

Jens Kosiol

**Formal Foundations for  
Information-Preserving Model  
Synchronization Processes Based  
on Triple Graph Grammars**





# Formal Foundations for Information-Preserving Model Synchronization Processes Based on Triple Graph Grammars

## Dissertation

zur Erlangung des Doktorgrades  
der Naturwissenschaften (Dr. rer. nat.)

dem Fachbereich Mathematik und Informatik  
der Philipps-Universität Marburg  
(Hochschulkenziffer 1180)  
vorgelegt von

Jens Kosiol, MA, MSc  
geboren in Siegen

Marburg, Juni 2022

Supervised by: Professor Dr. Gabriele Taentzer  
Philipps-Universität Marburg  
Marburg, Germany

Second referee: Professor Dr. Fernando Orejas  
Universitat Politècnica de Catalunya  
Barcelona, Spain

Third referee: Professor Dr. Reiko Heckel  
University of Leicester  
Leicester, UK

Date of submission: August 27, 2021  
Date of thesis defense: October 22, 2021

Für Tatjana,  
Jonathan, Jael und Jakob



# Abstract

Restoring consistency between different information-sharing artifacts after one of them has been changed is an important problem that arises in several areas of computer science. In this thesis, we provide a solution to the *basic model synchronization problem*. There, a pair of such artifacts (models), one of which has been changed, is given and consistency shall be restored. *Triple graph grammars* (TGGs) are an established and suitable formalism to address this and related problems. Being based on the *algebraic theory of graph transformation* and *(double-)pushout rewriting*, they are especially suited to develop solutions whose properties can be formally proven. Despite being established, many TGG-based solutions do not satisfactorily deal with the *problem of information loss*. When one model is changed, in the process of restoring consistency such solutions may lose information that is only present in the second model because the synchronization process resorts to restoring consistency by re-translating (large parts of) the updated model. We introduce a TGG-based approach that supports advanced features of TGGs (attributes and negative constraints), is comprehensively formalized, implemented, and is *incremental* in the sense that it drastically reduces the amount of information loss compared to former approaches. Up to now, a TGG-based approach with these characteristics is not available.

The central contribution of this thesis is to formally develop that approach and to prove its essential properties, namely correctness, completeness, and termination. The crucial new idea in our approach is the use of *repair rules*, which are special rules that allow one to directly propagate changes from one model to the other instead of resorting to re-translation. To be able to construct and apply these repair rules, we contribute more fundamentally to the theory of algebraic graph transformation. First, we develop a new kind of sequential rule composition. Whereas the conventional composition of rules leads to rules that delete and re-create elements, we can compute rules that preserve such elements instead. Furthermore, technically the setting in which the synchronization process we develop takes place is the category

of *partial triple graphs* and not the one of ordinary triple graphs. Hence, we have to ensure that the elaborate theory of double-pushout rewriting still applies. Therefore, we develop a (category-theoretic) construction of new categories from given ones and show that (i) this construction preserves the axioms that are necessary to develop the theory of double-pushout rewriting and (ii) partial triple graphs can be constructed as such a category. Together, those two more fundamental contributions enable us to develop our solution to the basic model synchronization problem in a fully formal manner and to prove its central properties.

# Zusammenfassung

Zwischen verschiedenen Artefakten, die Informationen teilen, wieder Konsistenz herzustellen, nachdem eines von ihnen geändert wurde, ist ein wichtiges Problem, das in verschiedenen Bereichen der Informatik auftaucht. Mit dieser Dissertation legen wir eine Lösung für das *grundlegende Modellsynchronisationsproblem* vor. Bei diesem Problem ist ein Paar solcher Artefakte (Modelle) gegeben, von denen eines geändert wurde; Aufgabe ist die Wiederherstellung der Konsistenz. Tripelgraphgrammatiken (TGGs) sind ein etablierter und geeigneter Formalismus, um dieses und verwandte Probleme anzugehen. Da sie auf der *algebraischen Theorie der Graphtransformation* und dem *(Double-)Pushout Zugang zu Ersetzungssystemen* basieren, sind sie besonders geeignet, um Lösungen zu entwickeln, deren Eigenschaften formal bewiesen werden können. Doch obwohl TGG-basierte Ansätze etabliert sind, leiden viele von ihnen unter dem *Problem des Informationsverlustes*. Wenn ein Modell geändert wurde, können während eines Synchronisationsprozesses Informationen verloren gehen, die nur im zweiten Modell vorliegen. Das liegt daran, dass solche Synchronisationsprozesse darauf zurückfallen Konsistenz dadurch wiederherzustellen, dass sie das geänderte Modell (bzw. große Teile von ihm) neu übersetzen. Wir schlagen einen TGG-basierten Ansatz vor, der fortgeschrittene Features von TGGs unterstützt (Attribute und negative Constraints), durchgängig formalisiert ist, implementiert und *inkrementell* in dem Sinne ist, dass er den Informationsverlust im Vergleich mit vorherigen Ansätzen drastisch reduziert. Bisher gibt es keinen TGG-basierten Ansatz mit vergleichbaren Eigenschaften.

Zentraler Beitrag dieser Dissertation ist es, diesen Ansatz formal auszuarbeiten und seine wesentlichen Eigenschaften, nämlich Korrektheit, Vollständigkeit und Termination, zu beweisen. Die entscheidende neue Idee unseres Ansatzes ist es, *Reparaturregeln* anzuwenden. Dies sind spezielle Regeln, die es erlauben, Änderungen an einem Modell direkt zu propagieren anstatt auf Neuübersetzung zurückzugreifen. Um diese Reparaturregeln

erstellen und anwenden zu können, entwickeln wir grundlegende Beiträge zur Theorie der algebraischen Graphtransformation. Zunächst entwickeln wir eine neue Art der sequentiellen Komposition von Regeln. Im Gegensatz zur gewöhnlichen Komposition, die zu Regeln führt, die Elemente löschen und dann wieder neu erzeugen, können wir Regeln herleiten, die solche Elemente stattdessen bewahren. Technisch gesehen findet der Synchronisationsprozess, den wir entwickeln, außerdem in der Kategorie der *partiellen Tripelgraphen* statt und nicht in der der normalen Tripelgraphen. Daher müssen wir sicherstellen, dass die für Double-Pushout-Ersetzungssysteme ausgearbeitete Theorie immer noch gültig ist. Dazu entwickeln wir eine (kategorientheoretische) Konstruktion neuer Kategorien aus gegebenen und zeigen, dass (i) diese Konstruktion die Axiome erhält, die nötig sind, um die Theorie für Double-Pushout-Ersetzungssysteme zu entwickeln, und (ii) partielle Tripelgraphen als eine solche Kategorie konstruiert werden können. Zusammen ermöglichen diese beiden grundsätzlichen Beiträge es uns, unsere Lösung für das grundlegende Modellsynchronisationsproblem vollständig formal auszuarbeiten und ihre zentralen Eigenschaften zu beweisen.

# Contents

<b>Abstract</b>	<b>v</b>
<b>Zusammenfassung</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Problem Statement . . . . .	1
1.2 General Approach and Technical Problems . . . . .	3
1.3 Contributions . . . . .	6
1.4 Structure . . . . .	10
<b>2 Bidirectional Transformations—State of the Art and our Approach</b>	<b>13</b>
2.1 A Survey on Bidirectional Transformations and Model Synchronization . . . . .	13
2.1.1 Non-TGG-Based Approaches . . . . .	14
2.1.2 TGG-Based Approaches . . . . .	20
2.2 Introduction to our Approach by Example . . . . .	29
<b>3 Adhesive Categories and Two Algebraic Approaches to Rewriting</b>	<b>45</b>
3.1 Variants of Adhesive Categories and their Basic HLR Properties . . . . .	46
3.2 Two Algebraic Approaches to Rewriting . . . . .	52
3.2.1 Nested Conditions and the Double-Pushout Approach . . . . .	52
3.2.2 The Sesqui-Pushout Approach to Rewriting . . . . .	58
3.2.3 Sequential Independence . . . . .	59
3.2.4 Concurrent Rules and the Concurrency Theorem . . . . .	61
3.3 Further Categorical Preliminaries and Additional HLR Properties . . . . .	64
3.3.1 Properties of Pushouts and Pullbacks and a Categorical Definition of Partial Morphisms . . . . .	65
3.3.2 Additional HLR Properties . . . . .	66

3.4	Typed Attributed (Triple) Graphs and Triple Graph Grammars . . . . .	74
<b>4</b>	<b>A Generalized Concurrent Rule Construction for Double-Pushout Rewriting</b>	<b>81</b>
4.1	Introduction . . . . .	81
4.2	Running Example . . . . .	84
4.3	The Short-Cut Rule Construction . . . . .	85
4.4	Generalized Concurrent Rules—Construction, Preservation, and Relations . . . . .	88
4.4.1	Construction . . . . .	89
4.4.2	Preservation Property of Applications of GCRs . . . . .	93
4.4.3	Relating Generalized Concurrent Rules to other Variants of Rule Composition . . . . .	95
4.5	Generalized Concurrent Rules—Central Results . . . . .	98
4.5.1	Central Results for the Case of $\mathcal{M}$ -Matching . . . . .	99
4.5.2	Central Results for the Case of General Matching . . . . .	113
4.5.3	Conditions for Language-Preserving Applications of Generalized Concurrent Rules . . . . .	125
4.6	Related Work . . . . .	127
4.7	Conclusion . . . . .	129
<b>5</b>	<b>Double-Pushout Rewriting in <math>S</math>-Cartesian Functor Categories</b>	<b>131</b>
5.1	Introduction . . . . .	131
5.2	$S$ -Cartesian Functor Categories and Their HLR Properties	135
5.2.1	$S$ -Cartesian Functor Categories . . . . .	135
5.2.2	Additional HLR Properties of $S$ -Cartesian Functor Categories . . . . .	149
5.2.3	$S$ -Cartesian Functor Categories over Graphs . . . . .	152
5.3	The Category of Typed Attributed Partial Triple Graphs . . . . .	154
5.4	Related Work . . . . .	173
5.5	Conclusion . . . . .	178
<b>6</b>	<b>Information-Preserving Model Synchronization</b>	<b>179</b>
6.1	Introduction . . . . .	179

6.2	Integrity-Preserving Triple Graph Grammars and the Basic Model Synchronization Problem . . . . .	182
6.2.1	Integrity-Preserving Triple Graph Grammars . . . . .	182
6.2.2	The Basic Model Synchronization Problem . . . . .	189
6.2.3	Derived Model Synchronization Operations . . . . .	193
6.3	Precedence Structure and Incremental Pattern Matching . . . . .	207
6.3.1	Markings, Covers, and Partial Precedence Graphs . . . . .	207
6.3.2	Delta-Induced Updates of Precedence Structure . . . . .	218
6.4	Information-Preserving Model Synchronization—Our Algorithm . . . . .	230
6.4.1	The Basic Setup . . . . .	230
6.4.2	The Synchronization Process . . . . .	232
6.4.3	Example Run of the Algorithm . . . . .	237
6.4.4	Central Formal Properties . . . . .	247
6.5	Related Work . . . . .	262
6.6	Conclusion . . . . .	266
<b>7</b>	<b>Conclusion and Outlook</b>	<b>271</b>
	<b>Bibliography</b>	<b>277</b>
	<b>Appendix</b>	<b>310</b>
<b>A</b>	<b><math>\mathcal{E}'</math>-<math>\mathcal{M}'</math> Pair Factorization of Morphisms in <math>\text{ATrG}_{\text{ATG}}</math></b>	<b>311</b>
<b>B</b>	<b>Further Proofs</b>	<b>315</b>
B.1	Proofs from Chapter 4 . . . . .	315
B.2	Proofs from Chapter 5 . . . . .	330
B.3	Proofs from Chapter 6 . . . . .	349
<b>C</b>	<b>Scientific CV</b>	<b>363</b>



## List of Figures

2.1	Type Graph for the Java AST-to-documentation example	30
2.2	TGG rules for the Java-AST-to-documentation example	32
2.3	Instance of the Java AST-to-documentation example	33
2.4	Precedence graph for the example instance shown in Fig. 2.3	33
2.5	Source and forward rules derived from <i>CreateRoot</i> and <i>CreateLeaf</i>	34
2.6	Result of user edit applied to instance from Fig. 2.3	36
2.7	Triple graph resulting from revoking the now invalid rule applications in Fig. 2.6	36
2.8	Repaired triple graph resulting from re-translating the triple graph from Fig. 2.7	37
2.9	Examples for <i>generalized concurrent</i> (or <i>short-cut</i> ) rules that allow for complex model edits	39
2.10	Repair rules obtained as operationalized GCRs	41
2.11	Source rule derived from <i>CollapseHierarchy</i>	42
2.12	Partial cover for the edited instance shown in Fig. 2.6	42
2.13	Precedence graph for the repaired triple graph (Fig. 2.8) induced by the repair rule application	44
3.1	Commuting square for definition of admissible monomorphisms	47
3.2	The bottom pushout square	48
3.3	Commutative cube over the pushout square from left	48
3.4	Illustration of the $\mathcal{M}$ pushout-pullback and $\mathcal{M}$ pullback-pushout decompositions	50
3.5	Guaranteeing square (1) to be a pullback square	52
3.6	A rule-based transformation in the double-pushout approach	54
3.7	Graphical representation of the definition of $\text{Shift}(b, \exists(a : P \rightarrow C, d))$	56
3.8	Final pullback complement	59

3.9	A rule-based transformation in the sesqui-pushout approach	59
3.10	Defining sequential independence	60
3.11	$E$ -dependency relation and $E$ -concurrent rule	62
3.12	An $E$ -related transformation	63
3.13	A concurrent rule that stems from monotonic rules	64
3.14	Illustrating pushout and pullback (de)composition	65
3.15	Pushout along an identity	66
3.16	Pullback of composed morphisms	66
3.17	Initial pushout and context object	72
3.18	First closure property of initial pushouts	73
3.19	Second closure property of initial pushouts	73
3.20	$\mathcal{M}$ -effective union	74
3.21	Schematic depiction of the structure of $E$ -graphs	76
3.22	Compatibility of graph and algebra homomorphism	76
4.1	Two refactoring rules for class diagrams (first line) and sequentially composed rules derived from them (second line)	84
4.2	Construction of the LHS and RHS of short-cut rule $r_1^{-1} \times_k r_2$	87
4.3	Construction of the interface $K$ of $r_1^{-1} \times_k r_2$	88
4.4	Compatibility of a common kernel with an $E$ -dependency relation	89
4.5	Common kernel for rules <i>removeMiddleMan</i> and <i>extractSubclass</i>	90
4.6	Construction of a generalized concurrent rule	92
4.7	Computing the interface and the left-hand morphism of <i>Ref2Gen_GCR</i>	93
4.8	Constructing transformation via generalized concurrent rule from transformation via the concurrent rule	94
4.9	Concurrent rule as generalized concurrent rule (where $v_1 = l_1 \circ u_1$ and $v_2 = r_2 \circ u_2$ )	96
4.10	Short-cut rule as generalized concurrent rule	97
4.11	Relations between the different kinds of rule composition	98
4.12	Interaction of initial pushouts with pullbacks	99
4.13	Obtaining $l' \in \mathcal{M}$ via $\mathcal{M}$ -effective unions	100
4.14	Proving the outer square of Fig. 4.13 to be a pullback	101
4.15	Counterexample to $\mathcal{M}$ -effective unions	101

4.16	Abstract counterexample to Proposition 4.6 in absence of $\mathcal{M}$ -effective unions . . . . .	102
4.17	Concrete counterexample to the validity of Proposition 4.6 in the category of simple graphs . . . . .	103
4.18	Definition of <i>appropriate enhancement</i> . . . . .	104
4.19	Illustrating the property of appropriate enhancement . . . . .	104
4.20	Proving appropriate enhancement . . . . .	105
4.21	Obtaining the common kernel $k : K_{\cap} \hookrightarrow V$ via pushout . . . . .	107
4.22	Ensuring $k$ to compute $K'$ . . . . .	108
4.23	Proving $\rho_1 *_E \rho_2$ to be applicable at $m$ . . . . .	111
4.24	Proving the square (1) to be a pullback. . . . .	112
4.25	Alternative construction of a generalized concurrent rule . . . . .	114
4.26	An individually identifying embedding . . . . .	116
4.27	Example showing Corollary 4.12 to not provide a necessary condition . . . . .	118
4.28	Counterexample to the factorization of the initial pushout over $k'$ through the one over $l_1$ . . . . .	120
4.29	Definition of appropriate enhancement in the case of general matching . . . . .	121
4.30	Illustrating the definition of <i>enhancement-free boundary</i> . . . . .	123
4.31	Characterizing valid matches for GCRs (for $t \geq 2$ ) . . . . .	126
5.1	Pushout of morphisms between injective functions in the arrow category, computed componentwise . . . . .	137
5.2	Pushout of morphisms between injective functions in the full subcategory of the arrow category that has injective functions as objects . . . . .	137
5.3	Identity morphism in $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ . . . . .	139
5.4	Composition of morphisms in $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ . . . . .	139
5.5	Creation of pullbacks by the inclusion functor . . . . .	141
5.6	Pushout along an $\mathcal{M}^{\text{cart}}$ -morphism . . . . .	143
5.7	$Dm'$ as isomorphism mediating between two pushout objects . . . . .	144
5.8	Pushout complement for one component . . . . .	147
5.9	Pullback resulting in pushout complement . . . . .	147
5.10	One $S$ -component of the morphism $\gamma : D \hookrightarrow G$ . . . . .	148
5.11	Pullback of coprojections of a coproduct along an $\mathcal{M}$ -morphism . . . . .	150

5.12	Square of partial morphisms . . . . .	156
5.13	Commuting square of partial morphisms . . . . .	156
5.14	Mapping a morphism between triple graphs to one between partial triple graphs . . . . .	158
5.15	Relevant snippet of the rule application $J(G) \Rightarrow_{J(p),J(m)} H'$	159
5.16	Showing the morphism $l : K \hookrightarrow L$ from <i>CollapseHierarchy</i> to be $S$ -cartesian . . . . .	161
5.17	The morphism $l : K \hookrightarrow L$ from <i>CollapseHierarchySrc</i> is not $S$ -cartesian . . . . .	161
5.18	Characterizing rule and match from <b>PTrGTG</b> ( <b>APTrGATG</b> )	163
5.19	Source rule $p^S$ of a rule $p$ . . . . .	166
5.20	Forward rule $p^F$ of a rule $p$ . . . . .	166
5.21	Original rule $p$ as concurrent rule of $p^S$ and $p^F$ . . . . .	169
5.22	Example for the necessity of (implicit) deletions of corre- spondence elements . . . . .	171
6.1	Schematic structure of synchronization process . . . . .	180
6.2	Negative constraint forbidding any two <b>Classes</b> to have the same <b>name</b> . . . . .	188
6.3	Rule <i>CreateLeaf</i> with constraint integrated as NAC . . . . .	188
6.4	Derived source-delta (where $\iota_D^{\tilde{S}} := \iota_G^{\tilde{S}} \circ del^{\tilde{S}}$ and $\sigma_{G'} :=$ $add^S \circ \sigma_D$ ) . . . . .	191
6.5	TGG rule as concurrent rule of forward rule and adapted source rule . . . . .	195
6.6	Rule <i>CreateLeafFwd</i> with constraint integrated as NAC . . . . .	196
6.7	Construction of a relaxed revocation rule from the inverse of a forward rule $r^F$ . . . . .	197
6.8	Examples for (relaxed) revocation rules derived from <i>Cre- ateLeaf</i> . . . . .	199
6.9	Construction of a relaxed repair rule from a given repair rule	201
6.10	Examples for (relaxed) repair rules . . . . .	204
6.11	Examples for consistency patterns . . . . .	207
6.12	Obtaining a transformation sequence from a partial prece- dence graph $PG_G$ . . . . .	216
6.13	Computation of the delta-induced partial cover . . . . .	219
6.14	Legal match for a relaxed repair rule . . . . .	225
6.15	Original triple graph . . . . .	237

6.16	Precedence graph for the triple graph from Fig. 6.15 . . . .	238
6.17	Source edit of the original triple graph . . . . .	239
6.18	Induced partial cover after the edit . . . . .	240
6.19	Partial triple graph after first repair step . . . . .	241
6.20	Forward-induced partial cover of first repair step . . . . .	241
6.21	Partial triple graph after second repair step . . . . .	242
6.22	Repair-induced partial cover after second repair step . . . .	242
6.23	Triple graph after third repair step . . . . .	243
6.24	Repair-induced partial cover after third repair step . . . .	244
6.25	Triple graph after fourth repair step . . . . .	245
6.26	Forward-induced partial cover after fourth repair step . . .	245
6.27	Triple graph after fifth repair step . . . . .	246
6.28	Repair-induced partial cover after fifth repair step . . . .	247
6.29	Resulting triple graph after termination of algorithm . . . .	248
6.30	Repair-induced partial cover after sixth repair step. The partial cover is even a precedence graph for the triple graph $H$ from Fig. 6.29. . . . .	249
6.31	The repair rule <i>MovePackageFwd</i> . . . . .	249
6.32	Example for a repair rule that implements undesired reuse . . .	259
A.1	The algebra homomorphism $q_A^X : T(Y + Z)^X /_{\equiv} \rightarrow A_{H^X}$ . . .	313
B.1	Proving the interaction of initial pushouts with pullbacks . . .	316
B.2	Obtaining the morphism $k'$ as pushout along $k$ . . . . .	317
B.3	Obtaining $l'' \in \mathcal{M}$ by $\mathcal{M}$ -effective unions . . . . .	319
B.4	Proving (1) (the outer square from Fig. B.3) to be a pullback .	319
B.5	Characterizing generalized concurrent rules . . . . .	320
B.6	Applying $\mathcal{M}$ pullback-pushout decomposition . . . . .	321
B.7	Showing the front faces to be pullbacks . . . . .	321
B.8	Providing a sufficient criterion for Construction 4.7 to result in a generalized concurrent rule . . . . .	322
B.9	Obtaining the factorization of the initial pushout over $k'$ through the one over $l'_1$ . . . . .	323
B.10	Computing the span $C'_{k'} \xleftarrow{x'_{k'}} B'_{k'} \xrightarrow{b'_{k'}} K_{\cap}$ needed to com- pute $V$ . . . . .	324
B.11	Cube formed by the top squares from Fig. B.10 . . . . .	325

B.12	Computing the common kernel $k$ . . . . .	326
B.13	Ensuring $k$ to compute $\rho_1 *_{E,k} \rho_2$ . . . . .	326
B.14	Applicability of concurrent rule at restricted match . . . . .	328
B.15	Obtaining the morphism $b_m^{**} : B_m \hookrightarrow K$ . . . . .	329
B.16	Morphism from the initial object $\Delta I$ in $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ . . . . .	331
B.17	Induced coprojections . . . . .	331
B.18	Induced coprojections constitute pullback squares . . . . .	332
B.19	Induced mediating morphism constitutes a pullback square . . . . .	333
B.20	Functoriality and preservation of $\mathcal{E}\text{-}\mathcal{M}'$ factorization systems . . . . .	334
B.21	Illustrating the construction and properties of the limit $C$ (for $ I  = 3$ ) . . . . .	335
B.22	Showing the typing of pushout and pullback objects and the mediating morphisms to be typed morphisms . . . . .	340
B.23	Showing the forward rule to stem from <b>APT<sub>r</sub>G<sub>ATG</sub></b> . . . . .	342
B.24	Partial morphism from $L$ to $L^F$ . . . . .	342
B.25	Global view of the morphism $g : D \hookrightarrow G$ . . . . .	343
B.26	A component of the morphism $g : D \hookrightarrow G$ . . . . .	343
B.27	Proving $m(l(K)) \subseteq D$ . . . . .	346
B.28	Preservation of the validity of NACs . . . . .	349
B.29	Proving revocation rules to belong to <b>APT<sub>r</sub>G<sub>ATG</sub></b> . . . . .	350
B.30	Embedding the LHS of a forward rule into the LHS of a forward rule of a derived short-cut rule . . . . .	352
B.31	Proving $a'$ to be a morphism in <b>APT<sub>r</sub>G<sub>ATG</sub></b> . . . . .	352
B.32	Proving a one-element partial precedence graph $PG_G$ to correspond to a one-step transformation . . . . .	354
B.33	Obtaining a transformation sequence from a partial prece- dence graph . . . . .	355
B.34	Switching independent transformations of monotonic rules . . . . .	357

# List of Tables

2.1 An overview of TGG-based incremental synchronization approaches . . . . .	28
---	----



## List of Algorithms

6.1	Structure of our synchronization process . . . . .	233
6.2	Synchronization process—termination criteria . . . . .	234
6.3	Synchronization process—function TRANSLATE . . . . .	235
6.4	Synchronization process—function REPAIR . . . . .	236
6.5	Synchronization process—function REVOKE . . . . .	236



# 1 Introduction

The central contribution of this thesis is the advancement of an established approach to the problem of model synchronization.

## 1.1 Motivation and Problem Statement

Across various areas of computer science, the issue arises that the very same information is represented in different ways in a variety of artifacts. Moreover—especially when considering modern distributed software development processes—these artifacts might be manipulated concurrently by several programmers and stakeholders. Keeping the shared information consistent when one (or several) of the artifacts have been changed is a substantial challenge. We provide two concrete examples where this problem occurs (compare [48] for a still valuable overview): In the area of databases, it is important to be able to extract *updatable views* from a database, which a user can employ to manipulate the stored data. The performed changes then have to be propagated to the database again. In model-driven (software) engineering (MDE), where models are the central artifacts of the development process, an activity diagram might represent the dynamic behavior and information flow of a system and a class diagram its static architecture. Shared information needs to be kept consistent between both models. The same, of course, applies to the code these diagrams formalize (or that is even derived from them). Specifically addressing MDE, two recent review articles even identified “[m]odel synchronization and propagation of changes across models edited within the collaborating team” [75, p. 20, and similarly 36] as one of the key challenges of the entire research field.<sup>1</sup>

---

<sup>1</sup>Quite generally, different kinds of artifacts—including code and other textual ones—can be considered to be *models*; hence the term *model synchronization*. Thus, when speaking of model synchronization in this thesis, by no means this means that we exclusively address MDE.

Quite naturally, one can develop a solution for every synchronization problem using some general-purpose programming language. A more principled approach, however, promises various advantages. For this, researchers from different areas of computer science investigate *bi-* (bx) or *multidirectional transformations* (mx) that address the challenges of model synchronization on a more fundamental level (see, e.g., [48, 113, 83, 44] for overviews). Distinct approaches differ in the ways in which consistency is defined and in the means they employ to restore consistency. They address different issues and have their individual strengths and weaknesses. Recurrent motives, however, which are also central to this thesis, are that a formalized, principled approach provides formal guarantees and the possibility of automation.

In this thesis, we address the *basic model synchronization problem*. This is the problem of restoring consistency after one of two interrelated models has been changed.<sup>2</sup> While this is one of the more mundane problems bx are intended to solve,<sup>3</sup> existing solutions can still be improved. One of the eminent shortcomings of many approaches is that they are not truly *incremental* in the sense that they can unnecessarily lose information in the process of consistency restoration. If approaches are able to avoid this information loss, they are often applicable in rather special situations only or it is in the responsibility of the user to specify consistency and/or even the restoration process in such a way that the outcome is correct and information loss is avoided. For this to work, however, the user of such an approach needs to be an expert for both domains between which consistency is to be restored as well as for the employed bx-paradigm. Even then, there are no (formal) guarantees that synchronization works as desired.

Summarizing, there is a lack of solutions to the basic model synchronization problem that meet the following requirements. The solution

---

<sup>2</sup>That terminology is due to [54]. Terminology is not completely unified in bx research, though. What we call the *basic model synchronization problem* has also been called *standard model synchronization* [183], *sequential model synchronization* [186], or *directed synchronization* [6], for example.

<sup>3</sup>For both transitions, namely from the situation where only one model has been changed to the one where *concurrent* editing takes place and from pairs of interrelated models to networks of such, distinct new problems emerge. One cannot always come to grips with these by merely employing solutions that have been engineered to address the basic model synchronization problem [183, 201]. Still, mature such solutions provide a groundwork that one can expand to be able to cope with the more general problems.

- should be widely applicable in the sense that it does not seriously restrict the kind of artifacts between which consistency is to be restored;
- should offer formal guarantees, in particular, the correctness of computed solutions;
- should avoid unnecessary loss of information;
- should provide the just mentioned qualities without placing the burden that these are actually met on the user. Concrete model synchronization processes that are correct and preserve information should instead automatically be constructed from data that a user can easier specify.

## 1.2 General Approach and Technical Problems

*Triple graph grammars* (TGGs) [195] are an established bx-paradigm that is marked by the following features. Being based on graphs makes TGGs rather universal. Many data structures allow a formalization as a graph such that the kind of artifacts for which TGGs can be used to develop synchronization processes are barely restricted.<sup>4</sup> Furthermore, the rich and mature theory of (*algebraic*) *graph transformation* [53, 54] applies to triple graphs. Thus, there exists a framework to conveniently reason about formal properties of TGG-based synchronization processes. Furthermore, that formal framework facilitates powerful methods for *static analyses*, which can be used to ensure that a particular grammar satisfies preconditions that guarantee desirable properties of the grammar [10, 54]. With regard to usability, being based on graphs TGGs can be equipped with an intuitive visual syntax. More importantly, however, one can develop model synchronization processes abstractly on the level of TGGs, i.e., independently of an individual grammar. These processes then can be applied to concrete instances. The operations needed for these processes can (automatically) be

---

<sup>4</sup>This is a statement about the extent of supported data structures and *not* about the kinds of consistency relations that can be expressed via TGGs. In this respect, TGGs trade expressiveness for decidability: TGGs are not Turing-complete; the membership problem of the language that a TGG defines is decidable (as, for example, mentioned implicitly in [196]).

derived from the grammar of interest. Thus, instead of working out a correct and incremental model synchronization process, a user merely needs to define a grammar that captures the intended relation between the domains of interest. While this still might be non-trivial [8], it usually will be a task of significantly reduced complexity. Moreover, the operations derivable from that grammar can serve to address a whole range of problems in the area of consistency management [215]. Besides the basic model synchronization problem, for instance one can use these operations to generate consistent models, check pairs of models for consistency, or address the concurrent model synchronization problem (where both models have been changed). Thus, the specification of a single artifact, the grammar, suffices for various purposes. Finally, (practical) comparison with other bx-formalisms and -tools shows that TGG-based model synchronization offers high formal guarantees while still being expressive, concise, and implementable in a way that scales reasonably well [7, 6]. For these reasons, TGGs constitute a promising starting point for developing a model synchronization approach that meets the requirements stated at the end of the last section.

A TGG consists of a set of rules that declaratively define how consistent models (formalized as graphs)<sup>5</sup> co-evolve. Applications of the rules simultaneously create three graphs, called *source*, *correspondence*, and *target graph*, respectively. Source and target graph constitute the models of interest, i.e., these are the models that share information and between which consistency has to be restored when one of them has been edited.<sup>6</sup> The correspondence graph contains explicit traceability information; formally, it is connected to both the source and the target graph via a *correspondence morphism*. As already indicated, the given TGG defines consistency, that is, a pair of a source and a target graph is consistent if there is a triple graph with these

---

<sup>5</sup>We therefore use “model” and “graph” interchangeably throughout this thesis. In formal parts, though, we will stick to the formal terminology of graph theory.

<sup>6</sup>Triple graph grammars are symmetric by nature and the distinction between source and target is a terminological one. In this thesis, we are concerned with the case where only the source or the target graph has been edited (and not both of them concurrently). For our approach, however, it does not make any difference which one of them has been changed. For purely economical reasons, we present our whole approach in the *forward direction*, assuming that the source graph has been edited and changes have to be propagated to the target (and correspondence) graph. Everything we present, by symmetry, also works in the *backward direction* in the case where the target graph has been edited.

graphs as source and target that can be generated using the rules of the grammar. To increase expressiveness and usability, the considered triple graphs are usually *typed* and *attributed* [54]. This ensures that it is possible to assign different roles to the elements of a triple graph (via typing) and to store data in it (via attribution). Further extensions to TGGs have been suggested, like additional attribute constraints [13, 145], additional constraints on the models or application conditions in the rules [196, 87, 107, 54], or amalgamated rules [155], which provide a for-each like application semantic for rules.

Once committed to TGGs, the central issue therefore, when aiming at a solution to the problem stated at the end of Sect. 1.1, is to prevent information loss. In existing TGG-based approaches to model synchronization, information loss especially occurs when a user edits the source graph in such a way that an existing element is put into a new context, i.e., in situations where (a part of) the graph needs to be parsed in a way that differs from the way in which it could be parsed before the edit. Instead of incrementally adapting the correspondence and target graph to the new situation, most existing approaches delete the concerned parts of the correspondence and target graph and restore consistency by re-translating (the respective part of) the source graph from scratch [86, 149, 151, 150]; in [104], the amount of information loss depends on the given grammar. Approaches that avoid that kind of information loss are situation specific [31] or are not adequately formalized [84, 85]. The only formal TGG-based approach that avoids information loss [185] is not implemented and not completely specified (it basically treats the resolution of indeterminism as a black box). One reason for why the problem of information loss is not adequately addressed in TGG-based approaches, despite the practical importance of the issue, might be that a foundational understanding of edits that put elements into new context is missing.

A second technical problem is that formally, the objects that are manipulated during such a model synchronization process might not constitute actual triple graphs any longer. Strictly speaking, the correspondence morphisms can be partial. The theory of algebraic graph rewriting (or, more generally, double-pushout rewriting of objects) is applicable to certain kinds of “structured objects”, i.e., objects that are composed of more basic objects that are connected via morphisms [141, 53]. This, for instance, covers triple graphs. However, this theory has not been extended to the

case where the basic objects of such structured objects might be connected via partial morphisms. But this is necessary to cover partial triple graphs.

Summarizing, solving the following two technical problems means to provide formal foundations that one should be able to use to develop TGG-based, information-preserving model synchronization processes:

- (1) providing a formal understanding of the re-use of elements in a new context; and
- (2) extending the theory of double-pushout rewriting to further kinds of structured objects.

### 1.3 Contributions

*In this thesis, we develop an approach to the basic model synchronization problem that meets the requirements stated at the end of Sect. 1.1.* As just motivated, we do so on the basis of TGGs. To provide formal foundations for this, we first address the two more technical problems discussed in the last section. We do so in a way that is not custom-tailored to our practical needs. Instead, we work on a much more general level, namely in the setting of so-called  $\mathcal{M}$ -adhesive categories [54]. In this way, we also generally advance the theory of algebraic graph transformation.

Combined, this leads to the following major contributions of this thesis.

- (1) We develop a theory that helps to formally understand edits that put elements into new context. Intuitively, such an edit “revokes” the rule that originally created the element and applies another rule instead to create it. Instead of taking this to be two rule applications, we can equivalently consider the application of their sequentially composed rule. However, this would amount to a deletion and recreation of the concerned element. We therefore extend the standard sequential composition of rules and additionally allow identifying elements that the first rule deletes with elements that the second rule creates and preserving them (instead of deleting and recreating). We introduce these kinds of rules as *generalized concurrent rules* (GCRs).

We develop this construction in a very general context, namely for double-pushout rewriting in  $\mathcal{M}$ -adhesive categories via rules that

can be equipped with nested application conditions and that can be applied at arbitrary matches. As a central result, we prove the *Generalized Concurrency Theorem* (Theorem 4.14) that states that a sequence of two rule applications can equivalently be replaced by the application of a GCR and clarifies under which circumstances the converse holds. We will exemplify how this asymmetry enables our desired application. Despite the fact that not every application of a GCR can be replaced by applications of its underlying rules, in a second important theorem we provide sufficient conditions for applications of GCRs that preserve the language of a given grammar (Theorem 4.16).

We have partially published the construction of and theory about generalized concurrent rules in [137]; an extended version is currently under review [138]. In [78], we introduced *short-cut rules*, a precursor to our GCRs.<sup>7</sup>

- (2) We develop a foundational theory that we can apply to show that rewriting of partial triple graphs is possible in basically the same way as it is for ordinary triple graphs. We do so by introducing what we call *S-cartesian functor categories*. Formally, *S*-cartesian functor categories are certain non-full subcategories of functor categories; we mainly consider the case of *S*-cartesian functor categories over adhesive categories. Our first important result is Theorem 5.8 that states that these kinds of categories are adhesive (in some variant) again. Furthermore, we show that also so-called *additional HLR properties* [54] are preserved by that construction (under reasonable conditions). This means, the whole elaborate theory of algebraic graph transformation is applicable in *S*-cartesian functor categories. We then discuss how to instantiate (typed attributed) partial triple graphs as an *S*-cartesian functor category. This contribution has been published in [133, 134].
- (3) We provide a fully formalized solution to the basic model synchronization problem that is based on TGGs and meets the requirements stated at the end of Sect. 1.1. We explicitly allow the attribution of

---

<sup>7</sup>In the introduction of the individual chapters of this thesis, we explain in more detail how these chapters relate to our publications.

the triple graphs and support *negative constraints* that increase the expressivity of the grammar by excluding triple graphs from the language that contain forbidden patterns. These are features that several (but not all) TGG-based synchronization processes do not support (at least not explicitly). Given technical assumptions that are the same as [150] or even weaker than [104, 54] assumptions in previous TGG-based approaches, we can prove our algorithm to terminate with correct output for reasonable input (Proposition 6.16 and Theorems 6.17 and 6.18). Compared to several former approaches [86, 149, 151, 150, 104], we are able to preserve more elements (i.e., to reduce information loss), and we are able to provide conditions—which are strict but reasonable enough to be expected to regularly be met in practice—under which the runtime of the algorithm only depends on the size of the edit and not on the size of the whole triple graph (Proposition 6.19).

Generally, our model synchronization process is based on the one by Lelebici (et al.) [150, 151] but extends it by the use of *repair rules*. This results in the substantial increase of incrementality. We obtain the repair rules from generalized concurrent rules that are computed from rules of the given grammar. Such a GCR would capture a (language-preserving) edit of the whole triple graph. For model synchronization, however, we assume that a user has edited the source graph and that we need to propagate this edit to the target graph. Our repair rules are therefore acquired from such GCRs by adapting these in such a way that we assume that their action on the source graph already has happened and we only have to retain the action on correspondence and target graph.<sup>8</sup> Intuitively, repair rules allow us to simultaneously parse the source graph in a new way (compatible with the edit that has happened) and change correspondence and target graph in a way that is consistent with the new way of parsing the source graph.

To direct our synchronization process, and to prove its correctness, we also develop a theory of *markings*, *precedence graphs*, and *partial*

---

<sup>8</sup>Technically, this means to extend the standard operationalization of the rules of a TGG to so-called *forward rules* [195] to the case of general rules, i.e., to rules that are also allowed to delete structure.

*covers*. A precedence graph is basically a parse graph for a triple graph, i.e., it holds information on how the triple graph can be constructed by the rules of the given grammar. The precedence graph consists of *markings* that correspond to the individual rule applications; in particular, a marking signifies by which rule its underlying elements have been created. The precedence graph also contains information about the dependencies between the markings. When a user edits the source graph, usually some of the markings will be rendered *invalid* (by deleted or added elements), implying that their underlying elements (if they are not all deleted anyhow) cannot any longer be created in the way (i.e., via the same rule application) they had been created in the original triple graph. This signifies exactly the situations in which several previous TGG-based approaches resort to the deletion of the target elements underlying the now invalid marking and the re-translation of the source elements. In contrast, we will use a repair rule that revokes the application that corresponds to the now invalid marking and instead applies a rule that fits the new context. In this way, partial covers allow us to give a precise and formal meaning to the intuition that the application of a repair rule replaces a former rule application (that was used to build the original triple graph) by another one (that fits to how the edit changed the source graph of the triple graph).

The model synchronization process we develop here, extends the one we published in [79, 80].

As can be seen from the publications, the model synchronization process presented in this thesis has been developed in close cooperation with the *Real-Time Systems Lab* (FG Echtzeitsysteme) at the Technical University Darmstadt, and, in particular, with Lars Fritsche. Based on the model synchronization process suggested by Erhan Leblebici [150], we jointly developed the principal ideas behind our information-preserving extension of that process. While Fritsche then focused on implementation, evaluation, application, and extension of the process (e.g., to also address the problem of *concurrent model synchronization*, where both source and target graph have been altered [77])—leading to the PhD thesis [76]—, I formalized the process and, based on that, proved its properties. With regard to formalization, short-cut rules and the first half of the Short-Cut Theorem

(stating that the sequential application of the two underlying rules can equivalently be replaced by the application of a short-cut rule [78, Synthesis case of Theorem 7]) are originally due to Lars Fritsche and Andy Schürr. In this thesis they are extended to generalized concurrent rules. I stated and proved all results in this thesis. The running example of this thesis is originally due to Erhan Leblebici and Lars Fritsche but I somewhat extended and adapted it to fit the needs of this thesis.

During writing this thesis, I contributed to research beyond its scope. Thematically, this further research is concerned with logic on graphs (or models) and the preservation or restoration of validity of constraints [178, 174, 175, 132, 177, 176, 135, 136] and with the static analysis of graph transformation systems [143, 146].

## 1.4 Structure

In Chapter 2, we convey an impression of how our approach to model synchronization works and relates to already existing ones. For this, we provide a short literature survey, mostly focusing on TGGs (Sect. 2.1), and, using an example, we illustrate the key ideas of our approach (Sect. 2.2). The example will serve as running example for this thesis. Chapter 3 recollects the double-pushout approach to the rewriting of objects and further central preliminaries. Throughout this thesis, however, we generally assume knowledge of basic category theory; standard references are [147, 2, 17]. An introduction to category theory that specifically addresses the needs for double-pushout rewriting can be found in [53, Appendix A]. Then the three central chapters follow; each corresponds to one of the three contributions discussed above. Chapter 4 presents *generalized concurrent rules* (GCRs), our new variant of sequentially composed rules. In Chapter 5, we work out the theory of  $S$ -cartesian functor categories and use these to introduce (typed attributed) partial triple graphs. Finally, Chapter 6 presents our actual model synchronization algorithm. After these central contributions, we summarize our work and suggest directions for further research in Chapter 7. Some particularly technical proofs are relegated to Appendix B. Related works are specifically discussed in each of the central chapters.

## Acknowledgments

Finishing a thesis is always a time to thank the people who accompanied one through the process.

First, I want to thank my supervisor Gabi Taentzer. Thank you very much for your trust, the great freedom I had, and the opportunities to participate in research beyond the contents of my thesis. I am also grateful that Fernando Orejas was willing to serve as a second examiner for this thesis and agreed to a rather strict schedule for the examination. Furthermore, I am much obliged to Reiko Heckel, who provided a third report. In obtaining the results of this thesis, I immensely profited from the fruitful collaboration and discussions with Lars Fritsche, Andy Schürr, and Gabi Taentzer. The presentation of the obtained results greatly improved through your critical remarks while working on our joint papers. Thank you very much!

Financially, my work on this thesis has been supported by the *German Research Foundation* in the context of the project *Triple Graph Grammars (TGG) 2.0: Reliable and Scalable Model Integration*.

I have been in the lucky situation to work in a pleasant environment. Many thanks to my colleagues, my office mates, and my co-authors. By and large, I had a very good time working on this thesis and you all contributed to that.

Last but not least, I want to thank my family and, most notably, my wife, Tatjana. Thank you all for your continuous support! Tatjana, thank you for enduring my repeated absent-mindedness and for always caring for our children, especially when deadlines were approaching or when I visited conferences!



## 2 Bidirectional Transformations—State of the Art and our Approach

In this chapter, we first present the state of the art for the broad topic of bidirectional transformations and model synchronization. Subsequently, we illustrate our own TGG-based information-preserving approach using a simple example.

### 2.1 A Survey on Bidirectional Transformations and Model Synchronization

In this section, we present an overview of research on model synchronization and bidirectional transformations (bx) more generally. This is a vast area and no structured literature review or other comprehensive survey exists. We do not intend do close this gap here. Instead, we aim to convey an impression of the different existing approaches and (recent) topics that have been investigated. It should become clear how TGGs relate to other approaches and why they are interesting. In a second part, we more closely review different TGG-based approaches to the basic model synchronization problem because these approaches are the ones that are most closely related to the work in this thesis. In the whole section, our focus is on approaches and not on tooling (as that is also the focus of this thesis). This section (considerably) extends [80, Sect. 9].

We start by mentioning existing review and introductory articles. Early overviews include [15, 198, 48, 113]. They discuss how bx is used in different areas of computer science, what different properties bx can have, typical problems bx shall solve, what approaches exist to solve these problems, or implemented tools. Naturally, the presentation is outdated in some places by now. Hidaka et al. develop a feature-based classification for

bx [106]. The focus is rather technical (than conceptual) and only a very specific kind of TGGs (bijective ones) are considered. Diskin et al. suggest a taxonomy for bx that in particular allows matching use cases to bx with appropriate properties [49]. The most recent article we are aware of that relates different bx approaches is a practical comparison of several existing tools [6]. Interestingly for our purposes, the TGG-tool *eMoflon* considered in that comparison scales reasonably well in most scenarios, is reasonably expressive (i.e., passes most of the test cases), and allows for a concise specification of the synchronization scenario. Finally, Abou-Saleh et al. provide an introductory overview over several bx-approaches in [1]. The approaches they distinguish are (i) relational, (ii) lenses, (iii) ordered, delta-based, categorical, and (iv) TGGs. We orientate ourselves by that categorization; however, we merge their categories (ii) and (iii).

Two important dimensions that can serve to classify bx-approaches are how consistency is defined and what means are employed to restore consistency. As in [1], we first structure our presentation according to how consistency is defined and discuss relational approaches and lenses. But subsequently, we additionally discuss approaches that do not quite fit the categorization or that are more adequately presented in terms of how they restore consistency. Finally, we focus on TGG-based approaches in more detail.

### 2.1.1 Non-TGG-Based Approaches

**The relational approach** Viewed abstractly, every bx-approach is *relational* in the sense that the pairs of models that are considered to be consistent define a relation. Therefore, relations offer an adequate mathematical formalism to compare between different bx-approaches. This point of view has in particular been stressed by Perdita Stevens [199, 200]. She expresses desirable properties of bx in terms of relations and shows how different such properties are interrelated. The QVT-R standard by the *Object Management Group* [169] defines a syntax to express such bidirectional consistency relations. Westfechtel [219] provides a recent and comprehensive review of the state and usability of QVT-R. His approach is to implement a variety of bidirectional transformations with different requirements and of different complexity using QVT-R. One important finding of the study is that it can be very intricate to define a relation in QVT-R in such a way

that the obtained bidirectional transformations behave as desired. Detailed knowledge of QVT-R might be needed. The study also substantiates the (well-known; see, e.g., [199] or [148] for a more recent overview) semantic issues of QVT-R: The definition of its semantics in the standard is ambiguous and incomplete in parts. With regard to tools, Medini QVT<sup>1</sup> seemed to be the only usable tool implementing QVT-R. There exist, however, translations of QVT-R into other formalisms. The newest of which we are aware is a translation into UML-RSDS (which is an executable subset of UML and OCL) [148]. Also translations between QVT-R and TGGs have been suggested [131, 90] (both address older versions of the standard, however).

**The lens approach** An important approach to bidirectional transformations are *lenses*. A (well-behaved) lens is a pair of (partial) functions  $get : C \rightarrow A$  and  $put : A \times C \rightarrow C$  subject to the consistency conditions, also called *round-tripping laws*,

$$put(get(c), c) = c \quad \text{for all } c \in C \quad (\text{GetPut})$$

and

$$get(put(a, c)) = a \quad \text{for all } (a, c) \in A \times C \quad (\text{PutGet})$$

whenever both functions are defined [73].<sup>2</sup> The idea is that the set  $C$  contains *concrete* and the set  $A$  *abstract views*; the concrete views are also called *sources*. The function  $get$  extracts an abstract view from a concrete one, and  $put$  serves to put back a change to a concrete view that has been performed on an abstract one. In particular, a lens can be understood to define a consistency relation by stipulating that each concrete state  $c$  corresponds to its abstract view  $get(c)$ . From the very beginning [73], *compositionality* has been one important topic in research on lenses. Various ways to compose well-behaved lenses, e.g., sequential composition, result in well-behaved lenses again. Thus, simple lenses can serve as building blocks to express more complex bidirectional transformations. In their original formulation, lenses are *state-based* and *asymmetric*: The domains of the  $get$ -

<sup>1</sup><http://projects.ikv.de/qvt>.

<sup>2</sup>With their definition of lenses, Foster et al. state requirements for bx that had already earlier been expressed in more special settings, e.g., for the view-update problem in relational databases [20]. For discussions, see, for example, [73, 118, 119].

and the *put*-functions are just sets of states,<sup>3</sup> but the relations between the different states are not taken into account in that formalism. Asymmetry means that one of the two domains is considered to be an abstraction of the other, i.e., a concrete state  $c$  contains all information that is present in its abstraction  $get(c)$ . Formally this can be recognized from the domains in the definition of the functions that constitute a lens. Both restrictions have sparked research addressing them, leading to ever new kinds of lenses that are defined on more structured or more sophisticated input and were developed for symmetric and asymmetric settings.

*Symmetric lenses* [110] are a (state-based) generalization of lenses to a setting where neither domain is an abstraction of the other, i.e., where states of each domain might contain private information. *Edit lenses* [111] are symmetric and translate between edits, which are formalized via monoids. This means, the lenses do not map between states any longer but between edit sequences of the two domains. An important class of lenses are *delta lenses*, introduced by Diskin et al. in an asymmetric and a symmetric variant [50, 51]. The central motivation for the introduction of these kinds of lenses is the observation that in a purely state-based formalization of the distinction between *model alignment* and *change propagation* gets blurred. When a view  $B = get(A)$  of a state  $A$  is computed and then changed to  $B'$ , to synchronize  $A$  with  $B'$ , i.e., to compute  $put(B', A)$ , one needs to know *how*  $A$  relates to  $B$  (e.g., which elements correspond to each other—model alignment) and how  $B$  was changed to obtain  $B'$  (a delta); this delta then has to be propagated to  $A$ . Diskin et al. [50] therefore suggest to not consider the states as a set but rather as a (connected) category: states are the objects and deltas the morphisms. The functions *get* and *put* are not merely defined on states any longer but also take deltas as input. The round-tripping laws are adapted accordingly. In their strictest variant, *get* becomes a functor. For the symmetric situation, Diskin et al. introduce *sd-lenses* [51]. This is more technical, but the intuition and motivation remain the same.

The lens framework also inspired research on *bidirectional programming*, i.e., programming where for a program an inverse program is automatically derived. With regard to lenses, both defining functions have served as

---

<sup>3</sup>Of course, the individual states have an inner structure; they may be strings, trees, models, . . .

input to derive the second one. First, there have been different approaches (and combinations thereof) to derive a suitable *put*-function from a given *get*-function such that the round-tripping laws are guaranteed to hold [e.g., 164, 210, 211]. Also for the symmetric case of model transformations, an approach to automatically derive a backward transformation from a given forward transformation has been suggested (addressing round-tripping laws that have been adapted to the symmetric setting) [220]. Others have argued for a *putback-based* approach [71, 127]: A *get*-function usually does not uniquely determine a corresponding *put*-function but, via the round-tripping laws, a given *put*-function determines a unique *get*-function. Moreover, in the asymmetric setting of lenses, the actual synchronization takes place via the *put*-function. So it might be desirable that programmers are able control its behavior in detail. The tool BiGUL [128] implements the putback-based approach. Beyond these exact approaches where the user specifies one of the transformations, there also exists work on synthesizing different kinds of lenses from examples [162, 167, 168].

Lenses allow for a rich and elegant mathematical theory. They can be characterized as coalgebras for a certain comonad [e.g., 180] or as algebra for a certain monad [118]. In a series of papers (e.g., [114, 115, 116, 117]), Johnson and Rosebrugh relate different kinds of lenses, different mathematical descriptions of lenses, and the symmetric and asymmetric situation. Most interesting with regard to our work in this thesis is that certain lenses are known to compute *universal updates*; this has been shown in the asymmetric [119] as well as in the symmetric case [116]. Intuitively, this means that the update that the *put*-function computes is the smallest possible one (it factors through all other possible updates). However, the only considered changes are additions of information; a dual theory holds for changes that delete information. But, to the best of our knowledge, there is no theory of such least-change lenses for the case where arbitrary changes have to be propagated. This is the setting in which we work in this thesis. Further work on lenses that shares motivation with our work in this thesis is the work of Wang et al. [213] that develops strategies to obtain incremental put-functions. They consider asymmetric lenses between polymorphically typed tree structures. For polymorphic put-functions, they determine places for edits of views for which it is possible to update source states in an incremental manner and develop methods to also perform these updates. For this they also have to introduce structures that relate elements of a

source state with their corresponding elements in its abstract view, a feature that TGGs automatically offer. While the setting in which we work is more general, the “local” approach of Wang et al. inspires interesting questions also for TGG-based approaches. In this thesis, as usually done in work on TGGs, we will argue “globally” in the sense that we state conditions on the given grammar that guarantee desirable behavior of our synchronization process. In the future, one could additionally determine locations at which and kinds of edits for which synchronization is possible in a correct and incremental manner, even in the absence of “global” guarantees.

It is important to note that lenses offer a generic view on bidirectional transformations. In principle, it is only required that two functions interact in a specified way. Thus, one can define lenses between all kinds of structures (strings, trees, graphs, models, . . .). It is also not determined how the functions that constitute a lens have to be specified (internally). Thus, whenever they interact in the right way, it is possible to understand forward and backward transformations that are obtained from a TGG as a (symmetric) lens. That it is possible to implement lenses via TGGs has in particular been argued for in the case of delta lenses [104, 5]. With *view triple graph grammars* [11] there also exists a notion of TGGs for the asymmetric setting, i.e., for the setting in which one model is an abstraction of the other. Therefore, lenses offer a complementary view of bx rather than an approach that competes with TGGs. With their correspondence morphisms, TGGs provide explicit support to align models from two different domains—a feature that often has to somehow be added in practical applications of lenses [21, 213, 112]. In case of TGGs, from the (declarative) specification of a single artifact—the grammar—support for a whole range of tasks in the area of consistency management can be obtained. This can be of great advantage in scenarios where the designer of the bidirectional transformation solution might not even be a programmer but rather an expert for the involved domains—a situation frequently addressed in MDE. In contrast, especially the putback-based approach to lenses allows for more fine-grained control of the synchronization processes. This can lead to solutions of higher quality. As we will see, it is also possible to increase the amount of control that a user has on TGG-based synchronization processes. But this counters the simplicity of use of TGGs.

**Further approaches** An important approach to model synchronization is to translate a given instance of such a problem into some constraint language and to employ a solver for that language to compute a solution for the model synchronization problem. Such approaches offer high formal guarantees. They are also apt to be extended by side conditions that a computed solution should satisfy. For example, the approach by Macedo and Cunha [158] computes least-change solutions to the basic model synchronization problem, where least-change is defined with regard to two metrics from which the user can choose. JTL [43, 69] and Echo [159, 158] are two tools implementing such an approach. A well-known drawback of these kind of approaches is that they usually do not scale and are therefore only applicable to rather small problem instances [6]. In a recent line of work [153, 150, 216, 214, 217]—that we will discuss more thoroughly in the next section—TGGs have been combined with such a constraint-based approach to address different problems in the area of consistency management. The general idea is to, on the base of the given grammar, translate the concrete problem instance into a (0-1) integer linear programming (ILP) problem, for which a solution is computed using some off-the-shelf solver.

Another approach to `bx` is the development of an (internal) domain-specific language (DSL) that facilitates the definition of consistency relations and/or transformations. Two recent such approaches are `BXtend` [37] and `NMF Synchronization` [109]. `BXtend` offers an internal DSL for `Xtend` and `NMF Synchronization` one for `C#`. The great advantage of such an approach is its flexibility. In the already mentioned recent practical comparison of `bx` implementations [6], the solutions that pass the most of the test cases are the ones that are based on `BXtend` and `NMF Synchronization`. Moreover, in both cases the solutions scale very well. However, in such an approach the quality of the solution, e.g., with regard to correctness and incrementality, largely depends on the user. `NMF Synchronization` [109] offers at least some guarantees: There, the user defines a consistency relation using (intra-model) lenses. The user is responsible to ensure that the round-tripping laws are satisfied. If that is the case, correctness and incrementality of the computed solutions is guaranteed.

### 2.1.2 TGG-Based Approaches

As already stated in the introduction, a *triple graph grammar* (TGG) consists of a set of rules (also called *triple rules*) that declaratively define how pairs of consistent models (a source and a target) co-evolve; a third graph explicitly encodes traceability information, i.e., it encodes which individual elements from source and target are considered to correspond to each other [195]. There exists a vast body of research literature on TGGs, partially summarized in two overview articles [196, 9]. Over the years, several TGG tools have been implemented; [108, 154] offer overviews. To the best of our knowledge, however, eMofflon [152, 215] is the only tool among these that is still actively maintained. TGGs have been applied in various academic and industrial settings; recent applications we are aware of include support for consistency management between SysML models and Event-B code in the context of railway systems engineering [218], the automation of test schedule generation [14], the finding of solutions to the virtual network embedding problem [207], and data integration in graph databases [3].

Given the rules of a TGG, further rules (then also called *operations*) can be derived from them. These can be used for various tasks in the area of consistency management (checking two models for consistency, translating a model to the other domain, basic and concurrent model synchronization, ...). The maybe most basic *operationalization* of a rule of a TGG—already presented in [195]—is its splitting into a *source* and a *forward rule*.<sup>4</sup> The source rule only performs the source actions that are specified by the original rule. In contrast, the forward rule assumes that these source actions have already taken place and performs the remaining action on the correspondence and target graph. The application of a triple rule is equivalent to the application of its source rule followed by an application

---

<sup>4</sup>In general, TGGs are symmetric (as already indicated in the introduction) and the distinction between source and target and forward and backward directions are purely terminological. A triple rule, for instance, can also be split into a *target* and a *backward rule*. For economic reasons, we only present one direction and always assume that the source graph has been changed and that the correspondence and target graphs shall be updated accordingly. We would like to explicitly note that all the approaches we discuss in the following work in both directions even if we present them in terms of propagating changes from source to target.

of its forward rule (when these are matched consistently).<sup>5</sup> This means that (provided a suitable control algorithm) forward rules can be used to *translate* a given source graph into a consistent triple graph from the language that the grammar defines (provided that the source graph stems from the projection of that language onto its source component). This already gives a state-based solution to the basic model synchronization problem: after the source (or target) graph has been edited, one can just newly translate the result to restore consistency; in the literature on TGGs this is usually called a *batch transformation* [195, 196, 9].

**Batch transformations** In particular in the first phase of research on TGGs, these batch transformations have been investigated thoroughly; see, e.g., [52, 196, 63, 105, 102, 126, 103]. Important landmarks of this research that are fundamental to later, more sophisticated approaches to model synchronization (and also to the work in this thesis) are the following: First, core properties that TGG-based model synchronization should satisfy have been established. Those include *correctness* (a model from the source domain is always translated into a model from the target domain that corresponds to it), *completeness* (all models from the source domain for which at least one corresponding target model exists can be translated), and *efficiency* (it should at least be possible to identify practically relevant classes of TGGs for which a translation is possible in polynomial time depending on the size of the model) [195, 196, 103].<sup>6</sup> The expressivity of TGGs has been increased by considering rules that are equipped with *negative application conditions* (NACs) [196, 63, 102, 126, 103]; a NAC

---

<sup>5</sup>The deeper reason for this behavior is that a triple rule is a so-called *concurrent rule* of its source and its forward rule [52].

<sup>6</sup>In papers like [63, 105, 102, 103], Ehrig, Hermann et al. also stress the importance of *functional behavior*, i.e., the property of a translation process to always compute the same result for the same input. While this is of course desirable for some applications, it somewhat contradicts a key idea behind TGGs: These are intentionally introduced to be able to express consistency *relations* between different domains [195]. Hence, only in the very special case of domains that are in bijection it will be possible to construct functional translators in both directions without systematically not being able to produce certain valid results. We therefore hardly discuss functional behavior in this thesis.

specifies forbidden context in whose presence the rule is not applicable.<sup>7</sup> While defining a TGG to consist of rules that are equipped with NACs is straightforward, the operationalization of such rules and the development of synchronization processes in their presence proved to be more challenging. Different methods have been developed to direct a TGG-based translation process [55, 105, 102, 126], including methods that reduce the amount of backtracking that is necessary in such a process [102, 126]. Finally, consequently taking a category-theoretical approach [e.g., 52, 103] allows one to uniformly develop theory for different kinds of graphs. Notably, this justifies using *typed attributed graphs* [53] as graphs in a triple graph, which is vital for practical applications. Moreover, it allows one to apply powerful *static analysis methods* that have been developed for algebraic graph transformation also in the case of TGGs.

**Incremental model synchronization** Obviously, it is not desirable, given a consistent pair of source and target model, to always restore consistency by re-translation after one of the models has been changed. It is inefficient and information that is private to the discarded model is lost. Therefore, different approaches have been developed that are *incremental* in the sense that, instead of discarding the unchanged model, it is at least partially preserved during the synchronization process. Existing approaches differ in the extent to which they are formalized, in the amount of elements they are able to preserve, whether they are implemented or not, and in additional features of TGGs (like NACs or attribute constraints) they support. We do not discuss efficiency because not every approach offers an efficiency analysis and, moreover, explaining and comparing different efficiency results would require to go into technical details to an extent that we want to avoid at this place. The (formal) approaches also differ in the preconditions under which their desirable properties have been proven. We postpone that discussion until Sect. 6.5, however, where we will have developed the necessary terminology. In the following, we offer an overview of TGG-based, incremental approaches to the basic model synchronization problem; a brief overview on this topic (focusing on least-change) is also available in [202].

---

<sup>7</sup>That NACs actually increase the expressivity of TGGs follows from general results on graph grammars [212].

**Precedence-driven incremental approaches** Using the terminology of [150], most of these approaches are *precedence-driven* [86, 84, 149, 4, 185]. This means, in some way or other information about in which order the elements of a consistent triple graph have been created is additionally stored. Whenever one of the graphs is changed and consistency shall be restored, it is checked which elements are (transitively) affected by that change. The unaffected part of the whole triple graph is preserved; for the affected part, correspondence and target graph are discarded (assuming that the source graph has been changed) and the respective part of the source graph is re-translated (using the forward rules of the rules of the grammar). Under favorable circumstances, this greatly reduces the part of the graph that has to be re-translated (compared to re-translation from scratch). However, the great drawback of most existing approaches is that they generally overestimate the parts of the graphs that have to be discarded. Intuitively, thinking in terms of parsing, the addition and deletion of elements to the source graph can cause that certain elements cannot any longer be parsed in the same way in which they had originally been parsed/created. Existing precedence-driven approaches mostly retain that trunk of an original parse tree that is not affected by the changes, also not transitively. This trunk is then extended to a parse tree for a consistent new triple graph by re-translation of the parts of the source graph that have not yet been parsed. But this ignores the possibility that, even if an element has been deleted or needs to be parsed in a new way, it might be possible to parse elements that depend(ed) on it in exactly the same way as before. In contrast, using repair rules, we will try to discard and re-parse locally whenever elements cannot be parsed as before and integrate this (small) change with the original way of parsing their dependent elements.

We shortly present individual approaches: Giese and Wagner [86] save the necessary precedence structure directly in the triple graph, namely in the correspondence graph. For this, they add edges between the correspondence nodes such that each correspondence node references all correspondence nodes that depend on it (i.e., nodes that have been created via a rule application that matched that correspondence node). This results in a directed acyclic graph (DAG) that encodes the dependencies between the rule applications that created the triple graph. When the source graph is changed and consistency shall be restored, they check if rule applications that are affected by the change are still valid. If that is not

the case (e.g., because an element that had been created by such a rule application has been deleted by the change), the affected rule applications are *revoked*. This means, the correspondence and target elements that this rule application had created are deleted; moreover, it is noted that the source elements that this rule application had created have to be translated again. Importantly, also rule applications that (transitively) depend on a revoked rule application are revoked. Subsequently, a re-translation process starts (using the forward rules of the grammar). The approach is implemented but not formalized ([85] provides a formalization of a batch transformation algorithm that is also developed in [86]) and, consequently, no proofs for interesting properties are given. NACs and additional attribute conditions are supported by the implementation. The approach suffers from the problem of information loss by the potentially unnecessary revocation of rule applications that depend on invalid rule applications. To address this problem, Giese and Hildebrandt [84] extend that approach. Besides minor optimizations, they propose rules that save nodes instead of deleting and re-creating them. In particular, they present a rule that directly propagates the movement of elements, i.e., the redirection of edges between existing elements. Moreover, they suggest to try a reuse of elements before deleting them. But they neither present a general construction for their rules nor formalize the reuse that takes place. Consequently, no proof of correctness is given and it remains unclear with what kind of situations the approach can actually deal.

In a similar vein, Greenyer et al. [91] propose to not delete elements directly but to mark them for deletion and to allow for their reuse in rule applications during synchronization. When the source graph has been changed, they check which of the rule applications that formerly created the triple graph still are valid. Rule applications that are not valid any longer are revoked. Here, this means that their (still existing) underlying source elements are marked as untranslated and their underlying correspondence and target elements are *marked for deletion*. Then, consistency is restored by (re-)translating the yet untranslated source elements using forward rules. In that process, the forward rules are allowed to reuse (and, thus, restore) suitable correspondence and target elements that are marked for deletion (instead of creating them from scratch). Only elements that cannot be reused are deleted at the very end of the synchronization process. As the approaches of Giese et al., this approach is implemented but comes without

any formalization and proof of correctness. The authors mention that they support attribute conditions, NACs are not discussed.

In contrast, the idea of reusing elements in model synchronizations has been rigorously formalized by Orejas and Pino [185]. The re-translation process of the source graph that they suggest is rather similar to the one of Greenyer et al. but formalized by the introduction of *forward translation rules with reuse*. To determine which part of the original triple graph might remain intact and which part of the source graph then has to be re-translated (with reuse of correspondence and target elements that are provisionally marked to be deleted) they use a dependency relation between the elements. However, this analysis might not be fine-grained enough—the triple graph from which the re-translation process starts might still be too large. In that case, they generically refer to backtracking, heuristics, or user interaction to obtain a suitable starting point. The reason for that deficiency is basically that their analysis can detect the affects of deletions that have been performed on the source graph but is not able to detect situations where the addition of elements provokes that previously existing elements have to be parsed in a new way. The algorithm is proven to be correct and incremental (in a technical sense that we introduce in Sect. 6.4.4). Their approach has not been implemented yet; NACs and additional attribute constraints are not discussed.

The approach by Anjorin [4] extends the one of Lauder et al. [149]; we therefore only discuss the one of Anjorin. In that approach, two structures are used to store precedence information: A *translation protocol*, i.e., a parse graph for the original triple graph, is used to determine which parts of the original triple graph can be retained. The correspondence and target elements that had been created by rule applications that are not valid any longer get deleted and the remaining source graph is re-translated. This re-translation process is directed by a *source precedence graph*, which contains information about *all possible* dependencies between source elements. The approach is formalized and implemented; NACs and attribute constraints are supported. By the more refined analysis technique, the approach can guarantee successful synchronization for a larger class of TGGs as, for instance, the approaches by Giese et al. [86, 84]. However, it also suffers from the problem of information loss by the necessity to revoke rule applications that (transitively) depend on invalid ones.

**Further incremental approaches** We present three further approaches that are not precedence-driven. Hermann et al. [104] propose a synchronization algorithm where, after an edit on the source part, first those correspondence elements are deleted that do not refer to an element in the source graph any longer. Thereafter, they parse the remaining triple graph to find a maximal still valid subgraph. This triple graph is used as a starting point to propagate the remaining changes from source to correspondence and target graphs using forward rules. The approach is completely formalized and proven to be correct, also for attributed TGGs and triple rules with NACs. It avoids some unnecessary deletions but there are some that still can occur. In fact, the amount of unnecessary deletion taking place in that approach is dependent on the given TGG rules; a concrete example for that is given in [202]. While that approach is definitely a valuable contribution towards information-preserving synchronization, repeated parsing for maximally consistent sub-models is highly inefficient and might not scale to large models. At least part of that approach is implemented as HENSHINTGG [70].

The approach by Lelebici (et al.) [151, 150] is a *reactive approach*. An *incremental pattern matcher* monitors which of the rule applications that created a given triple graph are still valid and notifies about new matches for forward rules. When, after the source graph has been changed, consistency shall be restored, first all matches that became invalid are revoked (i.e., the correspondence and target elements that the respective rule applications had created are deleted). These revocations might trigger further revocations of dependent rule applications (by invalidating them). In particular, also this approach suffers from exactly the same kind of information loss as most of the already discussed approaches. When all invalid matches have been revoked, the part of the source graph that is not accounted for by the remaining valid matches is re-translated. This is done using the matches for the forward rules that are provided by the incremental pattern matcher. The approach is formalized and implemented. It does not cover triple rules with NACs; support for attribute constraints is not explicitly discussed in the formalization but in the implementation the techniques from [4] are reused.

Weidmann et al. [217] combine a TGG-based specification with an ILP-based solution to the model synchronization problem. Actually, their approach addresses the more general concurrent model synchronization

problem but naturally it can also be used to synchronize models after only one of them has been changed. The basic idea is to understand possible matches for the triple rules (or their derived forward rules) in the modified triple graph as a search-space. Dependencies between rule applications and mutual exclusions are translated into a 0-1 ILP problem and an off-the-shelf solver is employed to compute its maximal solution. The objective function that is given to the solver can be refined by setting weights of certain parts (which is of high importance in their setting where concurrent changes might contradict each other and the weights implicitly control how such conflicts will be resolved). As this approach will consider all possible ways to still parse the given model, information loss is low. However, compared to purely rule-based approaches it suffers from scalability issues. The approach extends previous combinations of TGGs with ILP-solving that addressed other problems in the realm of consistency management [153, 150, 216, 214]. These are formalized and interesting properties (like correctness) have been proven. For the synchronization approach in [217], while formal definitions are given no properties are proven. It should not be too difficult, however, to transfer the previous results.

