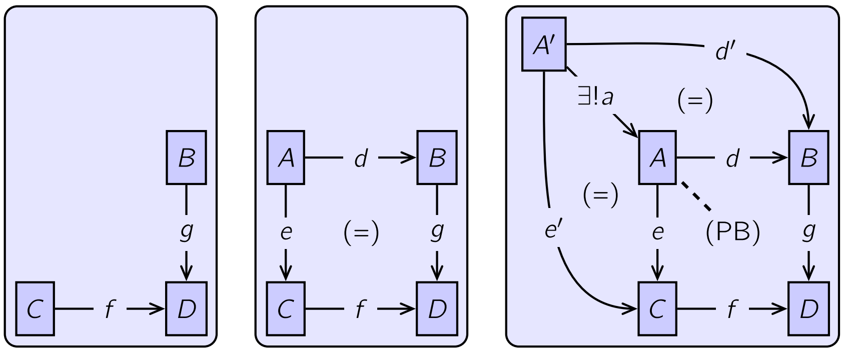
compositions and identities given by compositions and identities in **C**.

A *pullback of* f *and* g is then a final object (A, d, e) in **Cone**(f, g) (cf. Figure A.7c).

Similarly to coproducts, pushouts are also jointly epi, where the proof is also very similar. In fact, if we would extend the definition of jointly epimorphic morphisms to also cover sets with more than two morphisms then all colimits would be jointly epimorphic for the set of all morphisms into the colimit. For our purposes, it is, however, enough to know that pushouts are jointly epi.

Proposition A.10 (Pushouts are Jointly Epimorphic)
Given a category **C**, two morphisms $f: A \to B$ and $g: A \to C$ and a pushout (D, h, i) of f and g, then h and i are jointly epimorphic.

Beweis. Each object E with morphisms $j_1, j_2: D \to E$ satisfying $j_1 \circ h = j_2 \circ h$ and $j_1 \circ i = j_2 \circ h$ constitutes a comparison object $(E, j_1 \circ h, j_1 \circ i$ by $j_1 \circ h \circ g = j_1 \circ i \circ f$ (or equivalently a comparison object $(E, j_2 \circ h, j_2 \circ i)$ by $j_2 \circ \circ g = j_2 \circ i \circ f$). Then there has to be a unique morphism $k: D \to E$ with $k \circ h = j_1 \circ h$ and $k \circ i = j_1 \circ i$ (or equivalently



(a) Given situation (b) Cone of f and g (c) Pullback of f and g

Abbildung A.7.: Pullbacks of f and g

$k \circ h = j_2 \circ h$ and $k \circ i = j_2 \circ i$) by the definition of pushouts. Since both, j_1 and j_2, satisfy these equalities uniqueness implies $j_1 = k = j_2$. Hence, h and i are jointly epi. □

Finally, compositions and decompositions of pushouts and pullbacks are important preliminaries for the results in the main part of the thesis, especially for the corresponding results for minimal pushout complements and initial pushouts in Section 3.3. While pushouts and pullbacks can be composed to yield larger pushouts and pullbacks, decomposition is only possible if not only the large square but also a specific component square is given to be of the corresponding type.

Proposition A.11 (Composition and Decomposition of Pushouts and Pullbacks)
Let a category **C** and morphisms $f\colon A \to B$, $g\colon B \to C$, $h\colon A \to D$, $i\colon B \to E$, $j\colon C \to F$, $k\colon D \to E$ and $l\colon E \to F$, such that $i \circ f = k \circ h$ and $j \circ g = l \circ i$ (cf. Figure A.8), be given.

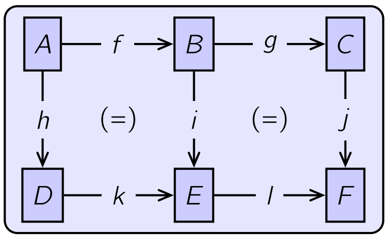
Composition: If (E, k, i) is a pushout of f and h and (F, l, j) is a pushout of g and i (both small squares are pushouts) then $(F, l \circ k, j)$ is a pushout of $g \circ f$ and h (the large square is a pushout). Vice versa, if (A, f, h) is a pullback of k and i and (B, g, i) is a pullback of l and j (both small squares are pullbacks) then $(A, g \circ f, h)$ is a pullback of $l \circ k$ and j (the large square is a pullback).