As can be seen from that discussion, most of the incremental approaches still suffer from the same kind of information loss. Instead of developing another approach to address this issue, in a guideline on how to develop a TGG Anjorin et al. [8] explain how certain kinds of rules in a TGG avoid that loss of information better than others (basically, by reducing certain dependencies between rule applications). Transforming a given TGG into that form, however, may change the defined language and thus, is not always applicable. In some cases that effect can be avoided by, e.g., designing suitable NACs for the rules and proving the equality of the generated model languages. But this is cumbersome and requires expert knowledge on TGGs by the user.

**Summary** Almost all TGG-based, incremental approaches to the basic model synchronization problem either suffer from information loss or are informal; Table 2.1 summarizes our discussion. Exceptions are [104, 217], which suffer from scalability issues, and [185], which is not implemented and (intentionally) not completely specified. Therefore, we suggest a new TGG-based, incremental approach to the basic model synchronization problem.

Table 2.1: An overview of TGG-based incremental synchronization approaches

	Information loss	Formalized	Implemented	Additional features
[86]	high	no	yes	NACs and ACs
[84]	low	no	yes	NACs and ACs
[91]	low	no	yes	ACs
[185]	low	yes	no	no
[149, 4]	high	yes	yes	NACs and ACs
[104]	moderate	yes	yes	NACs
[151, 150]	high	yes	yes	(ACs)
[217]	low	prepared	yes	ACs
ours	low	yes	yes	NACs (and ACs)

For this, we will combine the reactive approach of Leblebici et al. with techniques from the precedence-driven approaches, and the application of repair rules that we newly develop. We also use an incremental pattern matcher to be notified about former rule applications that became invalid by changes. But, as long as not all source elements that they created have been deleted, we first try to *repair* the rule application instead of directly revoking it. Intuitively, such a repair means to locally parse the source graph in a new way and to adapt the correspondence and target graph accordingly. Techniques from the precedence-driven approaches are used to ensure that we apply repair rules in such a way that their application is consistent with the other rule applications, i.e., that we (locally) obtain a parse graph of the triple graph. If repair is possible, we not only preserve elements of the rule application that we did not need to (completely) revoke—dependent rule applications might become valid again after the repair. In this way, we are able to preserve a similar amount of information as the other approaches that avoid information loss [84, 91, 185, 217]. We completely formalize our approach in the setting of typed attributed triple rules with a certain kind of NACs and prove central properties like correctness, completeness, and termination. Our approach is implemented [79, 80] and the implementation also reuses the support for additional attribute conditions that has been developed in [13, 4]. However, we do not formalize this.

**Outlook to concurrent model synchronization** We close this section with mentioning a topic that recently has attracted renewed interest in research on TGGs, namely *concurrent model synchronization*, where both source and target graph have been changed and consistency shall be restored. Naturally, approaches based on other formalisms exist (e.g., [221]), and also a TGG-based solution has been suggested [101]. But only recently, three new TGG-based approaches have been published that all reduce information loss. Besides the approach by Weidmann et al. [217], which we have already discussed above, Orejas et al. extended their approach via repair rules with reuse also to the concurrent case [186]. Moreover, also the approach that we present in this thesis has been generalized to that setting [77]. However, while our approach is implemented and evaluated, its complete formalization (based on the concepts developed in this thesis) remains future work.

## 2.2 Introduction to our Approach by Example

In this section, we illustrate our approach to model synchronization. Using a simple example, we explain the basic concepts as well as all main ingredients for our new synchronization process. In particular, we practically motivate the need for the theoretical results from Chapters 4 and 5. This section is intentionally rather informal; it is based on [80, Sect. 2]. Rigorous definitions of all employed concepts will be presented later (mostly in Sect. 3.4 and 6.2.1).

Graph transformations, and triple graph grammars in particular, are a suitable formal framework to reason about and to implement model transformations and synchronizations [30, 54].<sup>8</sup> A triple graph consists of three graphs, a *source*, a *target*, and a *correspondence graph*. While the source and target graphs belong to (different) modeling domains, the latter encodes which elements of these *correlate to each other*. This is done by mapping each element of the correspondence graph to an element of the source graph as well as to an element of the target graph (formally these are two *graph morphisms*). Elements connected via such a mapping are considered to be correlated. In practical applications, a *type graph* (or

---

<sup>8</sup>Therefore, as already stated in Chapter 1, we will use the terms “graph” and “model” interchangeably in this thesis.

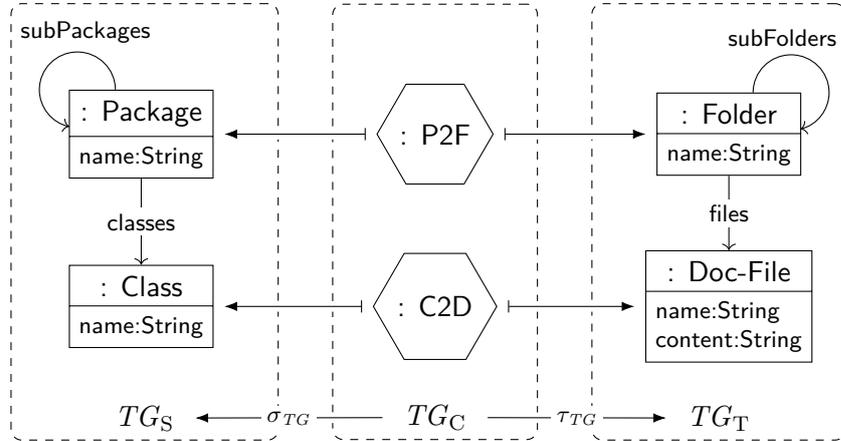


Figure 2.1: Type Graph for the Java AST-to-documentation example

*meta-model*) declares the syntax of instance triple graphs. Nodes may be equipped with additional attributes.

*Triple graph grammars* (TGGs) [195] declaratively define how consistent models co-evolve. This means that a triple graph is considered to be consistent if it can be derived from a start triple (e.g., the empty triple graph) using the rules of the given grammar. Furthermore, the rules can automatically be *operationalized* to obtain new kinds of rules, e.g., for translation and synchronization processes.

We illustrate our model synchronization process by synchronizing a Java AST (abstract syntax tree) and a custom documentation model as example. This example has been introduced by Leblebici et al. [155]; it is slightly modified to demonstrate the core concepts of our approach. We intentionally use a very simple example that yet suffices to illustrate its key ideas and concepts. Our approach has been tested and evaluated with more complex (and, in particular, less symmetric) grammars, including a more complex version of the grammar from this example [79, 80, 77]. We consider a Java AST model as *source* model and its documentation model as *target* model. Changes in a Java AST model have to be transferred to its documentation model and vice versa.

Figure 2.1 depicts the type triple graph that describes the syntax of our example triple graphs. The source side (formally: the graph  $TG_S$ )

specifies hierarchies of **Packages** that may contain **Classes**, whereas the target side (formally: the graph  $TG_T$ ) specifies hierarchies of **Folders** that may contain **Doc-Files**. The correspondence types **P2F** and **C2D** in between and their indicated mappings under the morphisms  $\sigma_{TG} : TG_C \rightarrow TG_S$  and  $\tau_{TG} : TG_C \rightarrow TG_T$ , respectively, declare that **Packages** might correspond to **Folders** and **Classes** to **Doc-Files**. Furthermore, except for the correspondence types, every node type has an attribute **name** of type **String** and **Doc-Files** have an additional attribute **content** of type **String**.

**TGG rules** Figure 2.2 shows the rule set of our example TGG consisting of three rules (assuming an empty start graph). All rules are presented in an *integrated* form: Instead of displaying LHS, RHS, and context as three separate graphs, just one graph is presented where the different roles of the elements are designated using markings (and color). *CreateRoot* creates a root **Package** together with a root **Folder** and a correspondence link in between. That these elements are created is indicated by the green color and the annotation with  $++$ . The names of the newly created nodes are set according to the input parameter **n**. The nodes' "names" **p**, **pf**, and **f** merely serve to identify elements and have no formal counterpart. *CreateSub* selects an existing pair **p1** and **f1** of corresponding **Package** and **Folder** (as indicated by the black color) and creates a new such pair. Finally, *CreateLeaf* creates a corresponding pair of a **Class** and a **Doc-File** under the same precondition as *CreateSub*.

TGG rules can be used to generate triple graphs; triple graphs generated by them are *consistent* by definition. An example is depicted in Fig. 2.3. It can be generated by first applying *CreateRoot* followed by two applications of *CreateSub* and an application of *CreateLeaf*. Starting with the empty triple graph, the first rule application just creates the elements **rootP** and **rootF** and the correspondence element in between. The second rule application matches these elements and creates **subP**, **subF**, **subD**, their respective incoming edges, and the correspondence element between **subP** and **subF**. The other two rule applications are performed similarly.

Such a *derivation sequence* can also be represented as a *precedence graph* where the nodes correspond to rule applications and edges to (*direct*) *dependencies* between them. We call the individual nodes of a precedence graph *markings*. For our purposes it will be enough to consider a single

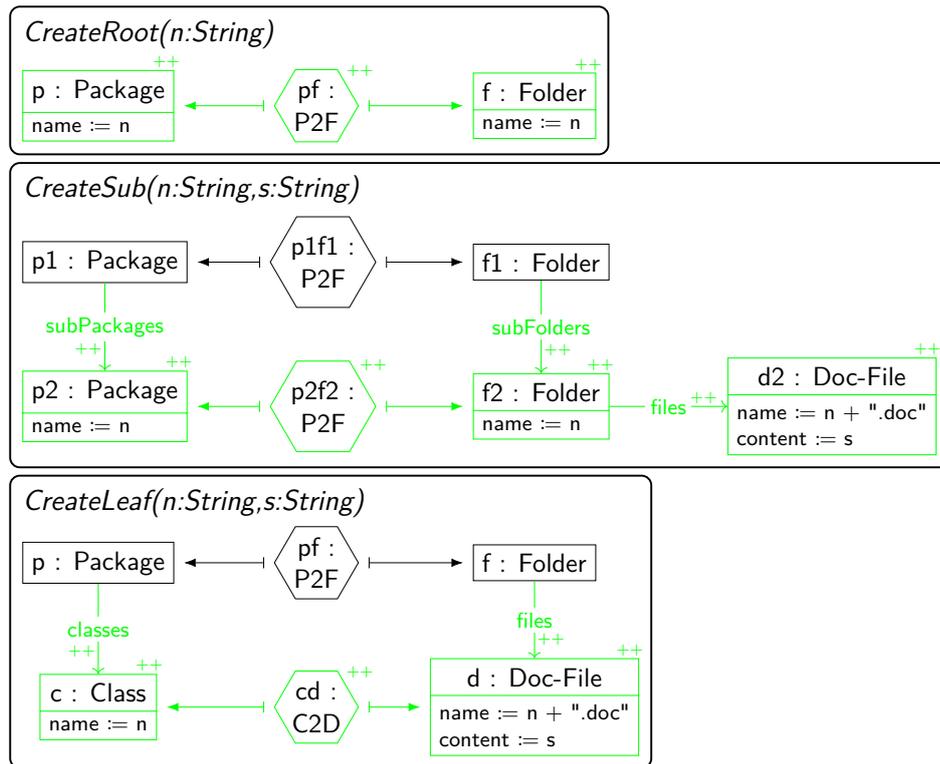


Figure 2.2: TGG rules for the Java-AST-to-documentation example

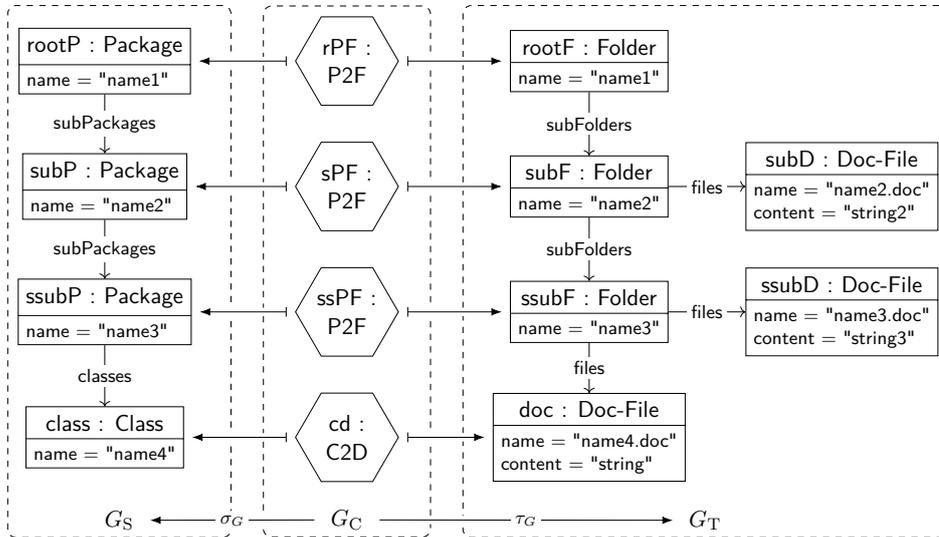


Figure 2.3: Instance of the Java AST-to-documentation example



Figure 2.4: Precedence graph for the example instance shown in Fig. 2.3

and very basic kind of dependency: A rule application depends on another one if the latter creates an element that the former matches. A precedence graph for the example instance shown in Fig. 2.3 is depicted in Fig. 2.4.

**Operationalization of TGG rules** A TGG can also be used for *translating* a model of one domain to a correlated model of a second domain. Moreover, a TGG offers support for *model synchronization*, i.e., for restoring the consistency of a triple graph that has been altered on one side. For these purposes, each TGG rule has to be *operationalized* to two kinds of rules: A *source rule* enables changes of source models (e.g., as performed by a user) while *forward rules* translate such changes to the target model.<sup>9</sup> The result of applying a source rule followed by an application of its corresponding forward rule (consistently matched) yields the same result as applying the

<sup>9</sup>Analogously, *target* and *backward rules* can be derived.

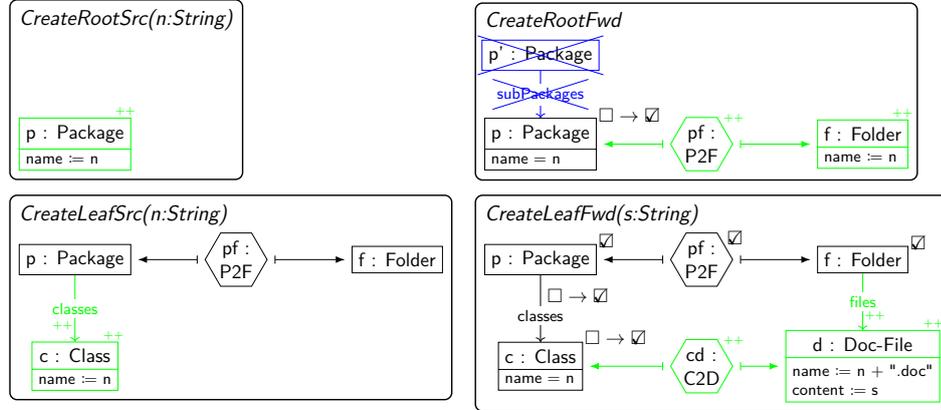


Figure 2.5: Source and forward rules derived from *CreateRoot* and *CreateLeaf*

TGG rule they originate from. Figure 2.5 shows the resulting source and forward rules for the rules *CreateRoot* and *CreateLeaf* from our example TGG.

Source rules only perform the source action of their underlying TGG-rule, whereas forward rules presume that this source action has already taken place and perform the remaining correspondence and target action of the underlying rule. For efficiency reasons, when employing forward rules for translation or synchronization of models, it is convenient to keep track of the elements that already have been translated. For this, in forward rules we annotate elements that need to be matched to already translated elements with  $\checkmark$  and those that are translated by a forward rule application with  $\square \rightarrow \checkmark$ ; newly created elements are considered to get “marked” without explicitly annotating this. Similar approaches to marking have been formalized using subrules [151] or by introducing translation attributes [102]. In contrast to these approaches, our marking concerns the whole triple graph and is not restricted to the source graph. The reason for this is that by initially retaining the original correspondence and target graph (instead of deleting all elements that are potentially affected by the user edit), during our synchronization processes the triple graphs will contain correspondence and target elements that might get deleted later on. Thus, markings on correspondence and target graph signify which elements are “safe” to match (for instance, because they will not get

deleted in the following). In Sect. 6.3.1, we formalize this and explain more details. Furthermore, forward rules may use *negative application conditions* (NACs) [53] which are depicted in blue. Using NACs, we are able to not only define necessary structure that has to be found but also the explicit absence of structural elements as in *CreateRootFwd*, where we forbid  $p$  to be matched to an element which has a parent package. The theory behind these so-called *filter NACs* is formalized by Hermann et al. [102] and they can be derived automatically from the rules of a given TGG when computing its forward rules.

Using these rules, we can translate Java AST to documentation models. Considering the one on the source side of the triple graph in Fig. 2.3, i.e., the graph  $G_S$ , this can be translated to a documentation model such that the result is the complete triple graph depicted in this figure. To obtain this result we apply *CreateRootFwd* at  $rootP$  (i.e., match  $p$  to  $rootP$ ), *CreateSubFwd* to translate Packages  $subP$  and  $ssubP$ , and finally *CreateLeafFwd* at the Class class. Note that *CreateSubFwd*, for example, is applicable when matching Packages  $p1$  and  $p2$  to the Packages  $rootP$  and  $subP$  of the source graph, respectively, since  $rootP$  was marked as translated by the application of *CreateRootFwd*. Without the NAC in *CreateRootFwd*, this rule would also be applicable at the elements  $subP$  and  $ssubP$ . Applying *CreateRootFwd* and translating these elements with it, however, would result in the edges from their parent Packages not being translatable any longer: there is no rule in our TGG rule set that creates edges between packages only. Hence, NACs can direct the translation process to avoid these dead-ends. Filter NACs are derived such that they only prevent rule applications leading to dead-ends.

**Existing approaches to model synchronization** Given a triple graph such as the one in Fig. 2.3, a developer may want to collapse the Package-hierarchy and deletes  $subP$  for that purpose, moving  $ssubP$  to Package  $rootP$  instead. The result is depicted in Fig. 2.6, where we omitted the name-attributes and identifiers and types of the correspondence nodes for brevity. The resulting object does not belong to the language of the given TGG. Formally, it is not even a triple graph any longer. Thus, the user edit has to somehow be propagated to restore consistency.

The approach to model synchronization that has been suggested by

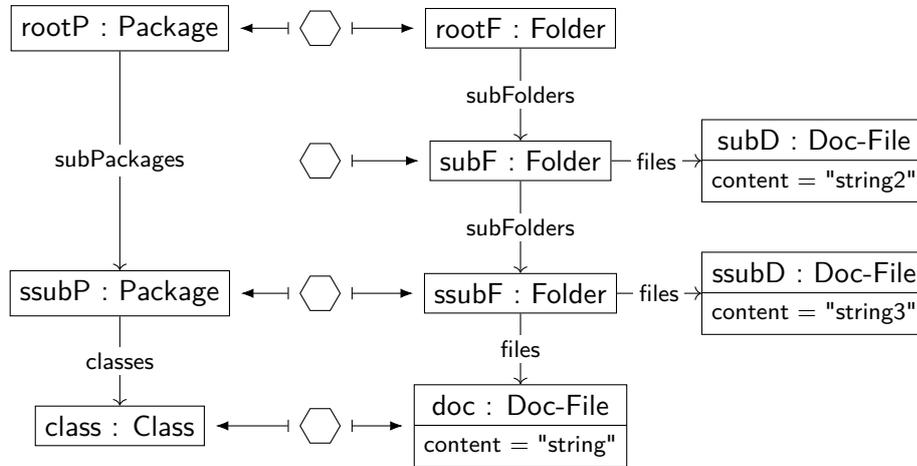


Figure 2.6: Result of user edit applied to instance from Fig. 2.3

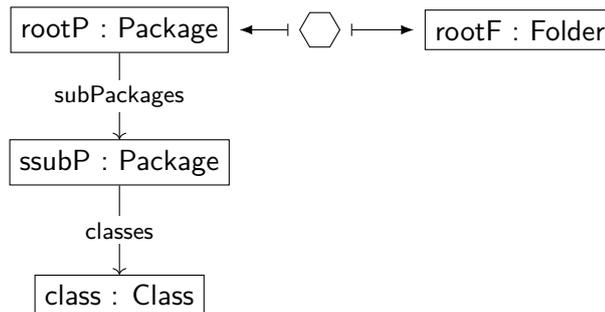


Figure 2.7: Triple graph resulting from revoking the now invalid rule applications in Fig. 2.6

Leblebici (et al.) [151, 150] employs an incremental pattern matcher to first recognize that the second to fourth rule applications that originally created the instance (see Fig. 2.4) are not *valid* any longer. Either elements that the rule application created or elements that the rule application matched are now missing. In that approach, the invalid rule applications are *revoked* on the correspondence and target part. In our example, this results in the triple graph depicted in Fig. 2.7. Given this triple graph, a re-translation process starts, using the forward rules derived from the

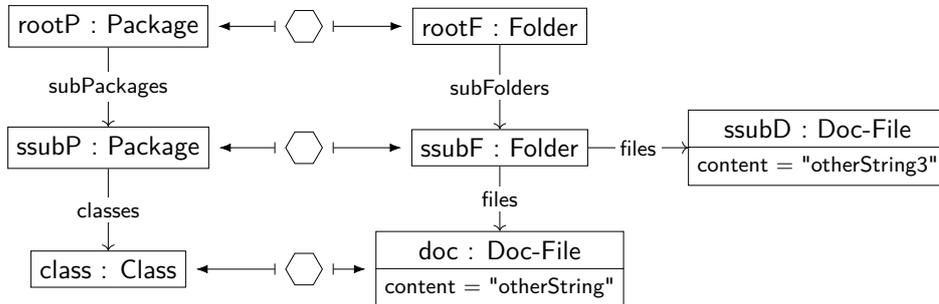


Figure 2.8: Repaired triple graph resulting from re-translating the triple graph from Fig. 2.7

given TGG. This leads to the valid instance depicted in Fig. 2.8. However, the source side does not contain any information about the value of the content-attributes of nodes `ssubD` and `doc` (which are set to `otherString3` and `otherString`, respectively). Thus, normally, these values are lost during the synchronization process and have to be manually restored by the user (if possible at all). Similarly, information that has been abstracted from in the formal representation (like layout information or internal unique identifiers of elements) is lost when deleting and recreating elements. Additionally, the computational effort of this approach can be rather high. In our example, as a whole hierarchy has to be re-translated, the effort of the synchronization process depends on the size of the whole model and not only on the size of the introduced user change. This illustrates exactly the kind of inefficiency and information loss under which many of the approaches we discussed in Sect. 2.1.2 suffer [86, 149, 4, 104, 151, 150].

**Model synchronization with repair rules** Our goal is to extend the model synchronization process of Leblebici (et al.) [151, 150] in such a way that the just discussed problems are avoided, at least for a reasonable class of TGGs. Our central idea is to additionally employ *repair rules* that allow directly propagating user edits like the one from our example. We are doing so in two steps: We first develop a new kind of rule composition, whose results are called *generalized concurrent rules*. These will allow us to derive new rules from a given TGG, which capture typical complex, consistency-preserving edits on a whole triple graph. This will be the topic

of Chapter 4. Secondly, we extend the operationalization of TGG-rules to this new class of rules to obtain repair rules. The intuition behind this is that the source rules correspond to complex user edits on the source model which the corresponding forward rules propagate to the target side.

Whenever an edit remains in the language of a given TGG, the original model can be transformed into the edited one by a series of applications of inverse rules of the TGG followed by applications of TGG rules; in the most extreme case, the inverse applications reduce the original model to the start graph of the grammar, and the other applications create the new model from scratch. However, a typical model edit, even a complex one, can usually be captured by rather short such sequences of rule applications. Moreover, where such a sequence of rule applications deletes and recreates elements, a “free” edit of a model, i.e., an edit that is not restricted to the use of TGG rules and their inverses, can just preserve these elements. We develop *generalized concurrent rules* (GCRs) to capture exactly this behavior. The construction of a GCR is a form of sequential rule composition, where elements that are deleted by the first rule and recreated by the second can be preserved. In our approach to model synchronization, we will consider GCRs that are computed from (concurrent rules of) inverse rules of rules of the given TGG as first input. The second input are (concurrent rules of) rules from the TGG. This accords with the idea of revoking (a series of) rule applications and replacing them by another one, while preserving selected elements.

Figure 2.9 depicts three GCRs that can be derived from our original three TGG rules in that way. *ChangeAttributeRoot* allows simultaneously changing the name-attribute of a correlated pair of `Package` and `Folder`. It is computed from the inverse of *CreateRoot* and *CreateRoot* itself. All elements except for the attribute values (formally: edges that connect the values of the attributes to the according nodes) are preserved. *MakeRoot* deletes the `subPackages`- and `subFolders`-edges between two pairs of correlated `Packages` and `Folders`. Additionally, the now superfluous `Doc-File d2` in `f2` is deleted. Graphically, the deletion is indicated via the red color and the “--” annotation. *MakeRoot* is computed from the inverse rule of *CreateSub* as first input and *CreateRoot* as second. Here, `Package p2`, `Folder f2`, and the correspondence element in between from the inverse of *CreateSub* are identified with `Package p`, `Folder f`, and the correspondence element in between from *CreateRoot* and preserved instead of deleted and recreated.

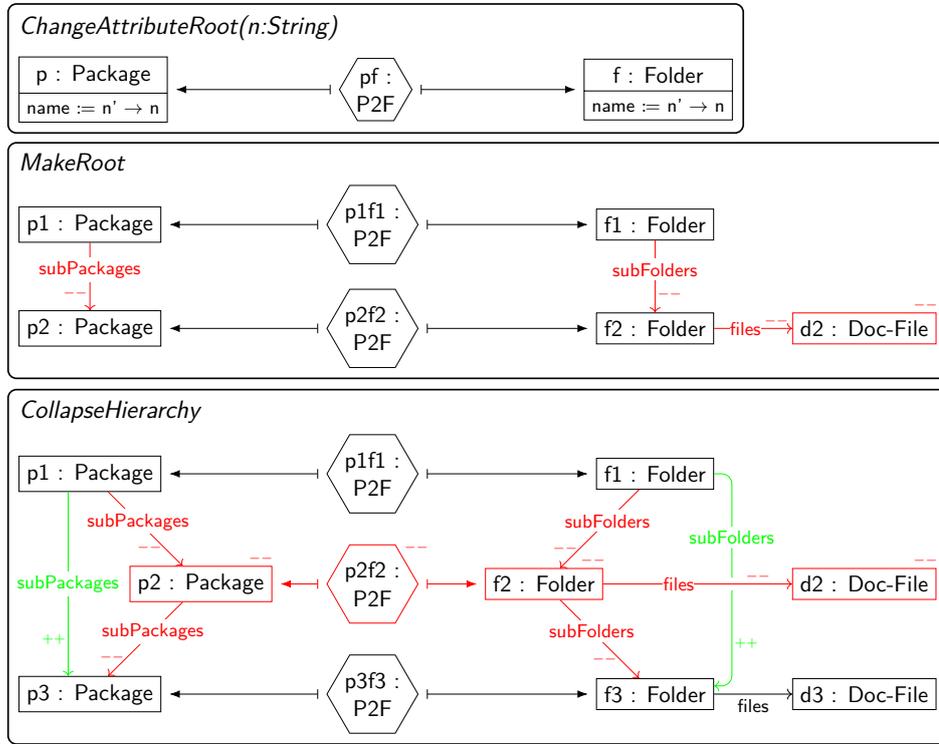


Figure 2.9: Examples for *generalized concurrent* (or *short-cut*) rules that allow for complex model edits

In this way, an application of *MakeRoot* transforms elements that had been created by an application of *CreateSub* into elements that have been created by *CreateRoot*. The last example, *CollapseHierarchy*, is more complex since its first input is a rule that sequentially applies the inverse of *CreateSub* two times. The second input is *CreateSub*, again. The resulting rule serves to collapse a `Package` and `Folder` hierarchy by one level.

In our example above, a user wants to transform the triple graph from Fig. 2.3 to the one depicted in Fig. 2.8. Using *CollapseHierarchy* and matching the `Package` nodes `p1`, `p2`, and `p3` to the packages `rootP`, `subP`, and `ssubP` (and the correspondence nodes and `Folders` accordingly), this transformation is performed with a single rule application. Moreover, the values of the content-attributes are preserved. Thus, GCRs derived from a given TGG

allow for complex, language-preserving user-edits of triple graphs. However, it is also easy to see that not all possible applications of them are language-preserving. The whole of Chapter 4 is devoted to develop a comprehensive theory of GCRs: their construction, their relation to other forms of rule composition, and their formal properties.

**Operationalization of generalized concurrent rules** A GCR transforms both models at once, i.e., (consistently) edits a whole triple graph. To employ GCRs in model synchronization processes, i.e., in the case where a user has edited one of two interrelated models, they also have to be operationalized. Again, the intuition is that the derived *source rule* corresponds to a possible user edit of the source part and the *forward rule* can serve to propagate that edit to the target side. We call forward rules that are derived from GCRs *repair rules* for that reason. Figure 2.10 depicts the three repair rules obtained from the GCRs from Fig. 2.9.

*ChangeAttributeRootFwd* sets the `name`-attribute of the matched `Folder` to the value of its corresponding `Package`, thus propagating an attribute change on the source side. *MakeRootFwd* deletes a `subFolders`-edge between two `Folders` and the `Doc-File` contained in the second one, transforming the second `Folder` `f2` into a root `Folder` as well. The two NACs ensure that the rule can only be applied when matching `Package` `p2` to a root `Package`. They are obtained from the filter NAC of *CreateRootFwd*. The newly employed annotation for source elements,  $\xi \rightarrow \square$ , signifies that a such annotated element shall be matched to an element that had been created via some rule application that has been rendered invalid by the change that the user has performed. Similarly, *CollapseHierarchyFwd* collapses a hierarchy on the correspondence and target part. As they will not be of central interest in this work, we only depict one of the according source rules, namely *CollapseHierarchySrc*—the most complex one—in Fig. 2.11. As to expect, it deletes the `Package` `p2` and the corresponding `subPackages`-edges between the `Packages` `p1` and `p3` and inserts `p3` in `p1`. Moreover, it deletes the reference to the deleted `Package`; formally this will amount to removing the correspondence element `p2f2` from the domain of the source correspondence morphism.

*Repair rules* allow propagating graph changes directly to the other graph to restore consistency. Revisiting our example from Fig. 2.6, it is possible

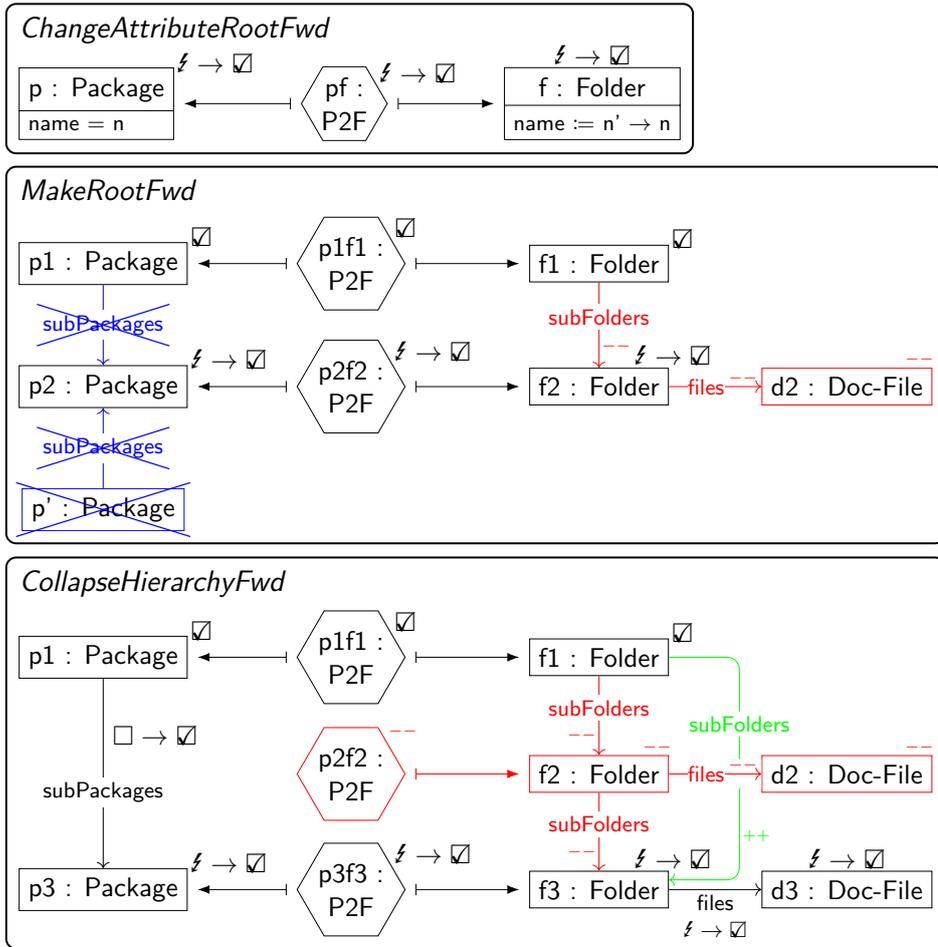
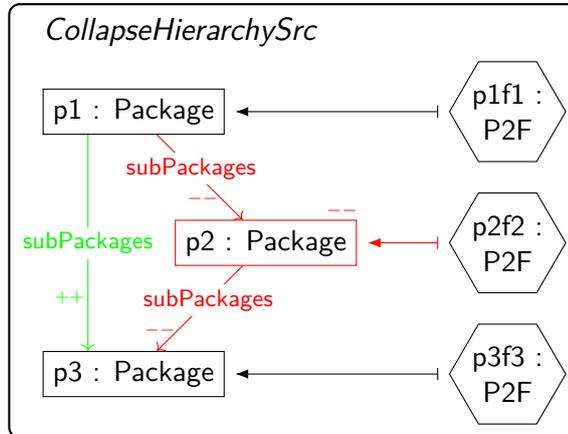


Figure 2.10: Repair rules obtained as operationalized GCRs

Figure 2.11: Source rule derived from *CollapseHierarchy*

to use *CollapseHierarchyFwd* to directly propagate the deletion of `subP` and the `subPackages`-edges and the creation of the new edge. The result is structurally the same as the one depicted in Fig. 2.8. However, the values of the `content`-attributes of Doc-Files `ssubD` and `doc` are not changed in this way. This means, compared to former approaches, this repair does not cause information loss and allows skipping the costly reversion process with the intermediate result in Fig. 2.7.



Figure 2.12: Partial cover for the edited instance shown in Fig. 2.6

To direct the synchronization process and, in particular, to decide when to apply which repair rule, we use a structure we call *partial cover*. Partial covers are kinds of “relaxed” precedence graphs. The original precedence graph from Fig. 2.4 for the example instance shown in Fig. 2.3 is not a precedence graph for the edited instance shown in Fig. 2.6 any longer but merely a partial cover (as depicted in Fig. 2.12). The two nodes `CreateSub1` and `CreateSub2` do not represent rule applications any more: elements the original rule applications created or matched have been deleted by the edit. We call such nodes *invalid*; they are indicated by their red border in

Fig. 2.12. And we cannot be sure whether, during a synchronization process, the rule application `CreateLeaf` can be maintained because it depends on now invalid patterns (indicated by the dashed border). However, invalid nodes in a partial cover signify that their underlying rule applications have to be repaired or revoked. We will therefore try to apply repair rules that have the same rule as an invalid node as their first input—matched in a way that is consistent with the original rule application. In our example, two subsequent applications of *CreateSub* have become invalid and *CollapseHierarchyFwd* (cf. Fig. 2.10) is a repair rule whose first input is exactly such a sequence. Moreover, on the source side, compared to a mere concurrent rule it additionally preserves exactly the `Package p3` that matches the `Package ssubP` that has been preserved by the user edit; all other source-elements that had been created by the rule applications `CreateSub1` and `CreateSub2` have been deleted. This information, extracted from the partial cover, shows that *CollapseHierarchyFwd* is a suitable candidate to try to directly propagate the user edit to the target side. For the rule to actually be applicable, the edit also needs to encompass the addition of the `subPackages`-edge between `rootP` and `ssubP` (which is the case in our example); this information is not represented in the partial cover. Hence, a partial cover informs us which repair rules to try and fixes partial matches for them; we will also use the (partial) order given by a partial cover to determine the order in which we try to repair invalid rule applications. Whether a repair rule suggested by a partial cover is actually applicable then has to be checked.

The second input of the repair rule *CollapseHierarchyFwd* is also the rule *CreateSub* (intuitively, *CollapseHierarchy* replaces a sequence of two applications of *CreateSub* by a single application of this rule). On the level of partial covers (or, precedence graphs), therefore, an application of *CollapseHierarchyFwd* (correctly matched) means that we can remove the sequence of two invalid `CreateSub`-nodes and replace it by a single (valid) one. Generally, an application of a repair rule also provides an *induced partial cover* for its result. For our example, it is depicted in Fig. 2.13. Here, the induced partial cover is even a precedence graph again; it witnesses that its underlying triple graph belongs to the language of the given TGG. We will use the observations how rule applications change a partial cover (and ultimately restore a precedence graph) to argue for the formal properties of our approach.



Figure 2.13: Precedence graph for the repaired triple graph (Fig. 2.8) induced by the repair rule application

Summarizing, the user edit of collapsing the `Package`-hierarchy corresponds to an application of the source rule of the GCR *CollapseHierarchy*, namely *CollapseHierarchySrc*, and the according update to the target side is performed by *CollapseHierarchyFwd*, which is the corresponding repair rule. We use a partial cover to detect that *CollapseHierarchyFwd* is indeed a suitable candidate to propagate the user edit. In Chapter 4, we will present the construction of generalized concurrent rules that serves to obtain our repair rules. Chapter 5 is devoted to develop the context in which our synchronization processes take place: It provides a formal framework that first allows rewriting objects like the one from Fig. 2.6—which is only a *partial triple graph* since the source correspondence morphism is not total any longer. In Sect. 5.3, we then instantiate this general framework to partial triple graphs. Chapter 6 provides the details on our incremental model synchronization process. In particular, we develop the theory of partial covers, which direct that process.

### 3 Adhesive Categories and Two Algebraic Approaches to Rewriting

In this chapter, we present two algebraic approaches to the rewriting of objects. We will mainly focus on the *double-pushout (DPO) approach*. This approach constitutes the common background for Chapters 4 to 6. But in Chapter 6, we also use the so-called *sesqui-pushout approach*, a generalization of DPO rewriting, because it will allow us to model the implicit deletion of elements (which cannot directly be done using DPO rewriting). We first introduce different variants of *adhesive categories* and basic properties thereof in Sect. 3.1. These kinds of categories offer a suitable formal framework for various categorical definitions of rewriting and for *double-pushout rewriting* in particular. In Sect. 3.2, we recall double- and sesqui-pushout rewriting and those central results for the DPO approach of which we make use later on. In Sect. 3.3, we recall general properties of pushouts and pullbacks and so-called *additional HLR properties*. These additional HLR properties allow proving some central results about DPO rewriting that cannot be obtained from the axioms of adhesive categories alone. Finally, we introduce *typed attributed (triple) graphs* as an adhesive HLR category in Sect. 3.4. Typed attributed triple graphs and *triple graph grammars* (TGGs) provide a sophisticated formal basis for developing reliable model synchronization processes. In particular, the incremental model synchronization process we develop in Chapter 6 is based on them. As this whole chapter only collects known definitions and results, we refer to the literature for proofs.

### 3.1 Variants of Adhesive Categories and their Basic HLR Properties

In the following, we introduce different variants of adhesive categories and present the distinctive properties that make such categories a suitable formal framework for rewriting.

Adhesive categories can be understood as categories where pushouts along monomorphisms behave like pushouts along injective functions in **Set**. They have been introduced by Lack and Sobociński [141] and offer a unifying formal framework for double-pushout rewriting. Later, amongst others, Ehrig et al. [53] and Heindel [98] introduced more general notions of adhesiveness that encompass practically relevant examples that are not adhesive. The formal basis for our work on model synchronization (see Chapter 6) are *typed attributed triple graphs*, which we introduce in Sect. 3.4. These constitute an *adhesive HLR category*; this follows directly from results presented, for example, in [53]. However, the canonical body of the theory of double-pushout rewriting has been lifted to the even more universal setting of  *$\mathcal{M}$ -adhesive categories* [58, 57, 54], including the *concurrent rule construction* we extend and generalize in Chapter 4. Yet another version of adhesiveness, namely *PVK square adhesiveness* [98], turns out to provide exactly the setting where the results we obtain in Chapter 5 hold. Therefore, we introduce all these different variants of adhesiveness.

The definitions of all variants of adhesiveness we present are based on the notions of *admissible classes of monomorphisms* and of *van Kampen squares* [141, 98].

**Definition 3.1** (Admissible class of monomorphisms). Given a category  $\mathcal{C}$ , a class  $\mathcal{M}$  of its monomorphisms (whose elements are referred to as  *$\mathcal{M}$ -morphisms* and depicted via hooked arrows  $m : A \hookrightarrow B$ ) is called *admissible* if (see Fig. 3.1)

- (1) the category  $\mathcal{C}$  has pullbacks along  $\mathcal{M}$ -morphisms and  $\mathcal{M}$  is closed under pullbacks, i.e., whenever  $n \in \mathcal{M}$  in the diagram to the right and  $g$  is arbitrary, their pullback exists and, moreover,  $m \in \mathcal{M}$ ,
- (2) the class  $\mathcal{M}$  contains all identity morphisms, and

$$\begin{array}{ccc}
 A & \xrightarrow{m} & B \\
 \downarrow f & & \downarrow g \\
 C & \xrightarrow{n} & D
 \end{array}$$

Figure 3.1: Commuting square for definition of admissible monomorphisms

- (3) the class  $\mathcal{M}$  is closed under composition, i.e., whenever  $n : A \hookrightarrow B$  and  $m : B \hookrightarrow C$  are  $\mathcal{M}$ -morphisms, their composition  $m \circ n$  is an  $\mathcal{M}$ -morphism, too.

The first two properties imply that an admissible class of monomorphisms contains all isomorphisms. Moreover,  $\mathcal{M}$  is closed under decomposition as well, i.e.,  $m \circ n, m \in \mathcal{M}$  implies  $n \in \mathcal{M}$  [54, Remark 4.3.]. In any category with pullbacks, the class of all monomorphisms is well-known to be admissible (see, e.g., [17, Exercises 5.7.2. and 2.9.2.] for properties (1) and (3)).

**Definition 3.2** ((Partial/vertical weak) van Kampen square). A pushout square as depicted in Fig. 3.2 is called a *van Kampen square* if, for any commutative cube as depicted in Fig. 3.3 with this pushout as bottom square and where the backfaces are pullbacks, the top square is a pushout if and only if both front faces are pullbacks.

Given an admissible class of monomorphism  $\mathcal{M}$ , a pushout square as depicted in Fig. 3.2 is a *vertical weak van Kampen square* if, for every commutative cube over it (as depicted in Fig. 3.3) where the backfaces are pullbacks and  $b, c, d \in \mathcal{M}$ , the top square is a pushout if and only if both front faces are pullbacks.

Given an admissible class of monomorphisms  $\mathcal{M}$ , a pushout square as depicted in Fig. 3.2 is a *partial van Kampen square* (PVK square) if, for every commutative cube over it (as depicted in Fig. 3.3) where the backfaces are pullbacks and  $b, c \in \mathcal{M}$ , the top square is a pushout if and only if both front faces are pullbacks and  $d \in \mathcal{M}$ .

We say that pushouts along  $\mathcal{M}$ -morphisms are *stable under pullback* (along  $\mathcal{M}$ -morphisms) if the first direction of the above equivalences holds,

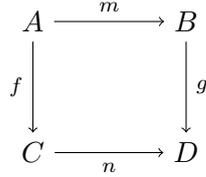


Figure 3.2: The bottom push-out square

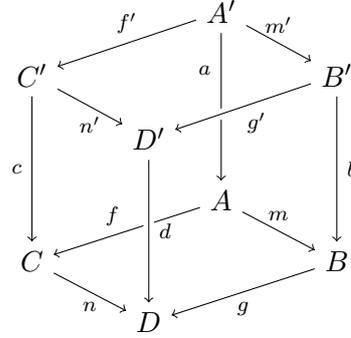


Figure 3.3: Commutative cube over the pushout square from left

i.e., when for every cube over a pushout with  $m \in \mathcal{M}$  like Fig. 3.3 where all side faces are pullbacks (and  $d \in \mathcal{M}$ ) the top face is a pushout.

**Definition 3.3** (Variants of adhesive categories).

**Adhesive** A category  $\mathcal{C}$  is *adhesive* if it has all pullbacks, has pushouts along monomorphisms, and pushouts along monomorphisms are van Kampen squares.

**Quasiadhesive** A category  $\mathcal{C}$  is *quasiadhesive* if it has all pullbacks, has pushouts along regular monomorphisms, and pushouts along regular monomorphisms are van Kampen squares.

**PVK square adhesive** A category  $\mathcal{C}$  with an admissible class of monomorphisms  $\mathcal{M}$  is *PVK square adhesive* if  $\mathcal{C}$  has pushouts along  $\mathcal{M}$ -morphisms and these are PVK squares.

**Adhesive HLR** A category  $\mathcal{C}$  with an admissible class of monomorphisms  $\mathcal{M}$  is *adhesive HLR* if  $\mathcal{C}$  has pushouts along  $\mathcal{M}$ -morphisms,  $\mathcal{M}$  is closed under pushouts, and the pushouts along  $\mathcal{M}$ -morphisms are van Kampen squares.

**$\mathcal{M}$ -adhesive** A category  $\mathcal{C}$  with an admissible class of monomorphisms  $\mathcal{M}$  is  *$\mathcal{M}$ -adhesive* if  $\mathcal{C}$  has pushouts along  $\mathcal{M}$ -morphisms,  $\mathcal{M}$  is closed under pushouts, and the pushouts along  $\mathcal{M}$ -morphisms are vertical weak van Kampen squares.

We shortly say that  $(\mathcal{C}, \mathcal{M})$  satisfies a certain variant of adhesiveness to express that  $\mathcal{C}$  does so with respect to the (admissible) class of monomorphisms  $\mathcal{M}$ .

*Remark 3.1.* In the first three cases, i.e., for adhesive, quasiadhesive, and PVK square adhesive categories, closedness of the respective class of monomorphisms under pushouts can be proved (see [141, Lemma 4.2 and Lemma 6.5] and [98, Lemma 4.1.], respectively) and thus, does not need to be required.

Heindel [97] or Ehrig, Golas, and Hermann [59] offer overviews of the known relations between the different variants of adhesiveness. Every adhesive category is adhesive HLR for  $\mathcal{M}$  being the class of all monomorphisms [53] as well as PVK square adhesive with respect to this class. A category that is adhesive HLR or PVK square adhesive with respect to an admissible class of monomorphisms  $\mathcal{M}$  is  $\mathcal{M}$ -adhesive with respect to this class. Heindel [98] showed that PVK square adhesiveness can equivalently be stated in terms of pushouts along  $\mathcal{M}$ -morphisms being *hereditary*, which means that they are pushouts also in the corresponding category of partial morphisms [125]. Important examples of adhesive categories include the category **Set** and various categories of *graph-like structures* like (typed) graphs and (typed) triple graphs [141, 53, 18]. Here, the basic category **Graph** is understood to be the functor category  $[\bullet \rightrightarrows \bullet, \mathbf{Set}]$ . Examples of categories that are adhesive HLR (for an appropriate choice of  $\mathcal{M}$ ) but not adhesive include typed attributed [53] and symbolic attributed graphs [184]. The category of *list sets* is an example of a category that is PVK square adhesive but not adhesive HLR [98, Theorem 3.19.]. Quasiadhesive categories have not been included in these overviews. However, results obtained by Lack and Sobociński ensure that in a quasiadhesive category  $\mathcal{C}$  the regular monomorphisms constitute an admissible class of monomorphisms that is furthermore closed under pushout [141, Proposition 6.4 and Lemma 6.5]. Thus, any quasiadhesive category is adhesive HLR (and  $\mathcal{M}$ -adhesive) with respect to the class of regular monomorphisms.

The following properties, also called *basic HLR properties*, are what makes variants of adhesive categories suitable frameworks for (double-pushout) rewriting. In fact, before adhesive categories (and their variants) had been found as an axiomatic basis, these properties, amongst others, were assumed to hold for developing that theory [60, 64]. Those properties have

$$\begin{array}{ccccc}
A & \xrightarrow{k} & B & \xrightarrow{r} & E \\
\downarrow l & & \downarrow f & & \downarrow v \\
(1) & & (2) & & \\
C & \xrightarrow{u} & D & \xrightarrow{w} & F
\end{array}$$

Figure 3.4: Illustration of the  $\mathcal{M}$  pushout-pullback and  $\mathcal{M}$  pullback-pushout decompositions

first been proven for adhesive categories [141] and subsequently for weaker variants like adhesive HLR [53, Theorem 4.26], PVK square adhesive [98, Section 4], and  $\mathcal{M}$ -adhesive categories [54, Theorem 4.22]. We state them in the general setting of  $\mathcal{M}$ -adhesive categories and will use them frequently in our proofs.

**Fact 3.1** (Basic HLR properties of  $\mathcal{M}$ -adhesive categories). *If  $(\mathcal{C}, \mathcal{M})$  is an  $\mathcal{M}$ -adhesive category, the following properties hold:*

- (1) *Pushouts along  $\mathcal{M}$ -morphisms are pullbacks.*
- (2) *If  $m$  in Fig. 3.2 is an  $\mathcal{M}$ -morphism, pushout complements for  $g \circ m$ , if existent, are unique (up to unique isomorphism).*
- (3)  *$(\mathcal{C}, \mathcal{M})$  has  $\mathcal{M}$  pushout-pullback decomposition. This means that given a diagram like the one depicted in Fig. 3.4 where the outer square (1) + (2) is a pushout, the right square (2) is a pullback,  $w \in \mathcal{M}$ , and  $l \in \mathcal{M}$  or  $k \in \mathcal{M}$ , then both (1) and (2) are pushouts and pullbacks.*
- (4) *The cube pushout-pullback property holds: Given a cube like the one depicted in Fig. 3.3 where all horizontal morphisms are  $\mathcal{M}$ -morphisms, the top square is a pullback, and the front squares are pushouts, then the bottom square is a pullback if and only if the squares to the back are pushouts.*

Furthermore, in all the introduced variants of adhesive categories the following results, which we will use in Chapter 5, hold.

**Fact 3.2** (Further properties of  $\mathcal{M}$ -adhesive categories [141, 53, 98]). *If a category  $(\mathcal{C}, \mathcal{M})$  fulfills one of the variants of adhesiveness defined*

above, the following properties hold (where  $\mathcal{M}$  is the class of all (regular) monomorphisms in the case of (quasi)adhesiveness):

- (1) Every  $\mathcal{M}$ -morphism is a regular monomorphism.
- (2) If  $\mathcal{C}$  is (quasi)adhesive, every functor category  $[\mathcal{X}, \mathcal{C}]$ , where  $\mathcal{X}$  is a small category, and every slice category  $\mathcal{C}/C$  for an object  $C$  of  $\mathcal{C}$  is (quasi)adhesive. Analogously, if  $(\mathcal{C}, \mathcal{M})$  is adhesive HLR, PVK square adhesive, or  $\mathcal{M}$ -adhesive, every functor category  $[\mathcal{X}, \mathcal{C}]$ , where  $\mathcal{X}$  is a small category, and every slice category  $\mathcal{C}/C$  for an object  $C$  of  $\mathcal{C}$  is so again. Here, the new class of  $\mathcal{M}$ -morphisms is given by natural transformations that consist of  $\mathcal{M}$ -morphisms componentwise or the restriction of  $\mathcal{M}$  to the morphisms in  $\mathcal{C}/C$ , respectively.

*Remark 3.2.* To the best of our knowledge, regularity of all  $\mathcal{M}$ -morphisms has only been proven for adhesive categories [141, Lemma 4.8.] (and is satisfied by definition in quasiadhesive ones) but exactly the same proof can be used for the other variants: If all pushouts along a certain morphism exist and are pullbacks, this morphism equalizes its co-kernel pair.

Moreover, we recall two technical lemmas for  $\mathcal{M}$ -adhesive categories that are needed in some of our proofs. Both allow recognizing certain squares to be pullbacks.

**Lemma 3.3** ( $\mathcal{M}$  pullback-pushout decomposition [88, Lemma B.2]). *In an  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$ , given a diagram like the one depicted in Fig. 3.4, if (1) + (2) is a pullback, (1) is a pushout, (2) commutes, and  $v \in \mathcal{M}$ , then (2) is a pullback.*

The next lemma, which appears as Lemma 2.2. in [82], is somewhat more general than  $\mathcal{M}$  pullback-pushout decomposition. Originally, Garner and Lack formulated the lemma for pushouts that are stable under pullback. We adapt it to the case of  $\mathcal{M}$ -adhesive and adhesive HLR categories.

**Lemma 3.4.** *Let  $(\mathcal{C}, \mathcal{M})$  be an  $\mathcal{M}$ -adhesive category and Fig. 3.5 a commuting diagram in that category. Then the square (1) is a pullback square provided that  $p, q \in \mathcal{M}$ , the bottom square (2) is a pushout with  $b_1 \in \mathcal{M}$  or  $b_2 \in \mathcal{M}$ , and the four remaining vertical squares are pullbacks. The statement holds in adhesive HLR categories with pullbacks even without assuming  $p, q \in \mathcal{M}$ .*

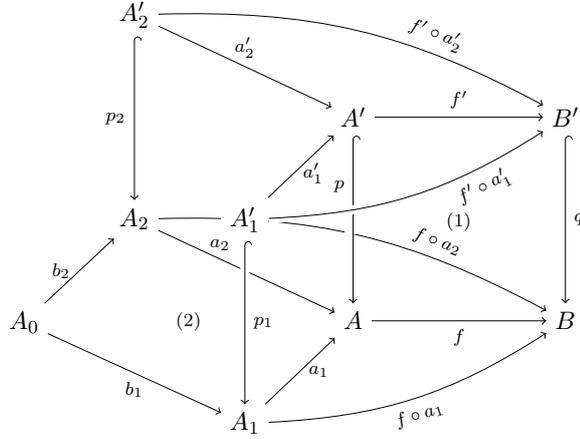


Figure 3.5: Guaranteeing square (1) to be a pullback square

### 3.2 Two Algebraic Approaches to Rewriting

In this section, we present the *double-* (Sect. 3.2.1) and the *sesqui-pushout approach* (Sect. 3.2.2) to the rewriting of objects. In the first part, we also present those concepts that are common to both approaches, in particular, the concept of a *rule*. To be able to equip rules with *application conditions*, we also recall the logic of (*nested*) *conditions* and *constraints* [94]. Subsequently, we discuss the *sequential (in)dependence* of rule applications (Sect. 3.2.3) and the construction of a *concurrent rule* (Sect. 3.2.4).

#### 3.2.1 Nested Conditions and the Double-Pushout Approach

In this section, we recall the definition of *constraints* and *conditions* and of constructions that allow “moving” conditions along morphisms or rules in such a way that their semantics is preserved.

Given an  $\mathcal{M}$ -adhesive category, *nested constraints* express properties of its objects whereas *nested conditions* express properties of morphisms [94]. Conditions are mainly used to restrict the applicability of rules. Constraints and conditions are defined recursively as trees of morphisms. For the definition of constraints, we assume the existence of an initial object  $\emptyset$  in the category  $\mathcal{C}$ .

**Definition 3.4** ((Nested) conditions and constraints). Let  $(\mathcal{C}, \mathcal{M})$  be an  $\mathcal{M}$ -adhesive category with initial object  $\emptyset$ . Given an object  $P$ , a *(nested) condition* over  $P$  is defined recursively as follows: **true** is a condition over  $P$ . If  $a : P \rightarrow C$  is a morphism and  $d$  is a condition over  $C$ ,  $\exists(a : P \rightarrow C, d)$  is a condition over  $P$  again. Moreover, Boolean combinations of conditions over  $P$  are conditions over  $P$ . A *(nested) constraint* is a condition over the initial object  $\emptyset$ .

*Satisfaction* of a nested condition  $c$  over  $P$  for a morphism  $g : P \rightarrow G$ , denoted as  $g \models c$ , is defined as follows: Every morphism satisfies **true**. The morphism  $g$  satisfies a condition of the form  $c = \exists(a : P \rightarrow C, d)$  if there exists an  $\mathcal{M}$ -morphism  $q : C \hookrightarrow G$  such that  $g = q \circ a$  and  $q \models d$ . For Boolean operators, satisfaction is defined as usual. An object  $G$  satisfies a constraint  $c$ , denoted as  $G \models c$ , if the empty morphism to  $G$  does so. A condition  $c_1$  over  $P$  *implies* a condition  $c_2$  over  $P$ , denoted as  $c_1 \implies c_2$ , if for every morphism  $g : P \rightarrow G$  with  $g \models c_1$  also  $g \models c_2$ . Two conditions are equivalent, denoted as  $c_1 \equiv c_2$ , when  $c_1 \implies c_2$  and  $c_2 \implies c_1$ . Implication and equivalence for constraints is inherited from the respective definition for conditions.

In the case of graphs, the graph constraints resulting from the above definition are expressively equivalent to a first-order logic on graphs [94, 189].

*Remark 3.3.* In the notation of graph conditions, we drop the domains of the involved morphisms and occurrences of **true** whenever they can unambiguously be inferred. Moreover, we introduce  $\forall(C_1, d)$  as an abbreviation for the graph condition  $\neg\exists(C_1, \neg d)$ . For example, we write  $\forall(C_1, \exists C_2)$  instead of  $\forall(a_1 : \emptyset \hookrightarrow C_1, \exists(a_2 : C_1 \hookrightarrow C_2, \text{true}))$ .

Adhesive categories (in all their variants) are a suitable formal framework for rule-based rewriting as defined in the double-pushout approach: Rules are a declarative way to define transformations of objects. They consist of a left-hand side (LHS)  $L$ , a right-hand side (RHS)  $R$ , and a common subobject  $K$ , the interface of the rule. In the case of (typed attributed) graphs, application of a rule  $p$  to a graph  $G$  amounts to choosing an image of the rule's LHS  $L$  in  $G$ , deleting the image of  $L \setminus K$ , and adding a copy of  $R \setminus K$ . This procedure can be formalized by two pushouts. Rules and their application semantics are defined as follows.

$$\begin{array}{ccccc}
 L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
 \downarrow m \models ac & & \downarrow d & & \downarrow n \\
 G & \xleftarrow{g} & D & \xrightarrow{h} & H
 \end{array}$$

Figure 3.6: A rule-based transformation in the double-pushout approach

**Definition 3.5** (Rules and transformations). A *rule*  $\rho = (p, ac)$  consists of a *plain rule*  $p$  and an *application condition*  $ac$ . The plain rule is a span of  $\mathcal{M}$ -morphisms  $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ ; the objects are called *left-hand side* (LHS), *interface*, and *right-hand side* (RHS), respectively. The application condition  $ac$  is a nested condition over  $L$ . If this application condition is of the form  $ac = \bigwedge_{i=1, \dots, k} \neg \exists (x_i : L \rightarrow N_i, \text{true})$ , it is called a *negative application condition* (NAC).

Given a rule  $\rho = (L \xleftarrow{l} K \xrightarrow{r} R, ac)$  and a morphism  $m : L \rightarrow G$ , a (*direct*) *transformation*  $G \Rightarrow_{\rho, m} H$  from  $G$  to  $H$  is given by the diagram in Fig. 3.6 where both squares are pushouts and  $m \models ac$ . If such a transformation exists, the morphism  $m$  is called a *match* and rule  $\rho$  is *applicable* at match  $m$ ; in this case,  $n$  is called the *comatch* of the transformation.

The inverse rule  $\rho^{-1}$  of a rule  $\rho = (L \xleftarrow{l} K \xrightarrow{r} R, ac)$  is the rule  $(R \xleftarrow{r} K \xrightarrow{l} L, \text{Left}(\rho, ac))$ , where  $\text{Left}(\rho, ac)$  is defined as introduced in the following. A plain rule  $p$  is *monotonic* if  $l : K \hookrightarrow L$  is an isomorphism, and *only deletes* if  $r : K \hookrightarrow R$  is an isomorphism. We denote monotonic rules as  $r : L \hookrightarrow R$ .

In several applications, the definition of a transformation is additionally restricted by requiring  $m \in \mathcal{M}$ ; we refer to this as  *$\mathcal{M}$ -matching*. For graphs,  $\mathcal{M}$ -matching (i.e., injective matching) is known to lead to more expressive grammars than the general matching [93]. In all variants of adhesive categories, the second pushout in the definition of a transformation always exists since  $r \in \mathcal{M}$ , and the pushout complement  $D$ , whenever it exists, is unique (up to unique isomorphism) since  $l \in \mathcal{M}$ . For a rule to be applicable at a match, thus, a pushout complement for  $m \circ l$  has to exist; a necessary and sufficient condition for this to be the case can be formulated

in terms of so-called *initial pushouts* (see Definition 3.20 and Fact 3.11). In categories of graph-like structures, also a more elementary characterization can be given in terms of the *gluing condition* [53, Fact 3.11]: There, a rule is applicable at a match  $m$  if and only if  $m$  does not identify an element that is to be deleted with any other element (*identification condition*) and  $m$  does not match a node that is to be deleted to a node with an incident edge that is not also designated to be deleted (*dangling-edge condition*). Notably, the dangling-edge condition is the only obstacle to the applicability of a rule in the setting of injective matching.

A transformation  $G \Rightarrow H$  comes with the notion of a *track morphism*. When the morphisms are functions between objects with elements, this is just the partial function that maps the elements of  $G$  that are not deleted by the transformation to their counterparts in  $H$ .

**Definition 3.6** (Track morphism). Let  $t : G \Rightarrow H$  be a direct transformation defined by the span  $G \xleftarrow{g} D \xrightarrow{h} H$ . The *track morphism*  $tr_t : G \dashrightarrow H$  is that span, considered as partial morphism from  $G$  to  $H$  with domain  $D$ . Given a transformation  $t : G_0 \Rightarrow G_1 \Rightarrow \cdots \Rightarrow G_n$ , its track morphism  $tr_t$  is defined as the composition of the individual track morphisms.

In the following, we recall the *Shift* and *Left* constructions that allow “moving” constraints along morphisms and rules, respectively. The case for morphisms assumes that there is a class  $\mathcal{E}'$  of pairs of morphisms with the same codomain such that every pair of morphisms with the same codomain can be factored into a pair of morphisms from  $\mathcal{E}'$  followed by an  $\mathcal{M}$ -morphism. We formally introduce such  $\mathcal{E}'$ - $\mathcal{M}$  *pair factorizations* of pairs of morphisms (together with the other additional HLR properties) in Sect. 3.3.2.

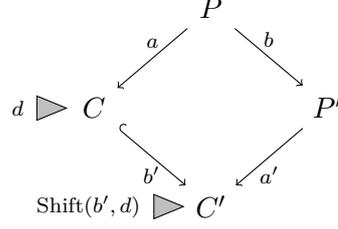
**Construction 3.7** (Shift along morphism). Given a condition  $c$  over an object  $P$  and a morphism  $b : P \rightarrow P'$ , the *shift of  $c$  along  $b$* , denoted as  $\text{Shift}(b, c)$ , is inductively defined as follows:

- If  $c = \text{true}$ ,

$$\text{Shift}(b, c) := \text{true} .$$

- If  $c = \exists (a : P \rightarrow C, d)$ ,

$$\text{Shift}(b, c) := \bigvee_{(a', b') \in \mathcal{F}} \exists (a', \text{Shift}(b', d)) ,$$


 Figure 3.7: Graphical representation of the definition of  $\text{Shift}(b, \exists(a : P \rightarrow C, d))$ 

where  $\mathcal{F}$  contains morphism pairs from  $\mathcal{E}'$  with one morphism from  $\mathcal{M}$  that complement  $a$  and  $b$  to a commutative diagram, i.e.,

$$\mathcal{F} := \{(a', b') \in \mathcal{E}' \mid b' \in \mathcal{M} \text{ and } b' \circ a = a' \circ b\} .$$

Note that the empty disjunction is equivalent to false.

- If  $c = \neg d$ ,

$$\text{Shift}(b, c) := \neg \text{Shift}(b, d) .$$

- If  $c = \bigwedge_{i \in I} c_i$ ,

$$\text{Shift}(b, c) := \bigwedge_{i \in I} \text{Shift}(b, c_i) .$$

**Fact 3.5** (Correctness of Shift [58, Lemma 3.11]). *In an  $\mathcal{M}$ -adhesive category  $\mathcal{C}$  with  $\mathcal{E}'$ - $\mathcal{M}$  pair factorization, given a nested condition  $c$  over an object  $P$  and a morphism  $b : P \rightarrow P'$ , for each morphism  $g' : P' \rightarrow G$*

$$g' \models \text{Shift}(b, c) \iff g := g' \circ b \models c .$$

Similarly, conditions can be “moved” along rules.

**Construction 3.8** (Shift over rule). Given a condition  $c$  over an object  $R$  and a plain rule  $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ , the *shift of  $c$  over  $p$* , denoted as  $\text{Left}(p, c)$ , is inductively defined as follows:

- If  $c = \text{true}$ ,

$$\text{Left}(p, c) := \text{true} .$$

- If  $c = \exists(a : R \rightarrow R^*, d)$ , consider the following diagram:

$$\begin{array}{ccccc}
L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
a^* \downarrow & & \downarrow & & \downarrow a \\
L^* & \xleftarrow{l^*} & K^* & \xrightarrow{r^*} & R^* \\
\triangle \uparrow & & & & \triangle \uparrow \\
\text{Left}(p^*, d) & & & & d
\end{array}$$

(2)                      (1)

If  $a \circ r$  has a pushout complement (1) and

$$p^* = (L^* \xleftarrow{l^*} K^* \xrightarrow{r^*} R^*)$$

is the rule derived by constructing the pushout (2), i.e., by applying the inverse rule  $p^{-1}$  at match  $a$  to  $R^*$ , then

$$\text{Left}(p, c) := \exists (a^* : L \rightarrow L^*, \text{Left}(p^*, d)) .$$

Otherwise, i.e., if the pushout complement (1) does not exist,

$$\text{Left}(p, c) := \text{false} .$$

- If  $c = \neg d$ ,

$$\text{Left}(p, c) := \neg \text{Left}(p, d) .$$

- If  $c = \bigwedge_{i \in I} c_i$ ,

$$\text{Left}(p, c) := \bigwedge_{i \in I} \text{Left}(p, c_i) .$$

**Fact 3.6** (Correctness of Left [58, Lemma 3.14]). *In an  $\mathcal{M}$ -adhesive category  $\mathcal{C}$ , given a nested condition  $c$  over an object  $R$  and a plain rule  $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ , for each direct transformation  $G \Rightarrow_{p, m, m^*} H$ , where  $m^*$  denotes the comatch of that transformation,*

$$m \models \text{Left}(p, c) \iff m^* \models c .$$

*Remark 3.4.* By the symmetric nature of double-pushout rewriting, a completely analogous construction Right shifts application conditions from the LHS of a rule to its RHS in a semantics-preserving way.

### 3.2.2 The Sesqui-Pushout Approach to Rewriting

In this section, we shortly introduce *sesqui-pushout rewriting* [46]. The distinctive property of sesqui-pushout rewriting (at least when restricted to linear rules as we do) is that it allows for deletion in unknown context. In contrast to DPO, a rule application is not blocked by the dangling-edge condition. Instead, such dangling edges are (implicitly) deleted as well. We will need this property in our model synchronization process: To guarantee termination, we follow a *top-down strategy*. We start change propagation at the root of a structure that is similar to a parse tree of the given model and continue to the leaves. During that procedure, consequences of decisions are passed down. Whenever we have to propagate a deletion of an element of the source model to the target model, this can therefore be a deletion in unknown context.

The definition of a sesqui-pushout transformation uses the concept of a *final pullback complement* (FPC).

**Definition 3.9** (Final pullback complement). Given a pair of composable morphisms  $m : L \rightarrow G$  and  $l : K \rightarrow L$ , a *final pullback complement* for  $m \circ l$  is an object  $D$  and a pair of composable morphisms  $g : D \rightarrow G$  and  $d : K \rightarrow D$  such that the resulting square is a pullback and universal in that respect: For any other pullback  $m \circ f = z \circ d'$  of  $m$  and any morphism  $f' : K' \rightarrow K$  such that  $l \circ f' = f$  (as depicted in Fig. 3.8), there exist a unique morphism  $z' : D' \rightarrow D$  such that  $g \circ z' = z$  and  $d \circ f' = z' \circ d'$ .

Compared to the definition of a DPO transformation, a transformation step in the sesqui-pushout approach replaces the first pushout by a final pullback complement. A rule is still defined to be a span of morphisms. While in the sesqui-pushout approach, it is possible to allow for arbitrary spans,<sup>1</sup> we do not need this kind of generality and restrict ourselves to the case of spans of  $\mathcal{M}$ -morphisms.