Decomposition: If $(F, l \circ k, j)$ is a pushout of $g \circ f$ and h and (E, k, i) is a pushout of f and h (the large and the first small square are pushouts) then (F, l, j) is a pushout of g and i (the second small square is a pushout). Vice versa, if $(A, g \circ f, h)$ is a pullback of $l \circ k$ and j and (B, g, i) is a pullback of l and j (the large and the second small square are pullbacks) then (A, f, h) is a pullback of k and i (the first small square is a pullback).

Beweis. Since the corresponding proof for pullbacks can be ontained by dualisation we only show this proposition for pushouts.

Composition: Suppose there is a comparison object (F', j', m') for the large square. Since $j' \circ g \circ f = m' \circ h$ this induces a comparison object $(F', j' \circ g, m')$ for the pushout (E, k, i) and yields a unique morphism $l' \colon E \to F'$ with $l' \circ k = m'$ and $l' \circ i = j' \circ g$. The last equality in turn means that (F', l', j') is a comparison object for the pushout (F, l, j) leading to a unique $f \colon F \to F'$ with $f \circ l = l'$ and $f \circ j = j'$. By composing with k we obtain $f \circ l \circ k = l' \circ k = m'$ and f is also a comparison morphism for the large square. Since each other morphism $f' \colon F \to F'$ with $f \circ j = j'$ and $f' \circ l \circ k = m'$ also satisfies $f' \circ l \circ i = f' \circ j \circ g = j' \circ g$ we get $f' \circ l = l'$ by uniqueness of l' and then $f' = f$ by uniqueness of f. Hence, f is also unique w. r. t. the equalities for the large square and the latter is a pushout.

Decomposition: Suppose there is a comparison object (F', l', j') for the second small square. Since $l' \circ i = j' \circ g$ we get $l' \circ k \circ h = l' \circ i \circ f = j' \circ g \circ f$ and $(F', l' \circ k, j')$ is a comparison object for the large square. Hence, there is a unique $f \colon F \to F'$ with $f \circ l \circ k = l' \circ k$ and $f \circ j = j'$. It follows that $f \circ l \circ i = f \circ j \circ g = j' \circ g = l' \circ i$ and since k and i are jointly epi also $f \circ l = l'$. Hence, f is a comparison morphism for the second small square. Each other morphism $f' \colon F \to F'$ with $f' \circ l = l'$ and $f' \circ j = j'$ also satisfies $f' \circ l \circ k = l' \circ k$ and uniqueness of f w. r. t. the latter equalities implies $f' = f$. Thus, f is also unique w. r. t. the equalities for the second small square and it is a pushout. □



$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "h"] \arrow[dr, phantom, "(=)"] & B \arrow[r, "g"] \arrow[d, "i"] \arrow[dr, phantom, "(=)"] & C \arrow[d, "j"] \\
D \arrow[r, "k"] & E \arrow[r, "l"] & F
\end{tikzcd}$$

Abbildung A.8.: Composition and decomposition of pushouts and pullbacks

Literaturverzeichnis

[AFR06] Daniel Amyot, Hanna Farah, Jean-François Roy. Evaluation of Development Tools for Domain-Specific Modeling Languages. In Gotzhein and Reed (editors), *System Analysis and Modeling: Language Profiles. Proc. SAM 2006*. LNCS 4320, pages 183–197. Springer, 2006.
doi:10.1007/11951148_12

[AHS90] Jiří Adámek, Horst Herrlich, George E. Strecker. *Abstract and Concrete Categories. The Joy of Cats*. Wiley, 1990.
http://katmat.math.uni-bremen.de/acc/

[Alv01] Harald Tveit Alvestrand. RFC 3066 – Tags for the Identification of Languages. Internet Engineering Task Force (IETF), January 2001. Best current practice.
http://tools.ietf.org/html/rfc3066

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

[BB08] Benjamin Braatz, Christoph Brandt. Graph Transformations for the Resource Description Framework. In Ermel et al. (editors), *Proc. GT-VMT 2008*. Electronic Communications of the EASST 10. 2008.
http://eceasst.cs.tu-berlin.de/index.php/eceasst/article/view/158

[BEB$^+$07] Christoph Brandt, Thomas Engel, Benjamin Braatz, Frank Hermann, Hartmut Ehrig. An Approach Using Formally Well-founded Domain Languages for Secure Coarse-grained IT System Modelling in a Real-world Banking Scenario. In *ACIS 2007 Proceedings*. 2007.
http://aisel.aisnet.org/acis2007/62/

[Bec04] Dave Beckett. RDF/XML Syntax Specification (Revised). World Wide Web Consortium (W3C), February 2004. Part of [W3C04].
http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

[BFM98] Tim Berners-Lee, Roy T. Fielding, Larry Masinter. RFC 2396 – Uniform Resource Identifiers (URI): Generic Syntax. Internet Engineering Task Force

(IETF), August 1998. Draft standard.
http://tools.ietf.org/html/rfc2396

[BFM05] Tim Berners-Lee, Roy T. Fielding, Larry Masinter. RFC 3986 – Uniform Resource Identifiers (URI): Generic Syntax. Internet Engineering Task Force (IETF), January 2005. Standard, obsoletes [BFM98].
http://tools.ietf.org/html/rfc3986

[BG04] Dan Brickley, R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. World Wide Web Consortium (W3C), February 2004. Part of [W3C04].
http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

[BHE09] Christoph Brandt, Frank Hermann, Thomas Engel. Security and Consistency of IT and Business Models at Credit Suisse Realized by Graph Constraints, Transformation and Integration Using Algebraic Graph Theory. In Halpin et al. (editors), *BPMDS 2009 and EMMSAD 2009 Proceedings*. LNBIP 29. Springer, 2009.
doi:10.1007/978-3-642-01862-6_28

[BHL01] Tim Berners-Lee, James Hendler, Ora Lassila. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American* 284(5):34–43, May 2001.
http://www.scientificamerican.com/article.cfm?id=the-semantic-web

[BHLT06] Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin. Namespaces in XML 1.0 (Second Edition). World Wide Web Consortium (W3C), August 2006.
http://www.w3.org/TR/2006/REC-xml-names-20060816/

[BM04] Paul V. Biron, Ashok Malhotra. XML Schema Part 2: Datatypes (Second Edition). World Wide Web Consortium (W3C), October 2004.
http://www.w3.org/TR/xmlschema-2/

[CHHK06] Andrea Corradini, Tobias Heindel, Frank Hermann, Barbara König. Sesqui-Pushout Rewriting. In Corradini et al. (editors), *Graph Transformations. Proc. ICGT 2006*. LNCS 4178, pages 30–45. Springer, 2006.
doi:10.1007/11841883_4

[DS05] M. Dürst, M. Suignard. RFC 3987 – Internationalized Resource Identifiers (IRIs). Internet Engineering Task Force (IETF), January 2005. Proposed standard.
http://tools.ietf.org/html/rfc3987

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, Gariele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer

Science. Springer, 2006.
doi:10.1007/3-540-31188-2

[EG94] Hartmut Ehrig, Martin Große-Rhode. Functorial theory of parameterized specifications in a general specification framework. *Theoretical Computer Science (TCS)* 135(2):221–266, December 1994.
doi:10.1016/0304-3975(94)90110-4

[EHKP91] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, Francesco Parisi-Presicce. From graph grammars to high level replacement systems. In Ehrig et al. (editors), *Graph Grammars and Their Application to Computer Science*. LNCS 532, pages 269–291. Springer, 1991.
doi:10.1007/BFb0017395

[Ehr79] Hartmut Ehrig. Introduction to the Algebraic Theory of Graph Grammars (A Survey). In Claus et al. (editors), *Graph Grammars and Their Application to Computer Science and Biology*. LNCS 73, pages 1–69. Springer, 1979.
doi:10.1007/BFb0025714

[EPN09] Fredrik Enoksson, Matthias Palmér, Ambjörn Naeve. An RDF Modification Protocol, based on the Needs of Editing Tools. In Sicilia and Lytras (editors), *Metadata and Semantics*. Pages 191–199. Springer, 2009.
doi:10.1007/978-0-387-77745-0_18
http://kmr.nada.kth.se/papers/SemanticWeb/Corfu07_Remote_Editing.pdf