**Definition 3.10** (Sesqui-pushout transformation). Given a rule  $\rho = (L \xleftarrow{l} K \xrightarrow{r} R, ac)$  and a morphism  $m : L \rightarrow G$ , a (*direct*) *transformation from  $G$*

<sup>1</sup>By its universal property, a FPC (if it exists) is unique (up to unique isomorphism) independently of the morphism  $l$ . In contrast, uniqueness of pushout complements (in  $\mathcal{M}$ -adhesive categories) depends on  $l$  being an  $\mathcal{M}$ -morphism. For graph-like structures, allowing the left morphism  $l$  in a rule to not be injective provides support for cloning of elements; see, e.g., [46, 157].

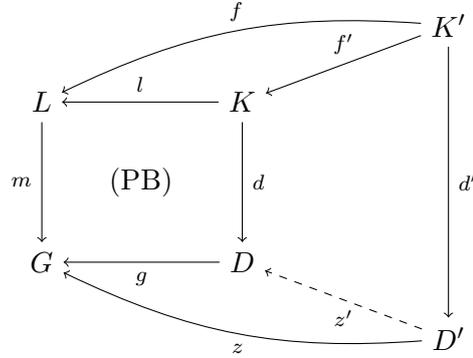


Figure 3.8: Final pullback complement

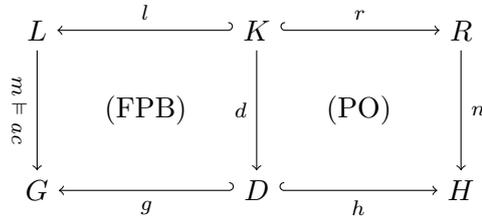


Figure 3.9: A rule-based transformation in the sesqui-pushout approach

to  $H$  in the sesqui-pushout approach, denoted as  $G \Rightarrow_{\rho, m}^{\text{SqPO}} H$ , is given by the diagram in Fig. 3.9, where the left square is a final pullback complement, the right one a pushout, and  $m \vDash ac$ .

### 3.2.3 Sequential Independence

Sequential independence of two rule applications intuitively means that none of these applications enables the other one when applied sequentially. This implies that the order of their application may be switched. Here, and also in the following section, we only present the theory for the case of double-pushout rewriting because we only need it in that context. For the according theory for sesqui-pushout rewriting we refer the reader to [46, 157, 45, 24].

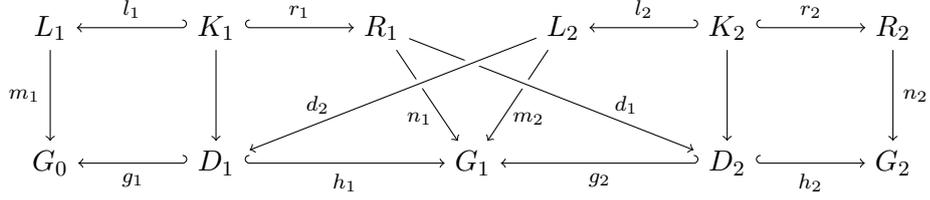


Figure 3.10: Defining sequential independence

**Definition 3.11** (Sequential independence). Given two rules  $\rho_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i, ac_i)$ , where  $i = 1, 2$ , two direct transformations  $t_1 : G_0 \Rightarrow_{\rho_1, m_1} G_1$  and  $t_2 : G_1 \Rightarrow_{\rho_2, m_2} G_2$  via the rules  $\rho_1$  and  $\rho_2$  are *sequentially independent* if there exist two morphisms  $d_1 : R_1 \rightarrow D_2$  and  $d_2 : L_2 \rightarrow D_1$  as depicted in Fig. 3.10 such that  $n_1 = g_2 \circ d_1$  and  $m_2 = h_1 \circ d_2$  and

$$m'_2 := g_1 \circ d_2 \models ac_2$$

$$m'_1 := h_2 \circ d_1 \models \text{Right}(L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1, ac_1) .$$

We say that transformation  $t_2$  is *independent* of transformation  $t_1$  when at least the morphism  $d_2$  with  $h_1 \circ d_2 = m_2$  exists such that  $g_1 \circ d_2$  is a match for  $\rho_2$ .

Sequential independence is closely related to the concept of *parallel independence*, which we do not explicitly introduce. Intuitively, parallel independence means that two rule applications do not disable each other when applied to the same object. The *Local Church–Rosser Theorem* relates both concepts and states that the order of sequentially independent rule applications might be switched. It is this formal property that we need later on.

**Theorem 3.7** (Local Church–Rosser Theorem [58, Theorem 4.7]). *Let  $(\mathcal{C}, \mathcal{M})$  be an  $\mathcal{M}$ -adhesive category. Given two parallel independent transformations  $G \Rightarrow_{\rho_1, m_1} H_1$  and  $G \Rightarrow_{\rho_2, m_2} H_2$ , there is an object  $X$  and transformations  $H_1 \Rightarrow_{\rho_2, m'_2} X$  and  $H_2 \Rightarrow_{\rho_1, m'_1} X$  such that both*

$$G \Rightarrow_{\rho_1, m_1} H_1 \Rightarrow_{\rho_2, m'_2} X$$

and

$$G \Rightarrow_{\rho_2, m_2} H_2 \Rightarrow_{\rho_1, m'_1} X$$

are sequentially independent.

Given two sequentially independent transformations

$$G \Rightarrow_{\rho_1, m_1} H_1 \Rightarrow_{\rho_2, m'_2} X \quad ,$$

there is an object  $H_2$  and sequentially independent transformations

$$G \Rightarrow_{\rho_2, m_2} H_2 \Rightarrow_{\rho_1, m'_1} X$$

such that  $G \Rightarrow_{\rho_1, m_1} H_1$  and  $G \Rightarrow_{\rho_2, m_2} H_2$  are parallel independent.

Sequential independence induces an equivalence relation on transformation sequences, called *switch equivalence*. It has first been studied for graphs [139].

The definition of sequential independence can then be extended to a sequence of rule applications longer than two. In Theorem 4.16, we will use this to identify language-preserving applications of GCRs.

**Definition 3.12** (Sequential independence from an application sequence). Given rules  $\rho$  and  $\rho_i$ , where  $1 \leq i \leq t$ , a transformation  $G_t \Rightarrow_{\rho, m} H$  is *sequentially independent from a sequence of transformations*

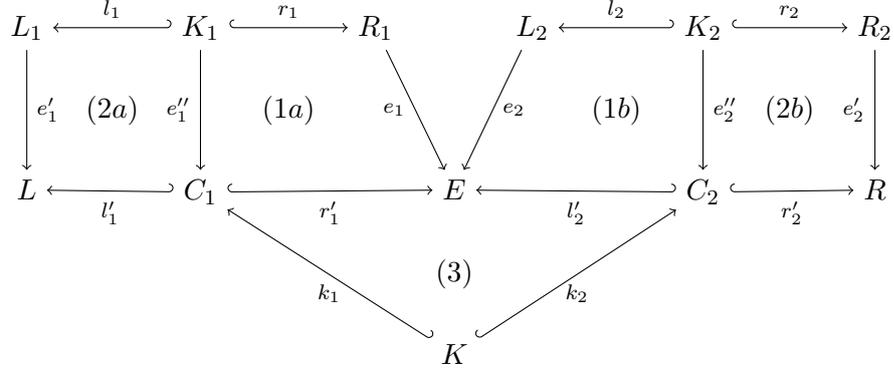
$$G_0 \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow_{\rho_2, m_2} \cdots \Rightarrow_{\rho_t, m_t} G_t \quad ,$$

where  $t \geq 2$ , if first,  $G_t \Rightarrow_{\rho, m} H$  and  $G_{t-1} \Rightarrow_{\rho_t, m_t} G_t$  are sequentially independent and then, iteratively, the transformations  $G_i \Rightarrow_{\rho, m'_i} G'_{i+1}$  (induced by the Local Church–Rosser Theorem) and  $G_{i-1} \Rightarrow_{\rho_i, m_i} G_i$  are sequentially independent for all  $1 \leq i \leq t-1$ .

### 3.2.4 Concurrent Rules and the Concurrency Theorem

Here, we recall *E-concurrent rules*, which combine the actions of two (possibly sequentially dependent) rules into a single one. Also their definition assumes a given class  $\mathcal{E}'$  of pairs of morphisms with the same codomain. In practical applications, this is often the class of jointly epic pairs of ( $\mathcal{M}$ -)morphisms.

**Definition 3.13** (*E*-concurrent rule). Given two rules  $\rho_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i, ac_i)$ , where  $i = 1, 2$ , an object  $E$  with morphisms  $e_1 : R_1 \rightarrow E$  and


 Figure 3.11:  $E$ -dependency relation and  $E$ -concurrent rule

$e_2 : L_2 \rightarrow E$  is an  $E$ -dependency relation for  $\rho_1$  and  $\rho_2$  if  $(e_1, e_2) \in \mathcal{E}'$  and the pushout complements (1a) and (1b) for  $e_1 \circ r_1$  and  $e_2 \circ l_2$  as depicted in Fig. 3.11 exist.

Given an  $E$ -dependency relation  $E = (e_1, e_2) \in \mathcal{E}'$  for rules  $\rho_1, \rho_2$ , their  $E$ -concurrent rule is defined as

$$\rho_1 *_E \rho_2 := (L \xleftarrow{l} K \xrightarrow{r} R, ac) ,$$

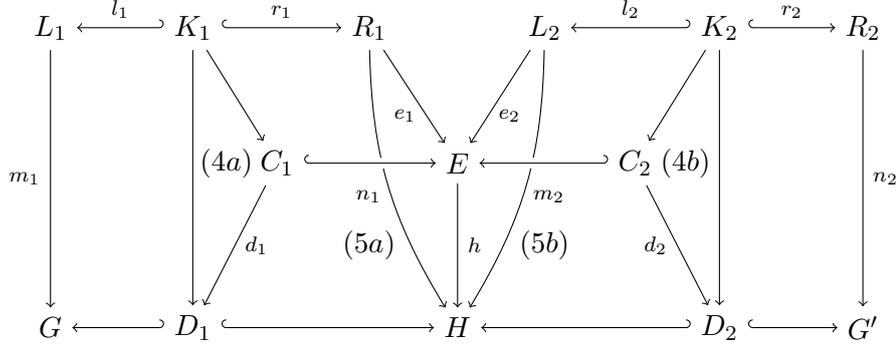
where  $l := l'_1 \circ k_1$ ,  $r := r'_2 \circ k_2$ , (1a), (1b), (2a), and (2b) are pushouts, (3) is a pullback (also shown in Fig. 3.11), and

$$ac := \text{Shift}(e'_1, ac_1) \wedge \text{Left}(p', \text{Shift}(e_2, ac_2)) ,$$

where  $p' := (L \xleftarrow{l'_1} C_1 \xrightarrow{r'_1} E)$ .

A transformation sequence  $G \Rightarrow_{\rho_1, m_2} H \Rightarrow_{\rho_2, m_2} G'$  is called  $E$ -related for the  $E$ -dependency relation  $(e_1, e_2) \in \mathcal{E}'$  if there exists  $h : E \rightarrow H$  with  $h \circ e_1 = n_1$  and  $h \circ e_2 = m_2$  and morphisms  $d_i : C_i \rightarrow D_i$ , where  $i = 1, 2$ , such that (4a) and (4b) commute and (5a) and (5b) are pushouts (see Fig. 3.12).

The Concurrency Theorem [53, Theorem 5.23] states that two  $E$ -related rule applications may be synthesized into the application of their  $E$ -concurrent rule and that an application of an  $E$ -concurrent rule may be analyzed into a sequence of two  $E$ -related rule applications.

Figure 3.12: An  $E$ -related transformation

**Theorem 3.8** (Concurrency Theorem [54, Theorem 5.30]). *In an  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$ , let two rules  $\rho_1, \rho_2$  and an  $E$ -dependency relation  $E = (e_1, e_2)$  for them be given.*

**Synthesis** *Given an  $E$ -related transformation sequence*

$$G \Rightarrow_{\rho_1, m_1} H \Rightarrow_{\rho_2, m_2} G' ,$$

*there is a direct transformation  $G \Rightarrow_{\rho_1 *_{E} \rho_2, m} G'$  via the  $E$ -concurrent rule.*

**Analysis** *Given a direct transformation  $G \Rightarrow_{\rho_1 *_{E} \rho_2, m} G'$  via the  $E$ -concurrent rule, there is an  $E$ -related transformation sequence*

$$G \Rightarrow_{\rho_1, m_1} H \Rightarrow_{\rho_2, m_2} G' .$$

*Synthesis and Analysis are inverse to each other (up to isomorphism).*

The question arises whether, given any transformation sequence

$$G \Rightarrow_{\rho_1, m_1} H \Rightarrow_{\rho_2, m_2} G' ,$$

there is an  $E$ -relation for which this sequence is  $E$ -related. The answer is “yes”, provided there exists an  $\mathcal{E}'$ - $\mathcal{M}$  factorization for pairs of morphisms; this is a special case of [54, Fact 5.29].

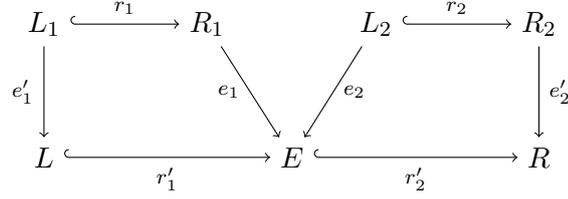


Figure 3.13: A concurrent rule that stems from monotonic rules

*Remark 3.5.* Later in this work (Sect. 6.2.3), we will consider the special cases of concurrent rules that stem from monotonic rules, i.e., from rules that only create structure, and concurrent rules that stem from their inverses, i.e., from rules that only delete structure. The computation of concurrent rules then simplifies as depicted in Fig. 3.13 (for the case of monotonic rules).

We will need the following two observations, which are evident from Fig. 3.13. They also hold (inductively) for the case in which the concurrent rule is computed from more than two rules.

- (1) A concurrent rule of monotonic rules is monotonic again. Likewise, a concurrent rule of rules that only delete is a rule that only deletes.
- (2) Given a concurrent rule  $L \hookrightarrow R$  derived from monotonic rules  $r_i : L_i \hookrightarrow R_i$ , where  $i = 1, \dots, t$ , for any RHS  $R_i$  there is a morphism  $\iota_i : R_i \rightarrow R$ . For  $t = 2$ , these are the morphisms  $r'_2 \circ e_1$  and  $e'_2$  (see Fig. 3.13); for  $t > 2$  they are obtained by further composition. Likewise, given a concurrent rule  $L \leftarrow R$  derived from rules that only delete, for any LHS  $L_i$  of any of these rules there is a morphism  $\iota_i : L_i \rightarrow L$ .

### 3.3 Further Categorical Preliminaries and Additional HLR Properties

While we generally assume knowledge of basic category theory, we still recall fundamental properties of pushouts and pullbacks that we use repeatedly in our proofs in Sect. 3.3.1. Section 3.3.2 then contains the definitions of further properties, called *additional HLR properties*, that are not implied

$$\begin{array}{ccccc}
 A & \xrightarrow{k} & B & \xrightarrow{r} & E \\
 \downarrow l & & \downarrow f & & \downarrow v \\
 C & \xrightarrow{u} & D & \xrightarrow{w} & F
 \end{array}
 \quad
 \begin{array}{ccc}
 & (1) & \\
 & & (2)
 \end{array}$$

Figure 3.14: Illustrating pushout and pullback (de)composition

by adhesiveness but are additionally needed to develop a sophisticated theory of rewriting.

### 3.3.1 Properties of Pushouts and Pullbacks and a Categorical Definition of Partial Morphisms

In our proofs, we repeatedly use the following well-known properties of pushouts and pullbacks.

**Fact 3.9** (Properties of pushouts and pullbacks). *In any category  $\mathcal{C}$ , pushouts and pullbacks satisfy the following properties.*

- (1) Pushout composition and decomposition: *Given a commuting diagram like Fig. 3.14 where (1) is a pushout, (1) + (2) is a pushout if and only if (2) is.*
- (2) Pullback composition and decomposition: *Given a commuting diagram like Fig. 3.14 where (2) is a pullback, (1) + (2) is a pullback if and only if (1) is.*
- (3) *A morphism is a monomorphism if and only if its kernel pair consists of the identity morphism.*
- (4) *A pushout along an isomorphism results in an isomorphism; in particular, one can choose this morphism to be an identity morphism as well (as depicted in Fig. 3.15). Likewise, a pullback along an isomorphism results in an isomorphism.*
- (5) *Given morphisms  $k : A \rightarrow B$  and  $f : B \rightarrow D$  with  $f$  being a monomorphism, the span  $B \xleftarrow{k} A \xrightarrow{id_A} A$  is a pullback of  $(f, f \circ k)$  (as depicted in Fig. 3.16).*

$$\begin{array}{ccc}
 A & \xrightarrow{k} & B \\
 \parallel & & \parallel \\
 A & \xrightarrow{k} & B
 \end{array}$$

Figure 3.15: Pushout along an identity

$$\begin{array}{ccc}
 A & \xrightarrow{k} & B \\
 \parallel & & \downarrow f \\
 A & \xrightarrow{f \circ k} & D
 \end{array}$$

Figure 3.16: Pullback of composed morphisms

Partial morphisms will play a central role in this thesis because for our model synchronization process we need to formalize the situation that a correspondence morphism of a triple graph or a comatch of a formerly valid rule application become partial. Our formalization of these situations will rest upon the following (simple) definition of partial morphisms in arbitrary categories. To simplify presentation, we refrain from identifying equivalent partial morphisms as usually done [190].

**Definition 3.14** (Partial morphism). A *partial morphism*  $a : A \dashrightarrow B$  is a span  $A \xleftarrow{\iota_A} A' \xrightarrow{a} B$  where  $\iota_A$  is a monomorphism;  $A'$  is called the *domain* of  $a$ .

In this thesis, we will frequently consider the case where the inclusion of the domain is not an arbitrary monomorphism but stems from a selected class  $\mathcal{M}$  of monomorphisms. This will always be an admissible class of monomorphisms, which ensures that composition of partial morphisms is defined (compare, e.g., [98]).

### 3.3.2 Additional HLR Properties

The basic HLR properties of adhesive categories (that have exemplarily been listed in Fact 3.1) are enough to ensure that double-pushout rewriting in these is deterministic and to prove a first central result, namely the *Local Church–Rosser Theorem* (Theorem 3.7) stating that sequentially independent transformations might be performed in arbitrary order. However, *additional HLR properties* are required to prove most of the well-known results like the *Parallelism*, the *Concurrency* (Theorem 3.8),

and the *Amalgamation Theorem* for double-pushout rewriting in these categories. Overviews can be found, for example, in [58, 57, 54]. In particular, the following additional properties are used in the known proofs for the following results (where we make use of a second class of morphisms  $\mathcal{M}'$  that does not need to coincide with  $\mathcal{M}$ ):

- The Parallelism Theorem requires the existence of *finite coproducts* that are compatible with  $\mathcal{M}$ . These can be conveniently constructed as pushouts if  $(\mathcal{M}\text{-})$ *initial objects* (Definition 3.16) exist.
- The Concurrency Theorem, as well as the completeness of critical pairs, are based on  $\mathcal{E}'\text{-}\mathcal{M}'$  *pair factorizations* (Definition 3.18) of pairs of morphisms where  $\mathcal{M}'$  has to satisfy the  $\mathcal{M}\text{-}\mathcal{M}'$  *pushout-pullback decomposition* property (Definition 3.17). If a category has coproducts, such a pair factorization can be constructed from an  $\mathcal{E}\text{-}\mathcal{M}'$  *factorization* of morphisms.
- *Initial pushouts* (Definition 3.20) are used in the proof of the Embedding Theorem.
- $\mathcal{M}$ -*effective unions* (Definition 3.21) are used in the proof of the Amalgamation Theorem.

In contrast to the basic HLR properties, additional HLR properties are either not known to hold in every  $\mathcal{M}$ -adhesive category or counterexamples are available. The results we obtain in Chapter 4 also depend on some of these additional properties. In Sect. 5.2.2, we will show how they are preserved under the construction of  $S$ -cartesian functor categories, which we develop in Sect. 5.2.1.

In the next definition, we summarize what it means for an  $\mathcal{M}$ -adhesive category to fulfill the additional HLR properties (see [54, Definition 4.23]). We introduce and discuss each of these properties subsequently.

**Definition 3.15** (Additional HLR properties). An  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$  satisfies the additional HLR properties if  $\mathcal{C}$  has

- (1) binary coproducts,
- (2) an  $\mathcal{E}\text{-}\mathcal{M}$  factorization of morphisms,

- (3) an  $\mathcal{E}'\text{-}\mathcal{M}'$  pair factorization of pairs of morphisms with the same codomain,
- (4) initial pushouts over  $\mathcal{M}'$ , and
- (5)  $\mathcal{M}$ -effective unions

for classes of (pairs of) morphisms  $\mathcal{E}, \mathcal{E}', \mathcal{M}'$  such that the  $\mathcal{M}\text{-}\mathcal{M}'$  pushout-pullback decomposition property holds and  $\mathcal{M}'$  is closed under pushouts and pullbacks along  $\mathcal{M}$ -morphisms.

**Binary Coproducts and ( $\mathcal{M}$ -)initial objects** The construction of *parallel rules* and, consequently, the Parallelism Theorem requires the existence of binary coproducts. Whenever an  $\mathcal{M}$ -adhesive category has such, finite coproducts are compatible with  $\mathcal{M}$  (of which we will make use in the proof of Proposition 5.13).

**Lemma 3.10** (Compatibility of coproducts with  $\mathcal{M}$  [58, Fact 4.9.]). *If a class of monomorphisms  $\mathcal{M}$  is closed under pushout, this class is compatible with finite coproducts, i.e., whenever*

$$\{f_i : A_i \hookrightarrow B_i\}_{i \in \{1, \dots, n\}}$$

*is a finite family of  $\mathcal{M}$ -morphisms and finite coproducts exist, the induced map*

$$f_1 + \dots + f_n : A_1 + \dots + A_n \hookrightarrow B_1 + \dots + B_n$$

*is an  $\mathcal{M}$ -morphism, too.*

When a category has an initial object, coproducts can be obtained as pushouts from that object.

**Definition 3.16** ( $(\mathcal{M})$ -initial object). An object  $I$  is *initial* in a category  $\mathcal{C}$  if for all objects  $A$  there is a unique morphism  $i_A : I \rightarrow A$ . Given an admissible class of monomorphisms  $\mathcal{M}$ , an initial object  $I$  is  *$\mathcal{M}$ -initial* if for every object  $A$  the corresponding unique morphism is an  $\mathcal{M}$ -morphism, i.e.,  $i_A \in \mathcal{M}$ .

**Factorization properties** *Factorization systems* are a common topic in category theory. In the literature on graph transformation, however, factorization of morphisms is introduced in a way that slightly differs from the presentation in standard textbooks such as [2]. The employed notion of factorization is normally weaker. But before discussing factorizations, we introduce properties that the additionally introduced class of morphisms  $\mathcal{M}'$  should satisfy with respect to  $\mathcal{M}$ .

For the construction of  $E$ -related transformation sequences for the Concurrency Theorem, and to prove the completeness of critical pairs, the morphism class  $\mathcal{M}'$  from the  $\mathcal{E}$ - $\mathcal{M}'$  factorization (or the  $\mathcal{E}'$ - $\mathcal{M}'$  pair factorization) has to fulfill the  $\mathcal{M}$ - $\mathcal{M}'$  *PO-PB decomposition property*. Moreover, to show important properties of initial pushouts over  $\mathcal{M}'$ -morphisms, this class needs to be closed under pushouts and pullbacks along  $\mathcal{M}$ -morphisms.

**Definition 3.17** ( $\mathcal{M}$ - $\mathcal{M}'$  PO-PB decomposition property. Closedness). A category  $\mathcal{C}$  with morphism classes  $\mathcal{M}$  and  $\mathcal{M}'$  has the  $\mathcal{M}$ - $\mathcal{M}'$  *PO-PB decomposition property* if, given the diagram depicted in Fig. 3.4 where the outer square (1) + (2) is a pushout, the right square (2) is a pullback,  $l \in \mathcal{M}$ , and  $w \in \mathcal{M}'$ , both (1) and (2) are pushouts and pullbacks.

The class  $\mathcal{M}'$  is closed under pushouts along  $\mathcal{M}$ -morphisms if, whenever a pushout like the one in Fig. 3.2 is given with  $m \in \mathcal{M}$ ,  $f \in \mathcal{M}'$ , also  $g \in \mathcal{M}'$ . Closedness under pullbacks is defined analogously.

Note that in  $\mathcal{M}$ -adhesive categories, whenever  $\mathcal{M} = \mathcal{M}'$ , the above defined properties are automatically true as they coincide with already known properties, namely the closedness of  $\mathcal{M}$  under pullback and pushout and  $\mathcal{M}$  pushout-pullback decomposition. In particular, the first two additional HLR properties (coproducts and  $\mathcal{E}$ - $\mathcal{M}$  factorization) imply the third in such a way that even  $\mathcal{M}' = \mathcal{M}$ . This is important in the presence of *application conditions* for rules: the *Shift-Construction* (see Construction 3.7) assumes an  $\mathcal{E}'$ - $\mathcal{M}$  factorization. Whenever application conditions are not needed, the weaker notion of an  $\mathcal{E}'$ - $\mathcal{M}'$  factorization might suffice because, as mentioned above, it is enough to establish, e.g., the Concurrency Theorem and completeness of critical pairs. Hence,  $\mathcal{E}'$ - $\mathcal{M}'$  factorizations are of independent interest and we introduce this weaker notion.

**Definition 3.18** ( $\mathcal{E}$ - $\mathcal{M}'$  factorization and  $\mathcal{E}'$ - $\mathcal{M}'$  pair factorization). A category  $\mathcal{C}$  with classes of morphisms  $\mathcal{M}'$  and  $\mathcal{E}$  has a (*unique*)  $\mathcal{E}$ - $\mathcal{M}'$

*factorization* if for every morphism  $f : A \rightarrow B$  in  $\mathcal{C}$  there are [unique (up to isomorphism)] morphisms  $e : A \rightarrow K \in \mathcal{E}$  and  $m : K \rightarrow B \in \mathcal{M}'$  such that  $f = m \circ e$ .

A unique  $\mathcal{E}$ - $\mathcal{M}'$  factorization is *functorial* if, for all morphisms  $f_i : A_i \rightarrow B_i$ , where  $i = 1, 2$  and  $f_i = m_i \circ e_i$  are the corresponding factorizations with  $e_i : A_i \rightarrow K_i$ ,  $m_i : K_i \rightarrow B_i$ , for every pair of morphisms  $u : A_1 \rightarrow A_2$  and  $v : B_1 \rightarrow B_2$  with  $v \circ f_1 = f_2 \circ u$  there exists a unique morphism  $d : K_1 \rightarrow K_2$  with  $d \circ e_1 = e_2 \circ u$  and  $m_2 \circ d = v \circ m_1$ .

Given a class of morphisms  $\mathcal{M}'$  and a class  $\mathcal{E}'$  of pairs of morphisms with the same codomain, the category  $\mathcal{C}$  has a (*unique*)  $\mathcal{E}'$ - $\mathcal{M}'$  *pair factorization* if for each pair of morphisms  $f_1 : A_1 \rightarrow B$ ,  $f_2 : A_2 \rightarrow B$  with the same codomain there are [unique (up to isomorphism)] morphisms  $e_1 : A_1 \rightarrow K$ ,  $e_2 : A_2 \rightarrow K$  and  $m : K \rightarrow B$  such that  $(e_1, e_2) \in \mathcal{E}'$ ,  $m \in \mathcal{M}'$ , and  $m \circ e_1 = f_1$  and  $m \circ e_2 = f_2$ .

*Remark 3.6.* If a category has binary coproducts, a (*unique*)  $\mathcal{E}$ - $\mathcal{M}'$  factorization induces a (*unique*)  $\mathcal{E}'$ - $\mathcal{M}'$  pair factorization in a canonical way by defining

$$\mathcal{E}' := \{(e_A : A \rightarrow C, e_B : B \rightarrow C) \mid [e_A, e_B] : A + B \rightarrow C \in \mathcal{E}\} ,$$

i.e., as those pairs of morphisms with common codomain for which the induced morphism from the coproduct of their domains to their common codomain belongs to  $\mathcal{E}$  (see, e.g., [54, Fact 4.28]).

In the literature on graph transformation, normally the above defined (*unique*)  $\mathcal{E}$ - $\mathcal{M}'$  factorizations are used. While these also suffice for our purposes in Chapter 4, the stronger notion of a factorization system turns out to be a suitable precondition for one result in Chapter 5 (namely, Proposition 5.15). Of the many equivalent possibilities to define a factorization system (for an overview see, e.g., [2, Sect. 14]), we choose a definition that extends the definition of a *unique*  $\mathcal{E}$ - $\mathcal{M}'$  factorization.

**Definition 3.19** ( $\mathcal{E}$ - $\mathcal{M}'$  factorization system). Given a category  $\mathcal{C}$  with a *unique*  $\mathcal{E}$ - $\mathcal{M}'$  factorization, this constitutes an  $\mathcal{E}$ - $\mathcal{M}'$  *factorization system* if

- (1) both  $\mathcal{E}$  and  $\mathcal{M}'$  are closed under composition and each contains all isomorphisms of  $\mathcal{C}$  and
- (2) the factorization of a morphism  $f = m \circ e$  with  $e \in \mathcal{E}$ ,  $m \in \mathcal{M}'$  is not only unique up to isomorphism but unique up to *unique* isomorphism.

*Remark 3.7.* If  $\mathcal{E}$  is a class of epimorphisms or  $\mathcal{M}'$  is a class of monomorphisms (which is frequently the case in applications), the second requirement is automatically true by the respective cancellation property of the class  $\mathcal{E}$  or the class  $\mathcal{M}'$  and, thus, the above definition becomes equivalent to the one used as Definition 3.3.1 in [192] (which we will use later).

**Initial pushouts** *Initial pushouts* are a way to generalize the set-theoretic complement operator categorically. They have been used, e.g., to construct a complement rule that is the basis for the Amalgamation Theorem [88, Theorem 4.4].

**Definition 3.20** (Boundary and initial pushout). Given a morphism  $m : L \rightarrow G$  in an  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$ , an *initial pushout over  $m$*  is a pushout (1) over  $m$  (as depicted below) such that  $b_m \in \mathcal{M}$  and (1) factors uniquely through every pushout (3) over  $m$  where  $b'_m \in \mathcal{M}$ . That is, for every pushout (3) over  $m$  with  $b'_m \in \mathcal{M}$ , there exist unique morphisms  $b_m^*, c_m^*$  with  $b_m = b'_m \circ b_m^*$  and  $c_m = c'_m \circ c_m^*$ .

If (1) is an initial pushout,  $b_m$  is called *boundary over  $m$* ,  $B_m$  the *boundary object*, and  $C_m$  the *context object with respect to  $m$* .

$$\begin{array}{ccc}
 B_m & \xrightarrow{b_m} & L \\
 \downarrow x_m & & \downarrow m \\
 C_m & \xrightarrow{c_m} & G
 \end{array}
 \quad (1)
 \qquad
 \begin{array}{ccccc}
 & & b_m & & \\
 & & \curvearrowright & & \\
 B_m & \xrightarrow{b_m^*} & D & \xrightarrow{b'_m} & L \\
 \downarrow x_m & & \downarrow & & \downarrow m \\
 C_m & \xrightarrow{c_m^*} & E & \xrightarrow{c'_m} & G \\
 & & \curvearrowleft & & \\
 & & c_m & & 
 \end{array}
 \quad (2) \quad (3)$$

In an  $\mathcal{M}$ -adhesive category, the square (2) in the above definition is also a pushout and  $b_m^*, c_m^* \in \mathcal{M}$ . In **Graph**, the context graph  $C_m$  of an initial pushout over  $m : L \rightarrow G$  contains every element of  $G$  that has more than one preimage under  $m$  and every element that has none. Moreover, it contains *boundary nodes*, i.e., the further nodes from  $G$  minimally necessary such that this structure constitutes a subgraph of  $G$ .  $B_m$  consists of these boundary nodes and the elements of  $L$  that share their image under  $m$  with another element. In particular, whenever  $m$  is injective,  $C_m$  is the

minimal completion of  $G \setminus m(L)$  (componentwise set-theoretic difference on nodes and edges) to a subgraph of  $G$ , and  $B_m$  contains the boundary nodes one has to add [53, Example 6.2]. This described behavior generalizes to arbitrary categories of presheaves over **Set**. In particular, initial pushouts over arbitrary morphisms exist in such categories [187, Theorem 2, or 18, Lemma 5.13]. In Chapter 4, we will use initial pushouts to provide a necessary and sufficient condition for the analysis case of our Generalized Concurrency Theorem (Theorem 4.14).

Initial pushouts can be used to characterize matches for which a rule is applicable.

**Fact 3.11** (Existence and uniqueness of contexts [53, Theorem 6.4]). *In an  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$  with initial pushouts, given a plain rule  $p = (L \xleftarrow{l} K \xrightarrow{r} R)$  and a morphism  $m : L \rightarrow G$ , the rule  $p$  is applicable at match  $m$  if and only if there exists a morphism  $b_m^* : B_m \rightarrow K$  with  $l \circ b_m^* = b_m$ , where  $B_m$  is the boundary object with respect to  $m$  and  $b_m$  is the boundary over  $m$ , i.e., where (1) is the initial pushout over  $m$  (compare Fig. 3.17).*

$$\begin{array}{ccccc}
 & & b_m^* & & \\
 & & \curvearrowright & & \\
 B_m & \xleftarrow{b_m} & L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
 & & \downarrow m & & \downarrow & & \\
 & & G & \xleftarrow{\quad} & D & & \\
 & & \downarrow & & \downarrow & & \\
 C_m & \xrightarrow{c_m} & G & \xleftarrow{\quad} & D & & 
 \end{array}$$

(1)

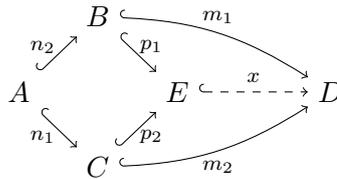
Figure 3.17: Initial pushout and context object

Moreover, initial pushouts enjoy the following closure property with respect to pushouts along  $\mathcal{M}'$ -morphisms.

**Fact 3.12** (Closure properties of initial pushouts [53, Lemma 6.5]). *Let  $(\mathcal{C}, \mathcal{M})$  be an  $\mathcal{M}$ -adhesive category with initial pushouts over  $\mathcal{M}'$  and  $\mathcal{M}'$  be closed under pushouts and pullbacks along  $\mathcal{M}$ -morphisms. Then, given the initial pushout (1) over a morphism  $a \in \mathcal{M}'$  and a pushout (2) along*



**Definition 3.21** ( $\mathcal{M}$ -effective unions). An  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$  has  $\mathcal{M}$ -effective unions if for each pushout of a pullback of a pair of  $\mathcal{M}$ -morphisms the induced mediating morphism belongs to  $\mathcal{M}$  as well, i.e., in each diagram like the one depicted in Fig. 3.20 where the outer square is a pullback of  $\mathcal{M}$ -morphisms and the inner one a pushout, the induced morphism  $x$  is an  $\mathcal{M}$ -morphism.

Figure 3.20:  $\mathcal{M}$ -effective union

Adhesive categories have effective unions [141, Theorem 5.1.]. So do practically relevant examples like typed attributed graphs (which constitute an adhesive HLR category that is not adhesive) [18, Lemma 5.24]; see [29, Table 1] for a short overview. The category of simple graphs, i.e., of graphs with at most one edge from one node to another, is a prominent example of an  $\mathcal{M}$ -adhesive category without  $\mathcal{M}$ -effective unions for  $\mathcal{M}$  being the class of regular monomorphisms: Johnstone et al. give a simple example showing that this category does not have  $\mathcal{M}$ -effective unions where  $\mathcal{M}$  is the class of regular monomorphisms [120, Corollary 20]; here, the regular monomorphisms are exactly the injective homomorphisms that reflect edges. Moreover, Heindel has shown this category to be  $\mathcal{M}$ -partial map adhesive (and, hence,  $\mathcal{M}$ -adhesive) with respect to that class  $\mathcal{M}$  [98, Lemma 3.11. and Lemma 3.13].

### 3.4 Typed Attributed (Triple) Graphs and Triple Graph Grammars

Finally, we introduce *typed attributed (triple) graphs* and *triple graph grammars* (TGGs). As these constitute the central object of our study—at least in Chapter 6—we illustrate these notions using our running example.

For model synchronization using (partial) triple graphs, the support of some concept of attribution is essential to be expressive enough to model examples with practical relevance. Numerous concepts for graph attribution have been suggested. We follow the approach first presented in [67] and in more detail in [53] because it seems to be the one that is employed most frequently in the literature. There, graphs are extended with special data nodes that stem from suitable algebras and attribute edges that connect between normal graph nodes or edges and data nodes. We present this approach to attribution of (triple) graphs. As a prerequisite, we shortly recall the definition of signatures and algebras in the spirit of [65].

**Definition 3.22** (Signatures and algebras). An (*algebraic*) *signature*  $\Sigma = (S, OP)$  consists of a set of sorts  $S$  and a family of operation symbols  $OP = (OP_{w,s})_{(w,s) \in S^* \times S}$ .

Given a signature  $\Sigma$ , a  $\Sigma$ -*algebra*  $A = ((A_s)_{s \in S}, (op_A)_{op \in OP})$  consists of

- a *carrier set*  $A_s$  for each sort  $s \in S$ ,
- a *constant element*  $c_A \in A_s$  for each constant symbol  $c : \rightarrow s \in OP$ , i.e., for each operation symbol over the empty word, and
- a mapping  $op_A : A_{s_1} \times \cdots \times A_{s_n} \rightarrow A_s$  for each operation symbol  $op : s_1 \dots s_n \rightarrow s \in OP$ .

Based on this definition of signatures and their algebras, we define attributed graphs.

**Definition 3.23** (Attributed graph and attributed graph morphism). Let  $\Sigma = (S, OP)$  be a signature with a designated subset  $S' \subseteq S$  of *attribute value sorts*. An *attributed graph*  $AG = (G, A)$  consists of an *E-graph*  $G$  and a  $\Sigma$ -algebra  $A = ((A_s)_{s \in S}, (op_A)_{op \in OP})$  such that  $\prod_{s \in S'} A_s = A_G$ . An *E-graph*  $G = (V_G, A_G, E_G, NA_G, EA_G, (src_j, tar_j)_{j \in \{G, NA_G, EA_G\}})$ , as schematically depicted in Fig. 3.21, consists of sets

- $V_G$  and  $A_G$ , the *graph* and *data nodes*, and
- $E_G, NA_G$ , and  $EA_G$ , the *graph*, *node attribute*, and *edge attribute edges*

and of *source* and *target functions*

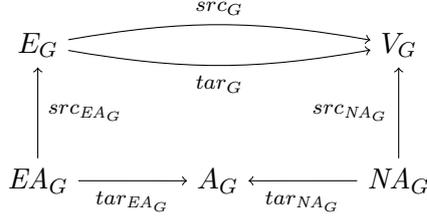
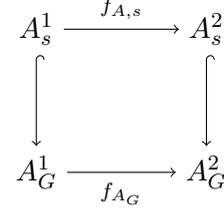
Figure 3.21: Schematic depiction of the structure of  $E$ -graphs

Figure 3.22: Compatibility of graph and algebra homomorphism

- $src_G, tar_G : E_G \rightarrow V_G$  for graph edges,
- $src_{NA_G} : NA_G \rightarrow V_G, tar_{NA_G} : NA_G \rightarrow A_G$  for node attribute edges, and
- $src_{EA_G} : EA_G \rightarrow E_G, tar_{EA_G} : EA_G \rightarrow A_G$  for edge attribute edges.

We call the sets  $V_G, E_G, NA_G$ , and  $EA_G$  of graph nodes and edges the *structural part* of an *attributed graph* and the data nodes  $A_G$  its *data part*.

An *attributed graph morphism*  $f : AG^1 \rightarrow AG^2$  between two attributed graphs  $AG^i = (G^i, A^i)$ , where  $i = 1, 2$ , consists of a pair  $f = (f_G, f_A)$  where  $f_G : G^1 \rightarrow G^2$  is an  $E$ -graph morphism and  $f_A : A^1 \rightarrow A^2$  is an algebra homomorphism such that for all  $s \in S'$  the induced diagram as depicted in Fig. 3.22 (where the vertical morphisms are the inclusions) commutes. An  $E$ -graph morphism  $f_G : G^1 \rightarrow G^2$  is a quintuple  $f_G = (f_{V_G}, f_{A_G}, f_{E_G}, f_{NA_G}, f_{EA_G})$  of functions  $f_X : X^1 \rightarrow X^2$  for  $X \in \{V_G, A_G, E_G, NA_G, EA_G\}$ , such that all components commute with the relevant source and target functions, e.g.,  $f_{V_G} \circ src_G^1 = src_G^2 \circ f_{E_G}$ .

Fixing a signature, one obtains a category of (typed) attributed graphs. These can then be used to define (typed) attributed triple graphs [54].

**Definition 3.24** (Typed attributed graphs and triple graphs). Let a signature  $\Sigma = (S, OP)$  and a designated subset  $S' \subseteq S$  of attribute value sorts be given.

- The *category of attributed graphs*, denoted as **AGraph**, has attributed graphs as objects where the algebra is a  $\Sigma$ -algebra and the data nodes are the corresponding elements of the carrier sets of the attribute value sorts and attributed graph morphisms as morphisms.

- Fixing an attributed graph  $ATG = (TG, Z)$  where  $Z$  is the final  $\Sigma$ -algebra, the *category of typed attributed graphs over  $ATG$*  is the slice category  $\mathbf{AGraph}/ATG$ , denoted as  $\mathbf{AGraph}_{ATG}$ .
- The *category of attributed triple graphs* is the functor category

$$[\bullet \leftarrow \bullet \rightarrow \bullet, \mathbf{AGraph}] ,$$

denoted as  $\mathbf{ATrG}$ . Given such a triple graph

$$G = (G_S \xleftarrow{\sigma_G} G_C \xrightarrow{\tau_G} G_T) ,$$

we refer to  $G_S, G_C$ , and  $G_T$  as *source, correspondence, and target graph (or part)*, respectively. The morphisms  $\sigma_G$  and  $\tau_G$  are called (*source and target correspondence morphism*).

- Fixing an attributed triple graph  $ATG = (TG_S \leftarrow TG_C \rightarrow TG_T)$  where each of the three graphs is attributed over the final  $\Sigma$ -algebra  $Z$ , the *category of typed attributed triple graphs over  $ATG$*  is the slice category  $\mathbf{ATrG}/ATG$ , denoted as  $\mathbf{ATrG}_{ATG}$ .

Note that a final  $\Sigma$ -algebra always exists. All carrier sets are just singletons (possibly identifying constant elements) and all mappings are constant (see [53, Definition B.11]). Regarding notation, to not overcrowd indices, we will also regularly use the letters  $S, C$ , and  $T$  as superscripts, instead, to denote that a certain object belongs to the source, correspondence, or target graph of a triple graph, respectively. In particular when working with a set of triple graphs  $\{G_i\}_{i \in I}$ , for example  $G_i^S$  will denote the  $i$ -th source graph and  $V_{G_i^S}$  its set of graph nodes.

*Remark 3.8.* When introducing the different variants of adhesiveness, we already mentioned  $\mathbf{Graph}$  as an example of an adhesive category and  $\mathbf{AGraph}$  as one of an adhesive HLR category (that is not adhesive). The class of monomorphisms  $\mathcal{M}$  with respect to which  $\mathbf{AGraph}$  is adhesive HLR is the class of morphisms that are injective on every graph component and isomorphisms on the algebra part [53]. Closedness of the different notions of adhesiveness under the slice construction (see Fact 3.2) immediately implies that  $\mathbf{AGraph}_{ATG}$  and  $\mathbf{ATrG}_{ATG}$  are adhesive HLR, as well.

In practical applications, it is often useful to assume the (triple) graphs of a rule to be typed over a *term algebra*  $T(X)$  where  $X$  is a set of variables

for the appropriate types. The (triple) graphs to be rewritten, in contrast, contain the common algebras for the given attribute value sorts. Normally, this will be the standard algebras of Booleans, Integers, Reals, Strings, etc. For a more thorough discussion see, e.g., [53].

**Example 3.1.** A typed attributed triple graph is depicted in Fig. 2.3; it is typed over the type graph from Fig. 2.1. Instead of displaying the typing morphism, we annotate each graph element with its type, i.e., with `Package`, `Folder`, etc. The hexagons in the middle constitute the correspondence graph. The arrows encode how an individual correspondence node is mapped by the correspondence morphisms. We also do not display attribute edges but write the type of that edge and the value it points to into its source node. Formally, for example, the string "name1" is a data node of the source graph  $G_S$  and a node attribute edge of type `name` connects the node `rootP` (of type `Package`) with this data node. Obviously, only data nodes with incoming attribution edge are displayed. The example does not make use of edge attribution.

We briefly introduce *triple graph grammars* (TGGs) [195] in the following; the extended version of TGGs that we consider for our model synchronization processes will be presented in more detail in Sect. 6.2.1. *Triple graph grammars* (TGGs) allow one to declaratively define consistency between models in a rule-based manner: If a triple graph is derivable via a given set of rules, it belongs to the language this set defines. Such a triple graph is then said to be *consistent*; its source and target graph *correspond to each other*. The two correspondence morphisms encode this correspondence on the level of individual elements. Classically, TGGs are defined for monotonic rules that are applied at injective matches [195]. However, when also considering attribution, this is not possible any longer: It is often necessary to be able to match two variables of the same type from the LHS of a rule to the same concrete value in a graph. To yet restrict transformations to rule applications that are at least injective on the structural part, *quasi-injective matches* have been introduced [104, 54].

**Definition 3.25** (Quasi-injective match). Given a plain rule  $p = (L \leftarrow K \hookrightarrow R)$  in  $\mathbf{AGraph}$  or  $\mathbf{AGraph}_{\text{ATG}}$ , a *quasi-injective match* for  $p$  is a match  $m = (m_G, m_A) : L \rightarrow G$  for  $p$  such that  $m_G$  is injective. Likewise, a *quasi-injective match*  $m = (m^S, m^C, m^T)$  with  $m^X = (m_G^X, m_A^X)$ , where

$X \in \{S, C, T\}$ , for a rule  $p$  in  $\mathbf{ATrG}$  or  $\mathbf{ATrG}_{\mathbf{ATG}}$  is a match for  $p$  such that all morphisms  $m_G^X$  are injective.

In the presence of application conditions, the use of quasi-injective matches necessitates the need to also adapt their semantics. We discuss this in Sect. 6.2.1.

**Definition 3.26** (Triple graph grammar). A *triple graph grammar* (TGG)  $GG = (\mathcal{R}, S, \mathbf{ATG})$  consists of a finite set of monotonic rules  $\mathcal{R}$  and a start triple graph  $S$  such that  $S$  and all graphs that occur in the rules are typed over  $\mathbf{ATG}$  and these as well as  $\mathbf{ATG}$  all have a finite structural part.

The *language defined by a TGG* is defined as

$$\mathcal{L}(GG) := \{G \in \mathbf{ATrG}_{\mathbf{ATG}} \mid \text{there exists } S \Rightarrow_{\mathcal{R}}^* G\} ,$$

i.e., it consists of all typed attributed triple graphs  $G$  that are derivable by finite sequences of applications of rules from  $\mathcal{R}$  starting at  $S$ . Here,  $\Rightarrow_{\mathcal{R}}$  denotes a transformation step via a rule from  $\mathcal{R}$  applied at a quasi-injective match and  $\Rightarrow_{\mathcal{R}}^*$  denotes its reflexive and transitive closure.

The *source language*  $\mathcal{L}^S(GG)$  of a TGG  $GG$  is the projection of its language to the source component, i.e.,

$$\mathcal{L}^S(GG) := \{G^S \mid \text{there exists } G = (G^S \leftarrow G^C \rightarrow G^T) \in \mathcal{L}(GG)\} .$$

The *target language* is defined analogously.

*Remark 3.9.* In this thesis, we always assume the start graph of a TGG to have an empty structural part and to contain suitable algebras of the given attribute value sorts (like the standard algebras for Boolean, Integer, Real, or String) as data part. Abusing notation, we denote this triple graph with  $\emptyset$ . Thus, formally,  $V_{\emptyset}^X = E_{\emptyset}^X = NA_{\emptyset}^X = EA_{\emptyset}^X = \emptyset$  for  $X \in \{S, C, T\}$ , whereas all  $A_{\emptyset}^X$  contain the elements of the suitable algebras.

**Example 3.2.** The rule set depicted in Fig. 2.2, together with the empty triple graph as start graph, constitutes a triple graph grammar that is typed over the type triple graph from Fig. 2.1. All displayed rules are monotonic, i.e., they only create elements. The triple graphs from Fig. 2.3 or Fig. 2.8 are part of the language defined by that grammar while the one from Fig. 2.7 is not. In the rules,  $n$ ,  $n'$ , and  $s$  are variables of type

String and, as such, can be matched to arbitrary concrete strings in a given triple graph. If we would not allow for quasi-injective matching, it would not be possible to apply *CreateSub* and *CreateLeaf* at matches that map the variables  $n$  and  $s$  to the same value (whereas this might constitute a reasonable default).

The operationalization of triple rules in *source* and *forward* (or, analogously, *target* and *backward*) rules is central for work with TGGs. Given a rule, its source rule only performs the rule's actions on the source graph while its forward rule propagates these to correspondence and target graph. Here, we present the classic definition for monotonic rules [195]. We later extend this definition to arbitrary rules (Definition 5.6).

**Definition 3.27** (Source and forward rule). Given a monotonic rule  $r = (L \hookrightarrow R)$  with  $L = (L_S \xleftarrow{\sigma_L} L_C \xrightarrow{\tau_L} L_T)$ ,  $R = (R_S \xleftarrow{\sigma_R} R_C \xrightarrow{\tau_R} R_T)$ , and  $r = (r_S, r_C, r_T)$  its *source rule*  $r^S$  is defined as

$$r^S := (L_S \xleftarrow{\emptyset} \emptyset \xrightarrow{\emptyset} \emptyset) \xrightarrow{(r_S, \emptyset, \emptyset)} (R_S \xleftarrow{\emptyset} \emptyset \xrightarrow{\emptyset} \emptyset) ,$$

where  $\emptyset$  denotes the empty graph and the empty morphism, respectively.

The corresponding *forward rule*  $r^F$  is defined as

$$r^F := (R_S \xleftarrow{r_S \circ \sigma_L} L_C \xrightarrow{\tau_L} L_T) \xrightarrow{(id_{R_S}, r_C, r_T)} (R_S \xleftarrow{\sigma_R} R_C \xrightarrow{\tau_R} R_T) .$$

The crucial property of the above operationalization is that the application of a source rule followed by an application of its forward rule (consistently matched) produces the same result as an application of their underlying triple rule [195]. The deeper reason for this is that a triple rule is a concurrent rule of its source and its forward rule [52]. We will extend this result also to the case of non-monotonic rules in Theorem 5.27.

**Example 3.3.** The source rules of the TGG from our running example allow creating *Packages* and *Classes* on the source side without changing correspondence and target graph; they are not depicted. The corresponding forward rules are given in Fig. 2.5. They can be used to propagate changes performed with the source rules to correspondence and target part. As evident from the examples, the application of a source rule followed by the application of the corresponding forward rule amounts to the application of the original triple rule if matched correctly.

# 4 A Generalized Concurrent Rule Construction for Double-Pushout Rewriting

In this chapter, we develop a new kind of sequential composition of rules that extends the concurrent rule construction.

## 4.1 Introduction

The composition of transformation rules has long been a topic of interest for (theoretical) research in graph transformation. Classical kinds of rule composition are the ones of *parallel* and *concurrent* [68] as well as of *amalgamated rules* [32]. Considering the *double-pushout approach* to graph transformation, these rule constructions have been lifted from ordinary graphs to the general framework of  $\mathcal{M}$ -adhesive categories and from plain rules to such with application conditions [53, 61, 88, 58]. These central forms of rule composition have also been developed for other variants of transformation, like *single-* or *sesqui-pushout rewriting* [156, 157, 24, 26].

In this chapter, we are concerned with simultaneously generalizing two variants of sequential rule composition in the context of double-pushout rewriting.<sup>1</sup> We develop *generalized concurrent rules* (GCRs), which comprise concurrent as well as so-called *short-cut rules* [78]. The concurrent rule construction, on the one hand, is optimized concerning *transient* model elements: An element that is created by the first rule and deleted by the second does not occur in a concurrent rule. A model element that is deleted by the first rule, however, cannot be reused in the second one. A short-cut rule, on the other hand, takes a rule that only deletes elements and a monotonic rule (i.e., a rule that only creates elements) and combines them

---

<sup>1</sup>We only address double-pushout rewriting because this is the context in which we apply our new theory in Chapter 6.

into a single rule where elements that are deleted and recreated may be preserved throughout the process. GCRs fuse both effects, the omission of transient elements and the reuse of elements, into a single construction.

The reuse of elements that is enabled by short-cut rules has two distinct advantages. First, information can be preserved. In addition, a rule that reuses model elements instead of deleting and recreating them is often applicable more frequently since necessary context does not get lost: Considering the double-pushout approach to graph transformation, a rule with higher reuse satisfies the *dangling-edge condition* more often in general. Our construction of GCRs provides the possibility of reusing elements when sequentially composing arbitrary rules. Hence, it generalizes the restricted setting in which we defined short-cut rules. We present our new theory in the general and abstract framework of double-pushout rewriting in  $\mathcal{M}$ -adhesive categories [141, 53].

In Chapter 6, the *repair rules* that constitute one central ingredient for our model synchronization process are based on GCRs. Actually, short-cut rules would be enough for that purpose because the rules of a TGG are monotonic. However, in [78] we did not develop the theory of short-cut rules to a degree that would completely suffice for our purpose: We only addressed  $\mathcal{M}$ -matching and did not include application conditions. Both, general matching and at least negative application conditions are needed in Chapter 6. Instead of merely adding these features to the construction of short-cut rules, we opted for addressing these issues in a more fundamental way. By developing generalized concurrent rules, we contribute more comprehensively to the theory of algebraic graph transformation: We add an interesting new kind of sequential rule composition. We envision possibilities for application beyond model synchronization, for example, the automated construction of complex (language-preserving) editing operations from simpler ones (which are not monotonic in general). In this thesis, however, we confine ourselves to developing the theory of GCRs and applying the special case of short-cut rules to the problem of model synchronization.

The content and structure of this chapter are as follows. In Sect. 4.2, we introduce an additional running example for this chapter and motivate the construction of GCRs by contrasting GCRs to concurrent rules. In contrast to rules from a TGG, the rules from this example are not monotonic. Section 4.3 recalls the construction of short-cut rules as introduced in [78]. In Sect. 4.4, we develop the construction of GCRs and prove some

of their fundamental properties. After the construction (Sect. 4.4.1), we prove the property that motivates their construction, namely the fact that—compared to the application of a concurrent rule—the application of a GCR preserves selected elements (Proposition 4.2 in Sect. 4.4.2). Next, we prove that the construction of GCRs generalizes indeed both, the concurrent as well as the short-cut rule constructions (Sect. 4.4.3). Section 4.5 presents further central results for GCRs. The heart of that section is split into two parts—Sect. 4.5.1 and 4.5.2—that contain the same results, first for the case of  $\mathcal{M}$ -matching and then also for the general case. The reason for this is that in the case of general matching technical difficulties arise that might obscure the intuitive ideas behind the construction of GCRs and why they behave like they do. We think that the less complex situation in the case of  $\mathcal{M}$ -matching is more illuminating and therefore warrants an independent presentation despite the introduced repetition. To mitigate this, we delegate all proofs for the general case to Appendix B.1. Contentwise, both those sections are built in the same way: We first characterize in two different ways under which conditions the GCR construction results in a rule. Thereafter, we present the Generalized Concurrency Theorem (Theorem 4.9 resp. Theorem 4.14), the central result of this chapter. It states that subsequent rule applications can be synthesized into the application of a GCR and characterizes the conditions under which the application of a GCR can be equivalently split up into the subsequent application of its two underlying rules. Section 4.5 concludes with another important theorem that provides sufficient conditions under which—assuming a given grammar—the application of a GCR preserves the language of that grammar (Theorem 4.16 in Sect. 4.5.3). While the later proof of correctness of our model synchronization approach in Sect. 6.4.4 does not need to invoke that theorem, the theorem still provides a deep reason for why our model synchronization approach works. Finally, we consider related work in Sect. 4.6 and conclude in Sect. 4.7.

This chapter is based on [137] but significantly extends it. There we only developed the theory for the case of  $\mathcal{M}$ -matching. The extension to the case of general matching is new, as is the theorem characterizing language-preserving applications of generalized concurrent rules (Theorem 4.16). We originally introduced short-cut rules in [78] but the results in this chapter considerably exceed that work. A journal article based on this chapter is currently under review [138].

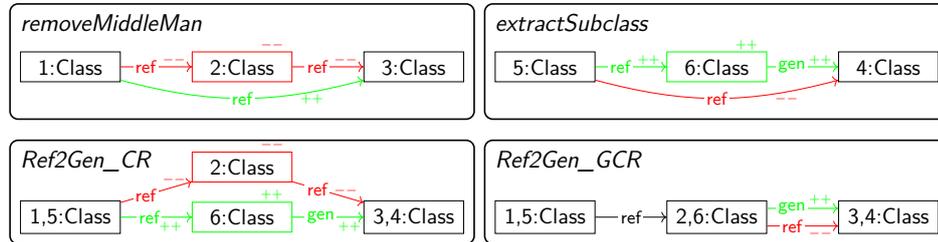


Figure 4.1: Two refactoring rules for class diagrams (first line) and sequentially composed rules derived from them (second line)

## 4.2 Running Example

In this section, we provide a short practical motivation for our new rule construction. It is situated in the context of model editing, more precisely class refactoring [74]. Refactoring is a technique to improve the design of a software system without changing its behavior. Transformation rules can be used to specify suitable refactorings of class models. For the sake of simplicity, we focus on the class structure here, where classes are just blobs. Two kinds of class relations are specified using typed edges, namely class references and generalizations; they are typed with *ref* and *gen*, respectively. All rules are depicted in an integrated fashion as introduced in Sect. 2.2.

The refactoring rules for our example are depicted in the first line of Fig. 4.1. The rule *removeMiddleMan* removes a Class that merely delegates the work to the real Class and directs the reference immediately to this, instead. The rule *extractSubclass* creates a new Class that is generalized by an already existing one. To not introduce unnecessary abstraction, the rule also redirects an existing reference to the newly introduced subclass.

Sequentially combining these two refactorings results in further ones. For example, this allows us to replace the second reference of a chain of two references with a generalization. The according concurrent rule is depicted as *Ref2Gen\_CR* in Fig. 4.1. It arises with *removeMiddleMan* as its first underlying rule and *extractSubclass* as its second, where Classes 1 and 5 and 3 and 4 are identified, respectively. The new reference-edge created by *removeMiddleMan* is deleted by *extractSubclass* and, thus, becomes transient. But the Class 2 that originally delegated the reference is deleted and cannot be reused. Instead, Class 6 has to be newly created to put

Classes 1 and 3 into this new context.

In many situations, however, it would be preferable to just reuse Class 2 and only replace the reference with a generalization. In this way, information (such as references, values of possible attributes, and layout information) is preserved. And, maybe even more importantly, when adopting the double-pushout approach, such a rule is typically applicable more often, namely also when the Class to which Class 2 is matched has adjacent edges. In our construction of generalized concurrent rules, we may identify elements deleted by the first rule and recreated by the second and decide to preserve them. In this example, our new construction allows one to also construct *Ref2Gen\_GCR* (Fig. 4.1) from *removeMiddleMan* and *extractSubclass*. In contrast to *Ref2Gen\_CR*, it identifies Class 2 deleted by *removeMiddleMan* and Class 6 created by *extractSubclass* and the respective incoming references and preserves them. This rule specifies a suitable variant of the refactoring *Remove Middle Man*, where the middle man is turned into a subclass of the real class.

### 4.3 The Short-Cut Rule Construction

We introduced *short-cut rules* as a special kind of sequential rule composition to construct (complex) edit rules from monotonic rules defining a grammar [78]. A *short-cut rule* composes a rule which only deletes (i.e., the inverse rule of a monotonic rule) with a monotonic rule into a single rule whose application has the same effect. The knack of the construction is that it allows identifying elements deleted by the first rule as recreated by the second. This results in these elements being preserved when the short-cut rule is applied. We defined the construction of short-cut rules in the context of adhesive categories but restricted to plain monotonic rules and monotonic matching. Here, we only briefly recall their construction to later show how the construction of GCRs encompasses the one of short-cut rules (Proposition 4.4). In particular, it will allow us to make use of short-cut rules in the setting of rules with negative application conditions matched at general matches in an adhesive HLR category. This is the setting in which we work in Chapter 6 to provide a solution to the basic model synchronization problem.

The construction of short-cut rules is based on a *common kernel* for

the two given rules. We shortly recall the definition of a common kernel for a pair of monotonic rules in the case of matches restricted to be monomorphisms (which is a special case of Definition 4.3). There, we understand a common kernel to embed a morphism  $k : K_\cap \hookrightarrow V$  into the morphisms  $l_1$  and  $r_2$ , i.e., into the left- and right-hand side of two given rules. However, in the construction of the short-cut rule, we consider the inverse rule  $r_1^{-1}$  of the first input rule  $r_1 : L_1 \hookrightarrow R_1$  (which means that the RHS of the original monotonic rule is its LHS); this is reflected in the following definition. In all of the following, we adapt the original notation from [78] to fit with the one used in this thesis.

**Definition 4.1** (Common kernel for monotonic rules). Given two plain, monotonic rules  $r_i : L_i \hookrightarrow R_i$ , where  $i = 1, 2$ , a *common kernel* for them is a monomorphism  $k : K_\cap \hookrightarrow V$  with monomorphisms  $u_i : K_\cap \hookrightarrow L_i$  and  $v_i : V \hookrightarrow R_i$  such that both induced squares, depicted below, constitute pullback squares.

$$\begin{array}{ccccc}
 L_1 & \xleftarrow{u_1} & K_\cap & \xrightarrow{u_2} & L_2 \\
 \downarrow r_1 & & \downarrow k & & \downarrow r_2 \\
 R_1 & \xleftarrow{v_1} & V & \xrightarrow{v_2} & R_2
 \end{array}$$

Given a common kernel  $k : K_\cap \hookrightarrow V$  for monotonic rules  $r_1$  and  $r_2$ , their short-cut rule  $r_1^{-1} \times_k r_2$  arises by gluing  $r_1^{-1}$  and  $r_2$  along  $k$ . The span  $L_1 \xleftarrow{u_1} K_\cap \xrightarrow{u_2} L_2$  contains the information on how to glue  $r_1^{-1}$  and  $r_2$  to receive the LHS  $L$  and the RHS  $R$  of the short-cut rule  $r_1^{-1} \times_k r_2$ . The morphism  $k : K_\cap \hookrightarrow V$  contains the information on how to construct its interface  $K$ . In case of **Graph**, for example,  $K$  is enhanced by including the elements of  $V \setminus K_\cap$ . This means, the difference between  $V$  and  $K$  specifies those elements that would have been deleted by  $r_1^{-1}$  and recreated by  $r_2$ .

**Definition 4.2** (Short-cut rule). In an adhesive category  $\mathbf{C}$ , given two monotonic rules  $r_i : L_i \hookrightarrow R_i$ , where  $i = 1, 2$ , and a common kernel  $k : L_\cap \hookrightarrow R_\cap$  for them, the *short-cut rule*

$$r_1^{-1} \times_k r_2 := (L \xleftarrow{l} K \xrightarrow{r} R)$$

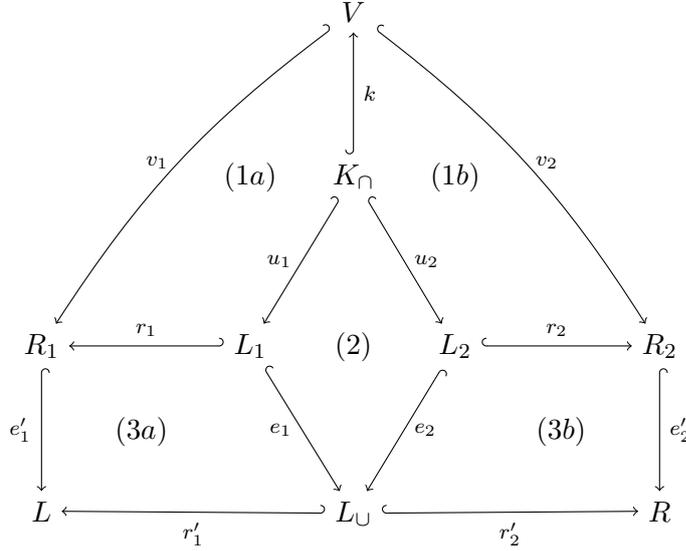


Figure 4.2: Construction of the LHS and RHS of short-cut rule  $r_1^{-1} \times_k r_2$

is computed by executing the following steps:

- (1) The union  $L_\cup$  of  $L_1$  and  $L_2$  along  $K_\cap$  is computed as pushout (2) in Fig. 4.2.
- (2) The LHS  $L$  of the short-cut rule  $r_1^{-1} \times_k r_2$  is constructed as pushout (3a) in Fig. 4.2.
- (3) The RHS  $R$  of the short-cut rule  $r_1^{-1} \times_k r_2$  is constructed as pushout (3b) in Fig. 4.2.
- (4) The interface  $K$  of the short-cut rule  $r_1^{-1} \times_k r_2$  is constructed as pushout (4) in Fig. 4.3.
- (5) Morphisms  $l : K \rightarrow L$  and  $r : K \rightarrow R$  are obtained by the universal property of the pushout (4).

The rules from Fig. 2.9 that have been discussed in Sect. 2.2 are examples for short-cut rules. They are derived from the rules that define the given

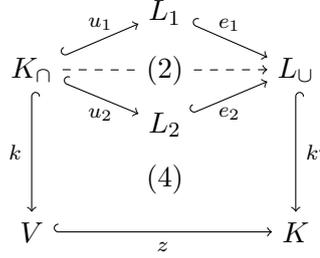


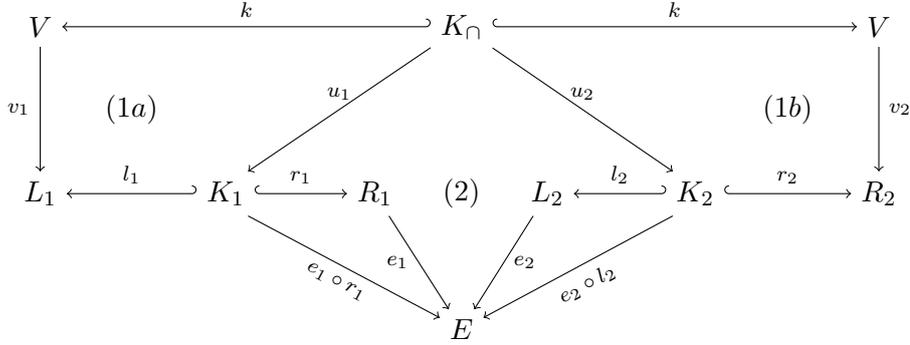
Figure 4.3: Construction of the interface  $K$  of  $r_1^{-1} \times_k r_2$

grammar. We do not further explain their computation. Instead, we will illustrate the computation of GCRs in more detail later on.

As central results pertaining to short-cut rules, we showed that they are indeed (linear) rules [78, Lemma 5] and proved the Short-Cut Theorem [78, Theorem 7]: An application of an inverse rule of a monotonic rule followed by one of a monotonic rule can be replaced (while preserving information) with one of a short-cut rule (synthesis); conversely (analysis), we provided sufficient conditions for when the application of a short-cut rule can be replaced by a sequential application of its underlying rules. Furthermore, in [79, Theorem 8] we gave sufficient conditions for applications of short-cut rules to be language-preserving with respect to a given grammar from which they are derived. In the following, we obtain all these results (and more) also for generalized concurrent rules. In particular, in the Generalized Concurrency Theorem we will present a necessary and sufficient condition for the analysis case, thus completely characterizing it.

#### 4.4 Generalized Concurrent Rules—Construction, Preservation, and Relations

In this section, we develop our construction of *generalized concurrent rules* (GCRs) in the context of an  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$  with an  $\mathcal{E}'$ - $\mathcal{M}$  pair factorization. We furthermore show their central preservation property and relate this new construction to the known ones of concurrent and of short-cut rules.

Figure 4.4: Compatibility of a common kernel with an  $E$ -dependency relation

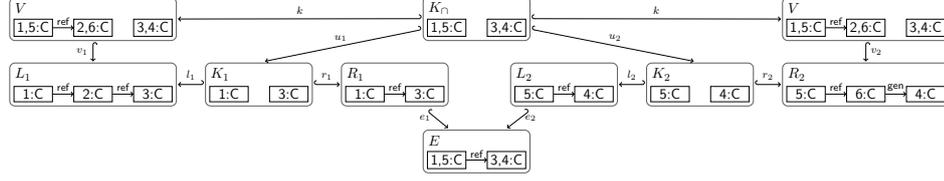
#### 4.4.1 Construction

Our construction of *generalized concurrent rules* combines the constructions of concurrent and short-cut rules into a single one. It is based on the choice of an  $E$ -dependency relation as well as of a *common kernel*. Intuitively, the  $E$ -dependency relation captures how both rules are intended to overlap (potentially producing transient elements) whereas the common kernel identifies elements that are deleted by the first rule and recreated by the second one. As both concepts identify parts of the interfaces of the involved rules, the construction of a GCR assumes an  $E$ -dependency relation and a common kernel that are *compatible*.

**Definition 4.3** ((Compatible) Common kernel). Given two rules  $\rho_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i, ac_i)$ , where  $i = 1, 2$ , a *common kernel* for them is an  $\mathcal{M}$ -morphism  $k : K_\cap \hookrightarrow V$  with morphisms  $u_i : K_\cap \rightarrow K_i$ ,  $v_1 : V \rightarrow L_1$ ,  $v_2 : V \rightarrow R_2$  such that both induced squares (1a) and (1b) in Fig. 4.4 are pullbacks.

Given additionally an  $E$ -dependency relation  $E = (e_1 : R_1 \rightarrow E, e_2 : L_2 \rightarrow E) \in \mathcal{E}'$  for  $\rho_1$  and  $\rho_2$ ,  $E$  and  $k$  are *compatible* if square (2) is a pullback.

*Remark 4.1.* In the case of  $\mathcal{M}$ -matching, we can safely assume that for any common kernel  $k$ , the embedding morphisms  $u_1$  and  $u_2$  are  $\mathcal{M}$ -morphisms (what then, as we will see, allows us to also assume that  $v_1$  and  $v_2$  stem

Figure 4.5: Common kernel for rules *removeMiddleMan* and *extractSubclass*

from  $\mathcal{M}$ ): Given an  $\mathcal{E}'$ - $\mathcal{M}$  pair factorization of morphisms, any sequence of two transformations  $t_1; t_2$  is  $E$ -related for the  $E$ -dependency relation  $E = (e_1, e_2) \in \mathcal{E}'$  that stems from factorizing  $(n_1, m_2)$ , i.e., from factorizing the comatch of the first and the match of the second transformation. As both  $n_1$  and  $m_2$  are  $\mathcal{M}$ -morphisms, the decomposition property of  $\mathcal{M}$ -morphisms implies that also  $e_1$  and  $e_2$  are. Then, closedness of  $\mathcal{M}$  under pullbacks ensures that  $u_1, u_2 \in \mathcal{M}$  for any common kernel that is compatible with  $E$ .

In the following, we will often suppress the morphisms  $u_i, v_i$  from our notation and just speak of a common kernel  $k : K_\cap \hookrightarrow V$ . As an  $\mathcal{M}$ -morphism  $k$  might constitute a common kernel for a pair of rules in different ways, we implicitly assume the embedding to be given.

**Example 4.1.** Figure 4.5 shows an  $E$ -dependency relation and a common kernel compatible with it for rules *removeMiddleMan* and *extractSubclass* (Fig. 4.1). The names of the nodes also indicate how the morphisms are defined. The concurrent rule for this  $E$ -dependency relation is the rule *Ref2Gen\_CR* in Fig. 4.1.

The following lemma is the basis for the construction of generalized concurrent rules; it directly follows from the definition of the interface  $K$  of a concurrent rule as pullback.

**Lemma 4.1.** *Given two rules  $\rho_1, \rho_2$ , an  $E$ -dependency relation  $E$ , and a common kernel  $k$  for  $\rho_1, \rho_2$  that is compatible with  $E$ , there exists a unique morphism  $p : K_\cap \rightarrow K$ , where  $K$  is the interface of the concurrent rule  $\rho_1 *_E \rho_2$ , such that  $k_i \circ p = e_i'' \circ u_i$  for  $i = 1, 2$  (compare the diagrams in Definitions 3.13 and 4.3). Whenever  $u_1, u_2 \in \mathcal{M}$ , also  $p \in \mathcal{M}$ .*

*Proof.* By compatibility of the common kernel  $k$  with  $E$  and the definition of a concurrent rule we have that

$$\begin{aligned} r'_1 \circ e''_1 \circ u_1 &= e_1 \circ r_1 \circ u_1 \\ &= e_2 \circ l_2 \circ u_2 \\ &= l'_2 \circ e''_2 \circ u_2 . \end{aligned}$$

Thus, by the universal property of the pullback computing  $K$ , we obtain a unique morphism  $p : K_\cap \rightarrow K$  such that  $k_i \circ p = e''_i \circ u_i$  for  $i = 1, 2$ . Moreover,  $p \in \mathcal{M}$  by decomposition of  $\mathcal{M}$ -morphisms, whenever  $u_1, u_2 \in \mathcal{M}$ .  $\square$

A GCR extends a concurrent rule by enhancing its interface  $K$  with the additional elements in  $V$  of a given common kernel. Formally, this means to compute a pushout along the just introduced morphism  $p$ .

**Construction 4.4.** Given two plain rules  $\rho_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$ , where  $i = 1, 2$ , an  $E$ -dependency relation  $E = (e_1 : R_1 \hookrightarrow E, e_2 : L_2 \hookrightarrow E) \in \mathcal{E}'$ , and a common kernel  $k : K_\cap \hookrightarrow V$  of  $\rho_1$  and  $\rho_2$  that is compatible with  $E$ , we construct the span

$$L \xleftarrow{l'} K' \xrightarrow{r'} R$$

as follows (compare Fig. 4.6):

- (1) Compute the concurrent rule  $\rho_1 *_E \rho_2 = (L \xleftarrow{l} K \xrightarrow{r} R)$ .
- (2) Compute  $K'$  as pushout of  $k$  along  $p$ , where  $p : K_\cap \rightarrow K$  is the unique morphism existing according to Lemma 4.1 (depicted twice in Fig. 4.6); in an  $\mathcal{M}$ -adhesive category this pushout always exists because  $k \in \mathcal{M}$ .
- (3) The morphism  $l' : K' \rightarrow L$  is the unique morphism with  $l' \circ p' = e'_1 \circ v_1$  and  $l' \circ k' = l'_1 \circ k_1$  that is induced by the universal property of the pushout computing  $K'$ . The morphism  $r'$  is defined analogously.

**Definition 4.5** (Generalized concurrent rule. Enhancement morphism). Given two rules  $\rho_1, \rho_2$  and an  $E$ -dependency relation  $E$  and a common kernel  $k$  of  $\rho_1$  and  $\rho_2$  that are compatible such that the span  $L \xleftarrow{l'} K' \xrightarrow{r'} R$

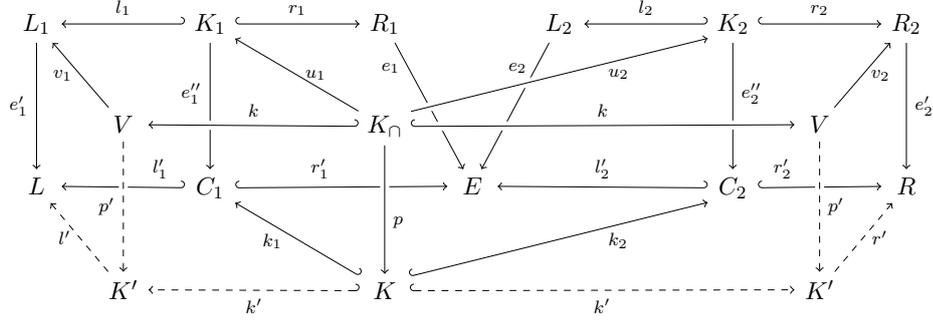


Figure 4.6: Construction of a generalized concurrent rule

obtained from Construction 4.4 consists of  $\mathcal{M}$ -morphisms, the *generalized concurrent rule* of  $\rho_1$  and  $\rho_2$ , given  $E$  and  $k$ , is defined as

$$\rho_1 *_{E,k} \rho_2 := (L \xleftrightarrow{l'} K' \xleftrightarrow{r'} R, ac)$$

with

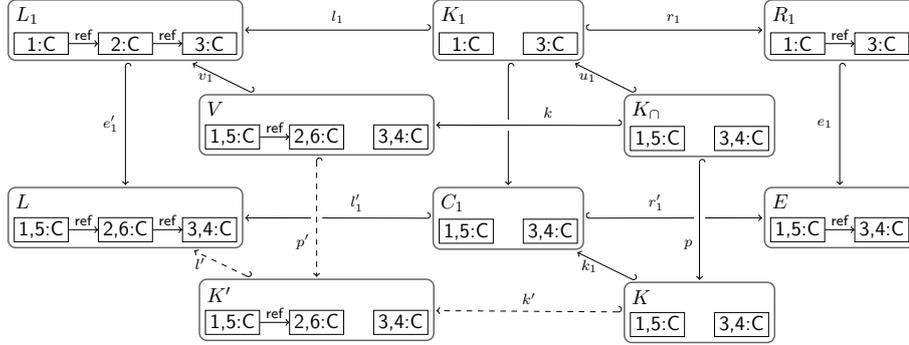
$$ac := \text{Shift}(e'_1, ac_1) \wedge \text{Left}(p', \text{Shift}(e_2, ac_2))$$

coinciding with the application condition of the concurrent rule  $\rho_1 *_{E,k} \rho_2$ .

The unique  $\mathcal{M}$ -morphism  $k' : K \hookrightarrow K'$  with  $l = l'_1 \circ k_1 = l' \circ k'$  and  $r = r'_2 \circ k_2 = r' \circ k'$  that is obtained directly from the construction is called *enhancement morphism*. We also say that  $\rho_1 *_{E,k} \rho_2$  is a GCR *enhancing*  $\rho_1 *_{E,k} \rho_2$ .

**Example 4.2.** *Ref2Gen\_GCR* is a GCR that enhances *Ref2Gen\_CR* (Fig. 4.1); it is constructed using the common kernel presented in Example 4.1. Figure 4.7 illustrates the computation of its interface  $K'$  and left-hand morphism  $l'$ . The pushout of  $k$  and  $p$  extends the interface of *Ref2Gen\_CR* by the Class 2,6 and its incoming reference.

*Note 4.1* (Assumptions and notation). For the rest of this chapter, we fix the following assumptions: We work in an  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$  with a given class  $\mathcal{E}'$  of pairs of morphisms with the same codomain such that

Figure 4.7: Computing the interface and the left-hand morphism of  $Ref2Gen\_GCR$ 

$\mathcal{C}$  possesses an  $\mathcal{E}'$ - $\mathcal{M}$  pair factorization.<sup>2</sup> Further categorical assumptions are mentioned as needed. Furthermore, we always assume two rules  $\rho_1, \rho_2$ , an  $E$ -dependency relation  $E$ , and a common kernel  $k$  for them to be given such that  $E$  and  $k$  are compatible. We consistently use the notations and names of morphisms as introduced above (and in Fig. 4.6).

#### 4.4.2 Preservation Property of Applications of GCRs

Next, we present a proposition that relates applications of concurrent rules with those of enhancing GCRs. It states that an application of a GCR leads to the same result as one of the enhanced concurrent rule; however, by its application, more elements are preserved (instead of being deleted and recreated). This is the distinct property that makes GCRs interesting. Moreover, this result will immediately imply the synthesis-case of the Generalized Concurrency Theorem.

**Proposition 4.2** (Preservation property of GCRs). *Let  $G_0 \Rightarrow_{\rho_1 *_{E} \rho_2, m} G_2$  be a transformation via concurrent rule  $\rho_1 *_{E} \rho_2$ , given by the span  $G_0 \xleftarrow{g_0} D \xrightarrow{g_2} G_2$ . For any GCR  $\rho_1 *_{E, k} \rho_2$  enhancing  $\rho_1 *_{E} \rho_2$ , there is a transformation  $G_0 \Rightarrow_{\rho_1 *_{E, k} \rho_2, m} G_2$ , given by a span  $G_0 \xleftarrow{g'_0} D' \xrightarrow{g'_2} G_2$ , and*

<sup>2</sup>We do not directly need this property in any of our proofs but it is assumed for the computation of application conditions of concurrent and, hence, also generalized concurrent rules. Moreover, it guarantees the existence of  $E$ -related transformations [54, Fact 5.29].

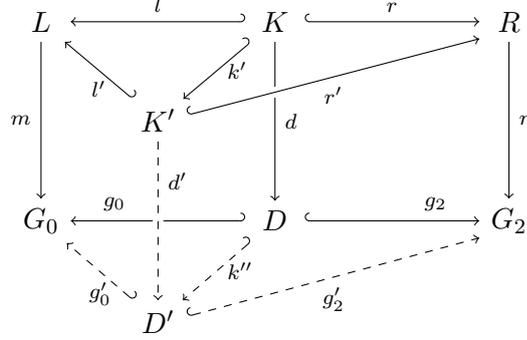


Figure 4.8: Constructing transformation via generalized concurrent rule from transformation via the concurrent rule

a unique  $\mathcal{M}$ -morphism  $k'' : D \hookrightarrow D'$  such that  $g'_i \circ k'' = g_i$  for  $i = 0, 2$ . Moreover,  $k''$  is an isomorphism if and only if the enhancement morphism  $k'$  is one.

*Proof.* Let  $G_0 \xleftarrow{g_0} D \xrightarrow{g_2} G_2$  be a span stemming from an application of  $\rho_1 *_E \rho_2$  at match  $m$  and  $d : K \rightarrow D$  be the according morphism from the interface of the rule to the context object of this transformation. Let  $k' : K \hookrightarrow K'$  be the enhancement morphism of  $\rho_1 *_E, k \rho_2$ , i.e., the unique  $\mathcal{M}$ -morphism with  $l' \circ k' = l$  and  $r' \circ k' = r$  provided by construction. Compare Fig. 4.8 for the following.

First, compute  $K' \xrightarrow{d'} D' \xleftarrow{k''} D$  as pushout of  $K' \xleftarrow{k'} K \xrightarrow{d} D$ . Note that  $k'' \in \mathcal{M}$  as it arises by pushout along  $k' \in \mathcal{M}$ . In particular,  $k''$  is an isomorphism if and only if  $k'$  is one by [Fact 3.9 \(4\)](#) (as this pushout is a pullback as well). Moreover, one computes

$$m \circ l' \circ k' = m \circ l = g_0 \circ d \text{ and } n \circ r' \circ k' = n \circ r = g_2 \circ d .$$

This means, by the universal property of that pushout, we obtain morphisms  $g'_i : D' \hookrightarrow G_i$  such that  $g'_i \circ k'' = g_i$  for  $i = 0, 2$ . By pushout decomposition, both induced squares (the front squares in Fig. 4.8) are pushouts. Moreover,  $m \models ac$ , where  $ac$  is the application condition of  $\rho_1 *_E, k \rho_2$ , since  $ac$  is also the application condition of  $\rho_1 *_E \rho_2$  (compare [Definitions 3.13](#) and [4.5](#)) and  $\rho_1 *_E \rho_2$  is applicable at  $m$ . In particular,  $G_0 \xleftarrow{g'_0} D' \xrightarrow{g'_2} G_2$  is a

transformation from  $G_0$  to  $G_2$  via  $\rho_1 *_{E,k} \rho_2$  at match  $m$  and the desired morphism  $k''$  exists.  $\square$

**Example 4.3.** Every match for the concurrent rule  $Ref2Gen\_CR$  is also one for the GCR  $Ref2Gen\_GCR$ . Moreover, the two results of the according applications will be isomorphic. The above proposition, however, formally captures that the application of  $Ref2Gen\_CR$  will delete more elements than the one of  $Ref2Gen\_GCR$ . The graph intermediately arising during the application of the former (not containing the class to which Class 2 had been matched) properly embeds into the one arising during the application of the latter.

### 4.4.3 Relating Generalized Concurrent Rules to other Variants of Rule Composition

In this section, we relate generalized concurrent rules to other variants of rule composition.

**Concurrent Rules** are the established technique of sequential rule composition in double-pushout rewriting. By definition, the left- and right-hand sides of a GCR coincide with the ones of the concurrent rule it enhances. One also directly obtains that a GCR coincides with its underlying concurrent rule if and only if its common kernel  $k$  is chosen to be an isomorphism.

**Proposition 4.3** (A concurrent rule is a GCR). *Given a concurrent rule  $\rho_1 *_{E,k} \rho_2$  and a GCR  $\rho_1 *_{E,k} \rho_2$  enhancing it, the enhancement morphism  $k' : K \hookrightarrow K'$  is an isomorphism if and only if  $k$  is one. In particular, whenever pullbacks of pairs of morphisms from  $\mathcal{E}'$  exist,  $\rho_1 *_{E,k} \rho_2$  coincides with  $\rho_1 *_{E,k} \rho_2$  (up to isomorphism) for  $k = id_{K_\cap}$ , where  $K_\cap$  is obtained by pulling back  $(e_1 \circ r_1, e_2 \circ l_2)$ .*

*Proof.* First, since pushouts and pullbacks along isomorphisms result in isomorphisms again (Fact 3.9(4)), and since in  $\mathcal{M}$ -adhesive categories pushouts along  $\mathcal{M}$ -morphisms are pullbacks,  $k'$  is an isomorphism if and only if  $k$  is one.

Furthermore, for every  $E$ -dependency relation  $E = (e_1, e_2)$ , if the pullback of that pair of morphisms exists, we can construct  $id_{K_\cap}$  as compatible common kernel as follows: We obtain  $u_i : K_\cap \rightarrow K_i$ , where  $i = 1, 2$ , by

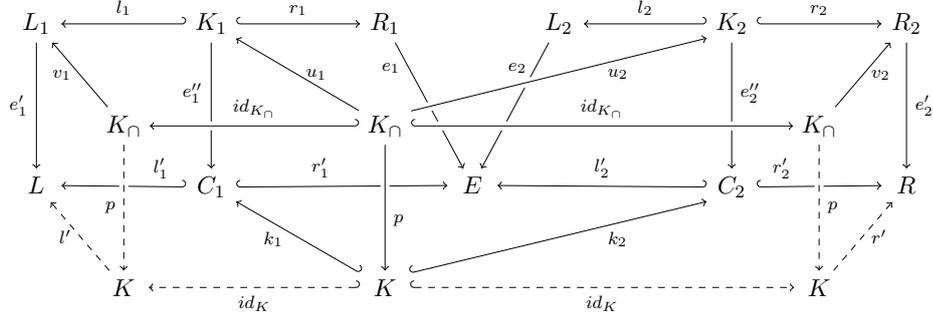


Figure 4.9: Concurrent rule as generalized concurrent rule (where  $v_1 = l_1 \circ u_1$  and  $v_2 = r_2 \circ u_2$ )

pulling back the pair of morphisms  $(e_1 \circ r_1, e_2 \circ l_2)$  (see Fig. 4.9). This pullback then exists because  $r_1, l_2 \in \mathcal{M}$ . Thus, it suffices to ensure that there are suitable morphisms  $v_1 : V = K_\cap \rightarrow L_1, v_2 : V = K_\cap \rightarrow R_2$  such that  $k$  indeed constitutes a common kernel for  $\rho_1$  and  $\rho_2$  compatible with  $E$ . This is guaranteed by Fact 3.9 (5) when setting  $v_1 := l_1 \circ u_1 : V = K_\cap \rightarrow L_1, v_2 := r_2 \circ u_2 : V = K_\cap \rightarrow R_2$  since  $l_1$  and  $r_2$  are monic (as  $\mathcal{M}$ -morphisms).

Finally, without loss of generality, we obtain  $k' = id_K : K \hookrightarrow K$  and  $p' = p$ . In particular,  $l' = l'_1 \circ k_1 = l$  and  $r' = r'_2 \circ k_2 = r$ .  $\square$

**Short-cut rules** are the very specific kind of sequentially composed rules that we recalled in Sect. 4.3. In an adhesive category, given a rule that only deletes and a rule that only creates, a short-cut rule combines their sequential effects into a single rule that allows identifying elements that are deleted by the first rule as recreated by the second and to preserve them instead. The construction of GCRs we present here now fuses our construction of short-cut rules with the concurrent rule construction. This means, we lift that construction from its very specific setting (adhesive categories and monotonic, plain rules) to a far more general one ( $\mathcal{M}$ -adhesive categories and general rules with application conditions). We already mentioned that this is of practical relevance because in Chapter 6 we work in that more general setting.

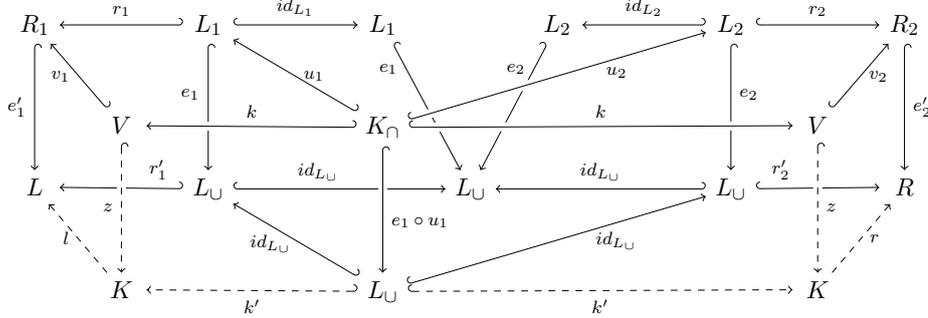


Figure 4.10: Short-cut rule as generalized concurrent rule

**Proposition 4.4** (A short-cut rule is a GCR). *Let  $\mathcal{C}$  be an adhesive category and the class  $\mathcal{E}'$  be such that it contains all pairs of jointly epic  $\mathcal{M}$ -morphisms. Let  $r_i = (r_i : L_i \hookrightarrow R_i)$ , where  $i = 1, 2$ , be two monotonic rules and  $k : K_\cap \hookrightarrow V$  a common kernel for them. Then the short-cut rule  $r_1^{-1} \times_k r_2$  coincides with the generalized concurrent rule  $r_1^{-1} *_{E,k} r_2$ , where  $E = (e_1, e_2)$  is given via the pushout of  $(u_1, u_2)$ .*

*Proof.* To make the constructions comparable, we consider the (equivalent) rules  $\rho_1^{-1} = (R_1 \xleftarrow{r_1} L_1 \xrightarrow{id_{L_1}} L_1)$  and  $\rho_2 = (L_2 \xleftarrow{id_{L_2}} L_2 \xrightarrow{r_2} R_2)$ , instead.

Applying the construction of GCRs to  $\rho_1^{-1}$  and  $\rho_2$  with  $E = (e_1 : L_1 \hookrightarrow L_\cup, e_2 : L_2 \hookrightarrow L_\cup)$  as  $E$ -dependency relation and  $k : K_\cap \hookrightarrow V$  as common kernel, results in the diagram depicted in Fig. 4.10 (where we employ the notation of Figs. 4.2 and 4.3): In this special case, the morphism  $p$  provided by Lemma 4.1 is already the morphism  $e_1 \circ u_1 = e_2 \circ u_2$ . Therefore, it is evident that this computes the same rule as the short-cut rule construction, i.e.,  $\rho_1^{-1} *_{E,k} \rho_2 = \rho_1^{-1} \times_k \rho_2$ . In particular, as pushout complements for a sequence of two morphisms with first morphism an identity always exist,  $E$  indeed is an  $E$ -dependency relation, as long as  $(e_1, e_2) \in \mathcal{E}'$ . Since this pair is computed as pushout along the pair of  $\mathcal{M}$ -morphisms  $(u_1, u_2)$ ,  $E$  is a pair of jointly epic  $\mathcal{M}$ -morphisms; a class of morphisms that is regularly included in (or even constitutes)  $\mathcal{E}'$  in practical applications.  $\square$

**Parallel and amalgamated rules** are further kinds of rules arising by composition. Whereas concurrent rules combine the sequential application

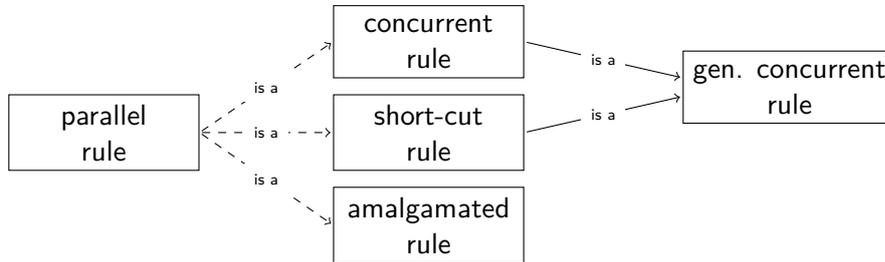


Figure 4.11: Relations between the different kinds of rule composition

of two rules, an amalgamated rule combines the application of two (or more) rules to the same object into the application of a single rule [32, 88]. In categories with coproducts, the parallel rule is just the sum of two rules; for plain rules it is a special case of the concurrent as well as of the amalgamated rule construction. A thorough presentation of all three forms of rule composition in the context of  $\mathcal{M}$ -adhesive categories, rules with application conditions, and general matching can be found in [58]. When introducing short-cut rules [78], we showed that their effect cannot be achieved by concurrent or amalgamated rules. Thus, by the above proposition, the same holds for GCRs; they indeed constitute a new form of rule composition. The relations between the different kinds of rule composition are summarized in Fig. 4.11. The lines from parallel rule are dashed as the indicated relations only hold in the absence of application conditions and, for short-cut rules, in the specific setting only in which these are defined.

## 4.5 Generalized Concurrent Rules—Central Results

In this section we present the central properties of generalized concurrent rules. We elaborate the conditions under which our construction results in a (linear) rule and characterize the kinds of rules that are derivable with our construction. As central result, we state the Generalized Concurrency Theorem.

We develop our theory in two steps, namely first for the case of  $\mathcal{M}$ -matching (Sect. 4.5.1) and subsequently for the general case (Sect. 4.5.2). Since the general case is much more involved, we think that in this way,

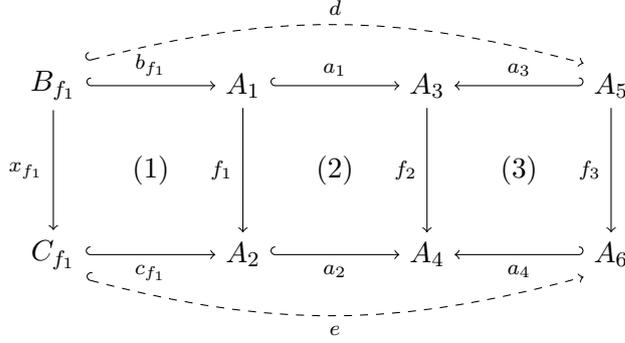


Figure 4.12: Interaction of initial pushouts with pullbacks

even if it involves a certain amount of duplication, it will be easier to convey the central intuitions for the construction that otherwise might get lost in the technical details.

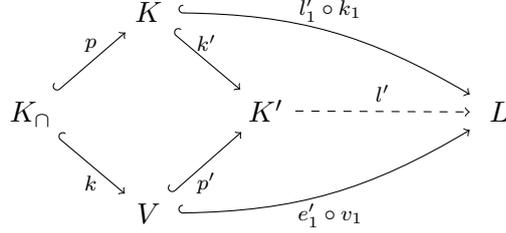
We start, however, with a general lemma that will be used in the proofs of both parts. It is concerned with the composition of an initial pushout with a pullback in  $\mathcal{M}$ -adhesive categories and seems to be new. Its proof is relegated to Appendix B.1.

**Lemma 4.5** (Interaction of initial pushouts with pullbacks). *In an  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$ , given a diagram like the one in Fig. 4.12 where (1) is an initial pushout, (2) a pullback with  $a_2 \in \mathcal{M}$ , and (3) a pushout such that  $a_3 \in \mathcal{M}$ , there are unique  $\mathcal{M}$ -morphisms  $d : B_{f_1} \hookrightarrow A_5$  and  $e : C_{f_1} \hookrightarrow A_6$  such that  $a_3 \circ d = a_1 \circ b_{f_1}$  and  $a_4 \circ e = a_2 \circ c_{f_1}$ . Moreover, the thereby induced square  $B_{f_1} \xrightarrow{d} A_5 \xrightarrow{f_3} A_6 \xleftarrow{e} C_{f_1} \xleftarrow{x_{f_1}} B_{f_1}$  is a pullback.*

*If  $(\mathcal{C}, \mathcal{M})$  is even adhesive HLR, this result already holds without the assumption  $a_2 \in \mathcal{M}$ . However, in this case  $d, e \in \mathcal{M}$  does not need to hold.*

#### 4.5.1 Central Results for the Case of $\mathcal{M}$ -Matching

The central assumption in this section is that all rules are applied using  $\mathcal{M}$ -morphisms as matches only. As discussed in Remark 4.1, this allows us to assume that also the morphisms  $u_i : K_\cap \hookrightarrow K_i$ , where  $i = 1, 2$ , are  $\mathcal{M}$ -morphisms. This will be the central technical ingredient that allows for

Figure 4.13: Obtaining  $l' \in \mathcal{M}$  via  $\mathcal{M}$ -effective unions

much simpler statements of the results and much simpler proofs compared to the general case.

### Characterizing Derivable Generalized Concurrent Rules ( $\mathcal{M}$ -Matching)

First, we characterize the GCRs derivable from a given pair of rules. We do so in two different ways, namely (i) characterizing possible choices for the morphisms  $v_1, v_2$  in a common kernel and (ii) characterizing the possible choices for enhancement morphisms  $k'$ .

**Proposition 4.6** (Embedding characterization of GCRs for  $\mathcal{M}$ -matching). *Let  $(\mathcal{C}, \mathcal{M})$  be an  $\mathcal{M}$ -adhesive category with  $\mathcal{M}$ -effective unions, and let  $k : K_\cap \hookrightarrow V$  be a common kernel for two rules  $\rho_i$  where the embedding morphisms  $u_i : K_\cap \hookrightarrow K_i$  are both  $\mathcal{M}$ -morphisms ( $i = 1, 2$ ).*

- (1) *The application of Construction 4.4 results in a GCR  $\rho_1 *_{E,k} \rho_2$  if and only if  $v_1, v_2 \in \mathcal{M}$ .*
- (2) *The assumption that  $\mathcal{M}$ -effective unions exist is necessary for this result to hold.*

*Proof.* First, if the application of Construction 4.4 results in a GCR, i.e., if  $l', r' \in \mathcal{M}$ , we immediately obtain  $l' \circ p' = e'_1 \circ v_1 \in \mathcal{M}$  by composition and then  $v_1 \in \mathcal{M}$  by decomposition of  $\mathcal{M}$ -morphisms. Analogously,  $r' \in \mathcal{M}$  implies  $v_2 \in \mathcal{M}$ .

For the converse direction, first Fig. 4.13 illustrates how  $l'$  is obtained by the universal property of the pushout computing  $K'$ . In particular,  $v_1 \in \mathcal{M}$  implies  $e'_1 \circ v_1 \in \mathcal{M}$  by composition of  $\mathcal{M}$ -morphisms. Thus, if the

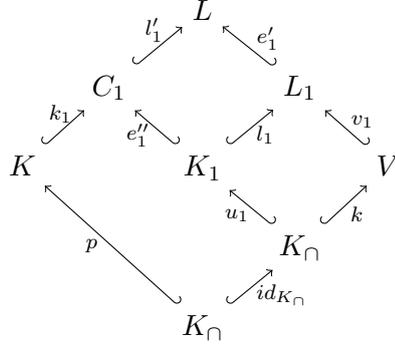


Figure 4.14: Proving the outer square of Fig. 4.13 to be a pullback

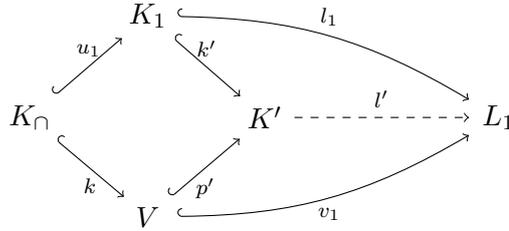


Figure 4.15: Counterexample to  $\mathcal{M}$ -effective unions

outer square is a pullback, unions being  $\mathcal{M}$ -effective implies that  $l' \in \mathcal{M}$ . To show the outer square to be a pullback, compare Fig. 4.14: The two top squares are pullbacks by assumption (the top square being a pushout along an  $\mathcal{M}$ -morphism) and the bottom square is a pullback according to Fact 3.9 (5). Then, pullback composition (Fact 3.9 (2)) implies that the whole square is a pullback, indeed.

Finally, we show  $\mathcal{M}$ -effective unions to be necessary for this result to hold by constructing an (abstract) counterexample otherwise. Whenever a category  $(\mathcal{C}, \mathcal{M})$  does not have  $\mathcal{M}$ -effective unions, there is a pullback of  $\mathcal{M}$ -morphisms  $l_1, v_1$  witnessing this; in particular  $l' \notin \mathcal{M}$  (as depicted in Fig. 4.15). Choose  $\mathcal{E}'$  such that it includes the class of pairs of jointly epimorphic  $\mathcal{M}$ -morphisms. Let  $\rho_1 = (L_1 \xleftarrow{l_1} K_1 \xrightarrow{id_{K_1}} K_1)$ ,  $\rho_2 = (K_\cap \xleftarrow{id_{K_\cap}} K_\cap \xrightarrow{id_{K_\cap}} K_\cap)$



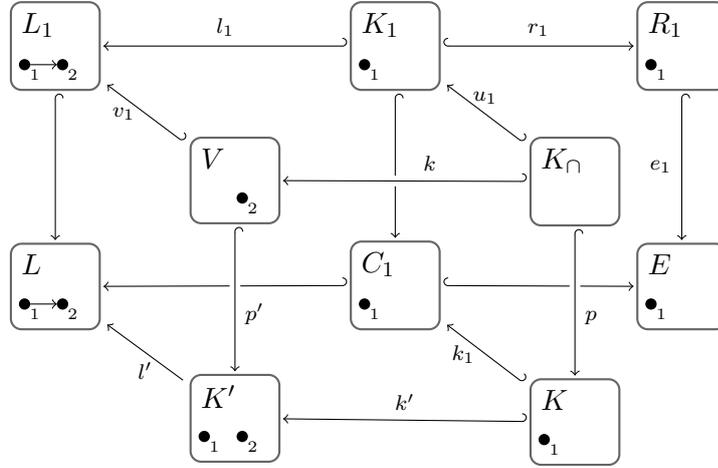


Figure 4.17: Concrete counterexample to the validity of Proposition 4.6 in the category of simple graphs

the interface of a GCR? The following simple example shows that not all of them do.

**Example 4.5.** In the category **Graph** (or **Set**), if  $p_1$  is the trivial rule  $\emptyset \leftarrow \emptyset \hookrightarrow \emptyset$  and  $p_2 = (\bullet \leftarrow \emptyset \hookrightarrow \bullet)$ , it is straightforward to verify that  $p_2$  can be derived as concurrent rule again (for the  $E$ -dependency object  $E = \bullet$ ). However,  $\bullet \leftarrow \bullet \hookrightarrow \bullet$  cannot be derived as GCR from these two rules since  $p_1$  does not delete a node.

It turns out that only elements that are deleted by the first rule and created by the second can be identified and incorporated into  $K'$ . The next proposition clarifies this connection using the language of initial pushouts.

**Definition 4.6** (Appropriately enhancing). In a category  $\mathcal{C}$  with initial pushouts, let  $\rho_1 *_E \rho_2 = L \xleftarrow{l} K \xrightarrow{r} R$  be an  $E$ -concurrent rule and  $k' : K \hookrightarrow K'$  be an  $\mathcal{M}$ -morphism such that there exist  $\mathcal{M}$ -morphisms  $l' : K' \hookrightarrow L$  and  $r' : K' \hookrightarrow R$  with  $l' \circ k' = l$  and  $r' \circ k' = r$ . Then  $k'$  is called *appropriately enhancing* if the following holds (compare Fig. 4.18): The boundary and context objects  $B_{k'}$  and  $C_{k'}$  of the initial pushout over  $k'$  factorize via  $\mathcal{M}$ -morphisms  $s_L, s_R : B_{k'} \hookrightarrow B_{l_1}, B_{r_2}$  and  $t_L, t_R : C_{k'} \hookrightarrow C_{l_1}, C_{r_2}$  as pullback

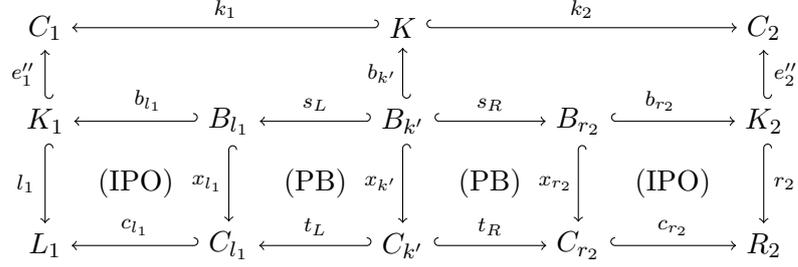
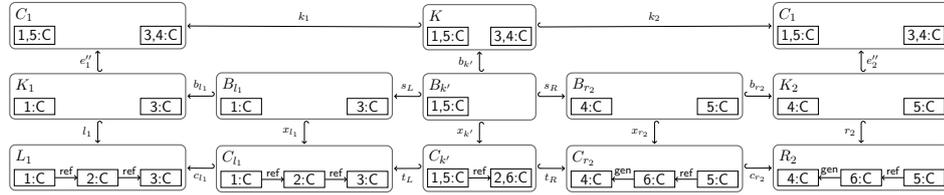
Figure 4.18: Definition of *appropriate enhancement*

Figure 4.19: Illustrating the property of appropriate enhancement

through the initial pushouts over  $l_1 : K_1 \hookrightarrow L_1$  and  $r_2 : K_2 \hookrightarrow R_2$  in such a way that  $k_1 \circ b_{k'} = e_1'' \circ b_{l_1} \circ s_L$  and  $k_2 \circ b_{k'} = e_2'' \circ b_{r_2} \circ s_R$ .

On the level of graph elements (in fact: arbitrary categories of presheaves over **Set**), this means the following: When considering  $K'$  as a subobject of  $L$  via  $l'$ , the elements of  $K' \setminus K$  have to be mapped to elements of  $L_1 \setminus K_1$ . When considering  $K'$  as a subobject of  $R$  via  $r'$ , the elements of  $K' \setminus K$  have to be mapped to elements of  $R_2 \setminus K_2$ .

**Example 4.6.** Figure 4.19 illustrates the notion of appropriate enhancement using our running example. The two inner squares constitute pullbacks, which means that the additional elements of  $K'$ , namely Class 2,6 and its incoming reference, are mapped via  $t_L$  to elements deleted by *removeMiddleMan* and via  $t_R$  to elements created by *extractSubclass*. Moreover, for both  $C_1$  and  $C_2$  the two possible ways to map Class 1,5 from  $B_{k'}$  to them coincide.

**Proposition 4.7** (Enhancement characterization for  $\mathcal{M}$ -matching). *Let  $(\mathcal{C}, \mathcal{M})$  be an  $\mathcal{M}$ -adhesive category with initial pushouts. Given an  $E$ -*

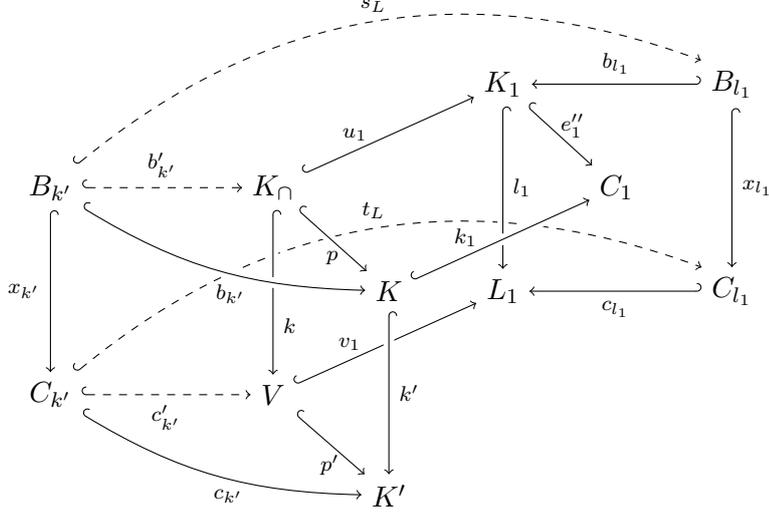


Figure 4.20: Proving appropriate enhancement

concurrent rule  $\rho_1 *_{E} \rho_2 = L \xleftarrow{l} K \xrightarrow{r} R$  and an  $\mathcal{M}$ -morphism  $k' : K \hookrightarrow K'$  such that there exist  $\mathcal{M}$ -morphisms  $l' : K' \hookrightarrow L$  and  $r' : K' \hookrightarrow R$  with  $l' \circ k' = l$  and  $r' \circ k' = r$ , the span  $L \xleftarrow{l'} K' \xrightarrow{r'} R$  is derivable as a GCR  $\rho_1 *_{E,k} \rho_2$  for a common kernel  $k$  that is embedded via  $\mathcal{M}$ -morphisms  $u_i : K_{\cap} \hookrightarrow K_i$ , where  $i = 1, 2$ , if and only if  $k'$  is appropriately enhancing.

*Proof.* First, let  $L \xleftarrow{l'} K' \xrightarrow{r'} R$  be a GCR, i.e., assume that there exists a common kernel  $k$  for  $\rho_1$  and  $\rho_2$  that is compatible with  $E$  such that  $\rho_1 *_{E,k} \rho_2 = L \xleftarrow{l'} K' \xrightarrow{r'} R$ . Compare Fig. 4.20 for the following.

The solid squares show the relevant part of the computation of  $\rho_1 *_{E,k} \rho_2$  (namely, the embedding of the common kernel  $k$  into  $l_1$ , the computation of  $K'$  as a pushout, and a part of the computation of  $K$  via pullback—the morphism  $k_1$ ) and the initial pushouts over  $k'$  (the bent square at the front) and  $l_1$  (the square to the very right). First, Fact 3.12 ensures that the boundary and context objects  $B_{k'}$  and  $C_{k'}$  of the initial pushout over  $k'$  also constitute the boundary and context objects of the initial pushout over  $k$  (as  $k'$  is computed as pushout along  $k$ ), where the necessary morphisms  $b'_{k'}, c'_{k'}$  are induced by initiality. Then, the sequence of three squares at the

center of the figure is as in the situation of Lemma 4.5: the first square is an initial pushout followed by a pullback and the opposing square is a pushout. Hence, Lemma 4.5 implies the existence of  $s_L$  and  $t_L$  yielding the required pullback square (bent square in the background).

Finally, using the commutativity of the whole diagram we compute

$$\begin{aligned} k_1 \circ b_{k'} &= k_1 \circ p \circ b'_{k'} \\ &= e''_1 \circ u_1 \circ b'_{k'} \\ &= e''_1 \circ b_{l_1} \circ s_L \end{aligned}$$

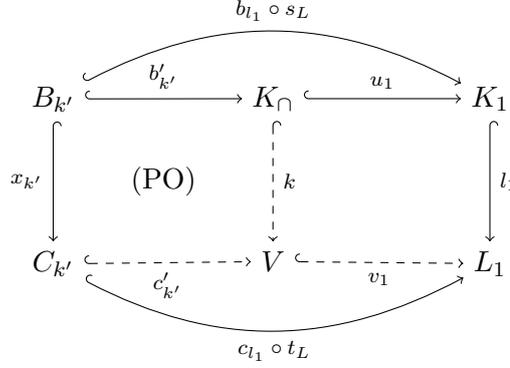
as was to be shown. The existence of  $s_R : B_{k'} \hookrightarrow B_{r_2}$  and  $t_R : C_{k'} \hookrightarrow C_{r_2}$  such that the induced square is a pullback and  $k_2 \circ b_{k'} = e''_2 \circ b_{r_2} \circ s_R$  is shown completely analogously.

For the other direction, assume  $k'$  to be appropriately enhancing. Using the equations  $k_1 \circ b_{k'} = e''_1 \circ b_{l_1} \circ s_L$  and  $k_2 \circ b_{k'} = e''_2 \circ b_{r_2} \circ s_R$ , we first compute

$$\begin{aligned} r'_1 \circ e''_1 \circ b_{l_1} \circ s_L &= r'_1 \circ k_1 \circ b_{k'} \\ &= l'_2 \circ k_2 \circ b_{k'} \\ &= l'_2 \circ e''_2 \circ b_{r_2} \circ s_R . \end{aligned} \tag{1}$$

We then compute  $(u_1 : K_\cap \hookrightarrow K_1, u_2 : K_\cap \hookrightarrow K_2)$  as pullback of  $(e_1 \circ r_1 = r'_1 \circ e''_1, e_2 \circ l_2 = l'_2 \circ e''_2)$ . Then, the universal property of this pullback and Eq. (1) imply the existence of a unique morphism  $b'_{k'} : B_{k'} \rightarrow K_\cap$  such that  $u_1 \circ b'_{k'} = b_{l_1} \circ s_L$  and  $u_2 \circ b'_{k'} = b_{r_2} \circ s_R$ ; moreover,  $b'_{k'} \in \mathcal{M}$  by decomposition of  $\mathcal{M}$ -morphisms. We then compute the object  $V$  as pushout of  $x_{k'}$  along this morphism  $b'_{k'}$ ; this results in Fig. 4.21: The left square is the computed pushout, resulting in the  $\mathcal{M}$ -morphism  $k : K_\cap \hookrightarrow V$ . The outer square is a pullback, which exists by assumption (composing the assumed pullback with the initial pushout over  $l_1$ ). The morphism  $v_1$  is obtained by the universal property of the pushout and makes the whole diagram commute. In particular, as the diagram commutes,  $l_1 \in \mathcal{M}$ , the outer square is a pullback, and the left square a pushout,  $\mathcal{M}$  pullback-pushout decomposition is applicable and ensures the right square to constitute a pullback. Completely analogously, one constructs the pullback embedding  $k$  into  $r_2$ .

With almost the same argument we ensure that  $k$  computes the correct interface  $K'$  (see Fig. 4.22): The left square, again, is the pushout computing

Figure 4.21: Obtaining the common kernel  $k : K_{\cap} \hookrightarrow V$  via pushout

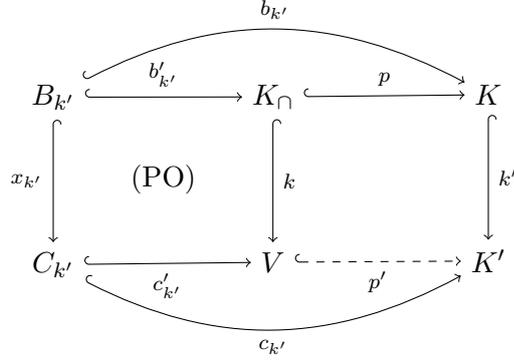
$V$  and the outer square the given initial pushout over  $k'$ . The morphism  $p$  is obtained as in Lemma 4.1; in particular,  $k_1 \circ p = e'_1 \circ u_1$ . Using this, we compute

$$\begin{aligned} k_1 \circ p \circ b'_{k'} &= e''_1 \circ u_1 \circ b'_{k'} \\ &= e''_1 \circ b_{l_1} \circ s_L \\ &= k_1 \circ b_{k'} . \end{aligned}$$

In particular, since  $k_1$  is a monomorphism,  $p \circ b'_{k'} = b_{k'}$ . This makes the upper part of Fig. 4.22 commute. Again, the universal property of the left pushout implies the existence of a morphism  $p'$  that makes the whole diagram commute. Furthermore, pushout decomposition implies the second square to be a pushout, as desired. Summarizing, we constructed a common kernel  $k$  that is compatible with  $E$  and computes the given object  $K'$  as interface.  $\square$

In **Graph** (and similar categories), the result above characterizes a GCR as a rule whose interface  $K'$  enhances the interface  $K$  of the enhanced concurrent rule by identifying elements of  $L_1 \setminus K_1$  and  $R_2 \setminus K_2$  with each other and including them in  $K'$ .

**Corollary 4.8** (Enhancement characterization in the category **Graph**). *In the category of (typed/attributed) graphs, given an  $E$ -dependency relation  $E$  for two rules  $\rho_1$  and  $\rho_2$ , every GCR  $\rho_1 *_{E,k} \rho_2$  enhancing  $\rho_1 *_{E} \rho_2$  is*

Figure 4.22: Ensuring  $k$  to compute  $K'$ 

obtained in the following way:  $K'$  arises by adding new graph elements to  $K$  and  $l'$  and  $r'$  extend the morphisms  $l$  and  $r$  in such a way that they (i) remain injective graph morphisms and (ii) under  $l'$  the image of every newly added element is in  $L_1 \setminus K_1$  and in  $R_2 \setminus K_2$  under  $r'$ .

Assuming finite graphs only, the number of GCRs enhancing a concurrent rule  $\rho_1 *_E \rho_2$  may grow factorially in  $\min(|L_1 \setminus K_1|, |R_2 \setminus K_2|)$ .

*Proof.* The categories of graphs, typed graphs, and attributed graphs are all known to meet the conditions of Proposition 4.7, i.e., they are  $\mathcal{M}$ -adhesive categories and have initial pushouts. Thus, the first part of the statement follows directly from the set-theoretic characterization of initial pushouts in these categories ([53, Example 6.2]).

For the second statement, consider the case of discrete graphs (i.e., of graphs without edges). Given a concurrent rule  $\rho_1 *_E \rho_2$ , there are

$$\sum_{i=0}^{\min(|L_1 \setminus K_1|, |R_2 \setminus K_2|)} i! \cdot \binom{|L_1 \setminus K_1|}{i} \cdot \binom{|R_2 \setminus K_2|}{i}$$

ways to derive a generalized concurrent rule from it as one extends  $K$  to  $K'$  by adding  $i$  elements to it which have to be mapped injectively to elements from  $L_1 \setminus K_1$  and  $R_2 \setminus K_2$ , respectively.  $\square$

### A Generalized Concurrency Theorem— $\mathcal{M}$ -Matching

In this section, we present our Generalized Concurrency Theorem that clarifies how sequential applications of two rules relate to an application of a GCR derived from them. The classical Concurrency Theorem states that a sequence of two rule applications can be replaced by an application of a concurrent rule (synthesis) and vice versa (analysis). The synthesis is still possible in the case of GCRs. The analysis, however, holds under certain conditions only. Next, we illustrate how the analysis might fail. Subsequently, we state the Generalized Concurrency Theorem.

**Example 4.7.** When *Ref2Gen\_GCR* (see Fig. 4.1) is applied at a match that maps Class 2,6 to a node with incoming or outgoing references or generalizations beyond the two references required by the match, the dangling-edge condition prevents the applicability of the underlying first rule *removeMiddleMan* at the induced match. The deletion of Class 2 would not be possible because of these additional adjacent edges. Hence, the analysis of the application of *Ref2Gen\_GCR* into sequential applications of *removeMiddleMan* and *extractSubclass* fails in that situation.

**Theorem 4.9** (Generalized Concurrency Theorem for  $\mathcal{M}$ -matching). *Let  $(\mathcal{C}, \mathcal{M})$  be an  $\mathcal{M}$ -adhesive category with  $\mathcal{E}'$ - $\mathcal{M}$  pair factorization and  $\mathcal{M}$ -effective unions.*

**Synthesis.** *For each  $E$ -related transformation sequence*

$$G_0 \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow_{\rho_2, m_2} G_2$$

*with  $m_1, m_2 \in \mathcal{M}$ , there exists a direct transformation*

$$G_0 \Rightarrow_{\rho_1 *_{E, k} \rho_2, m} G_2$$

*with  $m \in \mathcal{M}$  for each common kernel  $k$  for  $\rho_1$  and  $\rho_2$  that is compatible with  $E$ .*

**Analysis.** *Given a direct transformation*

$$G_0 \Rightarrow_{\rho_1 *_{E, k} \rho_2, m} G_2$$

*with  $m \in \mathcal{M}$ , there exists an  $E$ -related transformation sequence*

$$G_0 \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow_{\rho_2, m_2} G_2$$

*with  $m_1 = m \circ e'_1$  and  $m_1, m_2 \in \mathcal{M}$  if and only if  $G_0 \Rightarrow_{\rho_1, m_1} G_1$  exists, i.e., if and only if  $\rho_1$  is applicable at  $m_1$ .*

*Proof.* The *synthesis case* holds by virtue of the synthesis case of the Concurrency Theorem (see, e.g., [58, Theorem 4.17] for its statement in the context of  $\mathcal{M}$ -adhesive categories and rules with application conditions or [29, Theorem 2.7] for a statement in the same setting but specialized to  $\mathcal{M}$ -matching) and Proposition 4.2: Whenever such an  $E$ -related sequence of applications of  $\rho_1$  and  $\rho_2$  is given, the transformation  $G_0 \Rightarrow_{\rho_1 *_{E} \rho_2, m} G_2$  exists by the Concurrency Theorem, and, hence, the transformation  $G_0 \Rightarrow_{\rho_1 *_{E, k} \rho_2, m} G_2$  by Proposition 4.2. In particular, the Concurrency Theorem for the setting of  $\mathcal{M}$ -matching ensures that  $m \in \mathcal{M}$ .

For the *analysis case*, it suffices to show that  $\rho_1 *_{E} \rho_2$  is applicable at match  $m$  if  $\rho_1$  is applicable at  $m \circ e'_1$ . Applicability of  $\rho_2$  at a suitable match (from  $\mathcal{M}$ ) then, again, follows by the analysis case of the Concurrency Theorem (for  $\mathcal{M}$ -matching). The second direction of the stated equivalence is trivial. Notice that  $\mathcal{M}$ -matching allows us to assume that  $e_1 \in \mathcal{M}$  (see Remark 4.1) which implies  $e'_1 \in \mathcal{M}$  (by closedness of  $\mathcal{M}$  under pullbacks) which, in turn, implies  $m \circ e'_1 \in \mathcal{M}$  (by closedness of  $\mathcal{M}$  under composition). Thus,  $m_1, m_2 \in \mathcal{M}$  is clear.

Figure 4.23 displays the substance of the proof that  $\rho_1 *_{E} \rho_2$  is applicable at match  $m$  if  $\rho_1$  is applicable at  $m \circ e'_1$ . The solid lines are given and the dashed ones are constructed throughout the proof.

First, the outer square to the left is the pushout (with context object  $D_1$ ) that exists because of the applicability of  $\rho_1$  at  $m_1 := m \circ e'_1$ . Furthermore, the solid bent square at the front is the pushout (with context object  $D'$ ) that exists because of the applicability of  $\rho_1 *_{E, k} \rho_2$  at  $m$ . To show that also  $\rho_1 *_{E} \rho_2$  is applicable at  $m$ , it suffices to construct a pushout complement  $D$  for  $m \circ l$  with  $l := l'_1 \circ k_1$ . We construct this pushout as composition of two pushouts.

First, since  $m, l_1 \in \mathcal{M}$ ,  $\mathcal{M}$  pushout-pullback decomposition ensures that pulling back  $m$  and  $g_1$  decomposes the outer left square into two pushouts. However, the unique pushout complement of  $e'_1 \circ l_1$  is known to be given by  $l'_1 \circ e''_1$  such that (up to isomorphism)  $C_1$  is the object resulting from pulling back  $m$  and  $g_1$ . In particular, we obtain  $d'_1 : C_1 \hookrightarrow D_1$  such that both the left squares are pushouts.

Subsequently, we compute  $C'_1$  as pushout of  $k_1$  and  $k'$ . Its universal property induces the morphism  $l''$ ; in particular,  $l' = l'' \circ k'_1$  and  $l'' \in \mathcal{M}$  by the existence of  $\mathcal{M}$ -effective unions if the surrounding square can be shown to be a pullback. This surrounding square appears as left bottom square in the

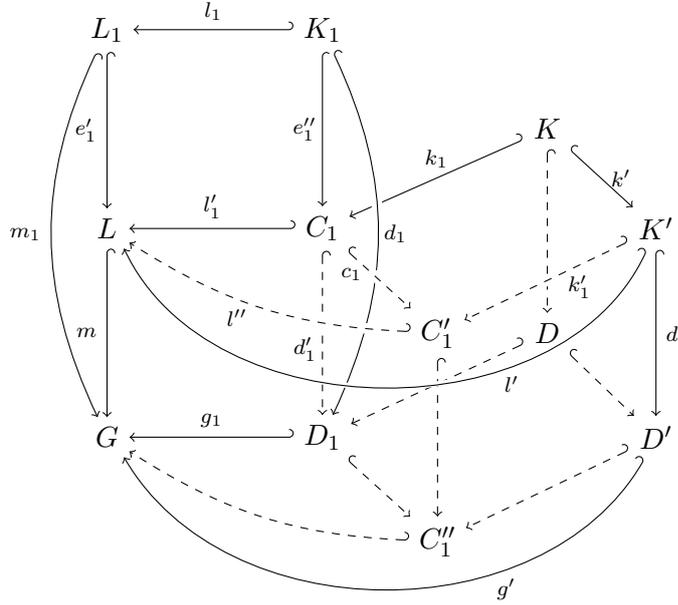


Figure 4.23: Proving  $\rho_1 *_E \rho_2$  to be applicable at  $m$

construction of the GCR (see Fig. 4.6). Figure 4.24 depicts this as square (1), composed with the pushout that computes  $K'$ . First, by Lemma 4.1 and construction, we know that  $k_1 \circ p = e''_1 \circ u_1$  and  $l' \circ p' = e'_1 \circ v_1$ . Moreover, pullback composition ensures that the span

$$C_1 \xleftarrow{e''_1 \circ u_1} K_\cap \xrightarrow{k} V$$

is the pullback of

$$V \xrightarrow{e'_1 \circ v_1} L \xleftarrow{l'_1} C_1$$

(this is because both the square at the top and at the back of the cube in Fig. 4.6 are known to be pullbacks). Summarizing, Fig. 4.24 commutes, the outer square is a pullback, the first square a pushout, and, moreover,  $l'_1 \in \mathcal{M}$ . Therefore,  $\mathcal{M}$  pullback-pushout decomposition is applicable and implies that square (1) is a pullback.

Continuing our main proof,  $C''_1$  then is computed as pushout of  $c_1$  and  $d'_1$ ; again, the morphism  $C''_1 \hookrightarrow G$  is obtained by its universal property.

$$\begin{array}{ccccc}
& & e'_1 \circ u_1 & & \\
& \curvearrowright & & \curvearrowleft & \\
K_{\cap} & \xrightarrow{p} & K & \xrightarrow{k_1} & C_1 \\
\downarrow k & & \downarrow k' & & \downarrow l'_1 \\
V & \xrightarrow{p'} & K' & \xrightarrow{l'} & L \\
& \curvearrowleft & e'_1 \circ v_1 & \curvearrowright & 
\end{array}
\quad \text{(PO)} \quad \text{(1)}$$

Figure 4.24: Proving the square (1) to be a pullback.

Now, invoking  $l' = l'' \circ k'_1$ , pushout decomposition, and uniqueness of  $C'_1$  as pushout complement for  $m \circ l''$ , the two vertical squares  $C'_1 \xrightarrow{l''} L \xrightarrow{m} G \leftarrow C''_1 \leftarrow C'_1$  and  $K' \xrightarrow{k'_1} C'_1 \leftarrow C''_1 \leftarrow D' \xleftarrow{d'} K'$  can be recognized to decompose the bent pushout at the front (that is given by the applicability of  $\rho_1 *_{E,k} \rho_2$  at  $m$ ) into two pushouts.

In particular, we obtained the top and the front faces of the right cube and they are all pushouts. Completing the cube by computing  $D$  as pullback and invoking the vertical weak van Kampen property of the right front face, the left back face of that cube is a pushout, as well. This means, since  $l = l'_1 \circ k_1$ , composing that pushout with the lower pushout of the two left ones constitutes the left pushout of a transformation of  $\rho_1 *_{E,k} \rho_2$  at match  $m$  with context object  $D$ . In particular,  $\rho_1 *_{E,k} \rho_2$  is applicable at  $m$ .

Finally, we did not explicitly mention application conditions but dealt with them implicitly, invoking the Concurrency Theorem for rules with application conditions. To at least somewhat motivate the definition of the application condition of a GCR (or concurrent rule) and for the convenience of the reader, we reproduce the relevant computation that can also be found in the proof of [58, Theorem 4.17]: Whenever  $m_1$  and  $m_2$  are the matches for  $\rho_1$  and  $\rho_2$  in an  $E$ -related transformation sequence, and  $m$  is the corresponding match for the (generalized) concurrent rule (or vice

versa in case analysis is possible), we have

$$\begin{aligned}
m_1 \models ac_1 \text{ and } m_2 \models ac_2 &\iff m \models \text{Shift}(e'_1, ac_1) \text{ and} \\
&h \models \text{Shift}(h, ac_2) \\
&\iff m \models \text{Shift}(e'_1, ac_1) \text{ and} \\
&m \models \text{Left}(p', \text{Shift}(h, ac_2)) \\
&\iff m \models \text{Shift}(e'_1, ac_1) \wedge \\
&\text{Left}(p', \text{Shift}(h, ac_2)) \\
&\iff m \models ac
\end{aligned}$$

where  $p' = L \xleftarrow{l'_1} C_1 \xrightarrow{r'_1} E$ ,  $m_1 = m \circ e'_1$ , and  $h : E \rightarrow G_1$  is the morphism with  $h \circ e_1 = n_1$  and  $h \circ e_2 = m_2$  that exists because the transformation sequence is  $E$ -related (compare Figs. 3.12 and 4.6). The computation relies on Facts 3.5 and 3.6.  $\square$

The existence of an  $\mathcal{E}'$ - $\mathcal{M}$  pair factorization is not needed in the actual proof but merely to be able to compute the application condition of the (generalized) concurrent rule. In the case of graphs and injective matching, the Generalized Concurrency Theorem ensures that the situation illustrated in Example 4.7 is the only situation in which the analysis of the application of a GCR fails: When restricting to injective matching, a violation of the dangling-edge condition is known to be the only possible obstacle to an application of a graph transformation rule [53, Fact 3.11]. We more comprehensively discuss the Generalized Concurrency Theorem after stating it for the case of general matching in the next section.

### 4.5.2 Central Results for the Case of General Matching

In this section, we restate the results of the last section in setting of arbitrary matches. All proofs in this section, which tend to be rather technical, are relegated to Appendix B.1.

#### Alternative Construction of Generalized Concurrent Rules

The construction of GCRs as introduced in Sect. 4.4.1 extends the one of concurrent rules in a straightforward way. So, it becomes obvious that every

concurrent rule is a GCR (for a common kernel being an isomorphism). Here, we introduce a second, equivalent construction that serves as basis for the proofs in this section.<sup>3</sup>

**Construction 4.7** (Alternative construction for concurrent rules). Given two plain rules  $\rho_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$ , where  $i = 1, 2$ , an  $E$ -dependency relation  $E = (e_1 : R_1 \hookrightarrow E, e_2 : L_2 \hookrightarrow E) \in \mathcal{E}'$ , and a common kernel  $k : K_\cap \hookrightarrow V$  of  $\rho_1$  and  $\rho_2$  that is compatible with  $E$ , we construct the span

$$L \xleftarrow{l'} K' \xrightarrow{r'} R,$$

where  $l' := l'' \circ k'_1$  and  $r' := r'' \circ k'_2$  as follows (compare Fig. 4.25):

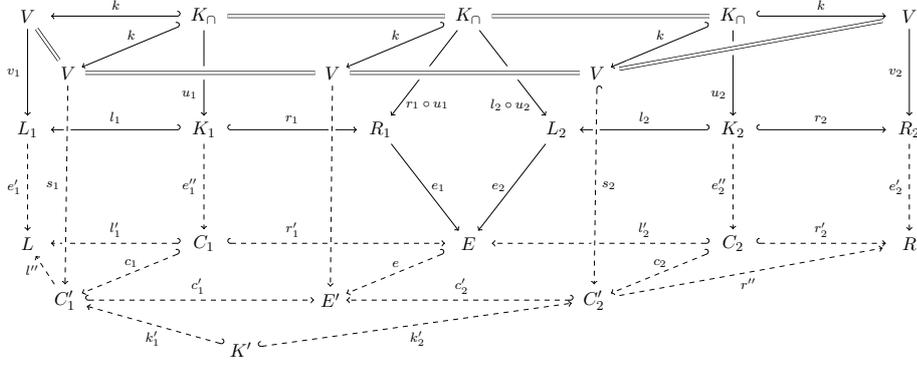


Figure 4.25: Alternative construction of a generalized concurrent rule

- (1)  $C_1$  and  $C_2$  are computed as pushout complements for  $e_1 \circ r_1$  and  $e_2 \circ l_2$ , respectively.
- (2)  $L$  and  $R$  are computed as pushouts of  $(l_1, e''_1)$ , and  $(r_2, e''_2)$ , respectively.
- (3)  $C'_1, E', C'_2$  are computed as pushouts of  $(k, e'_1 \circ u_1)$ ,  $(k, e_1 \circ r_1 \circ u_1 = e_2 \circ l_2 \circ u_2)$  and  $(k, e''_2 \circ u_2)$ , respectively.

<sup>3</sup>The reason for this is basically accidental: the simpler possibility to construct GCRs was discovered after obtaining the following proofs. Proofs that are directly based on the simpler construction are to appear in [138].

- (4) Morphisms  $l'' : C'_1 \rightarrow L$  and  $c'_1 : C'_1 \hookrightarrow E$  are obtained by the universal property of the pushout object  $C'_1$ ;  $r''$  and  $c'_2$  are obtained analogously.
- (5)  $K'$  is computed as pullback of  $(c'_1, c'_2)$ .

Both introduced constructions of spans are indeed equivalent and can serve as basis for a definition of generalized concurrent rules. For simplicity, we directly assume their left- and right-hand sides to be the same (and not only isomorphic): They are computed in the same way.

**Proposition 4.10** (Equivalence of constructions). *Given two rules  $\rho_1, \rho_2$  and an  $E$ -dependency relation  $E$  and a compatible common kernel  $k$  for them, let*

$$L \xleftarrow{l'^1} K'^1 \xrightarrow{r'^1} R$$

be constructed according to Construction 4.4 and

$$L \xleftarrow{l'^2} K'^2 \xrightarrow{r'^2} R$$

according to Construction 4.7. Then there exists an isomorphism  $i : K'^1 \rightarrow K'^2$  such that  $l'^2 \circ i = l'^1$  and  $r'^2 \circ i = r'^1$ .

### Characterizing Derivable Generalized Concurrent Rules (General Matching)

Again, we characterize the GCRs derivable from a given pair of rules in two different ways, namely (i) characterizing possible choices for the morphisms  $v_1, v_2$  in a common kernel and (ii) characterizing the possible choices for enhancement morphisms  $k'$ .

The embedding characterization of GCRs in the case of  $\mathcal{M}$ -matching (Proposition 4.6) is straightforward. Knowing the morphisms  $u_i$  to be  $\mathcal{M}$ -morphisms, the morphisms  $v_i$  need to be as well. The general case is more involved. The intuition behind the embedding characterization for that case (in categories like **Graph**) is that the morphisms  $v_i$  embedding  $V$  into the rules are only allowed to be as non-injective as the morphisms  $u_i$  are, i.e., they are not allowed to identify elements that are not already identified by  $u_i$ . This means that the morphisms  $v_i$  serve to individually identify elements from  $L_1 \setminus K_1$  with elements from  $R_2 \setminus K_2$ . The next proposition

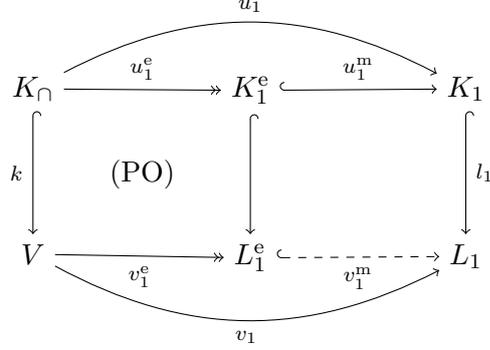


Figure 4.26: An individually identifying embedding

expresses that categorically. The greatest obstacle for a straightforward formulation of this result, however, is that it cannot be expressed by saying that the  $v_i$  are  $\mathcal{M}$ -morphisms when restricted to suitable  $\mathcal{M}$ -subobjects of  $V$ . We use an epi- $\mathcal{M}$  factorization of the morphisms  $u_i$  instead to express the desired property. Note that the proposition is proved for adhesive HLR categories. Whether it is possible to lift the result to a more general setting is an open question.

**Definition 4.8** (Individually identifying embedding). In a category with epi- $\mathcal{M}$  factorization of morphisms, given two rules  $\rho_i = (L_i \leftarrow K_i \hookrightarrow R_i, ac_i)$ , where  $i = 1, 2$ , morphisms  $u_i : K_\cap \rightarrow K_i$ ,  $v_1 : V \rightarrow L_1$ ,  $v_2 : V \rightarrow L_2$  embed a common kernel  $k : K_\cap \hookrightarrow V$  individually identifying if the morphisms  $v_i^e$  in the epi- $\mathcal{M}$  factorizations  $v_i = v_i^m \circ v_i^e$  can be obtained as the morphisms opposite to  $u_i^e$  in the pushout of  $u_i^e$  along  $k$ , where also  $u_i = u_i^m \circ u_i^e$  are the epi- $\mathcal{M}$  factorizations of the morphisms  $u_i$  (compare Fig. 4.26).

**Proposition 4.11** (Embedding characterization of GCRs for General Matching). Let  $(\mathcal{C}, \mathcal{M})$  be an adhesive HLR category with  $\mathcal{M}$ -effective unions and epi- $\mathcal{M}$  factorization of morphisms, and let  $k : K_\cap \hookrightarrow V$  be a common kernel for two rules  $\rho_i$  where the embedding morphisms  $u_i : K_\cap \rightarrow K_i$  are general ( $i = 1, 2$ ).