[EPS73] Hartmut Ehrig, Michael Pfender, Hans Jürgen Schneider. Graph-grammars: An algebraic approach. In *Switching and Automata Theory. Proc. SWAT 1973*. Pages 167–180. IEEE, 1973.
doi:10.1109/SWAT.1973.11

[GB92] Joseph A. Goguen, Rod M. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the ACM (JACM)* 39(1):95–146, January 1992.
doi:10.1145/147508.147524

[GB04] Jan Grant, Dave Beckett. RDF Test Cases. World Wide Web Consortium (W3C), February 2004. Part of [W3C04].
http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/

[Hay04] Patrick Hayes. RDF Semantics. World Wide Web Consortium (W3C), February 2004. Part of [W3C04].
http://www.w3.org/TR/2004/REC-rdf-mt-20040210/

[Her08] Matthias Hert. RDF Graph Transformation. Bridging between Ontologies. Diploma thesis, University of Zurich, February 2008.
http://www.ifi.uzh.ch/pax/uploads/pdf/publication/611/Thesis.pdf

[KC04] Graham Klyne, Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. World Wide Web Consortium (W3C), February 2004. Part of [W3C04].
http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[Lam07] Leen Lambers. Adhesive High-Level Replacement Systems with Negative Application Conditions. Technical report 2007/14, Technische Universität Berlin, 2007.
http://iv.tu-berlin.de/TechnBerichte/2007/2007-14.pdf

[LE90] Michael Löwe, Hartmut Ehrig. Algebraic approach to graph transformation based on single pushout derivations. In Möhring (editor), *Proc. Graph-Theoretic Concepts in Computer Science*. LNCS 484, pages 338–353. Springer, 1990.
doi:10.1007/3-540-53832-1_52

[LLD06] Dorel Lucanu, Yuan Fang Li, Jin Song Dong. RDF Framework Institutions. *Proceedings of the Romanian Academy* 7(1), 2006.
http://www.acad.ro/sectii2002/proceedings/doc2006-1/08-Luncanu.pdf

[Mar06] Draltan Marin. RDF formalization. Technical report TR/DCC-2006-8, Universidad de Chile, May 2006.
http://www.dcc.uchile.cl/TR/2006/TR_DCC-2006-008.pdf

[MH04] Deborah L. McGuinness, Frank van Harmelen. OWL Web Ontology Language Overview. World Wide Web Consortium (W3C), February 2004.
http://www.w3.org/TR/2004/REC-owl-features-20040210/

[MM04] Frank Manola, Eric Miller. RDF Primer. World Wide Web Consortium (W3C), February 2004. Part of [W3C04].
http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[MPG07] Sergio Muñoz, Jorge Pérez, Claudio Gutierrez. Minimal Deductive Systems for RDF. In Franconi et al. (editors), *The Semantic Web: Research and Applications. Proc. ESWC 2007*. LNCS 4519, pages 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

[PS08] Eric Prud'hommeaux, Andy Seaborne. SPARQL Query Language for RDF. World Wide Web Consortium (W3C), January 2008.
http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[SM08] Andy Seaborne, Geetha Manjunath. SPARQL/Update. A language for updating RDF graphs. April 2008.
http://jena.hpl.hp.com/~afs/SPARQL-Update.html

[Tol06] Juha-Pekka Tolvanen. MetaEdit+: integrated modeling and metamodeling environment for domain-specific languages. In *Proc. OOPSLA 2006*. Pages 690–691. ACM, 2006.
doi:10.1145/1176617.1176676

[Uni] The Unicode Consortium. The Unicode Standard, Version 5.1.0. Defined by: The Unicode Standard, Version 5.0 (Addison-Wesley, 2007), as amended by Unicode 5.1.0 (http://www.unicode.org/versions/Unicode5.1.0/).

[W3C04] World Wide Web Consortium (W3C). Resource Description Framework (RDF). February 2004. Consists of [MM04], [KC04], [Bec04], [Hay04], [BG04] and [GB04].
http://www.w3.org/RDF/

[Yer03] François Yergeau. RFC 3629 – UTF-8, a transformation format of ISO 10646. Internet Engineering Task Force (IETF), November 2003. Standard.
http://tools.ietf.org/html/rfc3629

Digital Object Identifiers (DOIs) can be resolved using the web service provided at http://dx.doi.org/.