- (1) The application of Construction 4.7 (or, equivalently, Construction 4.4) results in a GCR  $\rho_1 *_{E,k} \rho_2$  if and only if  $k$  is embedded

*individually identifying.*

- (2) *The assumption that  $\mathcal{M}$ -effective unions exist is necessary for this result to hold.*

As a corollary to the above proposition, in categories with initial pushouts we obtain an  $\mathcal{M}$ -subobject of  $V$ —namely the context object  $C_k$  of the initial pushout over  $k$ —such that if  $v_1, v_2$  are  $\mathcal{M}$ -morphisms when restricted to it, the construction results in a GCR. This condition can only be a sufficient and not a necessary one: As, for example, the complement of the image of a graph homomorphism does not need to be a graph itself, the above result implies that injectivity has to be checked on a part of a graph that does not need to be a graph again. We illustrate this and subsequently state the corollary.

**Example 4.8.** This example shows the following Corollary 4.12 to only provide a sufficient condition. Consider the computation of a GCR (in the category **Graph**) as partially depicted in Fig. 4.27. Again, the action of morphisms is indicated by node names. The names  $a$  and  $b$  stem from two nodes in the LHS of a second rule that are identified (by a morphism  $e_2$ ) in the graph  $E$ . A suitable second rule, for example, would be the rule that matches two nodes and creates a loop for each one. The morphism  $v_1 \circ c_k$  is not injective, and yet the result of the computation is a GCR (i.e.,  $l'$  is injective). The reason is that the identified nodes are already identified by  $u_1$ ; however, they are boundary nodes, i.e., they are needed in  $C_k$  for  $C_k$  to become a subgraph of  $V$ . The morphism  $v_1 \circ c_k$  is indeed injective on the two edges that are not part of  $K_\cap$ . However, these two edges alone do not constitute a graph.

**Corollary 4.12** (Sufficient criterion for constructing GCRs). *Let  $(\mathcal{C}, \mathcal{M})$  be an adhesive HLR category with initial pushouts,  $\mathcal{M}$ -effective unions, and epi- $\mathcal{M}$  factorization. Let  $\rho_1, \rho_2$  be two rules,  $E$  an  $E$ -dependency relation for these, and  $k$  a common kernel that is compatible with  $E$ . If  $v_i \circ c_k \in \mathcal{M}$  for  $i = 1, 2$ , where  $c_k : C_k \hookrightarrow V$  embeds the context object  $C_k$  of the initial pushout over  $k$  into  $V$  and  $v_1 : V \rightarrow L_1, v_2 : V \rightarrow R_2$  the common kernel into the rules, applying Construction 4.7 (or, equivalently, Construction 4.4) to this data results in a generalized concurrent rule  $\rho_1 *_{E,k} \rho_2$ .*

*Remark 4.2.* We presented the above result as a corollary to Proposition 4.11, i.e., under the assumption of an adhesive HLR category with

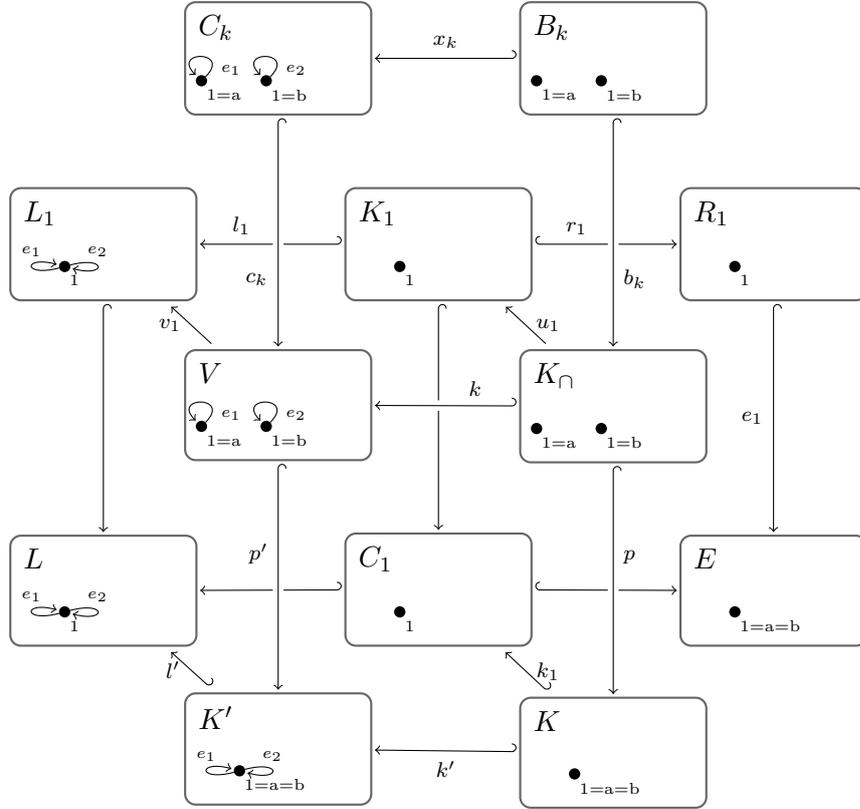


Figure 4.27: Example showing Corollary 4.12 to not provide a necessary condition

epi- $\mathcal{M}$  factorization. In that context, also Proposition 4.6 is a corollary to Proposition 4.11. But as Proposition 4.6, also the above corollary can be proven more generally for  $\mathcal{M}$ -adhesive categories and without assuming an epi- $\mathcal{M}$  factorization. We do not present the proof because it is very similar to the kinds of proofs that we present and because the result does not play any role in the rest of the thesis.

As in Sect. 4.5.1, we next answer the question which interfaces that extend the interface of a concurrent rule do indeed constitute the interface of a generalized concurrent rule. In the general case, the answer can also be given in terms of initial pushouts but is somewhat more involved. The ultimate reason for the increased complexity is that, when computing a

concurrent rule in the setting of general matching, individual elements of the LHS of its first underlying rule might be identified in the LHS of the concurrent rule (and likewise: individual elements of the RHS of its second underlying rule in the RHS of the concurrent rule). As a consequence, the initial pushout over an enhancement morphism does not need to factorize (uniquely) through the left morphism of the first rule (or the right morphism of the second). Instead, we have to use their opposite morphisms that arise in the computation of the concurrent rule and where the identification has taken place. The next example illustrates this.

**Example 4.9.** This example shows how, in the case of general matching, the initial pushout over  $k'$  does not need to factor through the one over  $l_1$ . Figure 4.28, again, depicts a (partial) computation of a GCR in **Graph**, where indices indicate the action of morphisms. Again, the rule that matches two nodes and creates a loop for each one is a suitable second rule for the computation and the names  $a$  and  $b$  stem from such a rule and a morphism  $e_2 : L_2 \rightarrow E$  that identifies  $a$  and  $b$ . As can be seen in the left top of the figure, there does not exist any morphism from  $C_{k'}$  to  $C_{l_1}$  with the desired properties: Every morphism would need to identify the edges  $e_1$  and  $e_2$  in  $C_{l_1}$  and consequently cannot be injective. The obstruction to the existence is the identification of nodes prescribed by morphism  $e_1 : R_1 \rightarrow E$  that is inherited by  $e_1''$  and  $e_1'$ . As  $e_1''$  and  $e_1'$  both identify the nodes accordingly, taking the initial pushout over  $l_1'$  instead of the one over  $l_1$  works just fine.

To accommodate for the additional complexity, we first have to adapt the definition of what it means to be *appropriately enhancing*.

**Definition 4.9** (Appropriately enhancing). In a category with initial pushouts, let  $\rho_1 *_E \rho_2 = L \xleftarrow{l} K \xrightarrow{r} R$  be an  $E$ -concurrent rule and  $k' : K \hookrightarrow K'$  be an  $\mathcal{M}$ -morphism such that there exist  $\mathcal{M}$ -morphisms  $l' : K' \hookrightarrow L$  and  $r' : K' \hookrightarrow R$  with  $l' \circ k' = l$  and  $r' \circ k' = r$ . Then  $k'$  is called *appropriately enhancing* if the following holds (compare Fig. 4.29): The boundary and context object of the initial pushout over  $k'$  factorize via  $\mathcal{M}$ -morphisms  $s_L, s_R : B_{k'} \hookrightarrow B_{l_1'}, B_{r_2'}$  and  $t_L, t_R : C_{k'} \hookrightarrow C_{l_1'}, C_{r_2'}$  as pullback through the initial pushouts over  $l_1' : C_1 \hookrightarrow L$  and  $r_2' : C_2 \hookrightarrow R$  in such a way that

$$k_1 \circ b_{k'} = b_{l_1'} \circ s_L \text{ and } k_2 \circ b_{k'} = b_{r_2'} \circ s_R . \quad (4.1)$$

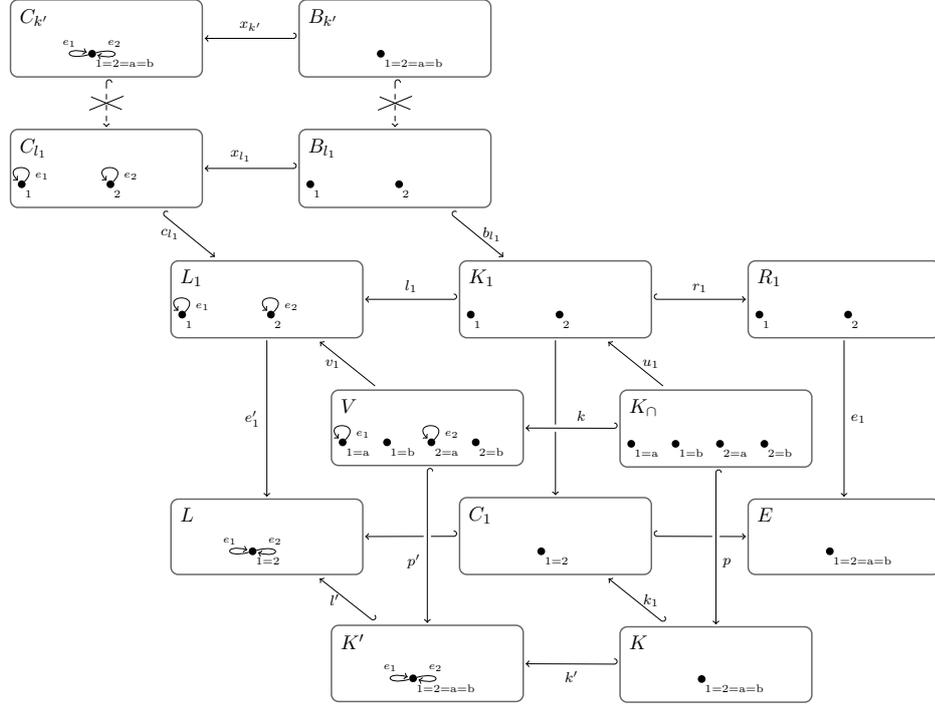


Figure 4.28: Counterexample to the factorization of the initial pushout over  $k'$  through the one over  $l_1$

The closure properties of initial pushouts (Fact 3.12) immediately ensure that Definition 4.9 generalizes Definition 4.6; therefore, we use identical names. When the morphisms  $e'_1$  and  $e''_1$  (see, e.g., Fig. 3.11) are both  $\mathcal{M}$ -morphisms, the initial pushout over  $l'_1$  offers also the one over  $l_1$ .

With the more general definition of appropriate enhancement, the statement of the proposition is now almost the same as in the case of  $\mathcal{M}$ -matching. The qualification that the morphisms  $u_i$  stem from  $\mathcal{M}$  is naturally obsolete. A prominent difference is that we can only prove the statement in the setting of adhesive HLR categories with pullbacks.

**Proposition 4.13** (Enhancement characterization of GCRs for general matching). *Let  $(\mathcal{C}, \mathcal{M})$  be an adhesive HLR category with initial pushouts,  $\mathcal{M}$ -effective unions, and pullbacks. Given an  $E$ -concurrent rule  $\rho_1 *_E \rho_2 =$*

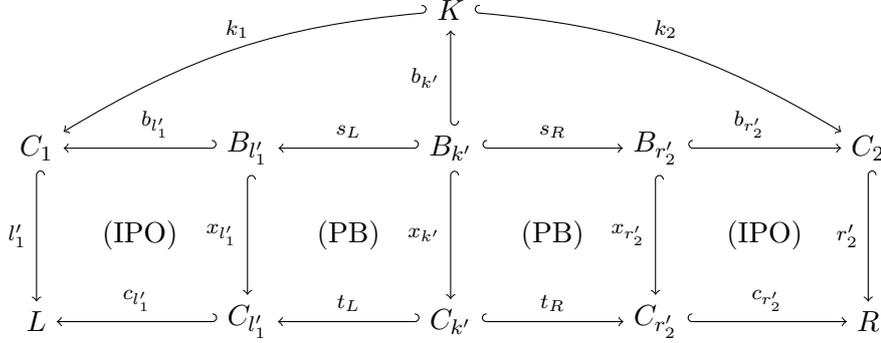


Figure 4.29: Definition of appropriate enhancement in the case of general matching

$L \xleftarrow{l} K \xrightarrow{r} R$  and an  $\mathcal{M}$ -morphism  $k' : K \hookrightarrow K'$  such that there exist  $\mathcal{M}$ -morphisms  $l' : K' \hookrightarrow L$  and  $r' : K' \hookrightarrow R$  with  $l' \circ k' = l$  and  $r' \circ k' = r$ , the span  $L \xleftarrow{l'} K' \xrightarrow{r'} R$  is derivable as a GCR  $\rho_1 *_{E,k} \rho_2$  for some common kernel  $k$  if and only if  $k'$  is appropriately enhancing.

Interestingly, Corollary 4.8 holds in the general setting exactly as in the setting of injective matching. This is because the enhancement morphism does not introduce identifications beyond those that take place in the construction of the underlying concurrent rule. Thus, also in the setting of general matching, in categories like **Graph** generalized concurrent rules are obtained from concurrent rules by identifying elements that the first underlying rule deletes with elements that the second rule creates and incorporating these into the extended interface.

### A Generalized Concurrency Theorem—General Matching

We close this section with presenting the Generalized Concurrency Theorem also for the case of general matching. With Example 4.7 we illustrated that even in the case of  $\mathcal{M}$ -matching, the Generalized Concurrency Theorem will necessarily be asymmetric. Analysis will not always be possible because GCRs satisfy the dangling-edge condition more often than their underlying rules. In the case of general matching, again, the situation becomes more complex. Here, analysis can also fail for the reason that the match for

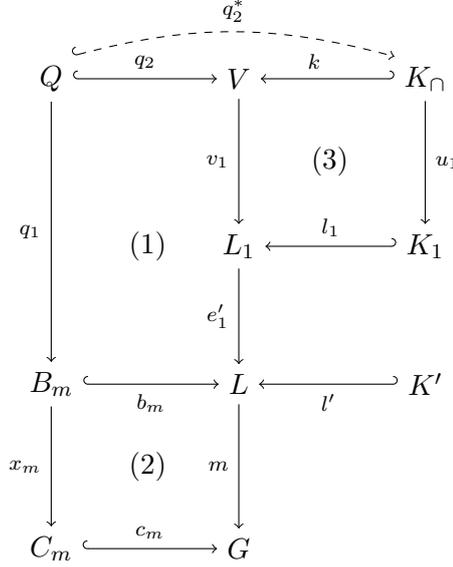
the GCR identifies one of its additionally preserved elements with some other elements. It turns out that both problems can be expressed in terms of the *boundary elements* of the match. For analysis to be possible, no boundary element of the match may be one of the elements that the GCR additionally preserves.

We first illustrate the additional situation in which analysis fails using a simple example from the category **Graph**. Then, we formally introduce what it means for a match that its boundary elements interact suitably with the enhancement of the GCR and subsequently state the Generalized Concurrency Theorem.

**Example 4.10.** In **Graph** (or **Set**), given the rules  $\bullet \leftrightarrow \emptyset \hookrightarrow \emptyset$  and  $\bullet \leftrightarrow \bullet \hookrightarrow \bullet\bullet$ , one can derive the GCR  $\bullet\bullet \leftrightarrow \bullet\bullet \hookrightarrow \bullet\bullet$ , enhancing the concurrent rule  $\bullet\bullet \leftrightarrow \bullet \hookrightarrow \bullet\bullet$ . We do not depict the simple computation. The GCR is applicable matched (non-injectively) to a single node (doing nothing); however, the concurrent rule is not. As in Example 4.7, an element by which the GCR is newly enhanced is also a boundary element of that match. This time, the node is a boundary node for sharing its image under the match with another one.

**Definition 4.10** (Enhancement-free boundary). In a category with initial pushouts, given a match  $m : L \rightarrow G$  for a generalized concurrent rule  $\rho_1 *_{E,k} \rho_2$ ,  $m$  has an *enhancement-free boundary* if, given the pullback  $(B_m \xleftarrow{q_1} Q \xrightarrow{q_2} V)$  of  $(B_m \xrightarrow{b_m} L \xleftarrow{e'_1 \circ v_1} V)$ , where  $b_m$  is the boundary of  $m$  and  $e'_1 \circ v_1$  maps  $V$  to  $L$  as specified in Construction 4.4 (see Fig. 4.6), there exists a morphism  $q_2^* : Q \rightarrow K_\cap$  such that  $k \circ q_2^* = q_2$  (see Fig. 4.30).

Using the property of enhancement-free boundary, we are able to state the Generalized Concurrency Theorem also for the case of arbitrary matching. For the analysis case, we additionally consider the case where the match for the GCR stems from  $\mathcal{M}$  and are able to regain a result similar to the one of Theorem 4.9. Importantly, in contrast to Theorem 4.9, this also holds for GCRs being obtained from  $E$ -dependency relations  $E = (e_1, e_2)$  with  $e_1, e_2 \notin \mathcal{M}$  (which also implies  $m_1, m_2 \notin \mathcal{M}$  for the induced matches for the underlying rules). Thus, rather than being a statement about the restricted setting of  $\mathcal{M}$ -matching, it is a statement about applications of GCRs where (in categories like **Graph**) no further identification between elements is made by the match of the GCR but all such desired identification


 Figure 4.30: Illustrating the definition of *enhancement-free boundary*

is already encoded in the dependency relation. We present this additional statement because it concerns the setting for which we are later able to present sufficient conditions of language-preserving applications of GCRs (see Theorem 4.16).

**Theorem 4.14** (Generalized Concurrency Theorem). *Let  $(\mathcal{C}, \mathcal{M})$  be an  $\mathcal{M}$ -adhesive category with  $\mathcal{E}'$ - $\mathcal{M}$  pair factorization,  $\mathcal{M}$ -effective unions, and initial pushouts.*

**Synthesis.** *For each  $E$ -related transformation sequence*

$$G_0 \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow_{\rho_2, m_2} G_2$$

*there exists a direct transformation*

$$G_0 \Rightarrow_{\rho_1 *_{E, k} \rho_2, m} G_2$$

*for each common kernel  $k$  for  $\rho_1$  and  $\rho_2$  that is compatible with  $E$ .*

**Analysis.** *Given a direct transformation*

$$G_0 \Rightarrow_{\rho_1 *_{E, k} \rho_2, m} G_2 \quad ,$$

there exists an  $E$ -related transformation sequence

$$G_0 \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow_{\rho_2, m_2} G_2$$

with  $m_1 = m \circ e'_1$  if and only if  $m$  has an enhancement-free boundary.

Moreover, if  $m \in \mathcal{M}$ , there exists an  $E$ -related transformation sequence

$$G_0 \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow_{\rho_2, m_2} G_2$$

with  $m_1 = m \circ e'_1$  if and only if

$$G_0 \Rightarrow_{\rho_1, m_1} G_1$$

exists (without the necessity to assume the existence of initial pushouts).

*Remark 4.3.* At least in the presence of initial pushouts, the Concurrency Theorem indeed becomes a corollary to our Generalized Concurrency Theorem. For this, one just needs to observe that whenever  $k$  is an isomorphism (see Proposition 4.3), the necessary morphism  $q_2^* : Q \rightarrow K \cap$  can always be obtained by composing  $q_2 : Q \hookrightarrow V$  with the inverse of  $k$ . Similarly, the Generalized Concurrency Theorem subsumes the Short-cut Theorem [78, Theorem 7].

As in the case of  $\mathcal{M}$ -matching, the existence of an  $\mathcal{E}'$ - $\mathcal{M}$  pair factorization is not needed in the actual proof but merely to be able to compute the application condition of the (generalized) concurrent rule.

The next corollary translates the analysis case of the above theorem into more concrete language in the case of graphs; for this, we use the terminology of *identification* and *dangling points* as introduced in [53, Definition 3.9].

**Corollary 4.15** (Analysis in the category **Graph**). *In the category of (typed, attributed) graphs, given a generalized concurrent rule  $\rho_1 *_{E,k} \rho_2$  derived from rules  $\rho_1$  and  $\rho_2$  and applicable at a match  $m : L \rightarrow G$ , the underlying concurrent rule  $\rho_1 *_E \rho_2$  is applicable at  $m$  (and consequently  $\rho_1$  and  $\rho_2$  at the induced matches) if and only if*

- (1) no dangling point of  $m$ , when considered as a match for  $\rho_1 *_E \rho_2$ , is among the elements that  $\rho_1 *_{E,k} \rho_2$  additionally preserves, i.e., no node that is to be deleted with incident edges that are to be preserved stems from  $K' \setminus K$ , and

- (2) no identification point of  $m$ , when considered as a match for  $\rho_1 *_{E} \rho_2$ , is among the elements that  $\rho_1 *_{E,k} \rho_2$  additionally preserves, i.e., no element that  $m$  identifies with another one stems from  $K' \setminus K$ .

### 4.5.3 Conditions for Language-Preserving Applications of Generalized Concurrent Rules

The Generalized Concurrency Theorem shows that a GCR might be applicable more often than the concurrent rule it enhances. Some of these additional applications might leave the language of a given grammar whereas the classical Concurrency Theorem clarifies that this is not possible using concurrent rules. The next theorem provides a sufficient condition for applications of GCRs to be language-preserving. The underlying notion is that of a *grammar* consisting of a start object and a set of rules; its *language* consists of all objects transitively derivable from the start object by applications of the rules of the grammar. The theorem generalizes a similar result we presented for short-cut rules [79, Theorem 8]. We discuss its practical relevance subsequently.

**Theorem 4.16** (Language-preserving applications). *In an  $\mathcal{M}$ -adhesive category  $(\mathcal{C}, \mathcal{M})$ , given a sequence of transformations*

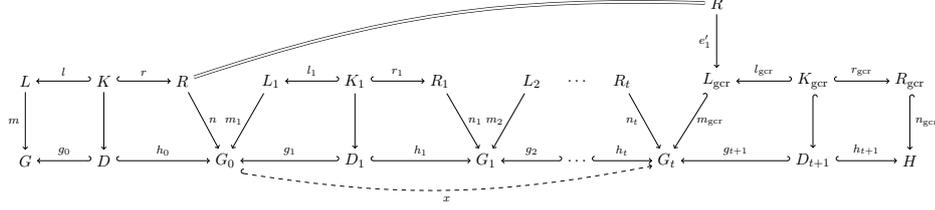
$$G \Rightarrow_{\rho, m} G_0 \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow_{\rho_2, m_2} \cdots \Rightarrow_{\rho_t, m_t} G_t \Rightarrow_{\rho^{-1} *_{E,k} \rho', m_{\text{gcr}}} H$$

with rules  $\rho_1, \dots, \rho_t$ , and  $\rho^{-1} *_{E,k} \rho'$  being a generalized concurrent rule of  $\rho^{-1}$  and  $\rho'$  such that  $m_{\text{gcr}} \in \mathcal{M}$  (as indicated in Fig. 4.31), there is a match  $m'$  for  $\rho'$  in  $G$  and a transformation sequence

$$G \Rightarrow_{\rho', m'} G'_0 \Rightarrow_{\rho_1, m'_1} G'_1 \Rightarrow_{\rho_2, m'_2} \cdots \Rightarrow_{\rho_{t-1}, m'_{t-1}} G'_{t-1} \Rightarrow_{\rho_t, m'_t} H ,$$

provided that

- (1) the application of  $\rho^{-1} *_{E,k} \rho'$  with match  $m_{\text{gcr}}$  is sequentially independent of the sequence of transformations  $G_0 \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow_{\rho_2, m_2} \cdots \Rightarrow_{\rho_t, m_t} G_t$ , and
- (2) the thereby implied match  $m'_{\text{gcr}}$  for  $\rho^{-1} *_{E,k} \rho'$  in  $G_0$  coincides with the comatch  $n$  when restricted to  $R$ , i.e.,  $x \circ n = m_{\text{gcr}} \circ e'_1$ , where  $x$  is the partial track morphism from  $G_0$  to  $G_t$  arising from composing the spans  $G_{i-1} \xleftarrow{e_i} D_i \xrightarrow{h_i} G_i$  for  $i = 1, \dots, t$ .

Figure 4.31: Characterizing valid matches for GCRs (for  $t \geq 2$ )

In particular, given a grammar such that  $\rho, \rho', \rho_1, \dots, \rho_t$  belong to its rules and  $G$  to its language, also the object  $H$  belongs to the language of this grammar.

*Proof.* Assume a sequence of transformations as required in the statement of the theorem to be given. First, by the sequential independence of the application of the GCR of the foregoing sequence of rule applications, we can iteratively apply the Local Church–Rosser Theorem (Theorem 3.7 or [58, Theorem 4.7]) to switch that application to the front and obtain the sequence

$$G \Rightarrow_{\rho, m} G_0 \Rightarrow_{\rho^{-1} *_{E, k} \rho', m'_{\text{gcr}}} G'_0 \Rightarrow_{\rho_1, m'_1} \cdots \Rightarrow_{\rho_{t-1}, m'_{t-1}} G'_{t-1} \Rightarrow_{\rho_t, m'_t} H .$$

Since  $m_{\text{gcr}} \in \mathcal{M}$ , also  $m'_{\text{gcr}} \in \mathcal{M}$  (this follows by composition of  $\mathcal{M}$ -morphisms). Hence, by the analysis condition on matches from  $\mathcal{M}$  for GCRs from the Generalized Concurrency Theorem, the application of  $\rho^{-1} *_{E, k} \rho'$  at  $m'_{\text{gcr}} \in \mathcal{M}$  can be decomposed into an application of  $\rho^{-1}$  followed by one of  $\rho'$  (at the appropriate matches) whenever  $\rho^{-1}$  is applicable at  $m'_{\text{gcr}} \circ e'_1$ . However,  $x \circ n = m_{\text{gcr}} \circ e'_1$  just means that  $m'_{\text{gcr}} \circ e'_1 = n$ . Obviously,  $\rho^{-1}$  is applicable at  $n$ , just reverting the first transformation. Thus, we obtain

$$\begin{aligned} & G \Rightarrow_{\rho, m} G_0 \Rightarrow_{\rho^{-1} *_{E, k} \rho', m'_{\text{gcr}}} G'_0 \Rightarrow_{\rho_1, m'_1} \cdots \Rightarrow_{\rho_{t-1}, m'_{t-1}} G'_{t-1} \Rightarrow_{\rho_t, m'_t} H \\ \equiv & G \Rightarrow_{\rho, m} G_0 \Rightarrow_{\rho^{-1}, n} G \Rightarrow_{\rho', m'} G'_0 \Rightarrow_{\rho_1, m'_1} \cdots \Rightarrow_{\rho_{t-1}, m'_{t-1}} G'_{t-1} \Rightarrow_{\rho_t, m'_t} H \\ \equiv & G \Rightarrow_{\rho', m'} G'_0 \Rightarrow_{\rho_1, m'_1} \cdots \Rightarrow_{\rho_{t-1}, m'_{t-1}} G'_{t-1} \Rightarrow_{\rho_t, m'_t} H , \end{aligned}$$

where  $m'$  is the match for  $\rho'$  provided by the Generalized Concurrency Theorem, and  $\equiv$  denotes the equivalence relation induced by isomorphisms

and switching of sequentially independent applications, called *switch equivalence* [139, 19].

In particular, if  $G$  belongs to the language defined by a grammar containing rules  $\rho', \rho_i$ , where  $i = 1, \dots, t$ , then also  $H$  belongs to that language.  $\square$

Given a language-defining grammar, a typical model edit amounts to revoking one (or more) of the rule applications that created the model and replacing it by another one.<sup>4</sup> Considering the situation described in Theorem 4.16 and depicted in Fig. 4.31, this frequently can be done by GCRs in situations where the underlying concurrent rule is not applicable: In the case of graphs, for example, whenever one of the applications of the rules  $\rho_i$  creates an edge incident to one of the nodes created by  $\rho$  such that this edge is still present in  $G_t$ ,  $\rho^{-1}$  (and, hence, every concurrent rule built from it) is not applicable to  $G_t$ . Thus, to replace the application of  $\rho$  by one of  $\rho'$ , one would first need to revoke the applications of the  $\rho_i$  before being able to perform that replacement; subsequently the  $\rho_i$  then need to be reapplied. A GCR might be directly applicable, provided it preserves the said node.

Note that, beyond language-preservation, the above theorem provides a foundation for *incremental parsing* after such edits: One obtains a derivation trace for  $H$  by just replacing  $\rho$  by  $\rho'$  in the known one for  $G_t$ . In particular, by repeatedly applying the classical Concurrency Theorem, the above result holds in the case where  $\rho$  and  $\rho'$  are concurrent rules. This means, not only single steps but even longer sequences of transformations might be replaced like this.

## 4.6 Related Work

We already compared GCRs with our own work on short-cut rules [78, 79] throughout this chapter and therefore focus on other works in the following.

After predecessors in specific categories (see, e.g., [68] for the case of graphs), a Concurrency Theorem for double-pushout rewriting in a rather general categorical setting has been presented in [60]. In that work, spans

---

<sup>4</sup>We stated and proved Theorem 4.16 for  $\rho$  and  $\rho'$  being rules of the given grammar. However, by the classical Concurrency Theorem and the associativity result of Behr and Sobociński [29], at least for the case of  $\mathcal{M}$ -matching  $\rho$  and  $\rho'$  might as well be themselves concurrent rules built from rules of the grammar.

$R_1 \leftarrow D \rightarrow L_2$  are used to encode the dependency information. When (variants of) adhesive categories had been established as axiomatic basis for double-pushout rewriting [141, 59], the construction of concurrent rules and an according Concurrency Theorem (now based on a co-span  $R_1 \rightarrow E \leftarrow L_2$  encoding the dependency information) was lifted to that setting: Ehrig et al. [53] present this for plain rules in the context of (weak) adhesive HLR systems, which was later extended to the case of rules with negative [144] and general application conditions [61]. It is this mature formulation of the Concurrency Theorem ( $\mathcal{M}$ -adhesive categories, application conditions, arbitrary matches), as also given in current surveys [58, 54], that we address in our generalization. While our construction of a GCR and the synthesis case of the Generalized Concurrency Theorem hold in exactly this context, we additionally assume the existence of initial pushouts or restrict to the case of  $\mathcal{M}$ -matching of GCRs for the analysis case. We are not aware of any other rule construction that allows for the kind of reuse of elements that we enable with GCRs.

Behr and Sobociński [29] prove that, in the case of  $\mathcal{M}$ -matching, the concurrent rule construction is associative (in a certain technical sense). Based on that result, they present the construction of a rule algebra that captures interesting properties of a given grammar. This has served, e.g., as starting point for static analysis of stochastic rewrite systems [25, 27]. Determining whether also our GCR construction is associative (in certain cases) is future work. Behr [24] also established a construction of concurrent rules (and associativity) for sesqui-pushout rewriting [46]; among the several categorical approaches to rewriting, this is the only variant beyond DPO we are aware of where this has been done. It seems that our construction of GCRs would be similarly applicable to sesqui-pushout rewriting; however, applied as is, it would have restricted expressivity in that context: In **Graph**, for instance, applying a rule according to the sesqui-pushout semantics implies to (implicitly) delete all edges that are incident to a deleted node. Our construction would need to be extended to allow also preserving such implicitly deleted elements, i.e., to identify such elements as recreated by a second rule application.

Finally, the automated generation of model edit rules from a given meta-model has been addressed by Kehrer et al. [124]. Besides basic operations that create or delete elements, they introduce *move* and *change rules*. One can check that these can be built as GCRs from the basic operations

but not as mere concurrent rules. While this scenario, together with our own application in the context of model synchronization (see Chapter 6), suggests that GCRs indeed capture typical properties of model edits, a systematic study of which edits can be captured as GCRs (and which GCRs capture typical model edits) remains future work.

## 4.7 Conclusion

In this chapter, we develop *generalized concurrent rules* as a construction that combines both concurrent and short-cut rules. We develop our theory of GCRs in the setting of double-pushout rewriting in  $\mathcal{M}$ -adhesive categories with rules with application conditions matched at general matches. We characterize GCRs in terms of (i) how common kernels can be embedded and (ii) in which ways they enhance concurrent rules. As a first central result (Theorems 4.9 and 4.14), we generalize the classical Concurrency Theorem. A second central result presents sufficient conditions for an application of a GCR to be language-preserving with respect to a given grammar (Theorem 4.16). Our construction promises to allow for the (automated) derivation of (complex) edit rules from given (simple) rules of a grammar. Moreover, controlled application of these (in the sense of the preconditions of Theorem 4.16) allows for incremental parsing of the results.

From a theoretical point of view, developing similar kinds of rule composition in the context of other categorical approaches to rewriting like the single- or the sesqui-pushout approach would be an interesting next step. Considering practical application scenarios, we are most interested in somehow classifying the derivable GCRs of a given pair of rules according to their use such that the ones relevant for the desired application can be computed efficiently.



# 5 Double-Pushout Rewriting in $\mathcal{S}$ -Cartesian Functor Categories

In this chapter, we develop a new way of constructing adhesive categories from given ones and apply this to develop a comprehensive rewriting theory for typed attributed partial triple graphs.

## 5.1 Introduction

With the development of a substantial theory for double-pushout rewriting in the context of adhesive categories [141, 53, 58, 57, 54], there is a growing interest to apply these concepts to “higher-order” structures. Here, one does not merely rewrite objects of a certain type, e.g., graphs, for which this theory of rewriting applies. Instead, composite objects are rewritten which assemble basic objects according to a fixed structure. Examples for such composite objects are graph rewriting rules themselves or triple graphs. An obvious approach to formalize such composite objects categorically is to fix the shape of the structure and to use functors of that shape into the corresponding category of base objects. Functor categories are known to preserve adhesiveness [141, Proposition 3.5.] and such an approach has, e.g., been successfully applied to triple graphs [53, 54]. In certain cases, however, a functor category contains too many objects. In the example of graph rewriting rules, the obvious functor category consists of all spans and not only of those with monic legs. Similarly, for the rewriting of objects, each one with a designated subobject, the obvious functor category, namely the arrow category, contains all arrows and not only the monomorphisms.

To allow for more expressiveness with respect to restrictions, we introduce  *$\mathcal{S}$ -cartesian functor categories* as special subcategories of functor categories. The basic idea is that the objects of such a category are “structures of fixed shape” of objects of a given base category such as the category of graphs. By “structure of fixed shape”, we mean a fixed number of base objects,

which are connected by base morphisms of specified kinds. In  $S$ -cartesian functor categories, it is possible to force some of these base morphisms to be monomorphisms. Formally, given a small category  $\mathcal{X}$  that encodes the shape of the structure of interest and a base category  $\mathcal{C}$ , we consider functor categories  $[\mathcal{X}, \mathcal{C}]$  and restrict them in the following sense: Fixing a set  $S$  of the morphisms of  $\mathcal{X}$ , the functors have to map each  $S$ -morphism to a monomorphism in  $\mathcal{C}$ . As a variant, the functor category may be restricted even further by fixing an allowed class  $\mathcal{M}$  of monomorphisms to which  $S$ -morphisms have to be mapped. To obtain a theory of transformations based on double-pushout rewriting in such restricted functor categories, we will show that we cannot take the full subcategory induced by such functors but need to restrict the allowed natural transformations between them as well.

$S$ -cartesian functor categories provide a unifying framework for several examples that have been investigated in the literature. These include the following:<sup>1</sup>

- (1) Elements of  $[\bullet \hookrightarrow \bullet, \mathcal{C}]$  can be understood as objects that come with a designated subobject. These are the categories that Kastenberg and Rensink prove to be adhesive in [122] where the morphisms between monomorphisms are required to be pullback squares. Taking **Set** as base category, the full category with injective functions as objects and commuting squares as morphisms was already known to be quasiadhesive [120]. Kastenberg and Rensink use their result to introduce a new concept of attribution for the case where  $\mathcal{C}$  is the category **Graph**.
- (2) Elements of  $[\bullet \leftarrow \bullet \hookrightarrow \bullet, \mathcal{C}]$  are the linear rules of  $\mathcal{C}$ . Rewriting of rules (by higher-order rules) is of practical relevance, e.g., in the context of genetic algorithms, where transformation rules can serve as mutation operators and it can be of interest to evolve the mutation operators themselves [208, 203]. Rewriting of rules has formally been studied by Machado (et al.) [160, 161], for example.

---

<sup>1</sup>We assume the base category  $\mathcal{C}$  to meet some variant of adhesiveness (with respect to a designated class of monomorphisms  $\mathcal{M}$ ), do not depict identities of  $\mathcal{X}$ , and mark the morphisms from the designated set  $S$  by a hooked arrow.

- (3) Let  $\mathcal{T}$  be a finite tree and  $\bar{\mathcal{T}}$  denote the tree where every edge of  $\mathcal{T}$  is marked as to be mapped to monomorphisms. After fixing an appropriate decoration of the nodes of  $\mathcal{T}$  with quantifiers and logical connectives, the elements of  $[\bar{\mathcal{T}}, \mathcal{C}]$  are nested conditions [94] in  $\mathcal{C}$  of a fixed structure. There does not exist a rewriting theory for nested conditions yet.
- (4) Let  $\mathcal{S}_n$  be a star-like shape: there exists a central node with  $n$  outgoing spans (the shape  $\bullet \leftarrow \bullet \rightarrow \bullet \leftarrow \bullet \rightarrow \bullet$  being the special case for  $n = 2$ ). Let  $\bar{\mathcal{S}}_n$  denote the same shape with the arrows pointing to the central node designated to be mapped to monomorphisms, i.e.,  $S$  consists of the namely morphisms to the central node. König et al. use (slice categories of)  $[\bar{\mathcal{S}}_n, \mathcal{C}]$  (where  $\mathcal{C}$  is a suitable category of models) to formalize a correspondence relation between  $n$  different (meta-)models via  $n$  partial morphisms from a correspondence model [129, 204, 130].
- (5) In this thesis, we use the special case  $n = 2$  of the above example to define *(typed attributed) partial triple graphs*.

These example categories have been introduced in the literature but, except for the second example, to the best of our knowledge, there is no theory on structure transformation over these categories yet. For the second example, it is known that the full subcategory (i.e., linear rules) of the corresponding functor category (i.e., general spans) is *not* adhesive and Machado (et al.) develop different ways to mitigate that problem [160, 161].

Our first contribution (in Sect. 5.2) is to provide a unifying definition of categories like the above example categories. The corresponding categories are called *S-cartesian functor categories*. We show how and under which circumstances these categories are adhesive. It turns out that, if the base category is a PVK square adhesive category and the class of morphisms is restricted to those natural transformations where all naturality squares induced by  $S$ -morphisms are pullback squares (i.e., *cartesian*), the resulting category is PVK square adhesive again (Theorem 5.8). We further present conditions under which the additional HLR properties are valid for  $S$ -cartesian functor categories. Hence, we show under which conditions the classical theory of double-pushout rewriting holds for  $S$ -cartesian functor categories. As an example, we show that choosing (attributed) graphs as base category  $\mathcal{C}$ , every  $S$ -cartesian functor category over  $\mathcal{C}$  meets a

variant of adhesiveness and the additional HLR properties hold as well. Hence, we obtain the full elaborate theory for double-pushout rewriting for  $S$ -cartesian functor categories over graphs and attributed graphs (Theorem 5.20 and Corollary 5.21).

Our practical motivation for this chapter is to have a comprehensive formal foundation at hand when, in Chapter 6, we base our model synchronization approach on the rewriting of typed attributed partial triple graphs. We therefore, as a second contribution (in Sect. 5.3), instantiate (typed attributed) partial triple graphs as an  $S$ -cartesian functor category over the category **Graph** (**AGraph**) of (attributed) graphs and thus automatically obtain results known from double-pushout rewriting like the *Local Church–Rosser* or the *Concurrency Theorem*. As an application and sanity check, we define the decomposition of a triple rule on partial triple graphs into a source and a forward rule in analogy to the procedure for TGGs [195] and prove its correctness (Theorem 5.27). Besides generalized concurrent rules, this is the second ingredient for the construction of the repair rules that we employ in our model synchronization approach. The repair rules will be obtained by decomposing certain GCRs (or, more specifically, short-cut rules) that are computed from the rules of the grammar.

The rest of this chapter is structured as follows. First, in Sect. 5.2 we develop the theory of  $S$ -cartesian functor categories that is subsequently applied to partial triple graphs in Sect. 5.3. After these main contributions, we discuss related work in Sect. 5.4 and conclude in Sect. 5.5. The often rather technical proofs of the results in Sect. 5.2.2 are presented in Appendix B.2.

This chapter is based on [134] (and its preceding conference version [133]). Here, we somewhat update the presentation and the discussion of related work and add technical lemmas that more closely relate an  $S$ -cartesian functor category with its surrounding functor category (Lemmas 5.10, 5.11, 5.25, and 5.26). These lemmas will be needed in Chapter 6. The most important new contributions beyond the original papers, however, are Lemma 5.28 and Corollary 5.29 that clarify how sesqui-pushout rewriting of (typed) attributed partial triple graph works.

## 5.2 *S*-Cartesian Functor Categories and Their HLR Properties

We are interested in investigating certain subcategories of functor categories  $[\mathcal{X}, \mathcal{C}]$  where  $\mathcal{C}$  satisfies some variant of adhesiveness. Particularly, these subcategories arise by restricting to those functors that map all morphisms from a designated set  $S$  of morphisms in  $\mathcal{X}$  to  $\mathcal{M}$ -morphisms in  $\mathcal{C}$ . We call such functors *S*- $\mathcal{M}$ -restricted. We first introduce these categories as *S*-cartesian functor categories in Sect. 5.2.1. It turns out that the keys to preserving adhesiveness are to consider a non-full subcategory and to require the base category  $\mathcal{C}$  to be PVK square adhesive. In Sect. 5.2.2, we discuss conditions under which also additional HLR properties are preserved by our construction. In Sect. 5.2.3, we apply the previously developed theory to the case of *S*-cartesian functor categories over graphs.

### 5.2.1 *S*-Cartesian Functor Categories

We begin this section with the definition of *S*- $\mathcal{M}$ -restricted functors and *S*-cartesian natural transformations between them. These are the objects and morphisms that constitute our categories of interest.

**Definition 5.1** (*S*- $\mathcal{M}$ -restricted functor). Given a small category  $\mathcal{X}$ , a subset  $S$  of the morphisms of  $\mathcal{X}$ , and an arbitrary category  $\mathcal{C}$  with a designated class of monomorphisms  $\mathcal{M}$ , an *S*- $\mathcal{M}$ -restricted functor is a functor  $F : \mathcal{X} \rightarrow \mathcal{C}$  such that for every morphism  $m \in S$  the morphism  $Fm$  is an  $\mathcal{M}$ -morphism.

**Example 5.1.** Let  $\mathcal{X} = \bullet \rightarrow \bullet$ ,<sup>2</sup>  $\mathcal{C} = \mathbf{Set}$ , and  $\mathcal{M}$  consist of all monomorphisms, i.e., all injective functions. Choose  $S$  to consist of the only non-identity morphism of  $\mathcal{X}$ . Then, an *S*- $\mathcal{M}$ -restricted functor maps the two objects of  $\mathcal{X}$  to sets  $A$  and  $B$  and the morphism between the two objects to an injective function  $m : A \hookrightarrow B$ . Hence, in this case *S*- $\mathcal{M}$ -restricted functors correspond to injective functions. In contrast, the functor category  $[\bullet \rightarrow \bullet, \mathbf{Set}]$  contains all arrows, injective or not, as objects. Analogously, arbitrary classes of monomorphisms  $\mathcal{M}$  of a category  $\mathcal{C}$  can be defined as *S*- $\mathcal{M}$ -restricted functors.

---

<sup>2</sup>Again, we do not depict identity morphisms.

Similarly, as already indicated in Sect. 5.1, the objects of interest of all examples presented there can be defined as  $S$ - $\mathcal{M}$ -restricted functors for suitable choices of  $\mathcal{X}$ ,  $S$ ,  $\mathcal{C}$ , and  $\mathcal{M}$ .

We are interested in lifting the theory of double-pushout rewriting to categories with  $S$ - $\mathcal{M}$ -restricted functors as objects. Whereas in the case of functor categories this can be done easily because the relevant computations can be performed componentwise, the same is not true for  $S$ - $\mathcal{M}$ -restricted functors. It turns out that the pushout of  $S$ - $\mathcal{M}$ -restricted functors, computed componentwise in the ambient functor category, does not need to result in an  $S$ - $\mathcal{M}$ -restricted functor again. This is even the case when the pushout is computed along natural transformations where every component is an  $\mathcal{M}$ -morphism. This means that pushouts, if they exist, at least lead to unexpected results. We present a simple example of this in the case of injective functions between sets. Subsequently, we argue why such a category cannot be adhesive (in any variant) with respect to the class of componentwise  $\mathcal{M}$ -morphisms. The counterexample below has already been presented in [161].

**Example 5.2.** Let  $\mathcal{C}$  be the adhesive category **Set** and  $\mathcal{X}$  the category  $\bullet \rightarrow \bullet$ . Consider the full subcategory of the functor category  $[\bullet \rightarrow \bullet, \mathbf{Set}]$  induced by those functors that map the only non-identity morphism to an injective function in **Set**, i.e., the category with injective functions as objects and commuting squares as morphisms as in Example 5.1 above. Let  $[n]$  denote a set with  $n$  elements and consider the commuting cube depicted in Fig. 5.1: The two squares in the back are a span of monomorphisms in that category. Computing the top and the bottom square as pushouts, i.e., computing the pushout of the two squares in the back in the category  $[\bullet \rightarrow \bullet, \mathbf{Set}]$ , results in a function that is not injective. In this case, one checks that the pushout in the category of injective functions is the identity morphism  $id_{[1]} : [1] \hookrightarrow [1]$  (see Fig. 5.2). In particular, it is not computed componentwise.<sup>3</sup> Furthermore, this pushout is not a pullback, which implies that the category of injective functions is not adhesive.

A reason why a full subcategory of  $S$ - $\mathcal{M}$ -restricted functors is not generally adhesive is given in terms of *regularity* of the involved morphisms:

<sup>3</sup>In [133], we wrongly claimed that this pushout did not exist.

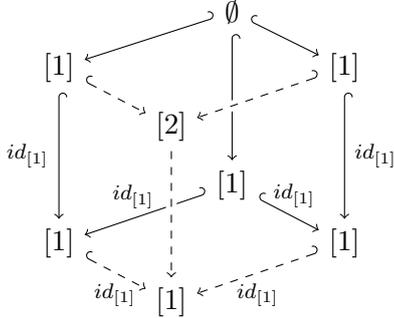


Figure 5.1: Pushout of morphisms between injective functions in the arrow category, computed componentwise

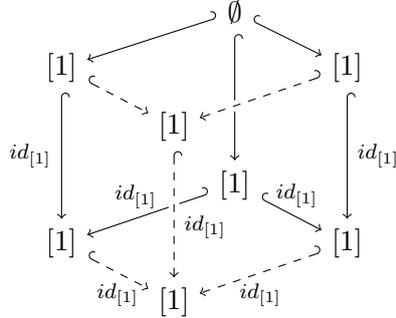


Figure 5.2: Pushout of morphisms between injective functions in the full subcategory of the arrow category that has injective functions as objects

Whereas every  $\mathcal{M}$ -morphism in any kind of adhesive category is a regular monomorphism (as recalled in Fact 3.2), it turns out that the componentwise  $\mathcal{M}$ -morphisms between  $S$ - $\mathcal{M}$ -restricted functors are not necessarily regular. In contrast, adapting [120, Lemma 14] to the situation of  $S$ - $\mathcal{M}$ -restricted functors, the naturality squares of a natural transformation between  $S$ - $\mathcal{M}$ -restricted functors need to additionally form pullback squares when a morphism that stems from  $S$  is involved.

**Fact 5.1** (Regular monos between  $S$ - $\mathcal{M}$ -restricted functors; cf. [120, Lemma 14]). *Let  $\mathcal{C}$  be a category that meets some kind of adhesiveness with respect to an admissible class of monomorphisms  $\mathcal{M}$ . Let  $\mathcal{X}$  be a small category and  $S$  a subset of its morphisms. Then, in the full subcategory of  $[\mathcal{X}, \mathcal{C}]$  that is induced by the  $S$ - $\mathcal{M}$ -restricted functors, given a natural transformation  $\mu : F \rightarrow G$  between  $S$ - $\mathcal{M}$ -restricted functors  $F$  and  $G$  such that every component  $\mu_x : Fx \hookrightarrow Gx \in \mathcal{M}$ ,  $\mu$  is a regular monomorphism if and only if for every  $m : x \rightarrow y \in S$ , the corresponding naturality square  $Gm \circ \mu_x = \mu_y \circ Fm$  is a pullback square in  $\mathcal{C}$ .*

Hence, if we are interested in obtaining an adhesive category consisting of  $S$ - $\mathcal{M}$ -restricted functors, it is necessary to restrict the natural transformations between them. This motivates the next definition.

**Definition 5.2** ( $S$ -cartesian natural transformation.  $S$ -cartesian functor category). Let an arbitrary category  $\mathcal{C}$  with designated class of monomorphisms  $\mathcal{M}$ , a small category  $\mathcal{X}$ , and a subset  $S$  of the morphisms of  $\mathcal{X}$  be given. A natural transformation  $\sigma : F \rightarrow G$  between two  $S$ - $\mathcal{M}$ -restricted functors is  $S$ -cartesian if for every morphism  $S \ni m : x \rightarrow y$  the corresponding naturality square  $\sigma_y \circ Fm = Gm \circ \sigma_x$ —where  $\sigma_x, \sigma_y$  denote the components of the natural transformation—is a pullback square.

An  $S$ -cartesian functor category is a category that has  $S$ - $\mathcal{M}$ -restricted functors as objects and  $S$ -cartesian natural transformations as morphisms between them. We denote such a category by  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ . By  $\mathcal{M}^{\text{cart}}$  we denote the class of  $S$ -cartesian natural transformations where each component stems from  $\mathcal{M}$ . Moreover,  $\mathcal{M}^{\text{comp}}$  denotes the class of natural transformations in  $[\mathcal{X}, \mathcal{C}]$  whose every component is an  $\mathcal{M}$ -morphism.

**Example 5.3.** Continuing Example 5.1 of injective functions as  $S$ - $\mathcal{M}$ -restricted functors, we now consider  $S$ -cartesian natural transformations between them. Given two injective functions  $m : A \hookrightarrow B$ ,  $m' : C \hookrightarrow D$  considered as  $S$ - $\mathcal{M}$ -restricted functors, a natural transformation between them is just a pair of functions  $f : A \rightarrow C, g : B \rightarrow D$  such that  $m' \circ f = g \circ m$ . Since the preimage of  $m$  and  $m'$  stems from  $S$ , an  $S$ -cartesian natural transformation between  $m$  and  $m'$  consists of functions  $f, g$  like above such that the resulting square not only commutes but is even a pullback. Hence, the  $S$ -cartesian functor category  $[\bullet \hookrightarrow \bullet, \mathbf{Set}_{\text{inj}}]_{\text{cart}}$  has injective functions as objects and pullback squares as morphisms between them. The class  $\mathcal{M}^{\text{cart}}$  consists of pairs of injective functions  $f : A \hookrightarrow C, g : B \hookrightarrow D$  such that there are injective functions  $m : A \hookrightarrow B, m' : C \hookrightarrow D$  completing the pair to a pullback square.

**Assumption 5.1.** In the rest of this section (i.e., throughout Sect. 5.2.1), we always assume a small category  $\mathcal{X}$ , a subset  $S$  of the morphisms of  $\mathcal{X}$ , and an arbitrary category  $\mathcal{C}$  with a designated class of monomorphisms  $\mathcal{M}$  to be given without referencing them explicitly. For simplicity, we assume  $\mathcal{M}$  to be an admissible class of monomorphisms, even if this is not necessary to prove several of the following results. If  $\mathcal{C}$  is required to meet some kind of adhesiveness, this is explicitly mentioned. In that case,  $\mathcal{M}$  is always assumed to be the class of monomorphisms with respect to which adhesiveness holds. Finally, to stress generality, in this section we denote

$$\begin{array}{ccc}
 Fx & \xrightarrow{id_{Fx}} & Fx \\
 Fm \downarrow & & \downarrow Fm \\
 Fy & \xrightarrow{id_{Fy}} & Fy
 \end{array}$$

 Figure 5.3: Identity morphism in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ 

$$\begin{array}{ccccc}
 Fx & \xrightarrow{\sigma_x} & Gx & \xrightarrow{\tau_x} & Hx \\
 Fm \downarrow & & \downarrow Gm & & \downarrow Hm \\
 Fy & \xrightarrow{\sigma_y} & Gy & \xrightarrow{\tau_y} & Hy
 \end{array}$$

 Figure 5.4: Composition of morphisms in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ 

morphisms in  $S$ -cartesian functor categories with greek letters (as they are natural transformations).

Since the composition of two pullbacks is a pullback and the identity natural transformation is trivially  $S$ -cartesian,  $S$ - $\mathcal{M}$ -restricted functors with  $S$ -cartesian natural transformations as morphisms indeed form a category.

**Proposition 5.2** ( $S$ -cartesian functor category). *An  $S$ -cartesian functor category  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  is indeed a category. Moreover, there is an inclusion functor  $I : [\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}} \hookrightarrow [\mathcal{X}, \mathcal{C}]$ .*

*Proof.* Let  $F$  be an arbitrary  $S$ - $\mathcal{M}$ -restricted functor from  $\mathcal{X}$  to  $\mathcal{C}$ . The identity natural transformation is a morphism in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  since for every  $S \ni m : x \rightarrow y$ , the square depicted in Fig. 5.3 is a pullback square.

Given two  $S$ -cartesian natural transformations  $\sigma : F \rightarrow G$  and  $\tau : G \rightarrow H$ , the composition  $\tau \circ \sigma$  is  $S$ -cartesian since for every  $S \ni m : x \rightarrow y$ , the composition of the two pullbacks depicted in Fig. 5.4 is a pullback. Associativity of composition and neutrality of the composition with the identity natural transformation are just inherited from the full functor category  $[\mathcal{X}, \mathcal{C}]$ .

Moreover, mapping every  $S$ - $\mathcal{M}$ -restricted functor from  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  to itself as object in  $[\mathcal{X}, \mathcal{C}]$  and every  $S$ -cartesian natural transformation from  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  to itself as morphism in  $[\mathcal{X}, \mathcal{C}]$  gives a faithful functor that is also injective on objects.  $\square$

Section 5.3 is devoted to develop the category of typed attributed partial triple graphs as an instantiation of this general framework. In particular,

it presents concrete examples illustrating the abstract notions. In this section, we prove that if a category  $\mathcal{C}$  is PVK square adhesive, categories  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  are PVK square adhesive again (Theorem 5.8). We first collect results that contribute to the proof of our main theorem. They are of independent interest as they determine that pushouts and pullbacks along  $\mathcal{M}^{\text{cart}}$ -morphisms are computed componentwise in  $S$ -cartesian functor categories. Recall that a functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  is said to *create (co-)limits* of a certain type  $\mathcal{J}$  if for every diagram  $D : \mathcal{J} \rightarrow \mathcal{C}$  and every (co-)limit for  $F \circ D$  in  $\mathcal{D}$  there exists a unique preimage under  $F$  that is a (co-)limit of  $D$  in  $\mathcal{C}$  [17]. Moreover, we repeatedly make use of the fact that if a (co-)limit of a certain type exists in the category  $\mathcal{C}$ , a functor category  $[\mathcal{X}, \mathcal{C}]$  has (co-)limits of this type as well and they are computed componentwise [17, Proposition 8.8.].

**Proposition 5.3** (Pullbacks in  $S$ -cartesian functor categories). *If a category  $\mathcal{C}$  has pullbacks, categories  $[\mathcal{X}, \mathcal{C}]$  and  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  have pullbacks and the inclusion functor  $I : [\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}} \hookrightarrow [\mathcal{X}, \mathcal{C}]$  creates these. In particular,  $[\mathcal{X}, \mathcal{C}]$  and  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  have pullbacks along  $\mathcal{M}^{\text{cart}}$ -morphism and  $I$  creates these.*

*Proof.* Since  $[\mathcal{X}, \mathcal{C}]$  is known to have these pullbacks if  $\mathcal{C}$  has, it only remains to show that the inclusion functor  $I$  creates these. Let two  $S$ -cartesian natural transformations  $\sigma : B \rightarrow D$  and  $\tau : C \rightarrow D$  between  $S$ - $\mathcal{M}$ -restricted functors  $B, C, D$  be given. Since  $I$  is an inclusion, we have to show that a componentwise computed pullback object  $A$  in  $[\mathcal{X}, \mathcal{C}]$  with natural transformations  $\eta : A \rightarrow B$  and  $\mu : A \rightarrow C$  is a pullback in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  as well. This means, we have to show (i) that  $A$  is an  $S$ - $\mathcal{M}$ -restricted functor and that  $\eta$  and  $\mu$  are  $S$ -cartesian and (ii) that given another  $S$ - $\mathcal{M}$ -restricted functor  $Q$  with  $S$ -cartesian natural transformations  $\nu_1 : Q \rightarrow B$  and  $\nu_2 : Q \rightarrow C$  such that  $\sigma \circ \nu_1 = \tau \circ \nu_2$ , the unique mediating natural transformation  $\epsilon : Q \rightarrow A$ , guaranteed to exist in  $[\mathcal{X}, \mathcal{C}]$ , is  $S$ -cartesian as well (i.e., is a morphism in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ ).

To show (i), let  $S \ni m : x \rightarrow y$  be an arbitrary morphism from  $S$ . Computing the pullbacks of the co-spans  $Bx \rightarrow Dx \leftarrow Cx$  and  $By \rightarrow Dy \leftarrow Cy$  leads to the left cube depicted in Fig. 5.5. These pullbacks exist in  $\mathcal{C}$  by assumption if  $\mathcal{C}$  has all pullbacks. In any case, they exist if, e.g., every component of  $\tau$  is an  $\mathcal{M}$ -morphism (since  $\mathcal{M}$  is admissible). The morphism  $Am$  is the unique morphism determined by the universal

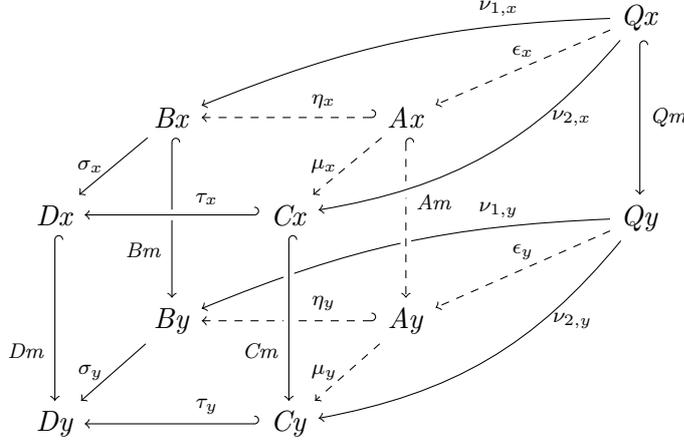


Figure 5.5: Creation of pullbacks by the inclusion functor

property of the pullback square at the bottom of the cube. Since the front faces are pullbacks (by assumption) and the top and the bottom square are computed as pullbacks, the faces to the back are pullbacks as well. Moreover, since  $\mathcal{M}$  is closed under pullbacks, the morphism  $Am$  is an  $\mathcal{M}$ -morphism. Hence,  $A$  is an  $S$ - $\mathcal{M}$ -restricted functor and  $\eta$  and  $\mu$  are  $S$ -cartesian.

(ii) Since the square  $Qx \rightarrow Cx \hookrightarrow Cy \leftarrow Qy \leftarrow Qx$  is a pullback by assumption and  $Ax \rightarrow Cx \hookrightarrow Cy \leftarrow Ay \leftarrow Ax$  is a pullback as well, pullback decomposition implies that the square  $Qx \rightarrow Ax \hookrightarrow Ay \leftarrow Qy \leftarrow Qx$  is also a pullback. Hence,  $\epsilon$  is an  $S$ -cartesian natural transformation.  $\square$

Every morphism in a functor category  $[\mathcal{X}, \mathcal{C}]$  where every component is a monomorphism in  $\mathcal{C}$  is a monomorphism; if  $\mathcal{C}$  has pullbacks, the converse is also true. The second statement follows directly from the characterization of monomorphisms as those morphisms whose kernel pairs are spans of identities (Fact 3.9 (3)). The next lemma characterizes the monomorphisms of  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  in an analogous fashion; its proof uses this characterization of monomorphisms in the same way.

**Lemma 5.4** (Monomorphisms in  $S$ -cartesian functor categories). *Let  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  be an  $S$ -cartesian functor category. Then, every morphism in  $[\mathcal{X}_S,$*

$\mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  where every component is a monomorphism in  $\mathcal{C}$  is a monomorphism. If  $\mathcal{C}$  has pullbacks, the converse is true.

*Proof.* Since  $I$  is a faithful functor from  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  to  $[\mathcal{X}, \mathcal{C}]$  and faithful functors reflect monomorphisms, a morphism in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  where every component is a monomorphism in  $\mathcal{C}$  is a monomorphism.

Given a monomorphism  $\mu : A \rightarrow B$  in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ , the span  $A \xleftarrow{id_A} A \xrightarrow{id_A} A$  is the kernel pair of  $\mu$  by [Fact 3.9 \(3\)](#). If  $\mathcal{C}$  has pullbacks, by [Proposition 5.3](#) this pullback is computed componentwise. Hence, for every  $x \in \mathcal{X}$ , the pullback of  $Ax \xrightarrow{\mu_x} Bx \xleftarrow{\mu_x} Ax$  in  $\mathcal{C}$  is given by  $Ax \xleftarrow{id_{Ax}} Ax \xrightarrow{id_{Ax}} Ax$ . Using the other direction of [Fact 3.9 \(3\)](#), this implies that every component  $\mu_x$  is a monomorphism.  $\square$

The last two results together imply that admissible classes of monomorphisms in  $S$ -cartesian functor categories can be defined componentwise.

**Corollary 5.5** (Admissibility of  $\mathcal{M}^{\text{cart}}$ ). *If  $\mathcal{M}$  is an admissible class of monomorphisms in a category  $\mathcal{C}$ ,  $\mathcal{M}^{\text{cart}}$  is an admissible class of monomorphisms in any  $S$ -cartesian functor category  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ .*

The next proposition states that also pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms are calculated componentwise in  $S$ -cartesian functor categories. In contrast to the case of pullbacks, the proof requires that  $\mathcal{C}$  is PVK square adhesive.

**Proposition 5.6** (Pushouts in  $S$ -cartesian functor categories). *Let  $\mathcal{C}$  be a PVK square adhesive category and  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  an  $S$ -cartesian functor category. Then  $[\mathcal{X}, \mathcal{C}]$  and  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  have pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms and the inclusion functor  $I : [\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}} \hookrightarrow [\mathcal{X}, \mathcal{C}]$  creates these.*

*Proof.* Because  $\mathcal{C}$  has pushouts along  $\mathcal{M}$ -morphisms,  $[\mathcal{X}, \mathcal{C}]$  has pushouts along  $\mathcal{M}^{\text{comp}}$ -morphisms and therefore also pushout along  $\mathcal{M}^{\text{cart}}$ -morphisms and they are computed componentwise. It remains to show that the inclusion functor  $I$  creates these.

Let two  $S$ -cartesian natural transformations  $\sigma : A \rightarrow B$  and  $\tau : A \rightarrow C$  between  $S$ - $\mathcal{M}$ -restricted functors  $A, B, C$  be given and let  $\tau \in \mathcal{M}^{\text{cart}}$ . Since  $I$  is an inclusion, we have to show that a componentwise computed pushout  $D$  in  $[\mathcal{X}, \mathcal{C}]$  with natural transformations  $\eta : B \rightarrow D$  and  $\mu : C \rightarrow D$  is a pushout in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  as well. This means, we have to show (i) that

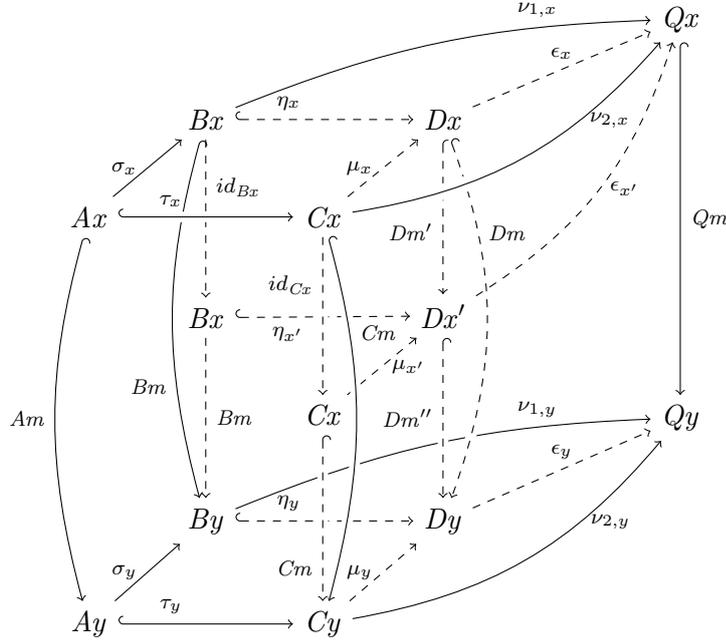


Figure 5.6: Pushout along an  $\mathcal{M}^{\text{cart}}$ -morphism

$D$  is an  $S$ - $\mathcal{M}$ -restricted functor and that  $\eta$  and  $\mu$  are  $S$ -cartesian and (ii) that given another  $S$ - $\mathcal{M}$ -restricted functor  $Q$  with  $S$ -cartesian natural transformations  $\nu_1 : B \rightarrow Q$  and  $\nu_2 : C \rightarrow Q$  such that  $\nu_1 \circ \sigma = \nu_2 \circ \tau$ , the unique mediating natural transformation  $\epsilon : D \rightarrow Q$ , guaranteed to exist in  $[\mathcal{X}, \mathcal{C}]$ , is  $S$ -cartesian as well (i.e., is a morphism in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ ).

To show (i), let  $S \ni m : x \rightarrow y$  be an arbitrary morphism from  $S$ . Computing the pushouts of the spans  $Bx \leftarrow Ax \hookrightarrow Cx$  and  $By \leftarrow Ay \hookrightarrow Cy$  leads to the left cube depicted in Fig. 5.6. These pushouts exist, since  $\mathcal{C}$  is PVK square adhesive and  $\tau_x, \tau_y \in \mathcal{M}$  by assumption. The morphism  $Dm$  is the unique morphism determined by the universal property of the pushout square at the top of the cube. That the front faces are pullbacks and  $Dm$  is an  $\mathcal{M}$ -morphism is a direct consequence of the bottom pushout being a PVK square, the back faces of the cube being pullbacks, and  $Bm, Cm \in \mathcal{M}$  by assumption. Hence,  $D$  is an  $S$ - $\mathcal{M}$ -restricted functor and  $\eta$  and  $\mu$  are  $S$ -cartesian.

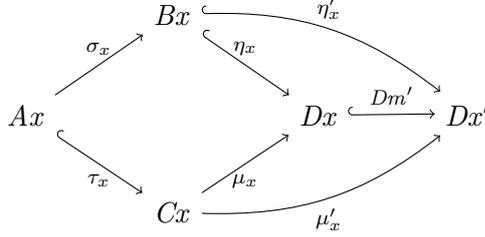


Figure 5.7:  $Dm'$  as isomorphism mediating between two pushout objects

(ii) It remains to show that  $Dx \rightarrow Qx \hookrightarrow Qy \leftarrow Dy \leftarrow Dx$  is a pullback square. For this, we mimic the proof of Lemma 2.2. in [82] in the context of PVK squares. First compute the pullback  $Dx'$  of  $Qx \hookrightarrow Qy \leftarrow Dy$  which exists since  $Qm \in \mathcal{M}$ . The universal property of this pullback implies the existence of a unique morphism  $Dm' : Dx \rightarrow Dx'$  such that  $Dm = Dm'' \circ Dm'$  and  $\epsilon_x = \epsilon_{x'} \circ Dm'$ . Thus, if  $Dm'$  is an isomorphism, the square  $Dx \rightarrow Qx \hookrightarrow Qy \leftarrow Dy \leftarrow Dx$  is a pullback square. We show that to be the case.

By composition of pullbacks, pulling back  $Dm''$  along  $\mu_y$  and  $\eta_y$ , respectively, results in  $Cx$  and  $Bx$  as pullback objects, again. This can be completed into a commuting cube by also pulling back  $\tau_y$  along  $Cm$  and  $\sigma_y$  along  $Bm$  both resulting in  $Ax$  as pullback object (not depicted). Since the bottom pushout of the cube is a PVK square and  $Bm, Cm \in \mathcal{M}$ , this exhibits also  $Dx'$  as pushout object of the span  $Bx \leftarrow Ax \hookrightarrow Cx$ . This means, we obtain Fig. 5.7 where both squares are pushouts. Hence,  $Dm' : Dx \rightarrow Dx'$  is the unique isomorphism mediating between the two pushout objects.  $\square$

The next proposition clarifies that, under rather general circumstances, PVK square adhesiveness is not only a sufficient but a necessary condition for the componentwise computation of pushouts in  $S$ -cartesian functor categories. While this result is not strictly necessary to develop our theory of rewriting in  $S$ -cartesian functor categories (for this, Proposition 5.6 suffices), it shows the intimate connection between the properties of an  $S$ -cartesian functor category and PVK square adhesiveness of the underlying base category.

**Proposition 5.7** (PVK square adhesiveness of suitable base categories). *Let  $\mathcal{C}$  be a category with an admissible class of monomorphism  $\mathcal{M}$  such that pushouts along  $\mathcal{M}$ -morphisms exist and are stable under pullback along  $\mathcal{M}$ -morphisms. If the inclusion functor  $I : [\bullet \hookrightarrow \bullet, \mathcal{C}_{\mathcal{M}}]_{\text{cart}} \hookrightarrow [\bullet \rightarrow \bullet, \mathcal{C}]$  creates pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms, the category  $\mathcal{C}$  is PVK square adhesive w.r.t.  $\mathcal{M}$ . In particular, if  $\mathcal{C}$  is adhesive HLR or  $\mathcal{M}$ -adhesive and  $I$  creates pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms,  $\mathcal{C}$  is also PVK square adhesive.*

*Proof.* Let  $\mathcal{C}$  be as stated above and  $I$  create pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms. We have to show that pushouts along  $\mathcal{M}$ -morphisms are PVK squares in  $\mathcal{C}$ . Assume a pushout like Fig. 3.2 with  $m \in \mathcal{M}$  and a commutative cube over it like Fig. 3.3 with  $b, c \in \mathcal{M}$  and backfaces pullbacks to be given. If both front faces are pullbacks and  $d \in \mathcal{M}$ , the stability of pushouts along  $\mathcal{M}$ -morphisms under pullback along  $\mathcal{M}$ -morphisms implies that the top face is a pushout. If the top face is a pushout, the fact that  $I$  creates pushouts implies that the whole cube is a pushout in  $[\bullet \hookrightarrow \bullet, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ . This means that the front faces are pullbacks and  $d \in \mathcal{M}$ . In summary, pushouts along  $\mathcal{M}$ -morphisms are PVK squares in  $\mathcal{C}$ .

For the statement about adhesive HLR and  $\mathcal{M}$ -adhesive categories, note that in both cases this implies that pushouts along  $\mathcal{M}$ -morphisms exist and are stable under pullback along  $\mathcal{M}$ -morphisms.  $\square$

Together, the obtained results guarantee that the construction of  $S$ -cartesian functor categories preserves adhesiveness.

**Theorem 5.8** (Preservation of adhesiveness). *Let  $\mathcal{C}$  be a PVK square adhesive category and  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  an  $S$ -cartesian functor category. Then  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  is PVK square adhesive w.r.t.  $\mathcal{M}^{\text{cart}}$ .*

*Moreover, if  $\mathcal{C}$  meets any other kind of adhesiveness listed in Definition 3.3 additionally, also this variant of adhesiveness is preserved w.r.t. the class  $\mathcal{M}^{\text{cart}}$ .*

*Proof.* By Corollary 5.5,  $\mathcal{M}^{\text{cart}}$  constitutes an admissible class of monomorphisms in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ . Since, in case  $\mathcal{C}$  is PVK square adhesive, the inclusion functor  $I : [\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}} \hookrightarrow [\mathcal{X}, \mathcal{C}]$  creates pullbacks and pushouts along such natural transformations (Propositions 5.3 and 5.6), these pushouts and pullbacks are calculated componentwise. In particular, their properties in  $\mathcal{C}$  also hold in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  which implies that the same variant of adhesiveness is met.  $\square$

Next we note that rule applications (of plain rules) are preserved and reflected by the inclusion functor  $I$ . This could also be stated by saying that  $I$  creates pushout complements. Intuitively, this means that whenever defined in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ , applying a rule in this category or in the full functor category  $[\mathcal{X}, \mathcal{C}]$  yields the same result. For notational simplicity, we suppress the inclusion functor  $I$  in the statement of this result.

**Proposition 5.9** (Preservation and reflection of rule applications). *Let  $\mathcal{C}$  be a PVK square adhesive category. Let  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  be an  $S$ -cartesian functor category,  $p = (L \xleftarrow{\lambda} K \xrightarrow{\rho} R)$  be a rule, and  $\mu : L \rightarrow G$  a match such that  $\lambda, \rho, \mu, L, K, R, G$  are morphisms and objects of  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ . Then  $p$  is applicable to  $G$  with match  $\mu$  in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  if and only if it is so in  $[\mathcal{X}, \mathcal{C}]$ . Moreover, the resulting objects of the two rule applications coincide (up to isomorphism).*

*Proof.* That a rule application in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  is a rule application in  $[\mathcal{X}, \mathcal{C}]$  is a direct corollary to Proposition 5.6 and Theorem 5.8.

Let the plain rule  $p$  be applicable to  $G$  at match  $\mu$  in  $[\mathcal{X}, \mathcal{C}]$ . Let  $D$  be the pushout complement computed in the first step of the rule application and  $\kappa : K \rightarrow D$  and  $\phi : D \rightarrow G$  the resulting morphisms. It suffices to show that  $\kappa$  and  $\phi$  are  $S$ -cartesian and that  $D$  is an  $S$ - $\mathcal{M}$ -restricted functor. Then, the pushout complement is a pushout complement in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  as well. That computing the second pushout of the rule application is the same in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  or in  $[\mathcal{X}, \mathcal{C}]$  is, again, stated by Proposition 5.6.

Let  $\mathcal{M} \ni m : x \rightarrow y$  be arbitrary. Figure 5.8 depicts the pushout complement at the components  $x$  and  $y$  with morphism  $Dm : Dx \rightarrow Dy$  which exists by assumption. Since  $\lambda$  and  $\mu$  are  $S$ -cartesian, the two vertical squares to the left are pullbacks and therefore, the outer square  $Kx \rightarrow Gx \hookrightarrow Gy \leftarrow Ky \leftarrow Kx$  is a pullback. Hence, if we show that  $Dx \rightarrow Gx \hookrightarrow Gy \leftarrow Dy \leftarrow Dx$  is a pullback square,  $Kx \hookrightarrow Ky \rightarrow Dy \leftarrow Dx \leftarrow Kx$  is a pullback square as well by pullback decomposition and  $Dm$  is an  $\mathcal{M}$ -morphism since it arises as pullback along the  $\mathcal{M}$ -morphism  $Gm$ . Computing the pullback of the co-span  $Gx \hookrightarrow Gy \leftarrow Dy$  (which exists in  $\mathcal{C}$  since  $Gm$  is an  $\mathcal{M}$ -morphism) results in an object  $Dx'$  with  $\mathcal{M}$ -morphism  $Dm' : Dx' \hookrightarrow Dy$ . Then, the pullback of  $\kappa_y$  along  $Dm'$  exists and results (up to isomorphism) in  $Kx$  again since it completes the pullback of the co-span  $Gx \hookrightarrow Gy \leftarrow Ky$ . Since the bottom square is a PVK square in

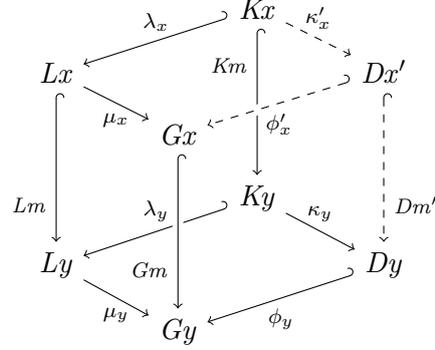
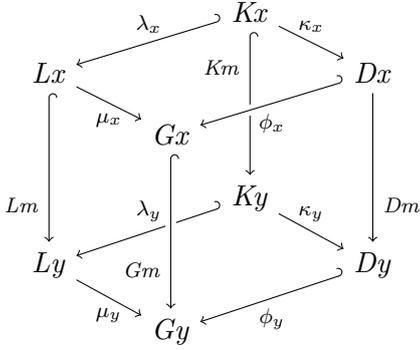


Figure 5.8: Pushout complement for one component      Figure 5.9: Pullback resulting in pushout complement

$\mathcal{C}$ ,  $Lm, Dm' \in \mathcal{M}$ , and all vertical faces are pullbacks, the top square is a pushout; the resulting cube is depicted in Fig. 5.9. Since pushout complements are unique in categories that are PVK square adhesive, there is an isomorphism  $i : Dx' \rightarrow Dx$  that is compatible with  $\phi_x$  and  $\phi'_x$ , i.e.,

$$\phi_x \circ i = \phi'_x .$$

Using this, we compute

$$\begin{aligned} \phi_y \circ Dm' &= Gm \circ \phi'_x \\ &= Gm \circ \phi_x \circ i \\ &= \phi_y \circ Dm \circ i \end{aligned}$$

which implies  $Dm' = Dm \circ i$  by monotonicity of  $\phi_y$ . Hence,  $i$  is also compatible with  $Dm'$  and  $Dm$ , and thus, the square at the right front in Fig. 5.8 is a pullback.  $\square$

The requirement on morphisms in  $S$ -cartesian functor categories to constitute pullback squares in the  $S$ -components is rather strict. In Sect. 5.3, a concrete example shows how this requirement restricts the expressiveness of rewriting in  $S$ -cartesian functor categories. For most of our practical purposes in Chapter 6, the achieved expressiveness still suffices. In some places, however, it does not. To elude this problem, we use the following

strategy: Instead of computing in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ , we take objects from that category but compute in the surrounding functor category  $[\mathcal{X}, \mathcal{C}]$  and then argue that the results belong to  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ . The following two lemmas will allow us to do that; again, we suppress the inclusion functor  $I : [\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}} \hookrightarrow [\mathcal{X}, \mathcal{C}]$  from our notation.

**Lemma 5.10** (Preservation property of deleting rules). *Let  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  be an  $S$ -cartesian functor category and let  $K, L, G$  be objects from it. Moreover, let  $\lambda : L \leftarrow K$  be a rule in  $[\mathcal{X}, \mathcal{C}]$  that only deletes and  $\mu : L \rightarrow G$  a match for that rule (i.e., the natural transformations  $\lambda$  and  $\mu$  are not necessarily  $S$ -cartesian). Then, the functor  $D \in [\mathcal{X}, \mathcal{C}]$  resulting from the corresponding transformation step is already an element of  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ , i.e., it is an  $S$ - $\mathcal{M}$ -restricted functor.*

Note that  $\lambda : K \leftarrow L$  means that  $\lambda \in \mathcal{M}^{\text{comp}}$ , i.e., every component of  $\lambda$  is an  $\mathcal{M}$ -morphism in  $\mathcal{C}$ ; however, its  $S$ -components do not need to constitute pullbacks.

*Proof.* We only have to show that for any  $s : x \rightarrow y \in S$ , we have  $Ds \in \mathcal{M}$ . For this, let  $\gamma : D \hookrightarrow G$  be the morphism (natural transformation) that results from the according transformation step  $t : G \Rightarrow_{\lambda, \mu} D$ . In particular, every component  $\gamma_x$  of  $\gamma$  is an  $\mathcal{M}$ -morphism (in  $\mathcal{C}$ ). We obtain  $\gamma_y \circ Ds = Gs \circ \gamma_x \in \mathcal{M}$  (see Fig. 5.10), which implies  $Ds \in \mathcal{M}$  by decomposition of  $\mathcal{M}$ -morphisms.  $\square$

$$\begin{array}{ccc}
 Dx & \xrightarrow{Ds} & Dy \\
 \gamma_x \downarrow & & \downarrow \gamma_y \\
 Gx & \xrightarrow{Gs} & Gy
 \end{array}$$

Figure 5.10: One  $S$ -component of the morphism  $\gamma : D \hookrightarrow G$

Virtually the same argument implies the similar next lemma.

**Lemma 5.11** (Preservation property of pullbacks). *Let  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  be an  $S$ -cartesian functor category and let  $A, B, C$  be objects from it. Moreover,*

let  $B \xrightarrow{\beta} A \xleftarrow{\gamma} C$  be a span in the surrounding functor category  $[\mathcal{X}, \mathcal{C}]$  with  $\beta \in \mathcal{M}^{\text{comp}}$ . Then, the functor  $P$  that results from pulling back that span is already an element of  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ , i.e.,  $P$  is an  $S$ - $\mathcal{M}$ -restricted functor.

Note again that  $\beta : B \hookrightarrow A \in \mathcal{M}^{\text{comp}}$  means that every component of  $\beta$  is an  $\mathcal{M}$ -morphism in  $\mathcal{C}$ , which, in particular, implies that a pullback of the span always exists in case  $\mathcal{M}$  is some admissible class of monomorphisms. However, the  $S$ -components of  $\beta$  do not need to constitute pullbacks. Therefore, the above lemma is not already covered by Proposition 5.3.

### 5.2.2 Additional HLR Properties of $S$ -Cartesian Functor Categories

In this section, we show under which circumstances the additional HLR properties, which we have recalled in Sect. 3.3.2, are preserved by the construction of  $S$ -cartesian functor categories. This is in the vein of [187] where the preservation of additional HLR properties is investigated for different ways to construct adhesive (HLR) categories; in particular also for functor categories. An overview of results for the more general case of  $\mathcal{M}$ -adhesive categories is given in [54]. While we aim at a certain generality in the following, our main goal is to provide criteria that are strong enough to show preservation in the case of the category of attributed graphs **AGraph** as base category for an  $S$ -cartesian functor category. We discuss the preservation properties in the same order as we introduced them in Sect. 3.3.2. All proofs are given in Appendix B.2.

**Binary coproducts and ( $\mathcal{M}$ -)initial objects** The existence of initial objects is always preserved under the construction of an  $S$ -cartesian functor category.

**Proposition 5.12** (Initial objects in  $S$ -cartesian functor categories). *Let  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  be an  $S$ -cartesian functor category with respect to an admissible class of monomorphisms  $\mathcal{M}$  of  $\mathcal{C}$ . If  $\mathcal{C}$  has an initial object  $I$ , the constant functor  $\Delta I : \mathcal{X} \rightarrow \mathcal{C}$  is an initial object in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ . If  $\mathcal{C}$  has an  $\mathcal{M}$ -initial object  $I$ , the constant functor  $\Delta I$  is an  $\mathcal{M}^{\text{cart}}$ -initial object in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ .*

Note that, if an  $S$ -cartesian functor category has an initial object and pushouts along the unique morphisms from the initial object, it has finite coproducts. In particular, if the base category  $\mathcal{C}$  has an  $\mathcal{M}$ -initial object, coproducts can be computed componentwise in the  $S$ -cartesian functor category, since pushouts along  $\mathcal{M}$ -morphisms are computed componentwise. We present an additional criterion for the existence of a componentwise computed coproduct that can be used in cases where no  $\mathcal{M}$ -initial object exists. For the statement, we need the following terminology: We say that coproducts are *stable under pullback along  $\mathcal{M}$ -morphisms* if pullbacks of  $\mathcal{M}$ -morphisms into coproducts along the coprojections of the coproduct result in coprojections of a coproduct again. This means that, given a diagram like the one depicted in Fig. 5.11 where  $m \in \mathcal{M}$ , if both squares are pullbacks, the top row is a coproduct diagram.

$$\begin{array}{ccccc}
 P_1 & \xrightarrow{p_1} & Y & \xleftarrow{p_2} & P_2 \\
 \downarrow & & \downarrow m & & \downarrow \\
 A & \xrightarrow{i_A} & A + B & \xleftarrow{i_B} & B
 \end{array}$$

Figure 5.11: Pullback of coprojections of a coproduct along an  $\mathcal{M}$ -morphism

**Proposition 5.13** (Coproducts in  $S$ -cartesian functor categories). *Let  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  be an  $S$ -cartesian functor category with respect to an admissible class of monomorphisms  $\mathcal{M}$  of  $\mathcal{C}$ . If  $\mathcal{C}$  has finite coproducts and these are stable under pullback along  $\mathcal{M}$ -morphisms,  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  has finite coproducts and they are computed componentwise.*

*Remark 5.1.* The above precondition of coproducts being stable under pullback along  $\mathcal{M}$ -morphisms is closely related to the definition of *extensive categories* [41]: In the proof of the proposition, we show that the converse is always true in variants of adhesive categories, i.e., in a commuting diagram where  $\mathcal{M}$ -morphisms connect two coproducts, both induced squares are pullbacks. This equivalence, but for all morphisms, can be used to define extensive categories. However, in the next section, we will discuss that the category of attributed graphs is an example that our requirements are strictly weaker than that of a category to be extensive.

**Factorization properties** Also the special properties that an additional class of morphisms  $\mathcal{M}'$  should satisfy are preserved without any assumptions.

**Proposition 5.14** (Decomposition and closedness in  $S$ -cartesian functor categories). *If  $\mathcal{C}$  is a PVK square adhesive category w.r.t. an admissible class of monomorphisms  $\mathcal{M}$  and  $\mathcal{M}'$  is a class of morphisms such that the  $\mathcal{M}$ - $\mathcal{M}'$  PO-PB decomposition property holds, the  $\mathcal{M}^{\text{cart}}$ - $\mathcal{M}'^{\text{cart}}$  PO-PB decomposition property holds in every  $S$ -cartesian functor category  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ .*

*Similarly, if the class  $\mathcal{M}'$  is closed under pushouts (or pullbacks) along  $\mathcal{M}$ -morphisms, the class of  $\mathcal{M}'^{\text{cart}}$ -morphisms is closed under pushouts (or pullbacks) along  $\mathcal{M}^{\text{cart}}$ -morphisms in every  $S$ -cartesian functor category  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$ .*

In contrast, we can only show preservation for factorization systems with additional properties.

**Proposition 5.15** ( $\mathcal{E}$ - $\mathcal{M}'$  factorization in  $S$ -cartesian functor categories). *Let  $\mathcal{C}$  be a category with an admissible class of monomorphism  $\mathcal{M}$  and unique  $\mathcal{E}$ - $\mathcal{M}'$  factorization. If this factorization is even an  $\mathcal{E}$ - $\mathcal{M}'$  factorization system and  $\mathcal{E}$  is stable under pullbacks along  $\mathcal{M}$ -morphisms,  $S$ -cartesian functor categories  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  have an  $\mathcal{E}$ - $\mathcal{M}'$  factorization system where  $\mathcal{E}$  and  $\mathcal{M}'$  are defined componentwise.*

*Remark 5.2.* While the conditions in the above proposition are rather strong, they still allow for a rich class of interesting instantiations: *Topoi* are known to be adhesive [142] (and thus, in particular, PVK square adhesive) and, moreover, in every topos the epimorphisms and monomorphisms form a factorization system and epimorphisms are closed under pullbacks (since every topos is a regular category); see, e.g., [33, Corollary 5.9.2 and Corollary 5.9.4]. From a practical point of view, this is interesting since categories of graph structures are topoi (in fact, presheaves over **Set**; see, e.g., [18, Sect. 5.1.] for a short overview) and hence categories like graphs, hypergraphs, typed graphs, labeled graphs, etc. all are suitable choices for base categories for  $S$ -cartesian functor categories when a factorization system is needed.

**Initial pushouts and  $\mathcal{M}$ -effective unions** The next proposition states that initial pushouts are preserved in  $S$ -cartesian functor categories provided that the base category has *intersections of  $\mathcal{M}$ -subobjects* which means that given a sink of  $\mathcal{M}$ -morphisms  $(c_i : C_i \hookrightarrow A')_{i \in I}$  (for some index class  $I$ ), this sink has a limit  $(C, (c'_i : C \hookrightarrow C_i)_{i \in I})$  such that  $c'_i \in \mathcal{M}$  for all  $i \in I$ . For the proof that this is the case, we suitably adapt the proof of the corresponding fact for general functor categories given in [187].

**Proposition 5.16** (Initial pushouts in  $S$ -cartesian functor categories). *Let  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  be an  $S$ -cartesian functor category where  $\mathcal{C}$  is PVK square adhesive w.r.t. an admissible class of monomorphisms  $\mathcal{M}$  such that  $\mathcal{C}$  has intersections of  $\mathcal{M}$ -subobjects and initial pushouts over  $\mathcal{M}'$ -morphisms. Then  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  has initial pushouts over morphisms where every component is from  $\mathcal{M}'$ .*

The property of having  $\mathcal{M}$ -effective unions is again preserved without the need of further assumptions.

**Proposition 5.17** ( $\mathcal{M}$ -effective unions in  $S$ -cartesian functor categories). *Let  $\mathcal{C}$  be a PVK square adhesive category with admissible class of monomorphisms  $\mathcal{M}$ . If  $\mathcal{C}$  has  $\mathcal{M}$ -effective unions, so has every  $S$ -cartesian functor category  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ .*

### 5.2.3 $S$ -Cartesian Functor Categories over Graphs

To sum up this part of the chapter and apply it, we show that **Graph** and **AGraph** are PVK square adhesive and meet all the assumptions that we introduced to ensure the preservation of the additional HLR properties. As a result, we obtain the whole established theory of double-pushout rewriting also for  $S$ -cartesian functor categories over those categories. Also the proofs of this section are given in Appendix B.2.

As mentioned in Sect. 3.1 and 3.4, (typed) attributed (triple) graphs are known to be adhesive HLR w.r.t. the class  $\mathcal{M}$  of attributed graph morphisms that consist of injective graph components and isomorphisms on the algebra part. Therefore, we give a simple sufficient condition for adhesive HLR categories to also be PVK square adhesive (w.r.t. the same class of monomorphism  $\mathcal{M}$ ) and apply it to the case of (typed) attributed (triple) graphs. This criterion has also been used in [18] in a slightly different context, namely to ensure  $\mathcal{M}$ -effective unions.

**Definition 5.3** (Additional decomposition of  $\mathcal{M}$ -morphisms). An admissible class of monomorphisms  $\mathcal{M}$  fulfills the *additional decomposition property* if for every  $m \circ n \in \mathcal{M}$ , where  $n \in \mathcal{M}$  and  $m$  is a monomorphism, also  $m \in \mathcal{M}$ .

**Lemma 5.18** (Additional decomposition implies PVK square adhesiveness). *Let  $(\mathcal{C}, \mathcal{M})$  be an adhesive HLR category. If  $\mathcal{C}$  has all pullbacks and  $\mathcal{M}$  fulfills the additional decomposition property, the category  $\mathcal{C}$  is also PVK square adhesive w.r.t.  $\mathcal{M}$ .*

It is easily verified that in the case of attributed graphs, the additional decomposition property is fulfilled by the class  $\mathcal{M}$ ; this is proved in [18, Lemma 5.24]. Moreover, it is known that all pullbacks exist [53, Remark 8.15.]. We thus obtain the following corollary.

**Corollary 5.19** (Attributed graphs are PVK square adhesive). *The category  $\mathbf{AGraph}$ , and also the categories  $\mathbf{AGraph}_{ATG}$ ,  $\mathbf{ATrG}$ , and  $\mathbf{ATrG}_{ATG}$  from Definition 3.24 are all PVK square adhesive.*

In fact, the same argument can be made in the case one chooses symbolic attributed graphs [184] as basis for attribution.

This means,  $\mathbf{Graph}$  or  $\mathbf{AGraph}$  are suitable choices to consider  $S$ -cartesian functor categories over them. In the next theorem, we also show that the additional HLR properties are preserved. When we construct partial triple graphs as such an  $S$ -cartesian functor category in the next section, we thus immediately obtain the rich theory of double-pushout rewriting for them.

**Theorem 5.20** (Additional HLR properties of  $S$ -cartesian functor categories over graphs). *Let  $\mathcal{X}$  be any small category and  $S$  an arbitrary set of its morphisms. Then the  $S$ -cartesian functor category  $[\mathcal{X}_S, \mathbf{Graph}_{\mathcal{M}}]_{cart}$  is adhesive and the  $S$ -cartesian functor category  $[\mathcal{X}_S, \mathbf{AGraph}_{\mathcal{M}}]_{cart}$  is PVK square adhesive and adhesive HLR w.r.t.  $\mathcal{M}^{cart}$ . Both categories fulfill the additional HLR properties. Moreover,  $[\mathcal{X}_S, \mathbf{Graph}_{\mathcal{M}}]_{cart}$  has an  $\mathcal{M}$ -initial object and  $[\mathcal{X}_S, \mathbf{AGraph}_{\mathcal{M}}]_{cart}$  has an initial object which is not  $\mathcal{M}$ -initial, in general.*

In the proof of the above theorem, we make use of the  $\mathcal{E}$ - $\mathcal{M}'$  factorization system of attributed graph morphisms where  $\mathcal{E}$  consists of the morphisms

that are componentwise surjective and  $\mathcal{M}'$  of the morphisms that are componentwise injective. This class  $\mathcal{M}'$  is closed under pullbacks and pushouts along  $\mathcal{M}$ -morphisms and  $\mathcal{M}$ - $\mathcal{M}'$  PO-PB decomposition holds [53, Lemma 11.16]. By Proposition 5.14 these properties are preserved for the componentwise defined classes in  $[\mathcal{X}_S, \mathbf{AGraph}_{\mathcal{M}}]_{\text{cart}}$ . Therefore, the following corollary holds.

**Corollary 5.21** (Rewriting theory for  $S$ -cartesian functor categories over graphs). *Given a small category  $\mathcal{X}$  and a subset  $S$  of its morphisms, the classical theorems obtained for double-pushout rewriting, as there are [57, 58, 54]*

- *the Local Church–Rosser Theorem,*
- *the Parallelism Theorem,*
- *the Concurrency Theorem,*
- *the Amalgamation Theorem,*
- *the Embedding Theorem,*
- *the Extension Theorem,*
- *completeness of critical pairs, and*
- *the Local Confluence Theorem*

*are valid in the  $S$ -cartesian functor categories  $[\mathcal{X}_S, \mathbf{AGraph}_{\mathcal{M}}]_{\text{cart}}$  and  $[\mathcal{X}_S, \mathbf{Graph}_{\mathcal{M}}]_{\text{cart}}$ .*

### 5.3 The Category of Typed Attributed Partial Triple Graphs

In this section, we use the general theory developed in the previous section to introduce the category of (typed) attributed partial triple graphs. We discuss double- and also sesqui-pushout rewriting in these categories. Central results are that these categories are adhesive HLR categories that encompass the classic category of (typed) attributed triple graphs (Propositions 5.22 and 5.23) and the extension of the operationalization of TGG

rules to rules that also delete (Theorem 5.27). Most proofs are given in Appendix B.2.

An attributed triple graph is defined as a functor from the shape  $\bullet \leftarrow \bullet \rightarrow \bullet$  to the category **AGraph** (Definition 3.24), i.e., an attributed triple graph is a span of attributed graphs. For our desired applications, these morphisms should be allowed to be partial. Based on the categorical notion of a partial morphism (Definition 3.14), we are hence interested in functors from the shape  $\bullet \leftarrow \bullet \rightarrow \bullet \leftarrow \bullet \rightarrow \bullet$  to the category **AGraph** where the two central morphisms are mapped to  $\mathcal{M}$ -morphisms. For brevity, we fix the following notation for this section and the corresponding proofs. In this section,  $\mathcal{A}$  always denotes the shape  $\bullet \leftarrow \bullet \rightarrow \bullet \leftarrow \bullet \rightarrow \bullet$ . We abbreviate the choice of the two central morphisms as  $S$ , i.e.,  $\bullet \leftarrow \bullet \leftrightarrow \bullet \leftrightarrow \bullet \rightarrow \bullet$  in our former notation, to  $\bullet \leftarrow - \bullet \rightarrow - \bullet$ .

**Definition 5.4** ((Attributed) partial triple graph).

- The category of *partial triple graphs* **PTrG** is defined as the  $S$ -cartesian functor category

$$[\bullet \leftarrow - \bullet \rightarrow - \bullet, \mathbf{Graph}_{\mathcal{M}}]_{\text{cart}} .$$

- The category of *attributed partial triple graphs* **APTrG** is defined as the  $S$ -cartesian functor category

$$[\bullet \leftarrow - \bullet \rightarrow - \bullet, \mathbf{AGraph}_{\mathcal{M}}]_{\text{cart}} .$$

*Remark 5.3.* By the definitions above, an object

$$G = (G_S \xleftarrow{\sigma_G} G_{\tilde{S}} \xrightarrow{\iota_{G_S}} G_C \xleftarrow{\iota_{G_T}} G_{\tilde{T}} \xrightarrow{\tau_G} G_T)$$

of **PTrG** or **APTrG** might equivalently be considered to consist of an (attributed) graph  $G_C$  with partial morphisms  $\sigma_G : G_C \dashrightarrow G_S$  and  $\tau_G : G_C \dashrightarrow G_T$  where  $G_{\tilde{S}}$  and  $G_{\tilde{T}}$  are the domains of  $\sigma_G$  and  $\tau_G$ , respectively. A morphism  $f : G \rightarrow H$  between (attributed) partial triple graphs then is a triple  $(f_S : G_S \rightarrow H_S, f_C : G_C \rightarrow H_C, f_T : G_T \rightarrow H_T)$  of (attributed) graph morphisms such that both induced squares of partial morphisms commute. In our context, such a square with two opposed partial morphisms (as depicted as square (1) in Fig. 5.12) *commutes* if there exists a morphism  $f_{\tilde{S}} :$

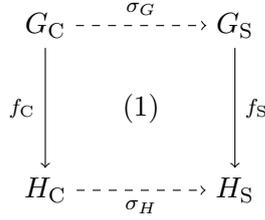


Figure 5.12: Square of partial morphisms

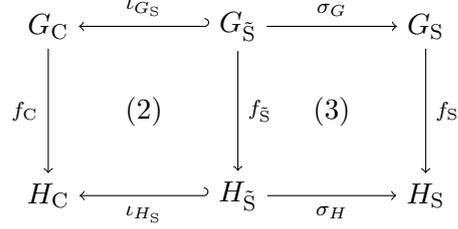


Figure 5.13: Commuting square of partial morphisms

$G_{\tilde{S}} \rightarrow H_{\tilde{S}}$  such that both arising squares (2) and (3) in Fig. 5.13 commute and, moreover, (2) is a pullback square. If  $f_{\tilde{S}}$  exists, it is necessarily unique since  $\iota_{H_S}$  is a monomorphism. This notion of a morphism between partial morphisms is stricter than, e.g., the *weak commutativity* used in [172] that does not require the square (2) to be a pullback square.

**Example 5.4.** The partial triple graph that is depicted in Fig. 2.6 is understood as an  $S$ - $\mathcal{M}$ -restricted functor from the category  $\bullet \leftarrow \bullet \dashrightarrow \bullet$  to the category of (attributed) graphs in the following way: The left object is mapped to its source graph, depicted to the left in the figure, the right object to the target graph depicted to the right, and the central object to the correspondence graph consisting of the four hexagons. The second and the fourth object are mapped to the respective domains of the correspondence morphisms to source and target graph. While the domain of the correspondence morphism to the target graph is the whole correspondence graph, the domain of the correspondence morphism to the source graph just consists of three of the hexagons. The outer morphisms of  $\mathcal{A}$  are mapped to the correspondence morphisms while the central morphisms are mapped to the inclusion of the domains of the correspondence morphisms into the correspondence graph, which are both injective.

By Corollary 5.19 and Theorem 5.20, (attributed) partial triple graphs are adhesive (HLR) and satisfy all of the additional HLR properties such that we obtain a rich theory of rewriting just by construction. The next proposition states that the category of (attributed) triple graphs is isomorphic to a full subcategory of the category of (attributed) partial triple graphs and that rule application is preserved and reflected by that inclusion. This

means that double-pushout rewriting of (attributed) partial triple graphs encompasses the known rewriting of (attributed) triple graphs.

**Proposition 5.22** (Inclusion of triple graphs into partial triple graphs. Preservation and reflection of rule applications). *The category  $\mathbf{TrG}$  ( $\mathbf{ATrG}$ ) of (attributed) triple graphs is isomorphic to a full subcategory of the category  $\mathbf{PTrG}$  ( $\mathbf{APTrG}$ ) of (attributed) partial triple graphs, i.e., there exists a full and faithful functor  $J : \mathbf{TrG} \rightarrow \mathbf{PTrG}$  ( $J : \mathbf{ATrG} \rightarrow \mathbf{APTrG}$ ) that is injective on objects.*

*Furthermore,  $J$  preserves and reflects rule applications. That is, given a rule  $p = (L \leftarrow K \hookrightarrow R)$  and a match  $m : G \rightarrow L$  in  $\mathbf{TrG}$  ( $\mathbf{ATrG}$ ), then  $p$  is applicable at  $m$  if and only if  $J(p)$  is applicable at  $J(m)$  in  $\mathbf{PTrG}$  ( $\mathbf{APTrG}$ ), where  $J(p)$  denotes the application of  $J$  to every component of the rule  $p$ . Moreover, the results coincide up to isomorphism, i.e., we have*

$$G \Rightarrow_{p,m} H \text{ implies } J(G) \Rightarrow_{J(p),J(m)} J(H)$$

and

$$J(G) \Rightarrow_{J(p),J(m)} H' \text{ implies } \exists H \text{ s.t. } G \Rightarrow_{p,m} H \text{ and } J(H) \cong H' .$$

*Proof.* The proofs for the attributed and non-attributed case are virtually the same. Let  $G = (G_S \xleftarrow{\sigma_G} G_C \xrightarrow{\tau_G} G_T)$  denote a triple graph and  $f = (f_S, f_C, f_T)$  a morphism between triple graphs. On objects, i.e., on triple graphs, define

$$J(G) := (G_S \xleftarrow{\sigma_G} G_C \xrightarrow{id_{G_C}} G_C \xleftarrow{id_{G_C}} G_C \xrightarrow{\tau_G} G_T)$$

and on morphisms define

$$J(f) := (f_S, f_C, f_C, f_C, f_T) .$$

First,  $id_{G_C} \in \mathcal{M}$ , and, as depicted in Fig. 5.14,  $f_C$  makes squares (1) and (2) commute (since  $(f_C, f_T)$  is a morphism between  $\sigma_G$  and  $\sigma_H$ ) and (1) into a pullback square (by Fact 3.9 (4)); the situation on the target part is analogous. Together this means that  $J$  actually maps to  $\mathbf{PTrG}$  ( $\mathbf{APTrG}$ ). That  $J$  indeed defines a functor that is faithful and injective on objects is straightforward to check. To see that  $J$  is also full, assume a morphism  $f = (f_S, f_{\tilde{S}}, f_C, f_{\tilde{T}}, f_T)$  between two partial triple graphs  $J(G)$  and  $J(H)$

$$\begin{array}{ccccc}
G_C & \xleftarrow{id_{G_C}} & G_C & \xrightarrow{\sigma_G} & G_S \\
\downarrow f_C & & \downarrow f_C & & \downarrow f_T \\
(1) & & (2) & & \\
H_C & \xleftarrow{id_{H_C}} & H_C & \xrightarrow{\sigma_H} & H_S
\end{array}$$

Figure 5.14: Mapping a morphism between triple graphs to one between partial triple graphs

in the image of  $J$  to be given. Since the two central squares of the four squares that are induced by the morphism are pullbacks by assumption (compare square (1) in Fig. 5.14 again), necessarily  $f_{\tilde{S}} = f_C = f_{\tilde{T}}$  for all such morphisms. Hence,  $J$  is full.

That rule application is preserved is a consequence of Proposition 5.6 and the way the inclusion functor  $J$  is defined: Since pushouts along morphisms where every component stems from  $\mathcal{M}$  are computed componentwise in both categories  $\mathbf{TrG}$  ( $\mathbf{ATrG}$ ) and  $\mathbf{PTrG}$  ( $\mathbf{APTrG}$ ), applying  $J$  to two pushouts that constitute a rule application (compare Fig. 3.6) in  $\mathbf{TrG}$  ( $\mathbf{ATrG}$ ), the resulting diagram consists of two pushouts in  $\mathbf{PTrG}$  ( $\mathbf{APTrG}$ ) and hence defines an application of  $J(p)$  at  $J(m)$ .

For the converse statement, i.e., that rule application is also reflected, assume an application  $J(G) \Rightarrow_{J(p), J(m)} H'$  be given. Figure 5.15 shows the part of that transformation that is central for the proof: It shows the computation of the pushout complement  $D'$ , at least on the source and correspondence part. Since the right bottom square needs to be a pullback square, Fact 3.9 (4) implies that  $\iota_{D_S}$  is an isomorphism. In particular,  $D = (D_S \xleftarrow{\sigma_D} D_C \xrightarrow{\tau_D} D_T)$  (for suitably chosen  $\sigma_D$ , depending on  $\iota_{D_S}$ ) is a pushout complement for  $m \circ l$  in  $\mathbf{TrG}$  ( $\mathbf{ATrG}$ ). In particular,  $p$  is applicable in  $\mathbf{TrG}$  ( $\mathbf{ATrG}$ ) with match  $m$  and  $J(D) \cong D'$ . That also  $J(H) \cong H'$  for the result  $H$  of that transformation follows similarly.  $\square$

In practical applications, the considered triple graphs are generally typed over a fixed triple graph. The next definition introduces typing of partial triple graphs over a fixed *triple graph*. This differs from the common definition of typing using a slice category construction (compare

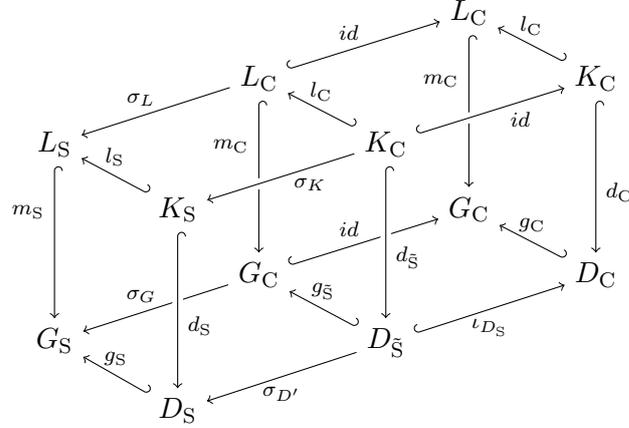


Figure 5.15: Relevant snippet of the rule application  $J(G) \Rightarrow_{J(p),J(m)} H'$

**Definition 3.24.** The reason for this is that, for scenarios such as model synchronization in the next chapter, it is convenient if the partial triple graphs are still typed over essentially the same triple graph as the original triple graph was.

**Definition 5.5** (Typed (attributed) partial triple graph). The category of *partial triple graphs typed over a fixed triple graph*  $TG$ , denoted  $\mathbf{PTrG}_{TG}$ , has morphisms

$$t_G : I(G) \rightarrow I(J(TG))$$

from  $[\mathcal{A}, \mathbf{Graph}]$  as objects. Here,  $J : [\bullet \leftarrow \bullet \rightarrow \bullet, \mathbf{Graph}] \rightarrow [\bullet \leftarrow \bullet \dashrightarrow \bullet, \mathbf{Graph}]$  is the inclusion functor from triple graphs to partial triple graphs, and  $I : [\bullet \leftarrow \bullet \dashrightarrow \bullet, \mathbf{Graph}] \rightarrow [\mathcal{A}, \mathbf{Graph}]$  is the inclusion functor from partial triple graphs to the ambient functor category. *Typed morphisms* are morphisms  $g : G \rightarrow H$  from  $\mathbf{PTrG}$  such that  $t_H \circ I(g) = t_G$ .

*Attributed partial triple graphs typed over a fixed attributed triple graph*  $ATG$  which is equipped with the final  $\Sigma$ -algebra, denoted  $\mathbf{APTrG}_{ATG}$ , are defined completely analogously, replacing graphs by attributed graphs.

**Example 5.5.** Figure 2.6 was already presented as a partial triple graph in Example 5.4. To consider it as still being typed over the same type

graph (Fig. 2.1) as the original triple graph from Fig. 2.3, we just restrict its typing morphism accordingly. Note that the resulting typing morphism is not  $S$ -cartesian but a morphism in  $[\mathcal{A}, \mathbf{Graph}]$ . Therefore, we did not define typed partial triple graphs as a slice category.

We exemplify rules and morphisms in  $\mathbf{APTrG}_{\text{ATG}}$  using the rule *CollapseHierarchy*, which is depicted in Fig. 2.9. Recall that its LHS  $L$  consists of the elements depicted in black or in red (not annotated or annotated with  $(--)$ ) while its interface  $K$  only consists of the black elements (not annotated). The mapping from  $K$  to  $L$  is injective in every component. In addition, all three correspondence nodes of  $L$  are in the domain of the according correspondence morphism to the source side and the two correspondence nodes that are already part of  $K$  are in the domain of the according correspondence morphism as well. This means that the induced square of morphisms is a pullback square. The whole situation is exemplary depicted in Fig. 5.16 where the integrated representation of Fig. 2.9 is replaced by a detailed one. The mapping of the correspondence nodes is indicated by their names. The domain of the source correspondence morphism is denoted by  $L_{\bar{\xi}}$  for the LHS  $L$  and by  $K_{\bar{\xi}}$  for the interface  $K$ .  $L_C$  and  $K_C$  denote the respective correspondence graphs. Analogous diagrams could be drawn for the target side and the RHS of the rule. Hence, the morphism from the interface  $K$  of *CollapseHierarchy* to its LHS is a morphism in  $\mathbf{APTrG}_{\text{ATG}}$ .

In contrast, the morphism from the interface  $K$  to the LHS of *CollapseHierarchySrc* as depicted in Fig. 2.11 is not a morphism in  $\mathbf{APTrG}_{\text{ATG}}$ : The according square  $K_{\bar{\xi}} \hookrightarrow K_C \hookrightarrow L_C \hookleftarrow L_{\bar{\xi}} \hookleftarrow K_{\bar{\xi}}$  is not a pullback square since the domain of the correspondence morphism to the source side in the interface graph only contains two elements and not three (compare Fig. 5.17). Theorem 5.27 explains why this is not a hindrance to our desired application.

The next result states that our obtained results still hold for the typed case.

**Proposition 5.23** (Adhesiveness of typed partial triple graphs). *Results analogous to Proposition 5.22 hold in the categories introduced in Definition 5.5:*

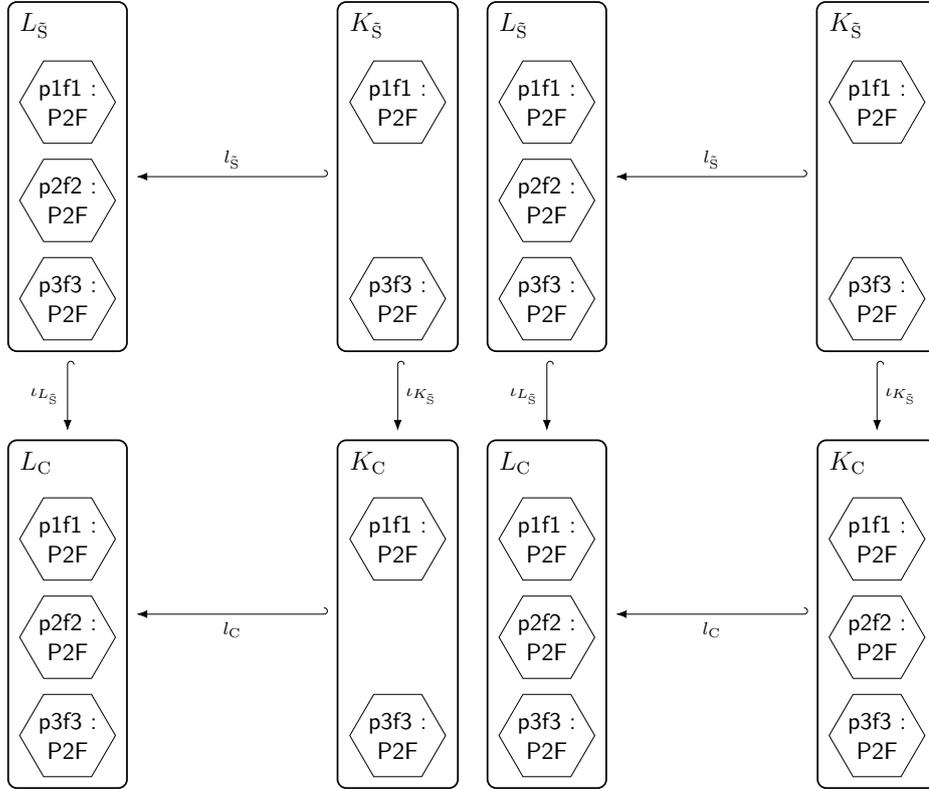


Figure 5.16: Showing the morphism  $l : K \hookrightarrow L$  from *CollapseHierarchy* to be  $S$ -cartesian

Figure 5.17: The morphism  $l : K \hookrightarrow L$  from *CollapseHierarchySrc* is not  $S$ -cartesian

- Given a triple graph  $TG$ , the category  $\mathbf{PTrG}_{TG}$  of partial triple graphs typed over  $TG$  is an adhesive category that satisfies the additional HLR properties.

Moreover,  $\mathbf{TrG}_{TG}$  is isomorphic to a full subcategory of the category  $\mathbf{PTrG}_{TG}$  such that rule applications are preserved and reflected (in the sense of Proposition 5.22).

- Given an attributed triple graph  $ATG$  equipped with the final  $\Sigma$ -algebra, the category  $\mathbf{APTrG}_{ATG}$  of attributed partial triple graphs typed over  $ATG$  is an adhesive HLR category (w.r.t.  $\mathcal{M}^{\text{cart}}$ ) that satisfies the

additional HLR properties. In this case, however, the existing initial object is not  $\mathcal{M}^{\text{cart}}$ -initial.

Moreover,  $\mathbf{ATrG}_{\text{ATG}}$  is isomorphic to a full subcategory of the category  $\mathbf{APTrG}_{\text{ATG}}$  such that rule applications are preserved and reflected (in the sense of Proposition 5.22).

In the proof of the above proposition, we show that pullbacks and pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms in  $\mathbf{PTrG}_{\text{TG}}$  ( $\mathbf{APTrG}_{\text{ATG}}$ ) are computed componentwise exactly as in  $\mathbf{PTrG}$  ( $\mathbf{APTrG}$ ). The same holds true for all other relevant computations. The necessary typing morphism is always obtained via the universal property (in case of colimits) or by composition. In the following, to significantly reduce the notational effort, we prove all our results in the untyped setting, even though we state them for the typed categories. This is justified by the (proof of the) above proposition: The exact same computations can be performed in the typed case.

The restriction to morphisms where certain squares are required to form pullback squares comes with some limitations. The next proposition clarifies these restrictions in the case of (typed attributed) partial triple graphs. We state which kinds of operations cannot be performed by rule application. Concretely, it is not possible to delete a reference (i.e., an element from the domain of a correspondence morphism) without deleting the referencing element (i.e., the according element in the correspondence graph), nor is it possible to create a reference from an already existing correspondence element. And for every matched correspondence element, a morphism also needs to match the according preimages in the domains of the two correspondence morphisms (if they exist) to become a valid match. In the statement of the proposition, we use shorthands like  $x \in L_C$  to say that  $x$  is any element (any kind of node or edge) of the (typed attributed) graph  $L_C$ . Consequently, we apply graph morphisms to such elements without explicitly stating the component of the morphism.

**Proposition 5.24** (Characterizing valid rules and matches). *Let a rule  $p = (L \xleftarrow{l} K \xrightarrow{r} R)$  and a morphism  $m : L \rightarrow G$  in  $[\mathcal{A}, \mathbf{Graph}]/TG$  (or  $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ) be given where  $L, K, R,$  and  $G$  are already objects from  $\mathbf{PTrG}_{\text{TG}}$  ( $\mathbf{APTrG}_{\text{ATG}}$ ), for instance,*

$$L = (L_S \xleftarrow{\sigma_L} L_{\tilde{S}} \xrightarrow{\iota_{L_S}} L_C \xleftarrow{\iota_{L_T}} L_{\tilde{T}} \xrightarrow{\tau_L} L_T)$$

and similarly for  $K, R, G$ . Then  $p$  is a rule with match already in  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ) if and only if (compare Fig. 5.18 for notation):

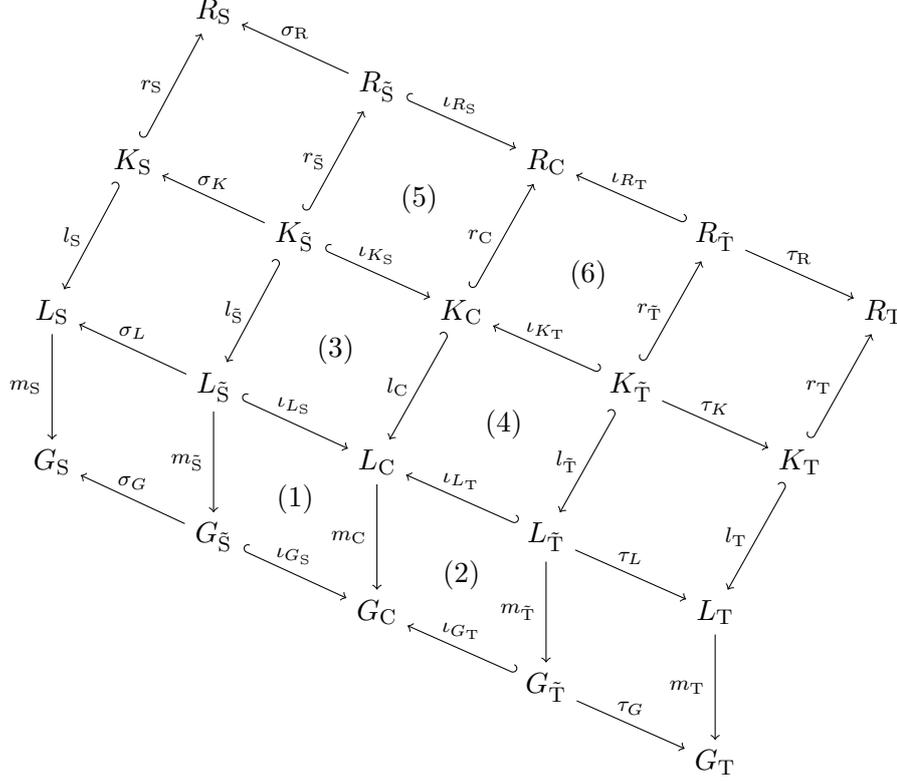


Figure 5.18: Characterizing rule and match from  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ )

**No isolated deletion.** If an element from  $L_C$  has a preimage in  $L_{\bar{S}}$ , i.e., belongs to the domain of the partial morphism to  $L_S$ , this element cannot be deleted from the domain of the partial morphism (i.e., not occur in  $K_{\bar{S}}$ ) without also being deleted from the correspondence graph (i.e., not occur in  $K_C$ ). The analogous statement holds for the target side. In formulas:

$$\forall x \in L_C \left( (x \in \iota_{L_S}(L_{\bar{S}}) \wedge x \in l_C(K_C)) \implies x \in l_C(\iota_{K_S}(K_{\bar{S}})) \right)$$

and analogously

$$\forall x \in L_C \left( (x \in \iota_{L_T}(L_{\bar{T}}) \wedge x \in l_C(K_C)) \implies x \in l_C(\iota_{K_T}(K_{\bar{T}})) \right) .$$

**No re-use in creation.** *If an element from  $R_C$  has a preimage in  $R_{\bar{S}}$ , i.e., belongs to the domain of the partial morphism to  $R_S$ , this element cannot be newly created in the domain of the partial morphism (i.e., not occur in  $K_{\bar{S}}$ ) without also being newly created in the correspondence graph (i.e., not occur in  $K_C$ ). The analogous statement holds for the target side. In formulas:*

$$\forall x \in R_C \left( (x \in \iota_{R_S}(R_{\bar{S}}) \wedge x \in r_C(K_C)) \implies x \in r_C(\iota_{K_S}(K_{\bar{S}})) \right)$$

and analogously

$$\forall x \in R_C \left( (x \in \iota_{R_T}(R_{\bar{T}}) \wedge x \in r_C(K_C)) \implies x \in r_C(\iota_{K_T}(K_{\bar{T}})) \right) .$$

**No isolated matching.** *Whenever a correspondence element from  $G_C$  is matched that belongs to the domain of the source correspondence morphism (i.e., occurs in  $G_{\bar{S}}$ ), for each preimage of the correspondence element the corresponding element in  $G_{\bar{S}}$  is also matched, i.e., has a preimage in  $L_{\bar{S}}$ . The analogous statement holds for the target side. In formulas:*

$$\forall x \in G_C \left( (\exists y_1 \in G_{\bar{S}}(x = \iota_{G_S}(y_1)) \wedge \exists y_2 \in L_C(x = m_C(y_2))) \implies \right. \\ \left. \exists z \in L_{\bar{S}}(m_{\bar{S}}(z) = y_1 \wedge \iota_{L_S}(z) = y_2) \right)$$

and analogously

$$\forall x \in G_C \left( (\exists y_1 \in G_{\bar{T}}(x = \iota_{G_T}(y_1)) \wedge \exists y_2 \in L_C(x = m_C(y_2))) \implies \right. \\ \left. \exists z \in L_{\bar{T}}(m_{\bar{T}}(z) = y_1 \wedge \iota_{L_T}(z) = y_2) \right) .$$

These results indicate that double-pushout rewriting has restricted expressiveness in  $S$ -cartesian functor categories. However, Fact 5.1, combined with Fact 3.2 (1), shows that  $\mathcal{M}^{\text{cart}}$  is the maximal choice possible for

an admissible class of monomorphisms with respect to which some kind of adhesivity can be obtained in categories consisting of  $S$ - $\mathcal{M}$ -restricted functors (at least, if this choice shall be based on  $\mathcal{M}$ ).<sup>4</sup> In the following, we will mitigate the restrictions by using the fact that certain rewritings of (typed attributed) partial triple graphs by more general rules result in (typed attributed) partial triple graphs, nevertheless.

The starting point for many practical applications of TGGs is the so-called *operationalization* of a rule which is a split of it into a *source* and a *forward rule* (or equivalently: a *target* and a *backward rule*), which we recalled in Definition 3.27. The source rule only performs the action of the original rule on the source part and the forward rule transfers this to the correspondence and the target part. As already recalled in Sect. 3.4, a basic result states that applying a rule to a triple graph is equivalent to applying the source rule followed by an application of the forward rule (if consistently matched). We give a comparable definition and result for (typed attributed) partial triple graphs. However, we generalize the operationalization of rules in two directions: Our rules are rules on partial triple graphs instead of triple graphs and, moreover, they are allowed to be deleting, whereas classically the rules of a TGG are monotonic [195]. To be able to do so, we need to deviate slightly from the original construction. Our source rules perform the action of the original rule on the source side. Moreover, the deletion-action on the domain of the source-correspondence morphism is performed. All other actions are performed by the forward rule. In general, the resulting source rules are not rules in  $\mathbf{PTrG}_{\mathbf{TG}}$  ( $\mathbf{APTrG}_{\mathbf{ATG}}$ ) any longer but in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ). However, the following theorem shows that the application of a rule in  $\mathbf{PTrG}_{\mathbf{TG}}$  ( $\mathbf{APTrG}_{\mathbf{ATG}}$ ) is equivalent to applying first the source and afterwards the forward rule (at a suitable match) in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ).

---

<sup>4</sup>In principle, it is possible to restate our results for full subcategories of functor categories that are induced by  $S$ - $\mathcal{M}$ -restricted functors. However, the above considerations imply that some variant of adhesivity can only be obtained with respect to the class of  $\mathcal{M}^{\text{cart}}$ -morphisms where  $S$ -components are pullbacks and not with respect to componentwise  $\mathcal{M}$ -morphisms. In particular, the expressiveness of rewriting is very similar and coincides if restricted to matches from  $\mathcal{M}^{\text{cart}}$ . This is regularly the case in practical applications. We choose to work with  $S$ -cartesian functor categories because we obtain a slightly more elegant result: adhesiveness in case the base category is (compared to quasiadhesiveness when working with full subcategories generated by  $S$ - $\mathcal{M}$ -restricted functors).

$$\begin{array}{c}
L^S = \\
\uparrow l^S \\
K^S = \\
\downarrow r^S \\
R^S =
\end{array}
\begin{array}{ccccccccc}
(L_S & \xleftarrow{\sigma_L} & L_{\bar{S}} & \xleftarrow{\iota_{L_S}} & L_C & \xleftarrow{\quad} & \emptyset & \xrightarrow{\quad} & \emptyset) \\
\uparrow l_S & & \uparrow l_{\bar{S}} & & \uparrow id & & \uparrow & & \uparrow \\
(K_S & \xleftarrow{\sigma_K} & K_{\bar{S}} & \xleftarrow{\iota_C \circ \iota_{K_S}} & L_C & \xleftarrow{\quad} & \emptyset & \xrightarrow{\quad} & \emptyset) \\
\uparrow r_S & & \uparrow id & & \uparrow id & & \uparrow & & \uparrow \\
(R_S & \xleftarrow{\tau_R \circ r_{\bar{S}}} & K_{\bar{S}} & \xleftarrow{\iota_C \circ \iota_{K_S}} & L_C & \xleftarrow{\quad} & \emptyset & \xrightarrow{\quad} & \emptyset)
\end{array}$$

Figure 5.19: Source rule  $p^S$  of a rule  $p$ 

$$\begin{array}{c}
L^F = \\
\uparrow l^F \\
K^F = \\
\downarrow r^F \\
R^F =
\end{array}
\begin{array}{ccccccccc}
(R_S & \xleftarrow{\tau_R \circ r_{\bar{S}}} & K_{\bar{S}} & \xleftarrow{\iota_C \circ \iota_{K_S}} & L_C & \xleftarrow{\iota_{L_T}} & L_{\bar{T}} & \xrightarrow{\tau_L} & L_T) \\
\uparrow id & & \uparrow id & & \uparrow l_C & & \uparrow l_{\bar{T}} & & \uparrow l_T \\
(R_S & \xleftarrow{\tau_R \circ r_{\bar{S}}} & K_{\bar{S}} & \xleftarrow{\iota_{K_S}} & K_C & \xleftarrow{\iota_{K_T}} & K_{\bar{T}} & \xrightarrow{\tau_K} & K_T) \\
\uparrow id & & \uparrow r_{\bar{S}} & & \uparrow r_C & & \uparrow r_{\bar{T}} & & \uparrow r_T \\
(R_S & \xleftarrow{\sigma_R} & R_{\bar{S}} & \xleftarrow{\iota_{R_S}} & R_C & \xleftarrow{\iota_{R_T}} & R_{\bar{T}} & \xrightarrow{\tau_R} & R_T)
\end{array}$$

Figure 5.20: Forward rule  $p^F$  of a rule  $p$ 

**Definition 5.6** (Source and forward rule). Let a rule  $p = (L \xleftarrow{l} K \xrightarrow{r} R)$  in  $\mathbf{APTTrG}_{\text{ATG}}$  be given. Then its *source rule*  $p^S = (L^S \xleftarrow{l^S} K^S \xrightarrow{r^S} R^S)$  is defined as depicted in Fig. 5.19 where  $\emptyset$  denotes the empty graph (or the empty graph equipped with the original rules' algebra at that point as data part), respectively. Its *forward rule*  $p^F = (L^F \xleftarrow{l^F} K^F \xrightarrow{r^F} R^F)$  is defined as depicted in Fig. 5.20.

We first note that at least the such constructed forward rules are indeed rules in  $\mathbf{APTTrG}_{\text{ATG}}$ ; in Example 5.5 we already saw that this is not generally true for the source rules.

**Lemma 5.25** (Forward rules in  $\mathbf{APTrG}_{ATG}$ ). *Let a rule  $p = (L \xleftarrow{l} K \xrightarrow{r} R)$  in  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ) be given and let  $p^F = (L^F \xleftarrow{l^F} K^F \xrightarrow{r^F} R^F)$  be its forward rule. Then  $l^F$  and  $r^F$  are  $\mathcal{M}$ -morphisms; in particular,  $r^F$  is a rule in  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ).*

Furthermore, for technical reasons we will later need that there is partial morphism from the LHS of a rule to the LHS of its forward rule; partiality emerges because the source elements that the rule deletes are not part any longer of the LHS of its forward rule.

**Lemma 5.26** (Partial morphism from  $L$  to  $L^F$ ). *Let a rule  $p = (L \xleftarrow{l} K \xrightarrow{r} R)$  in  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ) be given and let  $p^F = (L^F \xleftarrow{l^F} K^F \xrightarrow{r^F} R^F)$  be its forward rule. Then there is a partial morphism  $L \leftrightarrow A \hookrightarrow L^F$  in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ).*

The next theorem states the equivalence of applying a triple rule or applying its source rule first, followed by a correctly matched application of its forward rule. Obviously, analogous results hold for the other direction (comprising a *target* and a *backward rule*). In the statement, we again suppress the inclusion functor  $I$  into the functor category.

**Theorem 5.27** (Forward operationalization). *Let a rule  $p$  in  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ) with source and forward rules  $p^S$  and  $p^F$  be given.*

- (1) *Given a direct transformation  $G \Rightarrow_{p,m} H$  in  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ), there is a transformation sequence*

$$G \Rightarrow_{p^S, m^S} G' \Rightarrow_{p^F, m^F} H$$

*in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ) such that the matches satisfy the following set of equations*

$$\begin{array}{lll} m_S^S = m_S & m_S^F = n_S^S & m_T^F = m_{\bar{T}} \\ m_{\bar{S}}^S = m_{\bar{S}} & m_{\bar{S}}^F = n_{\bar{S}}^S & m_T^F = m_T \\ m_C^S = m_C & m_C^F = n_C^F & \end{array}$$

*where  $n^S$  is the comatch of the transformation via the source rule.*

(2) Given transformation steps

$$G \Rightarrow_{p^S, m^S} G' \Rightarrow_{p^F, m^F} H$$

in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ) where  $G$  and  $m$  are already elements of  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ) and the central column of the above equation holds, there is a direct transformation  $G \Rightarrow_{p, m} H$  in  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ), where  $m$  is defined via the outer columns of the above equations.

*Proof.* Let  $p, p^S$ , and  $p^F$  be given. The rules  $p^S$  and  $p^F$  are rules in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ). The first category is adhesive, the second adhesive HLR, and both have the necessary factorizations for morphisms. Hence, if we show that  $p$  is a concurrent rule for  $p^S$  and  $p^F$ , i.e.,  $p = p^S *_E p^F$  for a suitable dependency relation  $E$ , the Concurrency Theorem ([53, Thm. 5.23] or Theorem 3.8) is applicable and provides the desired result in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ), including the statement about the equations for the matches.

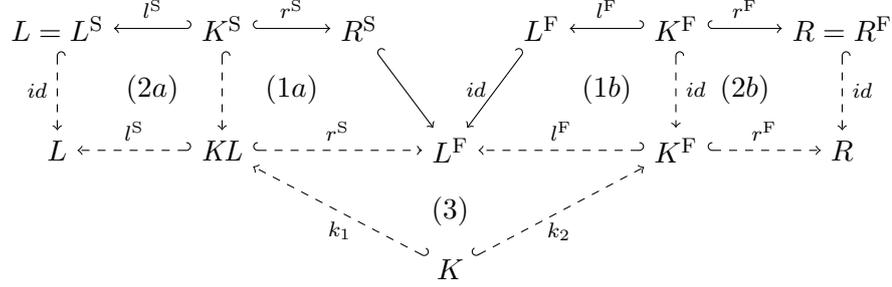
To see this, first observe that  $R^S$  and  $L^F$  map jointly epimorphic to  $L^F$ . Hence,  $R^S = L^F$  with the obvious morphisms is a natural choice for a dependency object. Moreover, the pushout complements for  $K^S \xrightarrow{r^S} R^S \hookrightarrow L^F$  and  $K^F \xrightarrow{l^F} L^F \xrightarrow{id} L^F$  exist and are given by  $KL$  and  $K^F$  again where  $KL := (K_S \leftarrow K_{\bar{S}} \hookrightarrow L_C \leftrightarrow L_{\bar{T}} \rightarrow L_T)$  (see (1a) and (1b) in Fig. 5.21). Computing the pushouts (2a) and (2b) results in  $L$  and  $R$ , respectively. One easily checks that computing the pullback (3) results in  $K$  and that  $l^S \circ k_1 = l$  and  $r^F \circ k_2 = r$ . In summary,  $p = p^S *_L p^F$ .

As already stated, applying the Concurrency Theorem gives the desired results at least in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ). Proposition 5.22 allows us to lift the results to  $\mathbf{PTrG}$  ( $\mathbf{APTrG}$ ):

- (1) Given a direct transformation  $G \Rightarrow_{p, m} H$  in  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ), by Proposition 5.22 this is a transformation in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ). Then, the Concurrency Theorem guarantees the existence of a transformation sequence

$$G \Rightarrow_{p^S, m^S} G' \Rightarrow_{p^F, m^F} H$$

in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ).


 Figure 5.21: Original rule  $p$  as concurrent rule of  $p^S$  and  $p^F$ 

(2) Given transformation steps

$$G \Rightarrow_{p^S, m^S} G' \Rightarrow_{p^F, m^F} H$$

in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ), the Concurrency Theorem guarantees the existence of a direct transformation  $G \Rightarrow_{p, m} H$  in  $[\mathcal{A}, \mathbf{Graph}]/TG$  ( $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ). Since  $G$  and  $m$  are already elements of  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ), by Proposition 5.22 this is a transformation already in  $\mathbf{PTrG}_{TG}$  ( $\mathbf{APTrG}_{ATG}$ ).  $\square$

**Example 5.6.** *CollapseHierarchy*, as depicted in Fig. 2.9, is split by our construction into *CollapseHierarchyFwd* and *CollapseHierarchySrc* as depicted in Figs. 2.10 and 2.11. Applying *CollapseHierarchy* to the triple graph from Fig. 2.3 such that nodes  $p_2$  and  $f_2$  are matched to  $\text{subP}$  and  $\text{subF}$ , respectively, gives the same result as first applying *CollapseHierarchySrc* at the according match, which gives the partial triple graph from Fig. 2.6 as intermediate result, and *CollapseHierarchyFwd* subsequently (with consistent match). Namely, both yield the triple graph from Fig. 2.8; however, with the content-attributes of  $\text{doc}$  and  $\text{ssubD}$  unaltered.

In the remainder of this section, we revisit the sesqui-pushout approach to rewriting and calculate final pullback complements in the category  $\mathbf{APTrG}_{ATG}$  of typed attributed partial triple graphs. To the best of our knowledge, this has not yet been done. Unfortunately, as an typed attributed partial triple graph consists of five attributed graphs each of which comprises an  $E$ -graph and an algebra, the concrete presentation of

final pullback complements in  $\mathbf{APTTrG}_{\text{ATG}}$  results in an absolute notational mess (even when typing remains implicit). To mitigate this at least a little bit, we restrict ourselves to the only case we practically need: we present final pullback complements for morphisms  $m \circ l$ , where  $l \in \mathcal{M}$  and  $m$  is quasi-injective. To further simplify the situation, we assume all involved  $\mathcal{M}$ -morphisms (in particular,  $l$  and the domains of the partial morphisms) to actually be inclusions and all isomorphisms between algebras to even be identities. We first informally summarize the computation: When restricting to linear rules in the category of graphs, the distinctive feature of a final pullback complement (compared to a pushout complement) is that it allows for deletion in unknown context [46]. Adjacent edges of nodes to be deleted are (implicitly) deleted as well. In this way, rule applications in the sesqui-pushout approach are not subject to the dangling-edge condition. However, this also shows that final pullback complements cannot be computed componentwise. In the case of typed attributed partial triple graphs, two further kinds of implicit deletion emerge: First, whenever an element of the source or target graph of a partial triple graph is deleted, if existent, its pre-image under the correspondence morphism is (implicitly) deleted from the respective domain. For the second kind of implicit deletion recall that by definition, morphisms between partial triple graphs have to induce pullback squares at the morphisms that include the domains of the correspondence morphisms into the correspondence graphs. Thus, when we compute  $K \xrightarrow{d} D \xrightarrow{g} G$  as a final pullback complement for  $K \xrightarrow{l} L \xrightarrow{m} G$ , the morphism  $d$  has to satisfy this requirement. This necessitates (implicit) deletion of elements from the correspondence graph. For example in Fig. 5.22, when compared to  $G$  the node 1 is deleted from the domain of the source correspondence morphism in  $D$ , it also has to be deleted from  $D_C$ , i.e., the correspondence graph of  $D$ . Otherwise, the pullback property would be violated. In particular, the implicit deletion of elements from the domain of the correspondence morphisms triggers further implicit deletions of correspondence elements. A similar consideration, however, shows that there cannot occur implicit deletion that is caused by propagating explicit deletions from the correspondence graph to the domains of the correspondence morphisms. The match necessary for this to happen would not be a morphism in  $\mathbf{APTTrG}_{\text{ATG}}$  because the required pullback property would be violated (compare also Fig. 5.18).

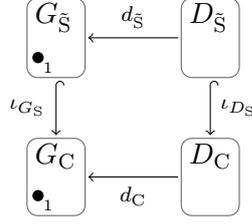


Figure 5.22: Example for the necessity of (implicit) deletions of correspondence elements

The next lemma clarifies the computation of final pullback complements in  $\mathbf{APTTrG}_{\text{ATG}}$  formally.

**Lemma 5.28** (FPCs in  $\mathbf{APTTrG}_{\text{ATG}}$ ). *Given morphisms  $K \xrightarrow{l} L \xrightarrow{m} G$  between typed attributed partial triple graphs, where  $l \in \mathcal{M}$  (treated as an inclusion) and  $m$  is quasi-injective, a final pullback complement  $K \xrightarrow{d} D \xrightarrow{g} G$  exists and is given by the following data.*

When

$$G = G_S \xleftarrow{\sigma_G} G_{\tilde{S}} \xrightarrow{\iota_{G_S}} G_C \xleftarrow{\iota_{G_T}} G_{\tilde{T}} \xrightarrow{\tau_G} G_T$$

with

$$G_X = \left( (V_G^X, A_G^X, E_G^X, NA_G^X, EA_G^X, (src_j^X, tar_j^X)_{j \in \{G, NA_G, EA_G\}}), A_{G^X} \right)$$

for  $X \in \{S, \tilde{S}, C, \tilde{T}, T\}$ , the partial triple graph  $D$  is the following subgraph of  $G$ : In every component, the algebra coincides with the one of  $G$ , i.e.,

$$A_{D^X} := A_{G^X} \text{ and } A_D^X := A_G^X$$

for all  $X \in \{S, \tilde{S}, C, \tilde{T}, T\}$ . For  $X \in \{S, T\}$

$$\begin{aligned} V_D^X &:= V_G^X \setminus m_{V_L}^X(V_L^X \setminus V_K^X) , \\ E_D^X &:= \{e \in E_G^X \mid src_G^X(e) \in V_D^X \wedge tar_G^X(e) \in V_D^X\} \setminus m_{E_L}^X(E_L^X \setminus E_K^X) , \\ NA_D^X &:= \{e \in NA_G^X \mid src_G^X(e) \in V_D^X\} \setminus m_{NA_L}^X(NA_L^X \setminus NA_K^X) , \\ EA_D^X &:= \{e \in EA_G^X \mid src_G^X(e) \in V_D^X\} \setminus m_{EA_L}^X(EA_L^X \setminus EA_K^X) . \end{aligned}$$

Moreover

$$V_D^{\tilde{S}} := \left\{ n \in V_G^{\tilde{S}} \mid \sigma_{V_G}(n) \in V_D^{\tilde{S}} \right\} \setminus \left( m_{V_L}^{\tilde{S}}(V_L^{\tilde{S}} \setminus V_K^{\tilde{S}}) \cup \left\{ n \in V_G^{\tilde{S}} \mid \exists n' \in V_G^{\tilde{T}}. \iota_{V_G^{\tilde{S}}}(n) = \iota_{V_G^{\tilde{T}}}(n') \wedge \tau_{V_G}(n') \notin V_D^{\tilde{T}} \right\} \right),$$

$$V_D^{\tilde{T}} := \left\{ n \in V_G^{\tilde{T}} \mid \tau_{V_G}(n) \in V_D^{\tilde{T}} \right\} \setminus \left( m_{V_L}^{\tilde{T}}(V_L^{\tilde{T}} \setminus V_K^{\tilde{T}}) \cup \left\{ n \in V_G^{\tilde{T}} \mid \exists n' \in V_G^{\tilde{S}}. \iota_{V_G^{\tilde{T}}}(n) = \iota_{V_G^{\tilde{S}}}(n') \wedge \tau_{V_G}(n') \notin V_D^{\tilde{S}} \right\} \right),$$

$$V_D^{\tilde{C}} := V_G^{\tilde{C}} \setminus \left( m_{V_L}^{\tilde{C}}(V_L^{\tilde{C}} \setminus V_K^{\tilde{C}}) \cup \left\{ n \in V_G^{\tilde{C}} \mid \exists n' \in V_G^{\tilde{S}}. \iota_{V_G^{\tilde{S}}}(n') = n \wedge \sigma_{V_G}(n') \notin V_D^{\tilde{S}} \right\} \cup \left\{ n \in V_G^{\tilde{C}} \mid \exists n' \in V_G^{\tilde{T}}. \iota_{V_G^{\tilde{T}}}(n') = n \wedge \tau_{V_G}(n') \notin V_D^{\tilde{T}} \right\} \right),$$

and analogously

$$E_D^{\tilde{S}} := \left\{ e \in E_G^{\tilde{S}} \mid \text{src}_G^{\tilde{S}}(e) \in V_D^{\tilde{S}} \wedge \text{tar}_G^{\tilde{S}}(e) \in V_D^{\tilde{S}} \wedge \sigma_{E_G}(e) \in E_D^{\tilde{S}} \right\} \setminus \left( m_{E_L}^{\tilde{S}}(E_L^{\tilde{S}} \setminus E_K^{\tilde{S}}) \cup \left\{ e \in E_G^{\tilde{S}} \mid \exists e' \in E_G^{\tilde{T}}. \iota_{E_G^{\tilde{S}}}(e) = \iota_{E_G^{\tilde{T}}}(e') \wedge \tau_{E_G}(e') \notin E_D^{\tilde{T}} \right\} \right),$$

$$E_D^{\tilde{C}} := \left\{ e \in E_G^{\tilde{C}} \mid \text{src}_G^{\tilde{C}}(e) \in V_D^{\tilde{C}} \wedge \text{tar}_G^{\tilde{C}}(e) \in V_D^{\tilde{C}} \right\} \setminus \left( m_{E_L}^{\tilde{C}}(E_L^{\tilde{C}} \setminus E_K^{\tilde{C}}) \cup \left\{ e \in E_G^{\tilde{C}} \mid \exists e' \in E_G^{\tilde{S}}. \iota_{E_G^{\tilde{S}}}(e') = e \wedge \sigma_{E_G}(e') \notin E_D^{\tilde{S}} \right\} \cup \left\{ e \in E_G^{\tilde{C}} \mid \exists e' \in E_G^{\tilde{T}}. \iota_{E_G^{\tilde{T}}}(e') = e \wedge \tau_{E_G}(e') \notin E_D^{\tilde{T}} \right\} \right),$$

$$NA_D^{\tilde{S}} := \left\{ e \in NA_G^{\tilde{S}} \mid \text{src}_G^{\tilde{S}}(e) \in V_D^{\tilde{S}} \wedge \sigma_{NA_G}(e) \in NA_D^{\tilde{S}} \right\} \setminus \left( m_{NA_L}^{\tilde{S}}(NA_L^{\tilde{S}} \setminus NA_K^{\tilde{S}}) \cup \left\{ e \in NA_G^{\tilde{S}} \mid \exists e' \in NA_G^{\tilde{T}}. \iota_{NA_G^{\tilde{S}}}(e) = \iota_{NA_G^{\tilde{T}}}(e') \wedge \tau_{NA_G}(e') \notin NA_D^{\tilde{T}} \right\} \right),$$

$$\begin{aligned}
NA_D^C := & \left\{ e \in NA_G^C \mid src_G^C(e) \in V_D^C \right\} \setminus \left( m_{NA_L}^C (NA_L^C \setminus NA_K^C) \cup \right. \\
& \left. \left\{ e \in NA_G^C \mid \exists e' \in NA_G^{\tilde{S}}. \iota_{NA_G^{\tilde{S}}}(e') = e \wedge \sigma_{NA_G}(e') \notin NA_D^{\tilde{S}} \right\} \cup \right. \\
& \left. \left\{ e \in NA_G^C \mid \exists e' \in NA_G^{\tilde{T}}. \iota_{NA_G^{\tilde{T}}}(e') = e \wedge \tau_{NA_G}(e') \notin NA_D^{\tilde{T}} \right\} \right) ,
\end{aligned}$$

where  $E_D^{\tilde{T}}$  and  $NA_D^{\tilde{T}}$  are defined analogously to the case of  $\tilde{S}$  and for  $X \in \{\tilde{S}, C, \tilde{T}\}$ , the set  $EA_D^X$  is defined analogously to the case of  $NA_D^X$ . Furthermore, all source and target functions of  $D$  and its morphisms  $\sigma_D, \tau_D, \iota_{D_S}$ , and  $\iota_{D_T}$  are defined as restriction of the according function or morphism in  $G$ .

Finally, the morphism  $g : D \rightarrow G$  is the inclusion, and the morphism  $d : K \rightarrow D$  is defined as  $m \circ l$ , i.e., it maps any element  $x \in K$  to  $g^{-1}(m(l(x))) \in D$ .

The concrete construction of final pullback complements clarifies how sesqui-pushout rewriting behaves in the category  $\mathbf{APTTrG}_{ATG}$ . We summarize this in the next corollary.

**Corollary 5.29** (Sesqui-pushout rewriting in  $\mathbf{APTTrG}_{ATG}$ ). *Given a rule  $\rho = (L \xleftarrow{l} K \xrightarrow{r} R, ac)$  in  $\mathbf{APTTrG}_{ATG}$  and an arbitrary quasi-injective morphism  $m : L \rightarrow G$  such that  $m \models ac$ , the transformation  $G \Rightarrow_{\rho, m}^{SqPO} H$  exists, i.e.,  $\rho$  is applicable at  $m$  in the sesqui-pushout approach.*

*Moreover, the resulting span  $G \xleftarrow{g} D \xrightarrow{h} H$  is a span of  $\mathcal{M}$ -morphisms and the comatch  $n : R \rightarrow H$  is quasi-injective.*

## 5.4 Related Work

Subcategories of functor categories have been investigated frequently in all possible variants (i.e., restricting the objects, the morphisms, or both). In [99], certain functor categories are restricted to contain only *cartesian natural transformations*, i.e., natural transformations where every naturality square is a pullback. These are used to generalize and characterize the concept of a Van Kampen cocone. In [40], cartesian natural transformations have been used to characterize strongly regular monads. Moreover, Johnstone et al. [120] have investigated under which conditions quasiadhesiveness is preserved by artin glueing, which is a special kind

of comma category construction. The category of injective functions (or, more generally, monomorphisms; compare Examples 5.1 and 5.2) can also be realized as a full subcategory of the arrow category which, in turn, can be obtained by artin glueing. Their results imply that injective functions (or monomorphisms), considered as full subcategory of the arrow category, constitute a quasiadhesive category if the base category is a quasitopos [120, Lemma 25]. In contrast, we receive the result that injective functions (or, more generally, monomorphisms) constitute an adhesive category when the base category is adhesive and additionally, the class of morphisms is restricted to the pullback squares between injective functions. The connection between these results is expressed in Fact 5.1 which is based on [120, Lemma 14]: Our additional restriction of the morphisms results in the exclusion of monomorphisms which are not regular.

Since being a monomorphism can be characterized in terms of a pullback,  $S$ - $\mathcal{M}$ -restricted functors can also be introduced as models for a very specific kind of sketch [23], namely for a sketch that only prescribes the cones that are necessary to express that the desired morphisms are to be mapped to monomorphisms. At least, this is possible in the case where  $\mathcal{M}$  is the class of all monomorphisms. It has been argued that sketches provide a suitable formal framework to model database schemas (see, e.g., [39, 119]). We are not aware, however, of any theory that has been developed for that specific kind of sketches, let alone theory considering the subcategories in which we are interested.

Maximova et al. introduced the concept of  $\mathcal{M}$ -functors to relate the transformations in different  $\mathcal{M}$ -adhesive categories [165, 166]. An  $\mathcal{M}$ -functor is a functor between  $\mathcal{M}$ -adhesive categories that maps  $\mathcal{M}$ -morphisms to  $\mathcal{M}$ -morphisms and preserves pushouts along  $\mathcal{M}$ -morphisms. Proposition 5.6 and Theorem 5.8 immediately imply that the inclusion functor  $I : [\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}} \hookrightarrow [\mathcal{X}, \mathcal{C}]$  that embeds an  $S$ -cartesian functor category into its ambient functor category is an  $\mathcal{M}$ -functor. However, we could not use the theory of  $\mathcal{M}$ -functors in any way because we first were able to establish adhesiveness of  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  by examining the properties of  $I$ . Maximova et al. also establish for  $\mathcal{M}$ -functors what we call preservation and reflection of rule applications (Proposition 5.9); they call this *translation and creation of transformations*. Interestingly, using the special structure of  $S$ -cartesian functor categories, we are able to show the reflection (creation) of transformations without any additional assumptions. Maximova et al.

are only able to show this for  $\mathcal{M}$ -functors that are bijections between the Hom-sets, which is a quite severe restriction. In our case, the inclusion functor  $I$  usually does not satisfy that assumption since  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  is a non-full subcategory of  $[\mathcal{X}, \mathcal{C}]$  in general. However, in [166] they also address application conditions, which we leave for future investigation.

We are not aware of further works which are closely related to our general approach. In particular, we are not aware of a unifying framework for the example categories we mentioned in the introduction. Therefore, in the following, we take up the examples again and discuss how they form instantiated  $S$ -cartesian functor categories. In addition, we shortly discuss related approaches that use partial morphisms in a way that is similar to ours.

The concept of attribution by Kastenbergh and Rensink in [122] is most closely related to our work. They use subgraphs for attribution and prove that the category of reflected monos is adhesive if  $\mathcal{C}$  is. This category is an  $S$ -cartesian functor category, namely  $[\bullet \hookrightarrow \bullet, \mathcal{C}_{\text{inj}}]_{\text{cart}}$  in our notation. This means that they construct a category with monomorphisms as objects in exactly the same way as we do. They do not generalize their construction to functors from an arbitrary small category  $\mathcal{X}$ , for different choices of  $S$ , or a restricted class of monomorphisms  $\mathcal{M}$ . Furthermore, they do not discuss more general variants of adhesiveness or additional HLR properties. Proving the results in Sect. 5.2.1, however, mostly reduces to inspecting the behavior of the  $S$ - $\mathcal{M}$ -restricted functors at a single morphism  $m \in S$ . Therefore, some of the proofs are similar in both approaches.

Double-pushout rewriting of graph transformation rules by so-called 2-rules has been extensively studied by Machado et al. in [161]. Applying a 2-rule to a rule does not need to result in a rule again since the resulting morphisms are not necessarily injective. This means that Machado et al. identify the problem we recall in Example 5.2. But instead of restricting the allowed morphisms (as we do), they equip their 2-rules with suitable *negative application conditions* (NACs). As their approach is specific to the rewriting of graph transformation rules and not directly generalizable to a purely categorical setting, this is not an option for our general setting. Considering the instantiation to typed graphs, it is not difficult to see that their approach using NACs is more expressive than ours: None of the involved morphisms needs to form a pullback square for a 2-rule to be applicable and to result in a rule again. In [160], Machado discussed the problem in a

more categorical setting: He considers the full subcategory of spans of graphs (i.e., of  $[\bullet \leftarrow \bullet \rightarrow \bullet, \mathbf{Graphs}]$ ) that is induced by monic spans, called  **$T$ -Rules** there. In our nomenclature, this is the full subcategory induced by the  $S$ - $\mathcal{M}$ -restricted functors where  $S$  consists of the two non-identity morphisms (i.e.,  $[\bullet \leftrightarrow \bullet \leftrightarrow \bullet, \mathbf{Graph}]$ ). He thus also restricts to the same kinds of objects but without restricting the morphisms of the resulting category. He shows that  **$T$ -Rules** are a reflective subcategory of the category of spans and that they cannot be adhesive. The construction of the left adjoint to the inclusion functor is based on purely categorical concepts. Therefore, it should be possible to use that idea for showing that the full subcategory induced by  $S$ - $\mathcal{M}$ -restricted functors is a reflective subcategory of the ambient functor category under rather mild assumptions (existence of certain colimits of sources of morphisms and existence of an epi- $\mathcal{M}$  factorization). However, it is also clear that this adjunction cannot generally extend to an adjunction of the  $S$ -cartesian functor category: We show that the inclusion of  $S$ -cartesian functor categories into the ambient functor category creates pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms (Proposition 5.6). This is false for the inclusion of  **$T$ -Rules** into the category of spans (which can be seen in Example 5.2). Machado suggests that the unit of the adjunction might be used to “correct” the results of rewriting. Concretely, the application of a 2-rule to a rule, i.e., an object of  **$T$ -Rules**, might be computed in the ambient category of all spans. This does not necessarily result in a rule again but the unit of the adjunction offers a definite way to associate a rule with the resulting span. In Theorem 5.27, we made use of the same idea when applying a rule to an  $S$ - $\mathcal{M}$ -restricted functor not in the  $S$ -cartesian functor category itself but in the ambient functor category. But our motivation is different as we want to use rules, which are not defined in the  $S$ -cartesian functor category to gain expressiveness. Moreover, we do not need to repair the results since the applied rules are derived in such a way that their applications result in  $S$ - $\mathcal{M}$ -restricted functors again.

Golas et al. [89] provide a formalization of TGGs which generalizes correspondence relations between source and target graphs as well. They use special typings for the source, target, and correspondence parts of a triple graph and introduce edges between correspondence nodes and source and target nodes instead of using graph morphisms. This means that they work in the category of (typed attributed) graphs and thus, automatically obtain the standard results for double-pushout rewriting,

which are known to hold in that category. Their approach allows for more flexible correspondence relations and more flexible deletion and creation of references than possible in the categories of partial triple graphs that we defined. In contrast, our approach also allows for references between edges. In addition, it is more in line with the standard formalization of TGGs where triple graphs are defined as spans. Moreover, their approach does not easily extend to further example categories that are instances of  $S$ -cartesian functor categories or different choices of base categories  $\mathcal{C}$ .

In contrast to triple graphs whose correspondences between elements are defined by total morphisms, partial morphisms have also been used to formalize the correspondence of elements in scenarios where more than two meta-models are involved [129, 204, 130]. These categories can also be seen as instantiated  $S$ -cartesian functor categories. One chooses a star of spans as shape, the morphisms to the central node as set  $S$ , and a suitable category of models as base category  $\mathcal{C}$  (as already mentioned in Sect. 5.1). To the best of our knowledge, these categories have not been investigated for adhesiveness, yet. Considering our practical application of partial triple graphs, we are interested in allowing for partial correspondence morphisms only to obtain more incremental synchronization processes, which allow intermediate partial correspondences. After finishing synchronization, overall correspondences are expected, as they are expressed by total morphisms. When relating more than two models, the definition of correspondence makes only sense based on partial morphisms as argued in [129].

Partial morphisms have long been a research topic in the area of graph transformation, in particular in connection with the single-pushout approach to graph transformation as presented in, e.g., [125, 156, 62]. Moreover, there has been research computing limits in categories of partial morphisms, e.g., [173], and relating properties of pushouts in a category to properties of a pushout in the according category of partial morphisms [98, 95]. In that line of research, the class of morphisms of a given category is enlarged by considering also partial morphisms. In contrast, our framework allows considering partial morphisms as objects but pushouts and pullbacks are still computed (componentwise) along total morphisms.

In [66], Ehrig et al. also consider certain functors as objects of a new category to model distributed objects. They prove (co-)completeness if the base category is (co-)complete. In contrast to  $S$ - $\mathcal{M}$ -restricted functors, their functors allow for structure change in the category  $\mathcal{X}$  and the considered

morphisms are accordingly quite different from those in  $S$ -cartesian functor categories.

## 5.5 Conclusion

In this chapter, we present  $S$ -cartesian functor categories, which are subcategories of functor categories that preserve adhesiveness of base categories. Adhesiveness is an important prerequisite to show theoretical results of double-pushout rewriting. Furthermore, we obtain sufficient conditions under which  $S$ -cartesian functor categories also preserve additional HLR properties such that the comprehensive theory of double-pushout rewriting is obtained for this new kind of categories.  $S$ -cartesian functor categories generalize several categories for which double-pushout rewriting has been defined separately so far. As a new application case, we present a category of (typed attributed) partial triple graphs. The inspection of this category (as well as the comparison with the rewriting of rewriting rules by Machado (et al.) [160, 161]) shows that, while still interesting in practice, the restriction to a certain kind of morphisms comes with a price. With Fact 5.1 we present a natural border to relax this condition. However, we are interested in studying further (categorical) conditions on rules in functor categories to result in  $S$ -restricted functors when being applied to one (similar to Theorem 5.27). Moreover, we want to investigate how far  $S$ -cartesian functor categories can be instantiated to further structures of interest in model-driven engineering and theoretical computer science. Nested graph constraints [94], for example, seem to be a good candidate. Furthermore, our various results on preservation and reflection of rule applications (Propositions 5.22 and 5.23 and Theorem 5.27) should be lifted from plain rules to the case of rules with (negative) application conditions. It is important to note, however, that even though we use rules with NACs in the next chapter, the results for plain rules we obtained in this chapter suffice for our purpose. The reason for this is that we do not need to transform a sequence of rule applications in one category to a sequence of rule applications in another category. Instead, for our purposes it will be enough that a rule is applicable and that this application results in a (typed attributed) partial triple graph. But NACs only influence the applicability of rules and not the result.

## 6 Information-Preserving Model Synchronization

In this chapter, we develop our approach to TGG-based, information-preserving model synchronization.

### 6.1 Introduction

Our approach to the basic model synchronization problem is an extension of the one introduced by Leblebici (et al.) in [150, 151]. Our crucial innovation is the application of repair rules that allow us to obtain a more incremental synchronization process. To direct these processes, in particular, to control the application of repair rules, and to argue for their formal properties, we introduce *precedence graphs* (PGs) [149, 4] and generalize these to *partial covers*—a technical contribution in its own right. Those kinds of objects contain static information about how to create the elements of a (partial) triple graph and the interdependencies between those elements. The central idea behind them is the simple observation that, since TGG-rules are non-deleting, the comatches of a sequence of rule applications surjectively “cover” the produced triple graph. We call such a family of comatches a precedence graph; the comatches constitute its nodes, and edges are given by the dependencies between the rule applications, i.e., they encode information about the order in which the rules have to be applied. Conversely, under suitable conditions, families of morphisms from the rules’ RHSs that surjectively “cover” a triple graph constitute a PG, i.e., encode a sequence of rule applications producing that graph. However, whenever a triple graph has been edited, a formerly given PG normally does not do so any longer: Newly added elements are not covered, yet, and some comatches might not be “valid” any longer, e.g., because elements they map to have been deleted. For the updated structure, the former PG then merely constitutes what we call a partial cover.

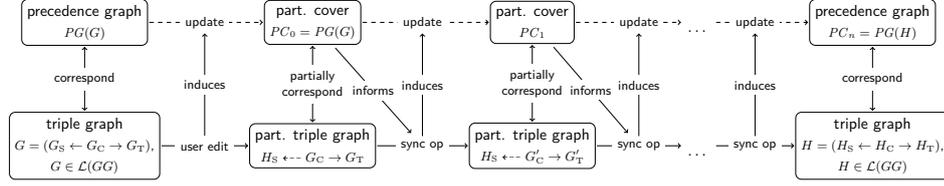


Figure 6.1: Schematic structure of synchronization process

Our synchronization algorithm takes this data into account and works on these two different levels: on the (partial) triple graph itself and the more abstract dependence structure. Figure 6.1 depicts the general structure of our approach (abstracting from almost all details): Given a triple graph from the language under consideration, that triple graph corresponds to a precedence graph. Whenever a user edits the triple graph, the resulting partial triple graph usually invalidates the precedence graph. However, the former precedence graph still serves as a partial cover and contains information about which parts of the updated source graph still have to be propagated or parsed differently. Guided by that information, we apply *synchronization operations* (derived from the rules of the given TGG) to the partial triple graph. But simultaneously, we observe how these applications affect the partial cover. As soon as the partial cover is actually a precedence graph again, synchronization has been successful. In this way, partial covers and precedence graphs not only direct the process but also guarantee its correctness.

As *synchronization operations*, we employ three different types of rules, namely *forward rules*, *repair rules*, and *revocation rules*. Forward rules are just the classical forward rules derived from a TGG. They serve to propagate additions of elements from the source to the target graph. Repair rules are forward rules of short-cut rules that have been derived from the given TGG. They serve to re-translate elements that had been accounted for by a rule application from the precedence graph but where this rule application is not valid any longer (either because of a user edit or because of the application of another synchronization operation). Finally, revocation rules are inverse rules of forward rules. They serve to propagate the deletion of elements and are used in situations, where no suitable repair rule is available.

The notable characteristics of our TGG-based approach are as follows.

- For adequate expressiveness, we support attributed triple graphs and negative constraints that restrict the language of a TGG; both are frequently needed in practice.
- Our approach is fully formalized and its central properties are proven. Most importantly, we prove that our synchronization process is *correct* (Theorem 6.18) and, under suitable circumstances, *terminates* and is *complete* (Proposition 6.16 and Theorem 6.17).
- Our approach is implemented, and implementation and formalization are in close correspondence.<sup>1</sup>
- Our approach avoids information loss. While we cannot establish the exact amount of preserved element in advance (i.e., without actually running the synchronization), Proposition 4.2 formally ensures that the application of repair rules preserves elements. With [185, 217] being the notable exceptions, other TGG-based approaches either suffer from information loss or are not formalized.

We start this chapter in Sect. 6.2 with formally presenting the kinds of TGGs we support, the synchronization problem for which our model synchronization algorithm provides a solution, and the synchronization operations we employ in our algorithm. Thereafter, we lay the formal basis to state our model synchronization algorithm and prove its properties by introducing precedence graphs and partial covers (Sect. 6.3). The crucial—but sometimes painfully technical—contribution of that section is the elaboration of the interaction of synchronization operations and partial covers (Sect. 6.3.2). Section 6.4 presents our actual algorithm and its formal properties. We also provide an additional example (extending our running example). Here, the hard work from the section before will pay off: based

---

<sup>1</sup>Even though implementation and evaluation are due to Lars Fritsche and not part of this thesis, their existence shows that the theory we develop in this thesis is practically viable. Implementation and evaluation are documented in our joint publications [79, 80, 77] and in his PhD thesis [76]. In contrast to the presentation in this thesis, the implementation offers no support for negative constraints, yet. Furthermore, only a fixed selected subset of repair rules is derived. Beyond this thesis, the tool offers support for additional constraints on the attribute values (with which the rules of the grammar can be equipped) and also implements a solution to the concurrent model synchronization problem. We present the approach that we took to deal with this more general problem in [77].

on that, the proofs of the properties of the algorithm are surprisingly short and intuitive. When comparing to related work in Sect. 6.5, we can consider details that were out of reach in Sect. 2.1. We conclude in Sect. 6.6. All proofs, except for the ones of the central results in Sect. 6.4, are given in Appendix B.3.

The publications [79, 80] are the basis of this chapter. However, we considerably exceed these. The exposition of markings, precedence graphs, and partial covers is completely new and allows for a far more general model synchronization process. In [80], we could only prove correctness of our process for specific kinds of edits and did not cover negative constraints.

## 6.2 Integrity-Preserving Triple Graph Grammars and the Basic Model Synchronization Problem

In this section, we formally introduce the *basic model synchronization problem* to which our approach offers a solution. In particular, we introduce the extended kind of triple graph grammars that we allow to be used to specify a consistency relationship, namely *integrity-preserving triple graph grammars* [126, 12]. Throughout this chapter, all (partial) triple graphs are considered to be *finitary* [81], i.e., while their algebras are allowed to be arbitrary, their sets of nodes, edges, and attribution edges are finite. For technical reasons, we also assume that all rules and constraints have a term algebra as their data part. The graphs to be rewritten will usually contain initial algebras; however, our approach does not rely on that.

### 6.2.1 Integrity-Preserving Triple Graph Grammars

Beyond plain TGGs as introduced in Definition 3.26, we support the extension of the consistency relationship that is defined by a TGG with *negative constraints* that forbid certain patterns to exist. As already reported in [12], this is equivalent to allowing the rules of the TGG to be equipped with certain kinds of *negative application conditions*. We start, however, with a short discussion regarding the semantics of constraints and conditions for attributed structures; a similar discussion can be found, e.g., in [100, Remark 2.10] or [54].

According to Definition 3.4, the satisfaction or violation of constraints and conditions hinges on the (non-)existence of certain  $\mathcal{M}$ -morphisms; this notion is introduced as  $\mathcal{M}$ -satisfiability in [94]. With this definition, and considering the special case of a negative application condition  $\neg\exists x : L \rightarrow N$  over the LHS  $L$  of some rule, a quasi-injective match  $m : L \rightarrow G$  trivially satisfies this NAC in the following two cases: (1) the chosen data algebra of  $N$  is not isomorphic to the one of  $G$  and (2) the match  $m$  identifies two data elements that are not identified by  $x$ .<sup>2</sup> Both are due to the fact that the  $\mathcal{M}$ -morphisms in the category of (typed) attributed (triple) graphs are isomorphisms on their data parts. However, both situations are common in applications. In particular, NACs are frequently attributed via some term algebra  $T(X)$ , whereas the graphs that are to be rewritten are attributed with the standard algebras for strings, integers, etc. (which are the initial ones) such that situation (1) is frequently given. Summarizing, for (typed) attributed (triple) graphs,  $\mathcal{M}$ -satisfiability leads to NACs that frequently do not restrict the applicability of rules. One alternative would be to define the satisfaction of application conditions via arbitrary morphisms (called  $\mathcal{A}$ -satisfiability in [94]). Indeed, both notions of satisfiability are expressively equivalent (under some mild assumptions on the concerned category) [94, Corollary 1]. However, expressing the non-identity of elements becomes much more involved when allowing arbitrary morphisms in the definition of satisfaction. Moreover, transforming a NAC (or a negative constraint) into an equivalent condition with regard to  $\mathcal{A}$ -satisfiability results in a condition that is not a NAC any more but truly nested (compare the construction after [94, Theorem 2]). Restricting to NACs (or negative constraints), as we are going to do in this section,  $\mathcal{M}$ -satisfiability is strictly more general. Indeed, when allowing arbitrary morphisms, we could not express the counting of elements as discussed in [92] (with regard to matching, the same phenomenon has been described in [93]).

Therefore, in this section we understand the satisfaction of constraints and conditions via quasi-injective morphisms. This is, in the following, satisfaction of a negative constraint or a NAC is defined as in Definition 3.4, just that the requirement “ $q \in \mathcal{M}$ ” is replaced by “ $q$  is quasi-injective”. For purely structural NACs (or constraints), i.e., a NAC  $x : L \rightarrow N$

---

<sup>2</sup>Dually, one obtains that *positive application conditions* always evaluate to false under these conditions.

where  $A_L = A_N$ ,  $NA_L = NA_N$ , and  $EA_L = EA_N$ , this leads to the familiar semantics as both  $\mathcal{M}$ -morphisms and quasi-injective morphisms are injective on the structural part. Moreover, the resulting semantics coincides with the one of *NAC schemata* that have been used in [100, 54] to address this very problem (for this, compare, e.g., [103, Remark 2.6]). While NAC schemata rely on a so-called *merge-construction* for application conditions, which allows the generalization of this approach to more general categorical settings, the approach via quasi-injective morphisms seems to be more accessible, when one is only interested in the case of graphs.

Next, we extend TGGs with negative constraints, and introduce *schema compliance* and *integrity-preserving triple graph grammars* [126, 12].<sup>3</sup> This extends the definition of a TGG by also considering (negative) constraints that additionally restrict the defined language.

**Definition 6.1** (Triple graph grammar with negative constraints. Schema compliance). *A triple graph grammar with negative constraints*

$$GG = (\mathcal{R}, S, NC, ATG)$$

consists of a finite set of monotonic rules  $\mathcal{R}$ , a start triple graph  $S$ , and a finite set  $NC$  of negative constraints, i.e., of constraints of the form  $\neg\exists C$ , such that  $S$  and all graphs that occur in the rules and the constraints are typed over  $ATG$  and these as well as  $ATG$  all have a finite structural part. Moreover, all graphs occurring in  $\mathcal{R}$  and in  $NC$  have term algebras as their data parts.

The language defined by a TGG with negative constraints is defined as

$$\mathcal{L}(GG) := \{G \in \mathbf{ATrG}_{ATG} \mid \text{there exists } S \Rightarrow_{\mathcal{R}}^* G \text{ and } G \models NC\} ,$$

i.e., it consists of all typed attributed triple graphs  $G$  that satisfy  $NC$  and are derivable by finite sequences of applications of rules from  $\mathcal{R}$  starting at  $S$ . Here,  $\Rightarrow_{\mathcal{R}}$  denotes a transformation step via a rule from  $\mathcal{R}$  applied at a quasi-injective match,  $\Rightarrow_{\mathcal{R}}^*$  denotes its reflexive and transitive closure, and  $G \models NC$  denotes satisfaction of each individual constraint  $c \in NC$ . Triple graphs from  $\mathcal{L}(GG)$  are also called *schema compliant*. *Source* and *target language*  $\mathcal{L}^S(GG)$  and  $\mathcal{L}^T(GG)$  are defined as projections on the respective components.

---

<sup>3</sup>In contrast to [12], to simplify notation, we directly understand the *schema* as part of the TGG.

Still, we always assume the start graph  $S$  to be a triple graph with empty structural part. As noted in Remark 3.9, we denote that triple graph as  $\emptyset$  even though its sets of attribute values are not empty. As our rewriting rules do not change the data part, all graphs of the language of a TGG (with constraints) have the same data part as  $\emptyset$ .

Of course, to avoid filtering, it is desirable that the rules of a given TGG with negative constraints cannot produce triple graphs that violate those constraints in the first place. Using the Shift-constructions introduced in Sect. 3.2.1, a triple graph grammar with negative constraints can be transformed into a triple graph grammar where the rules are equipped with negative application conditions in such a way that the defined languages coincide. First results in this direction have been obtained by Anjorin et al. in [12]. They showed that a TGG with such derived NACs is *integrity preserving*, which means that the application conditions only serve to prevent the introduction of constraint violations [126, 12]. We will not recall the formal definition but just introduce integrity-preserving TGGs as TGGs that have been obtained via that construction. Beyond [12], we lift the construction to attributed triple graphs and show that an integrity-preserving TGG actually produces *all* schema-compliant triple graphs from the original language (and not merely a subset of them).

To apply the Shift-construction, we have to choose an  $\mathcal{E}'$ - $\mathcal{M}'$  factorization of pairs of morphisms with respect to which the class  $\mathcal{F}$  occurring in the construction is defined. There are several possibilities for such a factorization in the category of typed attributed graphs [53, Example 9.22] (which can be lifted componentwise to the case of triple graphs). However, all these suggested factorizations have the common disadvantage that, when used as a basis for the Shift-construction, the resulting NACs will generally be infinite conjunctions. The reason for this is that with those choices for  $\mathcal{E}'$ , the computation of the NACs involves overlapping two attributed graphs in all possible ways. While this is fine for (finite) graphs, there are generally infinitely many ways to overlap the infinite term algebras that are used for attribution in rules and constraints. With our specific assumptions, however, namely the satisfaction of constraints and conditions being defined via quasi-injective morphisms, there exists another option. The intuition behind that factorization is that evaluation of variables should not be performed inside the NAC but by the morphism that checks for the (non-)satisfaction. Hence, the only identification of attribute values that

already takes place in the computed NAC is prompted by the existence of attribution edges pointing to the same value. We provide the details of that factorization in Appendix A. An alternative to that somewhat involved factorization would be to use *symbolic attributed graphs* [184] as foundation for attribution. There, only variables, which can be restricted by formulas, are part of the graphs and not a whole algebra. In this way, the graphs remain finite. However, a symbolic attributed graph usually represents a whole set of concretely attributed graphs. For use in TGGs this means that, when operationalizing its rules, one needs to transform the formulas of the rules into concrete assignments of values to variables in such a way that the formulas are satisfied [145]. To the best of our knowledge, this problem has not been addressed, i.e., there is no (automated) such translation available. Therefore, and because it is closer to the actual implementation in eMoflon, we stick with the formalization of attribution via E-graphs.

**Definition 6.2** (Integrity-preserving TGG). Given a triple graph grammar with negative constraints  $GG = (\mathcal{R}, \emptyset, NC, ATG)$ , its derived *integrity-preserving triple graph grammar*

$$GG^{NC} := (\mathcal{R}^{NC}, \emptyset, NC, ATG)$$

is the TGG whose set of rules  $\mathcal{R}^{NC}$  is defined as

$$\mathcal{R}^{NC} := \{\rho_i = (r_i, nac_i) \mid r_i \in \mathcal{R}\} ,$$

where  $NC = \{\neg\exists C_1, \dots, \neg\exists C_t\}$  and

$$nac_i := \bigwedge_{j=1}^t \text{Left}(r_i, \text{Shift}(\emptyset \rightarrow R_i, \neg\exists C_j)) .$$

Its language is defined as

$$\mathcal{L}(GG^{NC}) := \{G \in \mathbf{ATrG}_{\mathbf{ATG}} \mid \text{there exists } S \Rightarrow_{\mathcal{R}^{NC}}^* G\} .$$

*Source* and *target language*  $\mathcal{L}^S(GG^{NC})$  and  $\mathcal{L}^T(GG^{NC})$  are defined as projections on the respective components.

The standard results on the shift-constructions (Facts 3.5 and 3.6) immediately imply that an integrity-preserving TGG defines a sublanguage of

its underlying TGG with negative constraints, i.e., that all derivable triple graphs are schema compliant. Since we restrict ourselves to NACs and monotonic rules, the languages even coincide: In this way, it is excluded that a transformation sequence violates a negative constraint in between but results in a triple graph satisfying that constraint.

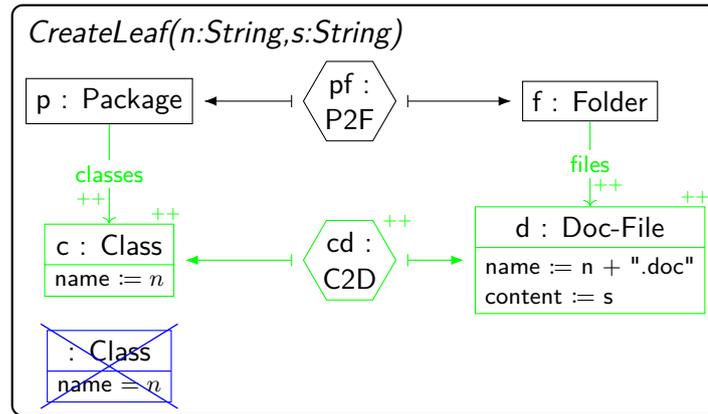
**Proposition 6.1** (Equivalence of defined languages). *Let  $GG = (\mathcal{R}, \emptyset, NC, ATG)$  be a TGG with negative constraints and  $GG^{NC} = (\mathcal{R}^{NC}, \emptyset, NC, ATG)$  its derived integrity-preserving TGG. Then*

$$\mathcal{L}(GG) = \mathcal{L}(GG^{NC}) .$$

*Remark 6.1.* The negative application condition with which we equip the rules of the integrity-preserving TGG is the *guaranteeing application condition* according to [94, Corollary 5]. For specific kinds of constraints and single-pushout rewriting in the category of graphs, this construction already dates back to [96]. When knowing that the start graph already satisfies the constraints of interest, the resulting application condition is much too complex because it also checks parts of the input graph to satisfy the constraint that are not affected by the application of the rule. This is unnecessary when validity of constraints is preserved throughout the transformation sequence. For their specific setting, Heckel and Wagner already reported that problem and provided a refined construction [96, Construction 3.3]. In [175, 176], we extended and generalized that refined construction. In particular, [176] presents it in the general setting of  $\mathcal{M}$ -adhesive categories and arbitrary nested constraints (addressing double-pushout rewriting). Moreover, relying on the notion of *graduated graph consistency* that we introduced in [135], we were able to show that in the case of negative constraints, the refined construction of application conditions results in the logically weakest application condition that still prevents introducing violations of the considered constraints [176, Theorem 1]. A similar simplification, also addressing negative constraints but stressing minimality of occurring patterns in the resulting application conditions (rather than logical weakness) can be found in [28]. We report on an implementation of our refined construction (and of the guaranteeing application condition) in [174]. The implementation is based on *Henshin* [16], a tool environment for in-place model transformations. Furthermore, there exists (a TGG-

$$\neg \exists \left( \begin{array}{|c|} \hline : \text{Class} \\ \hline \text{name} = n \\ \hline \end{array} \quad \begin{array}{|c|} \hline : \text{Class} \\ \hline \text{name} = n \\ \hline \end{array} \right)$$

Figure 6.2: Negative constraint forbidding any two Classes to have the same name

Figure 6.3: Rule *CreateLeaf* with constraint integrated as NAC

based) bridge between Henshin and eMoflon.<sup>4</sup> This means, the tooling is prepared to export TGG-rules from eMoflon to Henshin, to use Henshin to integrate (negative) constraints as NACs into these rules, and to re-export the results to eMoflon.

**Example 6.1.** A typical (negative) constraint is to require uniqueness of names. In our example scenario, one could require that no two Classes (or no two Packages) share the same name. Figure 6.2 visualizes that constraint and Fig. 6.3 shows the result when integrating that constraint as NAC into the rule *CreateLeaf*. The NAC just forbids a Class with the same name that shall be assigned to the newly created Class to already exist. This is already the optimized NAC. Computing the guaranteeing one would result in a NAC that additionally forbids two Classes with the same name to already exist.

The NACs of all rules of an integrity-preserving TGG prevent the introduction of the same forbidden patterns. This means, the application

<sup>4</sup>The code can be accessed at <https://github.com/eMoflon/emoflon-ibex-henshin>.

of such a rule can never introduce violations of the NACs of former rule applications. Because we need this property later, we state it in a separate lemma.

**Lemma 6.2** (Preservation of the validity of NACs). *Let  $GG = (\mathcal{R}^{NC}, \emptyset, NC, ATG)$  be an integrity-preserving TGG and*

$$\emptyset \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow \cdots \Rightarrow_{\rho_t, m_t} G_t$$

*a transformation sequence via rules from  $\mathcal{R}^{NC}$ ; for each  $1 \leq i \leq t - 1$  let  $tr_i : G_i \rightarrow G_t$  be the track morphism of the whole transformation sequence.*

*Then, for each  $1 \leq i \leq t - 1$*

$$tr_i \circ n_i \models \text{Right}(r_i, nac_i) \text{ ,}$$

*where the morphisms  $n_i$  are the comatches of the respective transformation steps and  $\rho_i = (r_i, nac_i)$ .*

## 6.2.2 The Basic Model Synchronization Problem

The *basic model synchronization problem* means to restore consistency between two related models after *one* of them has been changed. The more general *concurrent model synchronization problem* that addresses the question of restoring consistency after *both* models have been changed is out of the scope of this thesis; we contributed to research in that direction in [77].

We formally introduce *deltas* as changes of partial triple graphs, abstracting from the concrete way in which such a change is performed. In particular, a delta can capture some change performed by a user as well as the changes caused by rule applications during model synchronization. One formal intricacy has to be solved for that: Recall from Sect. 5.3 that in **APT<sub>r</sub>G<sub>ATG</sub>** it cannot be expressed that some correspondence element  $x$  of a partial triple graph  $G$  is in the image of a morphism  $f$  and belongs to the domain of one of the correspondence morphisms but that  $x$  does not also belong to the image of  $f$  when considered as an element of the domain of that correspondence morphism. This means, the exclusion (or inclusion) of an already *existing* correspondence element from (to) the domains of the correspondence morphisms cannot be expressed as morphism, even in cases

where this would result in a partial triple graph again. However, at least for exclusion, we need to formalize exactly that situation. It is common that a user deletes an element that is in the image of a correspondence morphism; a tool like eMoflon then (implicitly) deletes the preimage of that element from the domain of the correspondence morphism. However, the correspondence element itself is not deleted. As in Chapter 5, we address this problem by considering partial triple graphs as elements of the surrounding full functor category  $[\bullet \leftarrow \bullet \rightarrow \bullet \leftarrow \bullet \rightarrow \bullet, \mathbf{AGraph}]$ , also denoted as  $[\mathcal{A}, \mathbf{AGraph}]$  as in Sect. 5.3, where the required morphisms are available. For simpler notation, we do not explicitly indicate the inclusion functor.

Formally, a delta is just a span of injective morphisms with isomorphic data components.

**Definition 6.3** (Delta. Source- and target-delta. Induced source delta). Let  $G$  be a partial triple graph. A *delta for  $G$*  is a span of morphisms  $\delta = (G \xleftarrow{del} D \xrightarrow{add} G')$ , where  $D$  and  $G'$  are partial triple graphs and  $del$  and  $add$  are  $\mathcal{M}^{\text{comp}}$ -morphisms in  $[\mathcal{A}, \mathbf{AGraph}]/ATG$ , i.e., all their components are typed  $\mathcal{M}$ -morphisms between attributed graphs that are typed over the individual components of  $ATG$ . We say that  $\delta$  is a *source-delta* (*target-delta*) when  $del^C, del^T, add^C$ , and  $add^T$  ( $del^C, del^S, add^C$ , and  $add^S$ ) are isomorphisms.

Given a span of typed  $\mathcal{M}$ -morphisms  $G^S \xleftarrow{del^S} D^S \xrightarrow{add^S} G'^S$  where the attributed graphs are typed over the source component of  $ATG$ , its *derived source-delta*  $\delta = (G \xleftarrow{del} D \xrightarrow{add} G')$ , depicted in Fig. 6.4, is obtained by computing  $del^{\bar{S}}$  as pullback of  $del^S$  and  $\sigma_G$ ; all other morphisms are defined in the obvious way. A *derived target-delta* is defined analogously.

Using such spans to capture the change of an object is common; see, e.g., [205, 56, 183, 185, 54, 193]. Intuitively,  $G \setminus del(D)$  are the elements that get deleted, the elements of  $D$  are preserved, and the elements of  $G' \setminus add(D)$  are the elements that get newly created. In our specific setting of attributed graphs, the spans consisting of  $\mathcal{M}$ -morphisms implies that the data algebras of the graphs are not changed by a delta. A delta might stem from the application of a rule but it does not need to; we do not make any assumptions about that. In particular, deltas can capture free edits a user performs on a given (partial) triple graph. Practically, we

$$\begin{array}{c}
G = \\
\begin{array}{ccccccccc}
(G^S & \xleftarrow{\sigma_G} & G^{\tilde{S}} & \xleftarrow{\iota_G^{\tilde{S}}} & G^C & \xleftarrow{\iota_G^{\tilde{T}}} & G^{\tilde{T}} & \xrightarrow{\tau_G} & G^T) \\
\uparrow \text{del}^S & & \uparrow \text{del}^{\tilde{S}} & & \uparrow \text{id} & & \uparrow \text{id} & & \uparrow \text{id} \\
D = \\
(D^S & \xleftarrow{\sigma_D} & D^{\tilde{S}} & \xleftarrow{\iota_D^{\tilde{S}}} & G^C & \xleftarrow{\iota_G^{\tilde{T}}} & G^{\tilde{T}} & \xrightarrow{\tau_G} & G^T) \\
\uparrow \text{add}^S & & \uparrow \text{id} & & \uparrow \text{id} & & \uparrow \text{id} & & \uparrow \text{id} \\
G' = \\
(G'^S & \xleftarrow{\sigma_{G'}} & D^{\tilde{S}} & \xleftarrow{\iota_{G'}^{\tilde{S}} := \iota_D^{\tilde{S}}} & G^C & \xleftarrow{\iota_G^{\tilde{T}}} & G^{\tilde{T}} & \xrightarrow{\tau_G} & G^T)
\end{array}
\end{array}$$

Figure 6.4: Derived source-delta (where  $\iota_D^{\tilde{S}} := \iota_G^{\tilde{S}} \circ \text{del}^{\tilde{S}}$  and  $\sigma_{G'} := \text{add}^S \circ \sigma_D$ )

expect a user to somehow edit one model (source or target); formally, this can be captured as a span of  $\mathcal{M}$ -morphisms (on source or target domain). The tool environment then (implicitly) extends this edit to the derived source (or target) delta such that the original edit of the single model can be regarded as a delta of the whole triple graph. The computation of the derived delta is similar to the operation `fAln` in [104]. But instead of deleting correspondence elements from the correspondence graph, we delete them from the domain of the correspondence morphisms.

The *basic model synchronization problem* consists in restoring consistency between a pair of correlated models after one of them has been changed. We formally introduce it in the context of TGGs and source-deltas.

**Definition 6.4** (Basic model synchronization problem). Let  $GG$  be a(n integrity-preserving) TGG and  $G = (G^S \leftarrow G^C \rightarrow G^T)$  be a triple graph with  $G \in \mathcal{L}(GG)$ . Let  $\delta = (G \leftrightarrow D \leftrightarrow G')$  be a source-delta and  $G' = (H^S \leftarrow G^C \rightarrow G^T)$  be the partial triple graph obtained from its derived source-delta. The *basic model synchronization problem* is the task of finding a triple graph  $H = (H^S \leftarrow H^C \rightarrow H^T)$  such that  $H \in \mathcal{L}(GG)$ . The problem is analogously defined for target-deltas.

A procedure that computes a solution to the basic model synchronization problem is called a *model synchronization process* or *model synchronization algorithm*.

Of course, a model synchronization process should satisfy certain proper-

ties to be useful in practice. Numerous such properties have been suggested and discussed in the literature, compare, e.g., [73, 196, 199, 51, 54]. Terminology is not unified and not every concept directly fits for TGGs. In the following, we introduce properties that have often been discussed for TGGs [196, 55]; [196] serves as orientation for our terminology.

The maybe most basic property is the one of *correctness*: the computed result should belong to the language of the given TGG. We already required this in the definition of basic model synchronization problems.

**Definition 6.5** (Correctness). A model synchronization process is *correct* if for any given TGG  $GG$ , input triple graph  $G = (G^S \leftarrow G^C \rightarrow G^T)$ , and source-delta  $\delta = (G \leftrightarrow D \hookrightarrow G')$  with  $G' = (H^S \leftarrow G^C \rightarrow G^T)$  and  $H^S \in \mathcal{L}^S(GG)$ , the computed result  $H = (H^S \leftarrow H^C \rightarrow H^T)$  belongs to the language defined by  $GG$ , i.e.,  $H \in \mathcal{L}(GG)$ .

It is evident that  $H^S \in \mathcal{L}^S(GG)$  needs to hold for a model synchronization process to be able to obtain a result that belongs to the given language, at least when the source component  $H^S$  should not be altered. This delegates a lot of responsibility to the user as the construction and editing of valid models is highly non-trivial in general, in particular, when models comprise several thousand elements. As we will see, the approach to the model synchronization problem that we suggest is able to return valid models whose source component is at least “close” to the input. This practically interesting property has not often been addressed in research on TGGs (for an exception, see, for example, [217] in the context of the concurrent model synchronization problem).

*Completeness* means that for every sensible input the process is able to obtain a result.

**Definition 6.6** (Completeness). A model synchronization process is *complete*, if for any given TGG  $GG$ , input triple graph  $G = (G^S \leftarrow G^C \rightarrow G^T)$ , and source-delta  $\delta = (G \leftrightarrow D \hookrightarrow G')$  with  $G' = (H^S \leftarrow G^C \rightarrow G^T)$  and  $H^S \in \mathcal{L}^S(GG)$ , the process obtains a result  $H = (H^S \leftarrow H^C \rightarrow H^T)$ .

We close this section by mentioning some properties that have been discussed in the literature and which we do not introduce. We deliberately do not discuss *functional behavior* of model synchronization processes (see, e.g., [54]), i.e., the property of a synchronization process to always compute

the same output for the same input. If a synchronization process exhibits this behavior in both directions, it either means that the correspondence relation defined by the TGG is a bijection (which is a very restricted setting) or that systematically some results cannot be computed. Instead, we will present an indeterministic approach. Such an approach can always be made (more) deterministic by computing more than one solution, by user interaction, or by employing predefined policies.

Further properties that have been discussed include *(weak) invertibility* [51, 54], a property our approach does not generally satisfy. Intuitively, invertibility means that all information that has been added to a source model can also be propagated to the target model (and vice versa). This basically means that forward and backward synchronization become inverses of each other. Weak invertibility is less strict but still closely connected to functional behavior, which we want to avoid. The conditions under which (weak) invertibility has been proven for TGGs are very strict [54, Theorem 9.29]; under these circumstances we would be able to prove our synchronization process to be (weakly) invertible, too.

We will (more) formally introduce and discuss *incrementality*, *information loss*, and *hippocraticness* in Sect. 6.4.4, where we show in which sense our approach satisfies these.

### 6.2.3 Derived Model Synchronization Operations

In this section, we present the operations that constitute the basis for our model synchronization algorithm. In our setting, these operations are graph transformation rules that can be derived from the given TGG. We will employ three different kinds of rules: *forward rules*, *revocation rules* (inverses of forward rules), and *repair rules*. Forward rules serve to propagate additions from the source to the target model, whereas revocation rules serve to propagate deletions or to revoke translations that are not valid any longer. The use of both of them is standard. Beyond these, the use of repair rules is conceptually new. While some approaches used repair rules [84, 31], they were not able to derive them automatically and to provide correctness guarantees. Our repair rules are obtained as forward rules of short-cut rules that are derived from the given grammar. We derive all rules that delete elements, i.e., revocation rules and repair rules, in a basic and in relaxed variants. As we will perform model synchronization

using a top-down approach (viewed from the perspective of a precedence graph), it will be necessary to apply deleting rules in such a way that adjacent edges of deleted nodes are (implicitly) deleted as well. Thus, formally, we apply these rules using the sesqui-pushout approach, for which we have provided the theory in Sect. 5.3. This, in turn, might result in situations where elements that a rule shall delete have already been deleted implicitly by a previous rule application. Furthermore, on the source side, revocation rules might require elements that have been deleted by the user edit. The relaxed variants of deleting rules take these situations into account.

**Forward rules** We start with introducing forward rules; in all of the following, we directly assume an integrity-preserving TGG as introduced in Definition 6.2.

**Definition 6.7** (Forward rules with (filter) NACs). Let  $GG^{NC} = (\mathcal{R}^{NC}, \emptyset, NC, ATG)$  be an integrity-preserving TGG. For every rule  $\rho = (r : L \hookrightarrow R, nac) \in \mathcal{R}^{NC}$ , its *forward rule* is the rule

$$\rho^F := (r^F : L^F \hookrightarrow R^F, nac^F \wedge fnac) ,$$

where  $r^F : L^F \hookrightarrow R^F$  is defined as in Definition 3.27,  $fnac$  is a suitable filter-NAC that depends on  $GG$ , and

$$nac^F := \text{Right}(\iota, nac) ,$$

where  $\iota : L \hookrightarrow L^F$  is the embedding of the LHS  $L$  of  $\rho$  into the LHS  $L^F$  of  $\rho^F$  (considered as a rule).

The *required elements* of a forward rule  $\rho^F$  are structural elements that already stem from the LHS of the original rule  $\rho$ , i.e., the structural elements of  $L$ . Its *marking elements* are the structural elements that extend the LHS, i.e., the elements from  $L^F \setminus \iota(L)$ .

*Remark 6.2.* As  $\iota : L \hookrightarrow L^F$ , considered as a rule, is a monotonic rule itself, the computation of  $\text{Right}(\iota, nac)$  just amounts to computing the pushout of  $\iota$  along  $x : L \rightarrow N$  for every  $x \in nac$ . The intuition behind the definition is the following: As depicted in Fig. 6.5, the original TGG rule  $r$  (second line of the figure) can also be realized as a concurrent

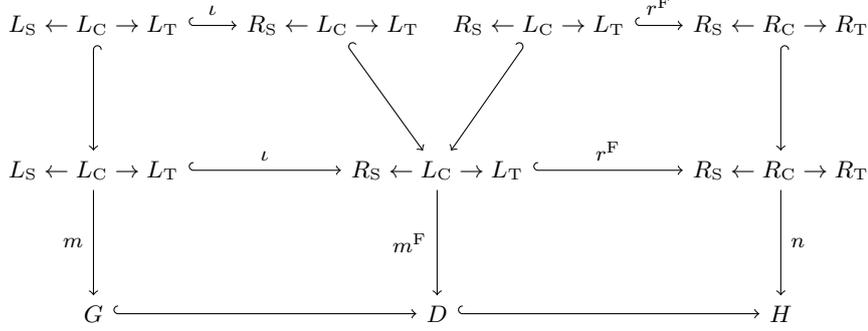


Figure 6.5: TGG rule as concurrent rule of forward rule and adapted source rule

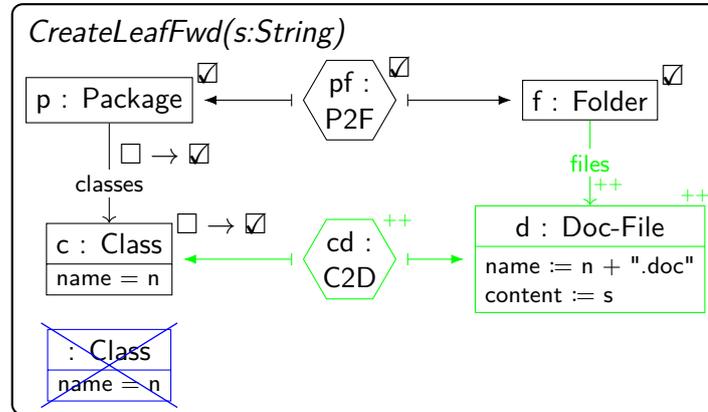
rule of a somewhat adapted source rule (containing the left-hand sides of correspondence and target part) and the forward rule. Thus, the definition of  $nac^F$  as  $\text{Right}(\iota, nac)$  by Fact 3.6 implies

$$m \models nac \iff m^F \models nac^F$$

in the case that  $m^F$  is the comatch of an application of  $\iota$  (considered as rule) at  $m$ . Alternatively, as all application conditions are assumed to stem from negative constraints, one could just recompute  $nac^F$  for  $r^F$  from the constraints.

*Filter NACs* serve to avoid dead-ends in translation and synchronization processes [126, 103]. They block rule applications that would result in triple graphs that cannot be completely translated. To simplify the presentation, we only consider one special kind of such filter NACs. There are more general variants of filter NACs and, in principle, our approach would allow employing also more complex filter NACs. However, the kind of filter NACs we discuss is, to the best of our knowledge, the only kind of filter NAC for which a completely automated construction (from the rules of the given grammar) is available. Therefore, this decision does not restrict the practical applicability of our approach.

*The kind of filter NACs that we consider forbid the presence of edges which are (i) incident to a marking node of the forward rule, (ii) not themselves marked by this forward rule, and (iii) for which no other rule is available that could mark the edge later on.* These kinds of NACs were

Figure 6.6: Rule *CreateLeafFwd* with constraint integrated as NAC

(independently) introduced by Hermann et al. in [103, Fact 3.7] and Klar et al. in [126, Sect. 4]. We will make use of the fact that this kind of NACs are always *source* or *target NACs*: A filter NAC of a forward rule only forbids certain edges in the source graph to exist. Dually, a filter NAC of a backward rule only forbids certain edges in the target graph. For the general definition of a filter NAC see [103, Definition 3.5, 54, Definition 8.16].

**Example 6.2.** The forward rules from our running example have already been depicted in Fig. 2.5. The required elements are exactly the elements annotated with  $\checkmark$  and the marking elements the ones annotated with  $\square \rightarrow \checkmark$ . The forward rule *CreateRootFwd* of *CreateRoot* is equipped with a filter NAC that forbids applying that rule at **Packages** with an incoming **subPackages**-edge. Applying that rule to such a **Package** would result in a dead-end for the current translation or synchronization process as there is no rule available that could translate the edge later on.

In Fig. 6.6, we additionally depict the forward version of *CreateLeaf* where the constraint that forbids two **Classes** to have the same **name** (see Fig. 6.2) has been integrated as a NAC. We will evaluate such NACs on that part of a currently given partial triple graph that has already been parsed. In this way, we will be able to at least translate one of two **Classes** with the same **name**.

$$\begin{array}{ccc}
R^{\text{RR}} & \xleftarrow{r^{\text{RR}}} & L^{\text{RR}} \\
\downarrow u & & \downarrow \\
R^{\text{F}} & \xleftarrow{r^{\text{F}}} & L^{\text{F}}
\end{array}
\quad (\text{PB})$$

Figure 6.7: Construction of a relaxed revocation rule from the inverse of a forward rule  $r^{\text{F}}$

**Revocation rules** Next, we introduce *revocation rules*. They serve to revoke the actions that are described by markings that have become invalid, either by a user edit or by the revocation of another rule application. As a revocation should not change the source graph (which is fixed throughout our model synchronization process), the inverses of forward rules constitute a suitable starting point to construct revocation rules. However, this is not enough because a match for such a rule might not exist any longer. We therefore define (relaxed) revocation rules as a suitable set of subrules of inverses of forward rules. They do not need to be equipped with an application condition because they cannot create structure that violates a constraint.

**Definition 6.8** ((Relaxed) revocation rule). Let  $GG = (\mathcal{R}^{\text{NC}}, \emptyset, \text{NC}, \text{ATG})$  be an integrity-preserving TGG. For every rule  $\rho = (r : L \hookrightarrow R, \text{nac}) \in \mathcal{R}^{\text{NC}}$ , its *revocation rule* is the rule

$$\rho^{\text{R}} := R^{\text{F}} \hookleftarrow L^{\text{F}} : r^{\text{F}} ,$$

i.e., the inverse rule of the (plain) forward rule  $r^{\text{F}}$  of  $\rho$ .

Given a revocation rule  $R^{\text{F}} \hookleftarrow L^{\text{F}} : r^{\text{R}}$ , a *relaxed revocation rule*

$$\rho^{\text{RR}} = R^{\text{RR}} \hookleftarrow L^{\text{RR}} : r^{\text{RR}}$$

is any rule that is obtained in the following manner:  $R^{\text{RR}}$  is a partial triple graph, there exists an  $\mathcal{M}^{\text{comp}}$ -morphism  $u : R^{\text{RR}} \hookrightarrow R^{\text{F}}$  in  $[\mathcal{A}, \mathbf{AGraph}] / \text{ATG}$  and  $r^{\text{RR}}$  is obtained by pulling back  $r^{\text{F}}$  along  $u$  as depicted in Fig. 6.7.

Whenever there is no need to explicitly distinguish between revocation rules and their relaxed variants, we will also use the shorter notation  $\rho^R$  for relaxed revocation rules.

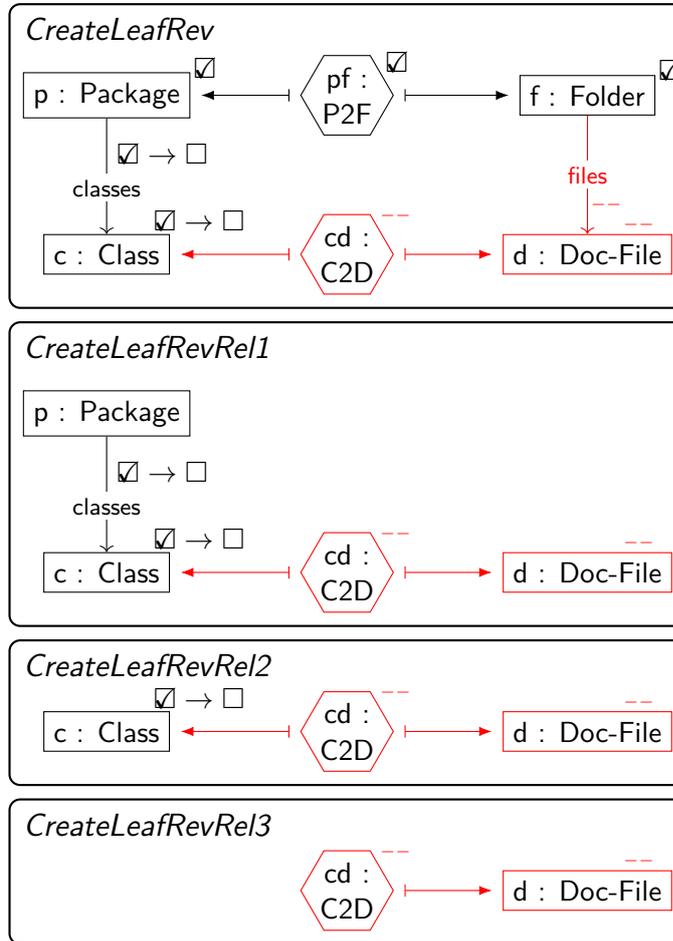
**Example 6.3.** Figure 6.8 depicts four revocation rules that can be derived from the rule *CreateLeaf*. The first is the “basic” revocation rule. It can be used in situations where it is not adequate any longer to translate the Class  $c$  with the forward rule of *CreateLeaf*. The other three rules are relaxed revocation rules. Their purpose is similar. *CreateLeafRevRel1*, for example, supposes a situation where the preceding rule application that created the correspondence between Package  $p$  and Folder  $f$  has already been revoked. This revocation has also implicitly deleted the *files*-edge and we therefore need a rule that does not demand to delete this edge again. In contrast, *CreateLeafRevRel2* and *CreateLeafRevRel3* are needed in situations in which a user deleted Package  $p$  and/or Class  $c$ . The annotation  $\square \rightarrow \square$  signifies that the respective elements become “unmarked” upon application of the rule. They have to be re-translated (in a different way) later on.

Even though morphisms from  $[\mathcal{A}, \mathbf{AGraph}]/ATG$  are used in the definition of revocation rules, they are actually rules in the category of typed attributed partial triple graphs. This is implied by the fact that the underlying forward rules are. Moreover, an application of a (relaxed) revocation rule does not change the source graph.

**Lemma 6.3** (Revocation rules are in  $\mathbf{APTrG}_{ATG}$ ). *Given a TGG  $GG$ , all derived (relaxed) revocation rules are rules in  $\mathbf{APTrG}_{ATG}$ , i.e., for any revocation rule  $\rho^R = R^F \leftarrow L^F : r^F$  or relaxed revocation rule  $\rho^{RR} = R^{RR} \leftarrow L^{RR} : r^{RR}$ , the morphism  $r^F$  or  $r^{RR}$  is an  $\mathcal{M}$ -morphism in  $\mathbf{APTrG}_{ATG}$ .*

*Moreover, the application of a (relaxed) revocation rule in the sesqui-pushout approach does not change the source graph of the partial triple graph it is applied to.*

**Repair rules** As a last operation, we introduce (*relaxed*) *repair rules*. Repair rules are forward rules of short-cut rules and serve to “re-parse” parts of a partial triple graph, i.e., to directly put elements into a new context without the need for deleting and recreating them. Relaxed repair rules, again, take into account that the deletions that are stipulated by

Figure 6.8: Examples for (relaxed) revocation rules derived from *CreateLeaf*

the underlying repair rule have already been performed. But whereas we could derive relaxed revocation rules for arbitrary subgraphs of the LHS of a revocation rule, the situation is more complex in the situation of repair rules. A repair rule is obtained from a short-cut rule  $\rho^{-1} \times_k \rho'$  and serves to replace a former application of the rule  $\rho$  by one of the rule  $\rho'$ . This means, when relaxing repair rules, we need to account for the situation that elements that  $\rho^{-1}$  should match or delete are already missing. But all

elements that are needed for an application of the forward rule  $\rho^F$  of  $\rho'$  need to be present. Thus, in relaxing repair rules, we need to prevent to remove elements that stem from the LHS of  $\rho^F$ .

To derive such relaxed repair rules from a repair rule (and also generally for the computation of application conditions of repair rules), we exploit the fact that the rules involved in the computations are related by various morphisms. Recall that the LHS of the second input rule  $\rho'$  for a short-cut rule is embedded into the LHS of that short-cut rule (via the morphism  $r'_1 \circ e_2$  in Fig. 4.2) and also into its interface (via the morphism  $k' \circ e_2$  in Fig. 4.3). As the next lemma shows, this implies that their respective forward rules are related in a similar way. As the statement is somewhat abstract, we briefly exemplify it using our running example: The LHS of the forward rule *CreateRootFwd* (see Fig. 2.5) just consists of the Package  $p$  (and its name-attribute); the short-cut rule *MakeRoot* (see Fig. 2.9, where we omitted the attributes for brevity) is computed with *CreateRoot* as its second input rule. The following lemma just states that we therefore will find an image of the Package  $p$  in the interface and the LHS of the forward rule of *MakeRoot* (depicted as repair rule *MakeRootFwd* in Fig. 2.10); this is the Package  $p_2$ . Similarly, we find an image of the LHS of *CreateSubFwd* (not depicted) in the interface and LHS of *CollapseHierarchyFwd* (also Fig. 2.10). Here, this consists of the Packages  $p_1$  and  $p_3$  and the subPackages-edge between them, the Folder  $f_1$ , and the correspondence node  $p_1f_1$ .

**Lemma 6.4** (Embedding of forward rules). *Let  $\rho^{-1} \times_k \rho' = (L^{\text{SC}} \leftarrow K^{\text{SC}} \hookrightarrow R^{\text{SC}})$  be a short-cut rule, which is derived from monotonic rules  $\rho$  and  $\rho'$  in  $\mathbf{ATrG}_{\text{ATG}}$ . Then there exists a morphism  $a_K : L'_F \rightarrow K^{\text{SC}}_F$  in  $\mathbf{APTrG}_{\text{ATG}}$  from the LHS  $L'_F$  of the forward rule  $\rho'_F$  to the interface  $K^{\text{SC}}_F$  of the forward rule of the short-cut rule. Consequently, there is also a morphism  $a_L : L'_F \rightarrow L^{\text{SC}}_F$  in  $\mathbf{APTrG}_{\text{ATG}}$  from  $L'_F$  to the LHS  $L^{\text{SC}}_F$  of the forward rule of the short-cut rule such that  $a_L = l^{\text{SC}}_F \circ a_K$ .*

**Definition 6.9** ((Relaxed) repair rule). Let  $GG = (\mathcal{R}^{\text{NC}}, \emptyset, \text{NC}, \text{ATG})$  be an integrity-preserving TGG. A *repair rule*

$$\rho^{\text{RP}} = (L^{\text{RP}} \xleftarrow{l^{\text{RP}}} K^{\text{RP}} \xrightarrow{r^{\text{RP}}} R^{\text{RP}}, \text{nac}^{\text{RP}} \wedge \text{fnac}^{\text{RP}})$$

derived from  $GG$  is any rule constructed in the following way:

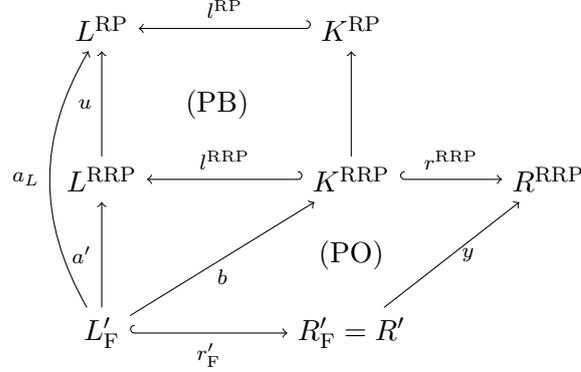


Figure 6.9: Construction of a relaxed repair rule from a given repair rule

- The underlying plain rule  $L^{\text{RP}} \leftrightarrow K^{\text{RP}} \hookrightarrow R^{\text{RP}}$  is the forward rule of a short-cut rule  $\rho^{-1} \times_k \rho' = (L^{\text{SC}} \leftrightarrow K^{\text{SC}} \hookrightarrow R^{\text{SC}})$  where  $\rho$  and  $\rho'$  are concurrent rules (of arbitrary length) built from rules from  $\mathcal{R}^{\text{NC}}$ .
- The NAC is defined as

$$nac^{\text{RP}} := \text{Shift}(a_L, nac') \text{ and } fnac^{\text{RP}} := \text{Shift}(a_L, fnac') ,$$

where  $a_L : L'_F \rightarrow L^{\text{SC}}$  is the morphism that exists according to Lemma 6.4 and  $nac'$  and  $fnac'$  are the NAC and filter NAC of  $\rho'^F$  (which are, in turn, obtained from the ones of the individual rules that compute  $\rho'$ ).

Given a repair rule  $\rho^{\text{RP}} = L^{\text{RP}} \xleftarrow{l^{\text{RP}}} K^{\text{RP}} \xrightarrow{r^{\text{RP}}} R^{\text{RP}}$ , a *relaxed repair rule*

$$\rho^{\text{RRP}} = (L^{\text{RRP}} \xleftarrow{l^{\text{RRP}}} K^{\text{RRP}} \xrightarrow{r^{\text{RRP}}} R^{\text{RRP}}, nac^{\text{RRP}} \wedge fnac^{\text{RRP}})$$

is any rule that is obtained in the following manner:  $L^{\text{RRP}}$  is a partial triple graph, there exists an  $\mathcal{M}^{\text{comp}}$ -morphism  $u : L^{\text{RRP}} \hookrightarrow L^{\text{RP}}$  in  $[\mathcal{A}, \mathbf{AGraph}]/ATG$  through which the the morphism  $a_L : L'_F \rightarrow L^{\text{RP}} = L^{\text{SC}}$  factors, and the rule is computed as follows (compare Fig. 6.9):

- (1)  $l^{\text{RRP}} : K^{\text{RRP}} \hookrightarrow L^{\text{RRP}}$  is obtained by pulling back  $l^{\text{RP}}$  along  $u$ .

- (2) When  $a'$  is the (unique) morphism that exists since  $a_L$  factors through  $u, b : L'_F \rightarrow K^{\text{RRP}}$  is the morphism induced by the universal property of the above pullback and  $l^{\text{RP}} \circ a_K = a_L = u \circ a'$  ( $a_K$  is not depicted in the figure).
- (3)  $r^{\text{RRP}} : K^{\text{RRP}} \hookrightarrow R^{\text{RRP}}$  is obtained by pushing out  $b$  along  $r'_F$ .
- (4) The NACs are computed by shifting along  $a'$ , i.e.,

$$nac^{\text{RRP}} := \text{Shift}(a', nac') \text{ and } fnac^{\text{RRP}} := \text{Shift}(a', fnac') .$$

As in the case of revocation rules, whenever there is no need to explicitly distinguish between repair rules and their relaxed variants, we will also use the shorter notation  $\rho^{\text{RP}}$  for relaxed repair rules.

Again, even though the computation is performed in  $[\mathcal{A}, \mathbf{AGraph}]/\text{ATG}$ , the resulting rules are actually rules in the category of attributed partial triple graphs.

**Lemma 6.5** (Repair rules are in  $\mathbf{APTrG}_{\text{ATG}}$ ). *Given a TGG  $GG$ , all derived (relaxed) repair rules are rules in  $\mathbf{APTrG}_{\text{ATG}}$ , i.e., for any repair rule  $\rho^{\text{RP}} = (L^{\text{RP}} \xleftarrow{l^{\text{RP}}} K^{\text{RP}} \xrightarrow{r^{\text{RP}}} R^{\text{RP}}, nac^{\text{RP}} \wedge fnac^{\text{RP}})$  or relaxed repair rule  $\rho^{\text{RRP}} = (L^{\text{RRP}} \xleftarrow{l^{\text{RRP}}} K^{\text{RRP}} \xrightarrow{r^{\text{RRP}}} R^{\text{RRP}}, nac^{\text{RRP}} \wedge fnac^{\text{RRP}})$  the morphisms  $l^{\text{RP}}$  and  $r^{\text{RP}}$  or  $l^{\text{RRP}}$  and  $r^{\text{RRP}}$  as well as all morphisms defining the NACs are ( $\mathcal{M}$ -)morphisms in  $\mathbf{APTrG}_{\text{ATG}}$ .*

*Remark 6.3.* The observations at the end of Sect. 3.2.4 show that the computation of a concurrent rule of monotonic rules is just a sequence of rule applications (compare Fig. 3.13). In the following, we assume that in the computation of the concurrent rule  $\rho$  that serves as first input for a short-cut rule  $\rho^{-1} \times_k \rho'$  from which a repair rule is derived, every rule application (transitively) depends on the first one. While this is not strictly necessary for our approach to work, it will greatly simplify the definition of a legal match for repair rules later on (which is already complex enough).

**Example 6.4.** Figure 6.10 displays examples for (relaxed) repair rules. The first one, *MakeRootFwd* (as already depicted in Fig. 2.10), is a repair rule that is obtained from *CreateSub* as first (inverse) input and *CreateRoot* as second. Shifting the filter NAC of *CreateRootFwd* (compare Fig. 2.5)

to the LHS of *MakeRootFwd* (along the morphism  $a_L$  whose existence is guaranteed by Lemma 6.4) results in the presented filter NAC with two parts in the following way: The forbidden referencing Package  $p'$  from *CreateRootFwd* could either be the Package  $p_1$  or an additional Package  $p'$ , again. The relaxed version *MakeRootFwdRel*, depicted below, omits only elements that stem from *CreateSub* (here: the complete first row), the first input of the underlying short-cut rule. The image of *CreateRootFwd* in *MakeRootFwd*, i.e., the Package  $p_2$  as mentioned above, is still present in *MakeRootFwdRel*. This version allows dealing with the situation where not only the deletion of the subPackages-reference but the deletion of also the root-Package has to be propagated. In our top down approach, the deletion of the root-Package is propagated first, which means to delete the corresponding Folder, which, in turn, triggers the implicit deletion of all outgoing subFolders-edges.

The third rule, *CollapseHierarchyFwd*, is a basic repair rule, again. Its first input is a concurrent rule of *CreateSub* with itself (creating a chain of two Packages and Folders at once—note that in the computation of this concurrent rule, the second rule application depends on the first), and its second input is also the rule *CreateSub*. The common kernel identifies the third row of *CollapseHierarchyFwd* as preserved instead of being deleted and recreated. The rule serves to directly propagate the collapse of a Package-hierarchy (by one level) to the corresponding Folder-hierarchy. As the forward rule of *CreateSub* does not have a filter NAC, *CollapseHierarchyFwd* does not get equipped with one. The final rule shows a relaxed version of *CollapseHierarchyFwd*. Again, the above described image of the LHS of *CreateSubFwd* in the one of *CollapseHierarchyFwd* is still a part of *CollapseHierarchyFwdRel*. The rule is crucial in situations in which a deleted Package contains more than one Package (and the user moves all of them). One application of *CollapseHierarchyFwd* deletes the Folder that corresponded to the deleted Package and (implicitly) the subFolder-edges to all dependent Folders. But only one of the dependent Folders is migrated to its new destination. The rule *CollapseHierarchyFwdRel* is needed to also migrate the rest.

The last example also shows that the relaxed variants of the rules allow us to handle situations in a lightweight fashion in which typically *amalgamation* is used; for sesqui-pushout rewriting, this concept has been

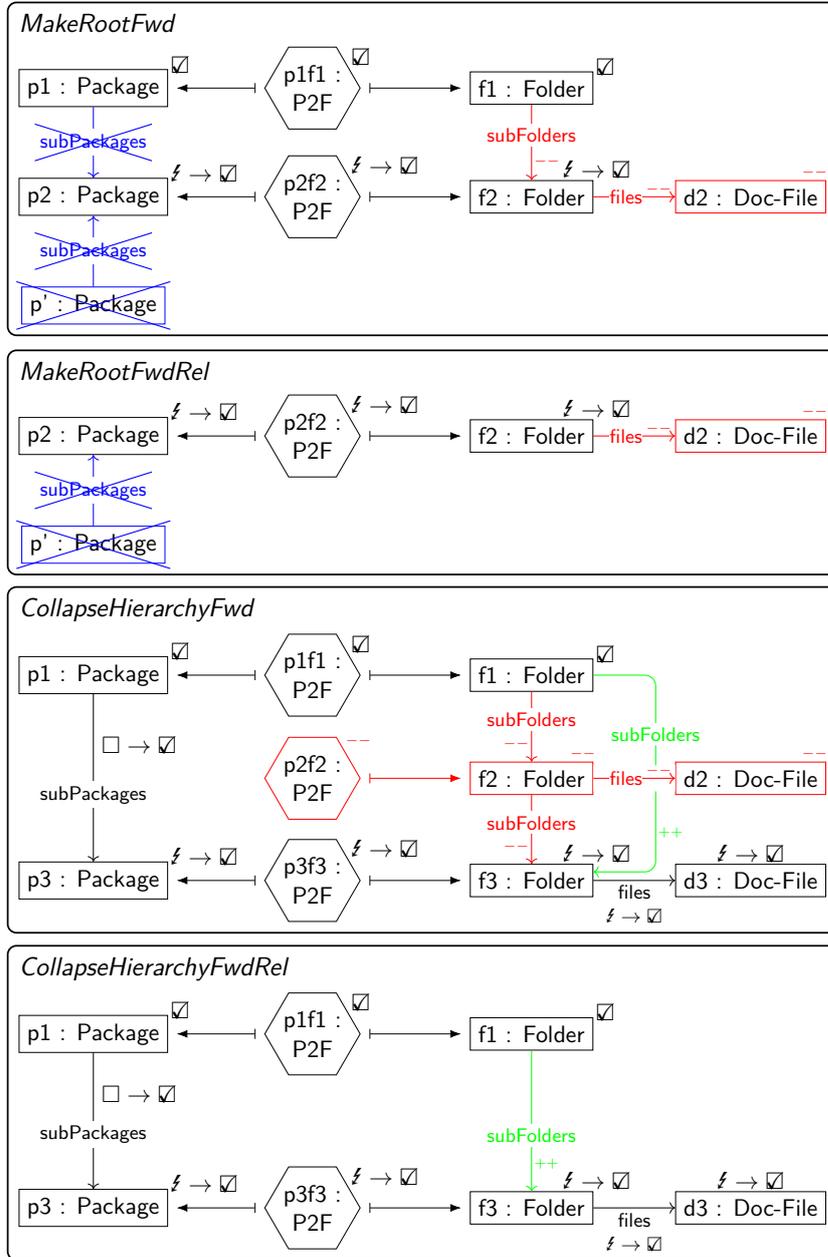


Figure 6.10: Examples for (relaxed) repair rules

developed by Löwe [157]. Consider the case where a **Package** with several sub-Packages is deleted, and some of these are deleted as well, some are distributed to other Packages, and some become root-Packages. The rule that revokes the creation of the first Package would constitute a *kernel rule* for rules that additionally revoke another of the deleted Packages and repair rules that additionally put the preserved Packages into their new contexts; for instance, a suitably relaxed variant of the revocation rule of *CreateSub* is a kernel rule of *CollapseHierarchyFwd*. The relaxed variants of revocation and repair rules then provide suitable *complement rules*; for instance, *CollapseHierarchyFwdRel* is one for *CollapseHierarchyFwd*. The computation of complement rules is performed from the view of the kernel rule—a complement rule performs the actions that have not yet (implicitly) been performed by the kernel rule. In contrast, our pullback-based computation of relaxed rules can be seen as starting from the result of an application of a kernel rule: the structure that is left after such an application is the structure that is still part of the relaxed rule. This will better fit with our synchronization process where an incremental pattern matcher monitors exactly which parts of a former valid rule application are still present.

**Consistency patterns** We close this section by introducing *consistency patterns*. These do not alter the structure of a partial triple graph but we use them in the next section to define partial covers and precedence graphs, which direct our model synchronization process. While we need information from the whole rule, we define consistency patterns just as RHSs of rules to stress that consistency patterns do not get applied. Furthermore, the NAC of the underlying rule and the filter NAC of its forward rule are suitably added to a consistency pattern to determine whether a morphism starting from it could possibly constitute a comatch.

**Definition 6.10** (Consistency pattern). Given an integrity-preserving TGG  $GG = (\mathcal{R}^{NC}, \emptyset, NC, ATG)$ , for any  $\rho_i = (r_i : L_i \hookrightarrow R_i, nac_i) \in \mathcal{R}^{NC}$ , the *consistency pattern derived from  $\rho_i$*  is the tuple

$$r_i^C := \left( R_i, nac_i^C \wedge fnac_i^C \right) ,$$

where  $nac_i^C$  and  $fnac_i^C$  are negative conditions over  $R_i$  and defined as follows:

$$nac_i^C := \text{Right}(r_i, nac_i)$$

and

$$fnac_i^C := \text{Right}(r_i^F : L_i^F \hookrightarrow R_i, fnac_i)$$

where  $r_i^F : L_i^F \hookrightarrow R_i^F = R_i$  is the forward rule of  $\rho_i$  and  $fnac_i$  its filter NAC.

The *required elements* of the consistency pattern  $r_i^C$  are those structural elements of  $R_i$  that have a preimage under the morphism  $r_i = (r_i^S, r_i^C, r_i^T)$ , i.e., the structural elements from  $r_i(L_i)$ . Its *marking elements* are the remaining structural elements, i.e., the elements from  $R_i \setminus r_i(L_i)$ .

*Remark 6.4.* As we assume all NACs to stem from negative constraints, the negative conditions  $nac_i$  can equivalently be defined as

$$nac_i := \bigwedge_{j=1}^t \text{Shift}(\emptyset \rightarrow R_i, \neg \exists C_j) ,$$

where  $\neg \exists C_1, \dots, \neg \exists C_t$  are the given negative constraints (compare Definition 6.2). This means, they can directly be obtained as part of the computation of the NACs of the rules of an integrity-preserving TGG.

**Example 6.5.** Figure 6.11 shows examples for consistency patterns. The pattern *CreateRootCon*, derived from the rule *CreateRoot*, is equipped with two negative conditions. The top one forbids **Package**  $p$  to be contained in another **Package** and stems from the filter NAC of *CreateRootFwd*. The second one forbids another **Package** with the same name to exist and stems from a negative constraint that forbids equal names also for **Packages** (as introduced for **Classes** in Example 6.1).

As *CreateSubFwd* does not have a filter NAC, the pattern *CreateSubCon* only contains negative conditions that stem from the constraint forbidding equality of names. Here, for simplicity, the part of the condition that forbids the equality of the names of **Packages**  $p_1$  and  $p_2$  is indicated as inequality inside of  $p_1$ . The forbidden extra node cannot express this as quasi-injective morphisms cannot identify it with the node  $p_1$ .

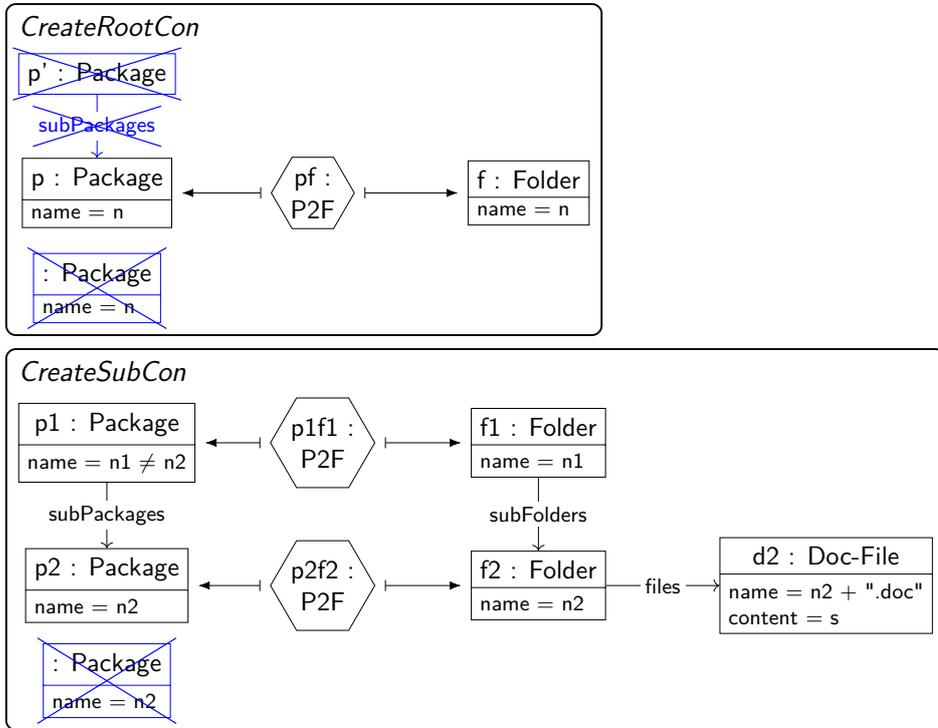


Figure 6.11: Examples for consistency patterns

## 6.3 Precedence Structure and Incremental Pattern Matching

In this section, we formally introduce (*partial*) *covers* and (*partial*) *precedence graphs* of (*partial*) triple graphs (Sect. 6.3.1) and investigate how these structures are affected by changes of the underlying (*partial*) triple graph (Sect. 6.3.2).

### 6.3.1 Markings, Covers, and Partial Precedence Graphs

Some of the following definitions are inspired by definitions already presented by Leblebici [150]. The central differences are that we lift the definitions from triple graphs to partial triple graphs, from application

sequences to precedence graphs (which abstract from the concrete order of independent rule applications), and from plain rules to rules with NACs.

In this part, also *partial morphisms* between partial triple graphs are considered, the idea being that a morphism that was obtained as a comatch of a transformation might become partial when a user edits the given triple graph: An element in the image of the comatch might be deleted. As the algebras (i.e., the data part) of our graphs are fixed during rewriting, it is enough to allow for partiality to occur on the structural part of partial triple graphs. This is achieved by considering every component of the domain of the partial morphisms to be an  $\mathcal{M}$ -*subobject* of the graph on which the partial morphism is defined. As above, to obtain the necessary expressivity, the domains of the partial morphisms are expressed via morphisms in  $[\mathcal{A}, \mathbf{AGraph}]/ATG$ .

**Definition 6.11** (Partial morphism between partial triple graphs). A *partial morphism*  $a : A \dashrightarrow B$  between partial triple graphs  $A$  and  $B$  from  $\mathbf{APTrG}_{ATG}$  is a span  $A \xleftarrow{\iota_A} A' \xrightarrow{a} B$ , where  $A'$  also is a partial triple graph from  $\mathbf{APTrG}_{ATG}$ ,  $\iota_A$  is an  $\mathcal{M}^{\text{comp}}$ -morphism in  $[\mathcal{A}, \mathbf{AGraph}]/ATG$ , and  $a$  is a morphism in  $\mathbf{APTrG}_{ATG}$ . The partial triple graph  $A'$  is called the *domain* of  $a$ .

Practically, the definition implies that a partial morphism between partial triple graphs is always defined on all data elements of the involved graphs (as every component of  $\iota_A$  is an isomorphism on its data part) and can only be defined on some edge if it is also defined on its source and target node (as the domain  $A'$  is also a partial triple graph). However, in contrast to Remark 5.3 or Proposition 5.24, allowing  $\iota_A$  to stem from  $[\mathcal{A}, \mathbf{AGraph}]/ATG$  implies that partial morphisms *do not need* to preserve definedness of the (partial) source and correspondence morphisms.

We will use *images* of families of partial morphisms to determine which parts of a partial triple graph still need to be translated from the source to the target side. *This serves as our formalization of marking, i.e., this is the way in which we keep track of already translated elements.* Again, as the algebras do not change, we concentrate on the image of the structural parts, which we call  $\mathcal{M}$ -*images*. Their definition uses a well-known decomposition of morphisms between attributed graphs: A morphism  $f = (f_G, f_A) : G \rightarrow H$  between ordinary typed attributed graphs can be split as  $(f_G, f_A) = (m_G, id) \circ (e_G, f_A)$ , where the epi-mono

factorization of  $f_G$  (on the structural parts of the graphs) provides  $m_G$  and  $e_G$ . This factorization is an *extremal  $\mathcal{E}$ - $\mathcal{M}$  factorization*, i.e., it factorizes  $f$  into an  $\mathcal{M}$ -morphism  $(m_G, id)$  and one that is extremal with respect to  $\mathcal{M}$  (see, e.g., [81]). In an  $\mathcal{M}$ -adhesive category, if it exists, such an extremal  $\mathcal{E}$ - $\mathcal{M}$  factorization is necessarily unique up to isomorphism [81, Fact 3.3]. This, in turn, ensures that it is meaningful to consider the codomain  $K$  of  $(e_G, f_A)$  as the  *$\mathcal{M}$ -image* of  $f$ :  $K$  is the “smallest”  $\mathcal{M}$ -subobject of  $H$  through which  $f$  factors. First, this factorization can be lifted componentwise to typed attributed partial triple graphs. Then, the factorization of a single morphism can be generalized to the one of a family of morphisms with the same codomain in exactly the same way as in Remark 3.6; one just uses arbitrary finite coproducts instead of binary ones. The next definition presents the resulting  $\mathcal{M}$ -subobject as  $\mathcal{M}$ -image of a family of partial morphisms, where partial morphisms are considered as ordinary morphisms from their domain. We give a more elementary, concrete definition, however.

**Definition 6.12** ( *$\mathcal{M}$ -image of a family of partial morphisms*). Given a finite family  $\Phi = \{\varphi_i : A_i \dashrightarrow B\}_{i \in I}$  of partial morphisms of typed attributed partial triple graphs with the same codomain and  $A'_i$  denoting their domains, its  *$\mathcal{M}$ -image*  $\text{Im } \Phi \subseteq B$  is the sub-partial-triple-graph<sup>5</sup> of  $B$  where every algebra coincides with the corresponding one of  $B$  and the structural part is given by the elements with preimage under one of the morphisms  $\varphi_i$ . This is, with

$$B^X = \left( (V_B^X, A_B^X, E_B^X, NA_B^X, EA_B^X, (src_j, tar_j)_{j \in \{B^X, NA_B^X, EA_B^X\}}), A^X \right)$$

for  $X \in \{S, \tilde{S}, C, \tilde{T}, T\}$  we define

$$\begin{aligned} (\text{Im } \Phi)^X = & \left( (V_{\text{Im } \Phi}^X, A_{\text{Im } \Phi}^X, E_{\text{Im } \Phi}^X, NA_{\text{Im } \Phi}^X, EA_{\text{Im } \Phi}^X, \right. \\ & \left. (src_j, tar_j)_{j \in \{(\text{Im } \Phi)^X, NA_{\text{Im } \Phi}^X, EA_{\text{Im } \Phi}^X\}}, \right. \\ & \left. A^X \right) \end{aligned}$$

<sup>5</sup>In contrast to *sub-partial-triple-graphs*, we will also speak of a *sub-triple-graph* of a partial triple graph  $G$  from  $\mathbf{APTTrG}_{\text{ATG}}$  when we want to stress that the sub-partial-triple-graph is basically a triple graph. Formally this means that this object is in the image of the inclusion functor  $I : \mathbf{ATrG}_{\text{ATG}} \hookrightarrow \mathbf{APTTrG}_{\text{ATG}}$  (see Proposition 5.23), or at least isomorphic to a partial triple graph inside that image.

via

$$\begin{aligned}
V_{\text{Im } \Phi}^X &:= \left\{ n \in V_B^X \mid \text{there are } i \in I \text{ and } n' \in V_{A'_i}^X \text{ s.t. } \varphi_i^X(n') = n \right\} \\
A_{\text{Im } \Phi}^X &:= A_B^X \\
E_{\text{Im } \Phi}^X &:= \left\{ e \in E_B^X \mid \text{there are } i \in I \text{ and } e' \in E_{A'_i}^X \text{ s.t. } \varphi_i^X(e') = e \right\} \\
NA_{\text{Im } \Phi}^X &:= \left\{ e \in NA_B^X \mid \text{there are } i \in I \text{ and } e' \in NA_{A'_i}^X \text{ s.t. } \varphi_i^X(e') = e \right\} \\
EA_{\text{Im } \Phi}^X &:= \left\{ e \in EA_B^X \mid \text{there are } i \in I \text{ and } e' \in EA_{A'_i}^X \text{ s.t. } \varphi_i^X(e') = e \right\}
\end{aligned}$$

and the source and target functions and also the correspondence morphisms are given by the corresponding restrictions, i.e.,  $\text{src}_{(\text{Im } \Phi)^X} : E_{\text{Im } \Phi}^X \rightarrow V_{\text{Im } \Phi}^X := \text{src}_{B^X} \upharpoonright_{(\text{Im } \Phi)^X}$ ,  $\sigma_{\text{Im } \Phi} := \sigma_B \upharpoonright_{(\text{Im } \Phi)^{\bar{s}}}$ , etc. By  $\iota_\Phi : \text{Im } \Phi \hookrightarrow B \in \mathcal{M}$  we denote the inclusion of the  $\mathcal{M}$ -image into  $B$ .

The following lemma, universally characterizing the  $\mathcal{M}$ -image, follows directly from the abstract construction indicated before its definition. It is also easily if somewhat tediously verified directly.

**Lemma 6.6** (Universal property of  $\mathcal{M}$ -image). *Let a finite family  $\Phi = \{\varphi_i : A_i \dashrightarrow B\}_{i \in I}$  of partial morphisms of typed attributed partial triple graphs with the same codomain and its  $\mathcal{M}$ -image  $\iota_\Phi : \text{Im } \Phi \hookrightarrow B$  be given. Then there are morphisms  $e_i : A'_i \rightarrow \text{Im } \Phi$  such that  $\iota_\Phi \circ e_i = \varphi_i$  for all  $i \in I$ , where  $A'_i$  are the respective domains. Moreover,  $\text{Im } \Phi$  satisfies the following universal property in this respect: For any  $\mathcal{M}$ -morphism  $m : Y \hookrightarrow B$  and family of morphisms  $e'_i : A'_i \rightarrow Y$  such that  $m \circ e'_i = \varphi_i$  for all  $i \in I$ , there exists a unique  $\mathcal{M}$ -morphism  $v : \text{Im } \Phi \hookrightarrow Y$  such that  $m \circ v = \iota_\Phi$ .*

To be able to model that formerly existing comatches might be “destroyed” when a user edits a model, we define *markings* to be partial morphisms; *partial covers* of a model are families of such markings.

**Definition 6.13** (Marking. Partial cover). Let a potentially partial triple graph  $G$  from  $\mathbf{APTTrG}_{\text{ATG}}$  and an integrity-preserving TGG  $GG = (\mathcal{R}^{NC}, \emptyset, NC, \text{ATG})$  be given. A (partial) *marking* of  $G$  is a (partial) quasi-injective morphism  $\varphi : R \dashrightarrow G$  in  $\mathbf{APTTrG}_{\text{ATG}}$ , where  $r^C = (R, \text{nac}^C \wedge \text{fnac}^C)$  is one of the consistency patterns of  $GG$ .

The *required elements* of a (partial) marking  $\varphi : R \dashrightarrow G$  are the elements in the image of the required elements of  $r^C$ , i.e., the structural elements

from  $\varphi(r(L))$ , where  $r : L \hookrightarrow R$  is the underlying rule of  $r^C$ . The *marked elements* of a (partial) marking are the elements in the image of the marking elements of  $r^C$ , i.e., the elements from  $\varphi(R \setminus r(L))$ .

A *partial cover of  $G$*  is a finite family  $PC_G = \{\varphi_i : R_i \dashrightarrow G\}_{i \in I}$  of potentially partial markings of  $G$  for some finite index set  $I$ . The image  $\text{Im } PC_G$  is the sub-partial-triple graph of  $G$  that is *covered by  $PC_G$* . The *marked elements* of a partial cover  $PC_G$  is the union of the marked elements of its individual markings.

Obviously, for a partial cover to correspond to a sequence of rule applications that creates the given graph, additional consistency conditions have to hold. We collect such consistency conditions and define *precedence graphs* based on them.

**Definition 6.14** (Consistency conditions for partial covers. Precedence graph). Let a (potentially partial) triple graph  $G$ , an integrity-preserving TGG  $GG$ , and a partial cover  $PC_G = \{\varphi_i : R_i \dashrightarrow G\}_{i \in I}$  of  $G$  be given.

Given two (potentially partial) markings  $\varphi_i : R_i \dashrightarrow G$  and  $\varphi_j : R_j \dashrightarrow G$ , where  $i, j \in I$ , marking  $\varphi_j$  *directly depends* on  $\varphi_i$ , denoted as  $\varphi_i <_d \varphi_j$ , if one of the required elements of  $R_j$  is matched by  $\varphi_j$  to an element of  $G$  that is marked by  $\varphi_i$ , i.e., whenever

$$\varphi_j(r_j(L_j)) \cap \varphi_i(R_i \setminus r_i(L_i)) \neq \emptyset .$$

*Dependence* is defined as the transitive closure of direct dependence and just denoted as  $<$ .

A partial cover  $PC_G$  is said to be

- *marking-consistent* if its above defined dependency relation is acyclic, i.e.,

$$\varphi_i < \varphi_j \implies \varphi_j \not< \varphi_i$$

for all  $i, j \in I$ ;

- *NAC-consistent* if no marking violates the NAC of its underlying consistency pattern, i.e., if for no  $i \in I$

$$e_i \not\# \text{nac}_i^C ,$$

where  $r_i^C = (R_i, \text{nac}_i^C \wedge \text{fnac}_i^C)$  is the underlying consistency pattern and  $e_i : R_i \rightarrow \text{Im } PC_G$  is the morphism with  $\iota_{\text{Im } PC_G} \circ e_i = \varphi_i$  that exists according to Lemma 6.6;

- *filter-NAC-consistent* if no marking violates the filter NAC of its underlying consistency pattern, i.e., if for no  $i \in I$

$$\varphi_i \not\models \text{fnac}_i^C ;$$

- *coherent* if no two markings mark the same element of  $G$ , i.e., if

$$\varphi_i(R_i \setminus r_i(L_i)) \cap \varphi_j(R_j \setminus r_j(L_j)) = \emptyset$$

for all  $i, j \in I$  where  $i \neq j$ ; and

- *internally complete* if every structural element (i.e., graph nodes and edges) of  $\text{Im } PC_G$  is marked by one of the markings  $\varphi_i$  and *complete* if every structural element of  $G$  is marked by one of the markings (i.e., if it is both internally complete and  $\text{Im } PC_G = G$ ).

We say that the partial cover is *consistent* if it is marking- and NAC-consistent (filter-NAC-consistency is *not* required).

A *partial precedence graph*  $PG_G$  for  $G$  is a consistent, coherent, and internally complete partial cover for  $G$ , where every marking is total. A *precedence graph* is a partial precedence graph where the underlying partial cover is even complete.

*Remark 6.5.* A series of comments on the above definition might be in order.

- When considering a (partial) precedence graph as a graph, the markings constitute its set of nodes and direct dependence defines the edges, i.e., there is an edge from  $\varphi_i$  to  $\varphi_j$  if and only if  $\varphi_i <_d \varphi_j$ . By *marking-consistency*, every (partial) precedence graph is a *directed acyclic graph* (DAG). In particular, its set of vertices can be sorted *topologically*, i.e., in such a way that in the resulting list, whenever  $\varphi_i <_d \varphi_j$ ,  $\varphi_i$  appears before  $\varphi_j$ ; this can even be done in linear time [121, Sect. 2.6]. Therefore, in all of the following, we will always assume (partial) precedence graphs (and also marking-consistent partial covers) to be given in the form  $PG_G = \{\varphi_1, \dots, \varphi_t\}$  such that  $\varphi_j \not<_d \varphi_i$  for all  $1 \leq i < j \leq t$  and, slightly abusing the set notation, say that  $\{\varphi_1, \dots, \varphi_t\}$  is topologically sorted.

- We define required and marked elements for the whole triple graph and not only its source graph. That differs from definitions of marking or translation attributes that only take the source elements into account as, for example, [104, 150]. The difference is that in these approaches (in different ways) consistency is restored by first determining a certain sub-triple-graph of the given partial triple graph. All correspondence and target elements that do not belong to it are deleted and a valid triple graph is obtained by re-translating the yet untranslated parts of the source graph using forward rules. In particular, the forward rules can freely match the existing correspondence and target elements. In contrast, in our approach there usually are correspondence and target elements in the currently given partial triple graph that are not yet part of the graph we are able to parse at that moment of the algorithm. Thus, to not introduce cyclic dependencies or to use elements that get deleted later on, we also need markings for the correspondence and target graph.
- We define filter NACs to be evaluated on the given partial triple graph  $G$  but the NACs to be evaluated on the image of the given partial cover. This is because both serve different purposes in our model synchronization process. Given a partial cover, we want to restore a (partial) precedence graph from it; as we will see in a moment, its image represents a triple graph (from the language of the given TGG). Therefore, the violation of a NAC of a consistency pattern in the image of a partial cover shows that this image violates one of the given constraints (by construction of these NACs). In particular, this image can never be extended to a schema compliant triple graph. Thus, in validating the NACs for the image of the partial cover, we ensure that the part that we are currently able to parse does not violate one of the negative constraints. In a certain sense, we do not care for constraint violations in the surrounding partial triple graph  $G$ ; these will be recognized when it is not possible to extend a given partial cover to further parse  $G$  without introducing a NAC violation for the then enlarged image.

In contrast, filter NACs signal that the context around the structure that we are currently able to parse is such that certain attempts to parse it must lead to dead-ends. Hence, we need to take the

whole partial triple graph  $G$  into account when evaluating filter NACs. Particularly, compared to the other consistency conditions, the violation of a filter NAC does not indicate that the partial cover cannot encode how to parse its image. It merely means that it will never be possible to extend the partial cover with further markings in such a way that the whole underlying graph  $G$  is marked and the other consistency conditions are still met. This is the reason why filter-NAC-consistency is not included in the definition of a (partial) precedence graph.

We illustrate the difference using the consistency pattern *CreateRootCon* from our running example (see Fig. 6.11): Assume a valid triple graph to be given that contains a root-*Package* with name  $n$ . A marking with underlying consistency pattern *CreateRootCon* signals that that part of the triple graph can be created using the rule *CreateRoot*. When a user adds a second *Package* with name  $n$  that does not change the validity of the triple graph we are currently able to parse. In our current model synchronization process, we will not (yet) try to constructively resolve that situation: We will just never parse the newly added *Package* with the forbidden name. In contrast, if a user adds another *Package* in the hierarchy above the currently considered one, the filter NAC violation signals that this *Package* cannot any longer be parsed using *CreateRoot*—at least if the newly added *Package* shall also become part of the considered triple graph. We are able elegantly resolve that situation (using a repair rule) and therefore evaluate filter NACs in the broader context. Note that the difference between the two situations is that in this second situation, the user provided an edit that remained in the source language of the given grammar, which is not the case in the first.

- The definitions of dependence and coherence as introduced above are closely related to the definitions of *sequential* and *parallel (in)dependence* of transformations (see Definition 3.11 and [53, Definition 3.15]). Intuitively, direct dependence between markings corresponds to sequential dependence between corresponding rule applications. Similarly, coherence corresponds to parallel independence of corresponding rule applications; in that case, though, of applications of the inverse rules. We need the concepts of dependence and coherence of markings,

however, to first determine when and how there exist transformations that correspond to a set of markings. Moreover, we need the concepts to be defined also in the absence of such transformations. Therefore, we cannot use the familiar notions of (in)dependence of transformations to define dependence and coherence of markings.

- In the definitions above, several related variants of *partiality* occur. First, the covered structure is allowed to be a partial triple graph. These are the objects occurring throughout our model synchronization processes. Secondly, partial markings are partial morphisms. We need this notion of partiality because, when the underlying partial triple graph changes, a formerly total marking might become partial: An element it matches can be deleted. Thirdly, a partial precedence graph might not cover its underlying structure completely: There can be elements in its underlying partial triple graph that are not in the image of the markings. Partial covers might be partial in both the second and third sense.

Partial precedence graphs correspond to sequences of rule applications and vice versa: Individual markings can be considered as comatches of a transformation sequence and the comatches of a transformation sequence can be assembled into a family of markings constituting a partial precedence graph. We first present the two transformations and state the precise relationship in a subsequent theorem. We start with a lemma that reconstructs a transformation sequence from a partial precedence graph. It can be seen as a generalization of [150, Lemma 4].

**Lemma 6.7** (From partial precedence graphs to transformation sequences). *Let an integrity-preserving TGG  $GG = (\mathcal{R}^{NC}, \emptyset, NC, ATG)$ , a possibly partial triple graph  $G$  from  $\mathbf{APTrG}_{ATG}$  whose algebras coincide with the ones from  $\emptyset$ , and a partial precedence graph  $PG_G = \{\varphi_1, \dots, \varphi_t\}$  of  $G$  in topological order be given. For  $1 \leq i \leq t$ , let  $\rho_i = (r_i : L_i \hookrightarrow R_i, nac_i)$  be the underlying rule of the consistency pattern  $r_i^C = (R_i, nac_i^C \wedge fnac_i^C)$  that is the base of the marking  $\varphi_i : R_i \rightarrow G$ . Then there is a transformation sequence*

$$t = (\emptyset \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow \dots \Rightarrow_{\rho_t, m_t} G_t) ,$$

such that (see Fig. 6.12)

- (1)  $G_t \cong \text{Im } PG_G$ ; in particular,  $\text{Im } PG_G \in \mathcal{L}(GG)$ , and
- (2)  $\iota_{PG_G} \circ tr_i \circ n_i = \varphi_i$  for all  $1 \leq i \leq t$ . Here,  $\iota_{PG_G} : \text{Im } PG_G \hookrightarrow G$  is the inclusion of the image of  $PG_G$  into  $G$ ,  $tr_i : G_i \rightarrow G_t$  is the track morphism including  $G_i$  into  $G_t$ , and  $n_i$  is the comatch of the respective transformation.

We denote this transformation sequence  $t$  as  $\text{Trafo } PG_G$ .

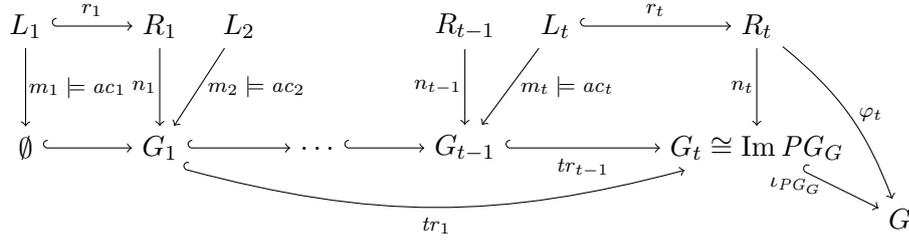


Figure 6.12: Obtaining a transformation sequence from a partial precedence graph  $PG_G$

In the other direction, transformation sequences induce partial precedence graphs (by composing the comatches with the track morphisms).

**Lemma 6.8** (From transformation sequences to partial precedence graphs). *Let an integrity-preserving TGG  $GG = (\mathcal{R}^{NC}, \emptyset, NC, ATG)$ , a partial triple graph  $G$  from  $\mathbf{APTTrG}_{ATG}$ , and a transformation sequence  $t = (\emptyset \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow \dots \Rightarrow_{\rho_t, m_t} G_t)$  via rules from  $GG$  be given such that there exists an  $\mathcal{M}$ -morphism  $u : G_t \hookrightarrow G \in \mathcal{M}$ . For  $1 \leq i \leq t$  let*

$$\varphi_i := u \circ tr_i \circ n_i : R_i \rightarrow G$$

be the morphism resulting from composing the  $i$ -th comatch  $n_i : R_i \rightarrow G_i$  with the  $i$ -th track morphism  $tr_i : G_i \rightarrow G_t$  and  $u : G_t \hookrightarrow G$ . Then the family

$$\Phi := \{\varphi_1, \dots, \varphi_t\}$$

is a (topologically sorted) partial precedence graph such that  $\text{Im } \Phi \cong G_t$ .

We denote this partial precedence graph as  $\text{PG}(t)$ .

Bundling these results shows the correspondence between sub-triple-graphs  $U \subseteq G$  of a partial triple graph  $G$  that belong to the language of the given TGG and partial precedence graphs of  $G$ .

**Theorem 6.9** (Correspondence result for partial precedence graphs). *Let an integrity-preserving TGG  $GG = (\mathcal{R}^{NC}, \emptyset, NC, ATG)$  and a possibly partial triple graph  $G$  from  $\mathbf{APTTrG}_{ATG}$  whose algebras coincide with the ones from  $\emptyset$  be given. Then sub-triple-graphs  $U \subseteq G$  with  $U \in \mathcal{L}(GG)$  correspond to partial precedence graphs of  $G$  with  $U$  as their image by virtue of the transformation sequences constructing them. For any transformation sequence  $t$  (via rules of  $GG$  and starting at  $\emptyset$ ) and any partial precedence graph  $PG_G$  of  $G$  this correspondence satisfies*

$$\text{Trafo}(PG(t)) \equiv t$$

and

$$PG(\text{Trafo}(PG_G)) = PG_G$$

where  $\equiv$  denotes switch equivalence. Moreover, there is a one-to-one correspondence between topological sortings of  $PG(t)$  and transformation sequences  $t'$  that are switch-equivalent to  $t$ .

The above theorem shows that (in our setting of monotonic rules) switch equivalent transformation sequences coincide with topological sortings of a certain partial order. While, as already mentioned, one such topological sorting can be found in linear time (depending on the number of nodes and edges), the task of finding all topological sortings of a given partial order is known to be #P-complete [35]. In our model synchronization algorithm, we therefore restrict ourselves to working with the given precedence graph in its current sorting instead of searching for another sorting that might be beneficial for the task at hand.

For our purposes, it is important to note that partial precedence graphs can be understood as representing an intermediate result of a synchronization process.

**Definition 6.15** (Partial synchronization graph). Let  $GG = (\mathcal{R}^{NC}, \emptyset, NC, ATG)$  be an integrity-preserving TGG,  $G = (G_S \leftarrow G_C \rightarrow G_T)$  some partial triple graph from  $\mathbf{APTTrG}_{ATG}$ , and  $PG_G$  a partial precedence graph for  $G$ . Let

$$\text{Im } PG_G = (U_S \xleftarrow{\sigma_U} U_C \xrightarrow{\tau_U} U_T) \in \mathcal{L}(GG)$$

be the sub-triple-graph of  $G$  that exists according to Lemma 6.7 and  $\iota_{PG_G}^S : U_S \hookrightarrow G_S$  the source part of the according inclusion morphism. Then the triple graph

$$G_S \xleftarrow{\iota_{PG_G}^S \circ \sigma_U} U_C \xrightarrow{\tau_U} U_T$$

is the *partial synchronization graph* represented by  $PG_G$ .

The last results show the connection between transformation sequences and partial precedence graphs; it is established by considering the individual markings as comatches of transformation steps and vice versa. Our model synchronization process uses a partial cover and basically replaces markings that cannot (any longer) contribute to a solution by more suitable markings. A marking cannot any longer contribute to a solution when elements that it requires or marks have been deleted or if its filter NAC is violated. Keeping a marking whose filter NAC is violated would mean that we could never extend the partial cover to a precedence graph for the underlying partial triple graph. The next definition formalizes this. We do not need to consider the violation of ordinary NACs because our starting point will be NAC-consistent partial covers and all operations that we apply during model synchronization preserve this property (as we will prove in the next section).

**Definition 6.16** (Invalidity of markings). Let a partial triple graph  $G$  and a partial cover  $PC_G = \{\varphi_1, \dots, \varphi_t\}$  of  $G$  be given. A marking  $\varphi_i$  of  $PC_G$  is *invalid* if either

- (1) the morphism  $\varphi_i$  is not total, or
- (2)  $\varphi_i \not\leq \text{fnac}_i^C$ .

A marking  $\varphi_i$  is *potentially invalid* if there exists an invalid marking  $\varphi_j$  such that  $\varphi_j < \varphi_i$ . A marking is *valid*, if it is neither invalid nor potentially invalid.

### 6.3.2 Delta-Induced Updates of Precedence Structure

Changing a (potentially partial) triple graph via a delta affects a partial cover defined for it. In this section, we discuss the changes that

$$\begin{array}{ccccc}
 R_i & \xleftarrow{\iota_{R_i}} & R'_i & \xleftarrow{p_i} & R''_i \\
 & & \downarrow \varphi_i & & \downarrow q_i \\
 & & G & \xleftarrow{del} & D & \xrightarrow{add} & G'
 \end{array}
 \quad (\text{PB})$$

Figure 6.13: Computation of the delta-induced partial cover

deltas induce on partial covers. For the whole section, we assume that an integrity-preserving TGG  $GG = (\mathcal{R}^{NC}, \emptyset, NC, ATG)$  is given; all markings, operationalized rules, etc. are obtained from  $GG$ . All occurring (partial) triple graphs stem from **APT<sub>r</sub>G<sub>ATG</sub>**; except for the graphs from the rules and the constraints, such graphs are assumed to be equipped with the same algebras as the start graph  $\emptyset$ .

Given a partial cover of a partial triple graph and a delta of that, the partial cover can be adapted to become a partial cover of the resulting partial triple graph.

**Definition 6.17** (Delta-induced partial cover). Given a partial cover  $PC_G = \{\varphi_i : R_i \dashrightarrow G\}_{(1 \leq i \leq t)}$  and a delta  $\delta = (G \xleftarrow{del} D \xrightarrow{add} G')$  of a partial triple graph  $G$ , the *delta-induced partial cover of  $G'$*

$$PC_{G'}^\delta = \{\varphi'_i : R_i \dashrightarrow G'\}_{(1 \leq i \leq t)}$$

consists of the markings

$$\varphi'_i := (R_i \xleftarrow{\iota'_{R_i}} R''_i \xrightarrow{\varphi'_i} G')$$

where  $\iota'_{R_i} := \iota_{R_i} \circ p_i$  and  $\varphi'_i := add \circ q_i$  are obtained via pulling back  $\varphi_i$  along  $del$  (in  $[\mathcal{A}, \mathbf{AGraph}]/ATG$ ) as depicted in Fig. 6.13. Whenever the delta is induced by a transformation  $t : G \Rightarrow_{\rho, m} H$ , we also denote its delta-induced partial cover as  $PC_{G'}^t$ .

Note that  $\iota'_{R_i}$  is an  $\mathcal{M}^{\text{comp}}$ -morphism in  $[\mathcal{A}, \mathbf{AGraph}]/ATG$  by closedness of these under pullback and composition; in particular, each  $\varphi'_i$  is guaranteed to be a partial morphism, again.

**Example 6.6.** Consider the precedence graph from Fig. 2.4 for the triple graph from Fig. 2.3 and the user edit that results in the partial triple graph from Fig. 2.6. In that situation, the delta-induced partial cover still consists of the same nodes and edges. Of course, the underlying morphisms change: Their codomain becomes the edited partial triple graph. And the morphisms represented by `CreateSub1` and `CreateSub2` become partial because elements to which they mapped (the `Package subP` and its incident edges) have been deleted.

On the level of individual markings, a delta might cause a formerly total marking to become partial or introduce a violation for its filter NAC. In particular, it might cause formerly valid markings to become (potentially) invalid. On a global level however, (most of) the introduced consistency conditions for partial covers, namely consistency, coherence, and internal completeness, are preserved under deltas.

**Lemma 6.10** (Preservation of consistency conditions for delta-induced partial covers). *Let a partial triple graph  $G$ , a partial cover  $PC_G$  of  $G$ , a delta  $\delta = (G \xleftarrow{del} D \xrightarrow{add} G')$ , and the corresponding delta-induced partial cover  $PC_{G'}^\delta$  of  $G'$  be given. Then, if  $PC_G$  is marking-consistent, NAC-consistent, coherent, or internally complete, so is the delta-induced partial cover  $PC_{G'}^\delta$ . Furthermore, a topological sorting for  $PC_G$  is also one for  $PC_{G'}^\delta$ .*

Next, we introduce *legal matches* for the different kinds of rules that we apply during model synchronization and extend the definition of delta-induced covers for them. These legal matches are matches at which the application of the respective kinds of rules preserves the consistency conditions for partial covers, which we also prove in the following. The correctness of our model synchronization process rests upon the preservation of these consistency conditions. Therefore, during those, we only apply rules at their legal matches (which is the reason for calling them “legal”).

We start with defining legal matches for forward rules. As already discussed for markings, we do not want to evaluate NACs for the whole given partial triple graph but merely for that part that is represented by the currently given partial cover. Hence, we cannot evaluate a NAC at the rule’s match but need to evaluate it at a morphism where the codomain is

restricted accordingly. This morphism is provided by Lemma 6.6, and we use it in the definition of legal matches for forward and for repair rules.

**Definition 6.18** (Image-restriction). Given a quasi-injective morphism  $m : L \rightarrow G$  from the left-hand side of a forward or a repair rule to a partial triple graph and a partial cover  $PC_G$  of  $G$ , let  $\text{Im } \Phi$  be the  $\mathcal{M}$ -image of the family of morphisms  $PC_G \cup \{m\}$ . The *image-restriction*  $m_{\text{Im}}$  of  $m$  is the unique morphism  $m_{\text{Im}} : L \rightarrow \text{Im } \Phi$  with  $\iota_\Phi \circ m_{\text{Im}} = m$  that exists according to Lemma 6.6.

**Definition 6.19** (Legal match for forward rule). Let a partial triple graph  $G$ , a partial cover  $PC_G$  of  $G$ , and a forward rule  $\rho^F = (r^F : L^F \hookrightarrow R^F, nac^F \wedge fnac)$  be given. A *legal match for  $\rho^F$  with respect to  $PC_G$*  is a quasi-injective morphism  $m^F : L^F \rightarrow G$  such that

- (1)  $m_{\text{Im}}^F \models nac^F$ , where  $m_{\text{Im}}^F$  is the image-restriction of  $m^F$ ,
- (2)  $m^F \models fnac$ ,
- (3)  $m^F$  matches every required element of  $\rho^F$  to an element that is marked by one of the valid markings of  $PC_G$ , and
- (4)  $m^F$  matches every marking element of  $\rho^F$  to an element that is not already marked by  $PC_G$ .

Given the transformation  $t : G \Rightarrow_{\rho^F, m^F} G'$ , the partial cover

$$PC_{G'}^F := PC_{G'}^t \cup \{n^F\}$$

that arises as union of the delta-induced partial cover  $PC_{G'}^t$  of the transformation  $t$  with the comatch  $n^F$  is called *forward-induced partial cover*.

**Lemma 6.11** (Preservation of consistency conditions for forward-induced partial covers). *Let a partial triple graph  $G$ , a partial cover  $PC_G$  of  $G$ , and a forward rule  $\rho^F = (r^F : L^F \hookrightarrow R^F = R, nac^F \wedge fnac)$  be given. Let*

$$t : G \Rightarrow_{\rho^F, m^F} G'$$

*be a transformation step where  $m^F$  is a legal match with respect to  $PC_G$ . Then, if  $PC_G$  is marking-consistent, NAC-consistent, filter-NAC-consistent,*

coherent, or internally complete, so is the forward-induced partial cover  $PC_{G'}^F$ . In particular, if  $PC_G$  is even a partial precedence graph, so is  $PC_{G'}^F$ .

Moreover, if one of the markings of  $PC_G$  is invalid and  $1 \leq j_0 \leq t$  is the smallest index such that  $\varphi_{j_0}$  is an invalid marking, then there exists a topological sorting

$$\{\varphi'_1, \dots, \varphi'_{j_0-1}, \dots, n^F, \varphi'_{j_0}, \dots\}$$

of  $PC_{G'}^F$ , i.e., a topological sorting of the forward-induced partial cover that maintains the order of the first  $j_0 - 1$ -th elements and sorts  $n^F$  in front of  $\varphi'_{j_0}$ .

The intuition behind the topological sorting in the above lemma is that, because  $\rho^F$  only matches validly marked elements and validly marked elements (by definition) do not depend on invalid markings,  $n^F$  and all markings it depends on might as well be sorted in front of the first invalid marking.

We define the revocation of a rule using sesqui-pushout transformations. This ensures that the revocation of an invalid marking is always possible (by implicit deletion of adjacent edges).

**Definition 6.20** (Legal match for revocation). Let a partial triple graph  $G$ , a partial cover  $PC_G$  of  $G$ , and a (relaxed) revocation rule  $r^R : R^R \leftrightarrow L^R$  be given. A *legal match*  $m^R : R^R \rightarrow G$  with respect to  $PC_G$  is a match for  $r^R$  (i.e.,  $m^R$  is a quasi-injective morphism) such that there exists an invalid marking  $\varphi_i \in PC_G$  such that

- (1) all  $\varphi_j \in PC_G$  with  $\varphi_j < \varphi_i$  are valid and
- (2)  $r^R$  and  $\varphi_i$  have the same underlying rule,  $\text{dom } \varphi_i \cong R^R$ , and  $m^R \cong \varphi_i$  (as typed partial morphisms between typed attributed partial triple graphs).

Given the transformation  $t : G \Rightarrow_{r^R, m^R}^{\text{SqPO}} G'$ , the partial cover

$$PC_{G'}^R := PC_{G'}^t \setminus \varphi'_i$$

that arises by removing the delta-induced partial marking  $\varphi'_i$  of  $\varphi_i$  from the delta-induced partial cover  $PC_{G'}^t$  is called *revocation-induced partial cover*. We call  $\varphi_i$  the *revoked marking* of  $t$ .

While the next lemma is technically trivial, it is central for the completeness of our model synchronization approach. It ensures that every invalid marking can be revoked (without changing the source graph according to Lemma 6.3).

**Lemma 6.12** (Possibility of revocation). *Given a partial triple graph  $G$  and a partial cover  $PC_G$  of  $G$ , for every  $\varphi_i : R_i \dashrightarrow G \in PC_G$  with underlying rule  $r_i : L_i \hookrightarrow R_i$  there exists a (relaxed) revocation rule  $r_i^{\text{RR}} : L_i^{\text{RR}} \hookrightarrow R_i^{\text{RR}}$  that is derived from  $r_i$  such that  $R_i^{\text{RR}} \cong \text{dom } \varphi_i$ . Moreover, if  $\varphi_i \in PC_G$  is invalid and such that  $\varphi_j$  is valid for all  $\varphi_j < \varphi_i$ ,  $\varphi_i$  is a legal match for that (relaxed) revocation rule.*

**Lemma 6.13** (Preservation of consistency conditions for revocation-induced partial covers). *Let a partial triple graph  $G$ , a partial cover  $PC_G$  of  $G$ , and a (relaxed) revocation rule  $r^{\text{R}} : R^{\text{R}} \hookrightarrow L^{\text{R}}$  be given. Let*

$$t : G \Rightarrow_{r^{\text{R}}, m^{\text{R}}}^{\text{SqPO}} G'$$

*be a transformation step, where  $m^{\text{R}}$  is a legal match for  $r^{\text{R}}$  with respect to  $PC_G$ . Then, if  $PC_G$  is marking-consistent, NAC-consistent, filter-NAC-consistent, coherent, or internally complete, so is the revocation-induced partial cover  $PC_{G'}^{\text{R}}$ . Moreover, if  $PC_G$  is coherent,  $\varphi_i$  is the marking revoked by  $t$ , and  $\varphi_j$  is a valid marking from  $PC_G$  such that  $\varphi_i \not\prec \varphi_j$ , then  $\varphi_j' \in PC_{G'}^{\text{R}}$  is valid.*

The statement about the validity of markings in the revocation-induced partial cover means that revocations can only trigger dependent markings to become invalid.

Finally, we define legal matches for (relaxed) repair rules and prove the same preservation properties as above also for the application of these kind of rules at their legal matches. Repair rules are the most complex kind of rules we use, and also the definition of their legal matches is somewhat involved. Recall that a (relaxed) repair rule ultimately stems from a short-cut rule  $\rho^{-1} \times_k \rho'$ , where  $\rho$  and  $\rho'$  are *concurrent compositions* of rules from the given TGG. The intuition is that the application of  $\rho^{-1} \times_k \rho'$  revokes an application of  $\rho$  and replaces it with one of  $\rho'$  while preserving the elements specified by the common kernel  $k$ . A repair rule, being a forward rule of such a short-cut rule, assumes that this kind of edit has

already been performed on the source graph (by some user) and performs the remaining actions on the correspondence and target graphs. Therefore, a (relaxed) repair rule must be matched in such a way that

- (1) it indeed revokes an application of  $\rho$  and
- (2) it positions the application of  $\rho'$  without introducing cyclic dependencies or multiply marked elements.

To ensure this, we formulate four central conditions on legal matches for repair rules in the next definition. The first serves to ensure that there actually exist markings that correspond to a sequence of the applications of the underlying rules of  $\rho$ . The second condition ensures that there is actually a reason to revoke  $\rho$ . The third is a technical assumption that prevents the introduction of cyclic dependencies, and the fourth condition ensures that a marking-consistent, coherent, and internally-complete partial cover remains so. We illustrate these conditions using our running example after formally presenting them in the following definition.

For the formal statement recall that, when  $\rho = R \leftrightarrow L$  and  $\rho' = L' \hookrightarrow R'$ , for every rule  $r_i : R_i \leftrightarrow L_i$  that contributes to  $\rho$ , there is a morphism  $\iota_i : R_i \rightarrow R$  and, likewise, for every rule  $r_j : L_j \hookrightarrow R_j$  that contributes to  $\rho'$ , there is a morphism  $\iota_j : R_j \rightarrow R'$  (see Remark 3.5). The RHS  $R$  of  $\rho$ , in turn, embeds (by construction) into the LHS  $L^{\text{SC}}$  of  $\rho^{-1} \times_k \rho'$  and that LHS is connected via a partial morphism with the LHS  $L_{\text{SC}}^{\text{F}} = L^{\text{RP}}$  of its forward rule (Lemma 5.26). Finally, the LHS  $L_{\text{F}}'$  of the forward rule of  $\rho'$  embeds into the LHS  $L^{\text{RP}}$  (or  $L^{\text{RRP}}$ ) of each derived (relaxed) repair rule (see Lemma 6.4 and Definition 6.9).

**Definition 6.21** (Legal match for repair). Let a partial triple graph  $G$ , a partial cover  $PC_G = \{\varphi_1, \dots, \varphi_t\}$  of  $G$  in topological sorting, and a (relaxed) repair rule

$$\rho^{\text{RRP}} = (L^{\text{RRP}} \leftrightarrow K^{\text{RRP}} \hookrightarrow R^{\text{RRP}}, \text{nac}^{\text{RRP}} \wedge \text{fnac}^{\text{RRP}})$$

be given. A *legal match for  $\rho^{\text{RRP}}$  with respect to  $PC_G$*  is a quasi-injective morphism  $m^{\text{RRP}} : L^{\text{RRP}} \rightarrow G$  with  $m_{\text{Im}}^{\text{RRP}} \models \text{nac}^{\text{RRP}}$ , where  $m_{\text{Im}}^{\text{RRP}}$  is the image-restriction of  $m^{\text{RRP}}$ , and  $m^{\text{RRP}} \models \text{fnac}$  such that the following conditions are satisfied (see Fig. 6.14):

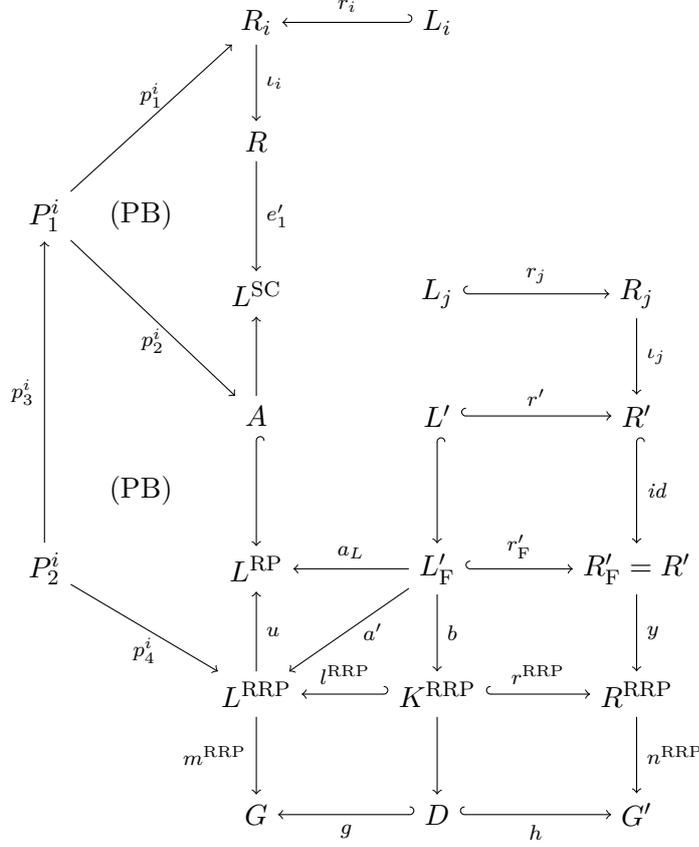


Figure 6.14: Legal match for a relaxed repair rule

- (1) *Availability of markings that are to be revoked:* Let  $L^{SC} \leftarrow A \hookrightarrow L^{RP}$  be the partial morphism from the LHS  $L^{SC}$  of the short-cut rule  $\rho^{-1} \bowtie_k \rho'$  to the LHS of its derived repair rule that exists according to Lemma 5.26. For every RHS  $R_i$  of a rule that contributes to the concurrent rule  $\rho$ , let  $P_2^i$  be computed via the two pullbacks depicted to the left in Fig. 6.14. Then, for every such  $R_i$ , if  $P_2^i \neq \emptyset$ , there exists an index  $1 \leq s \leq t$  such that  $R_i = R_s$ , where  $R_s$  is the underlying consistency pattern of  $\varphi_s$ , and the partial marking  $\varphi_s : R_s \dashrightarrow G$  is

isomorphic to the partial morphism

$$R_i \xleftarrow{p_1^i \circ p_3^i} P_2 \xrightarrow{m^{\text{RRP}} \circ p_4^i} G .$$

- (2) *Necessity of revocation:* When  $1 \leq i_0 \leq t$  is the index of the marking that corresponds to the first of the underlying rules of  $\rho$  (which exists according to the requirement above), then  $\varphi_{i_0}$  is invalid and every marking  $\varphi_j$  such that  $\varphi_j < \varphi_{i_0}$  is valid.<sup>6</sup>
- (3) *Absence of potentially cycle-inducing dependencies:* Let  $I_0 := \{i_0, \dots, i_z\}$  be the indexes of the partial markings  $\varphi_{i_0}, \dots, \varphi_{i_z}$  that correspond to the underlying rules of the concurrent rule  $\rho$  as guaranteed to exist by the above requirements. Then there do not exist markings  $\varphi_{i_u}$  and  $\varphi_{i_v}$  in  $I_0$  and a marking  $\varphi_q \in PC_G \setminus I_0$  such that  $\varphi_{i_u} < \varphi_q < \varphi_{i_v}$ .
- (4) *Safe repair:* The induced match  $m^{\text{RRP}} \circ a'$  for the forward rule  $\rho'^{\text{F}}$  of the concurrent rule  $\rho'$

- (1) matches every required element of  $\rho'^{\text{F}}$  to an element that is marked by one of the valid markings of  $PC_G$  (except for the ones whose match is already fixed by the one for  $\rho$ ). In particular, with

$$I_1 := \{1 \leq j \leq t \mid \text{there exists } i_u \in I_0 \text{ with } \varphi_{i_u} < \varphi_j\}$$

being the set of indexes of partial markings that (transitively) depend on the markings  $\varphi_{i_0}, \dots, \varphi_{i_z}$ , none of the “freely matchable” required elements of  $\rho'^{\text{F}}$  is matched to an element that is marked by one of the markings with index from  $I_1$ , i.e., we have that

$$m^{\text{RRP}}(a'(R'_F \setminus L'_F)) \cap \varphi(R_{i_u} \setminus r_{i_q}(L_{i_q})) = \emptyset$$

for every  $i_q \in I_1$  (compare Fig. 6.14 again).

---

<sup>6</sup>This point of the definition is the item in which we use that  $\rho$  is computed as a concurrent rule in a sequence of rule applications that (transitively) all depend on the first one (see Remark 6.3). To still control which markings remain valid after the application of a (relaxed) repair rule would require an even more complex definition if  $\rho$  were an arbitrary concurrent rule (of monotonic rules).

- (2) Moreover,  $m^{\text{RRP}} \circ a'$  matches every marking element of  $\rho^{\text{F}}$  that is *not* an element that the short-cut rule  $\rho^{-1} \times_k \rho'$  additionally preserves (i.e., any marking element that does not belong to the common kernel  $k$ ) to an element that is not already marked by  $PC_G$ .

Given the transformation

$$t : G \Rightarrow_{\rho^{\text{RRP}}, m^{\text{RRP}}}^{\text{SqPO}} G' ,$$

the partial cover

$$\left( PC_{G'}^{\text{RRP}} := PC_{G'}^t \setminus \{\varphi'_i \mid i \in I_0\} \right) \cup \{n^{\text{RRP}} \circ y \circ \iota_j\} ,$$

where  $j$  ranges over the rules contributing to  $\rho'$ , is called *repair-induced partial cover*.

As in the case of revocation rules, Corollary 5.29 ensures that a (relaxed) repair rule is always applicable at a legal match.

While the above definition is technical, the intuition behind it is not too difficult. The technicalities arise because (i) we consider concurrent rules as input to the repair rules and (ii) because we have to account for elements that have been deleted (either by the user edit or implicitly during the previous synchronization process). In the following example we illustrate and explain the individual conditions of the above definition.

**Example 6.7.** Condition (1) serves to ensure that there actually exist markings that correspond to a sequence of the applications of the underlying rules of  $\rho$ . The requirement that the pullbacks compute the partial morphisms of the markings ensures that we actually completely revoke the correspondence and target structure that has been created via  $\rho^{\text{F}}$  (except for the elements that are to be preserved via  $\rho'^{\text{F}}$ ). It ensures that we have to choose the largest possible relaxed repair rule; the exception for  $\emptyset$  allows us to skip rules that have already been revoked and whose former marking therefore has already been removed from the current partial cover. To illustrate this, consider the (relaxed) repair rules *CollapseHierarchyFwd* and *CollapseHierarchyFwdRel* as depicted in Fig. 6.10. We are only allowed to apply these repair rules in situations where indeed two subsequent

applications of *CreateSub* have created a hierarchy of **Packages** and **Folders**. Furthermore, by the kind of partial covers that actually occur in our synchronization processes, the condition to coincide with existing markings will also ensure that we cannot use *CollapseHierarchyFwdRel* when, for instance, an image for **Folder f2** still exists. We have to use a suitable larger repair rule then. Skipping of empty domains serves the following purpose: Assume a **Package** in a hierarchy of **Packages** being deleted that contains more than one **Package**, which are re-distributed to other **Packages**. This situation can completely be repaired with one application of *CollapseHierarchyFwd* followed by a suitable number of applications of *CollapseHierarchyFwdRel*. However, through the application of *CollapseHierarchyFwd*, the marking that matched the same elements as the correspondence node **p2f2**, **Folder f2**, and **Doc-File d2** (and the corresponding edges) is removed from the considered partial cover. Thus, when later applying *CollapseHierarchyFwdRel*, it is not available any longer, even if an according rule technically contributed to the computation of *CollapseHierarchyFwdRel* (it just has been completely removed in the relaxation process).

Condition (2) just ensures that there actually is a reason to revoke  $\rho$ . The first underlying marking of  $\rho$  being invalid and not being dependent on another invalid marking serves to ensure termination of our model synchronization process. The dependencies in a partial cover will dictate the order in which we repair or revoke markings and such repairs or revocations will never invalidate markings on which they depend. In our example from Sect. 2.2, if, for instance, the user had additionally prefixed the hierarchy of **Packages** with another **Package**, this would invalidate the first marking **CreateRoot** of the precedence graph from Fig. 2.4. Condition (2) now requires that we first had to fix that situation before we would be able to repair the dependent markings **CreateSub<sub>1</sub>** and **CreateSub<sub>2</sub>** using *CollapseHierarchyFwd*.

Condition (3) is a technical assumption that prevents the introduction of cyclic dependencies. When a match for the concurrent rule  $\rho$  “skips” an intermediate marking, i.e., in the situation  $\varphi_{i_u} < \varphi_q < \varphi_{i_v}$ , it would in principle be possible that a rule  $r_j$  that contributes to the repair rule uses the element of  $\varphi_q$  that causes the dependency for  $\varphi_{i_v}$  but preserves the element that causes the dependency of  $\varphi_q$  on  $\varphi_{i_u}$ . This would create a cyclic dependency between  $\varphi'_j$  and  $\varphi'_q$  in the repair-induced partial cover. With *CollapseHierarchyFwd* such a situation is not possible. In the concurrent

rule that serves as first input, all required elements of the second rule in that computation are elements that the first rule creates.

Condition (4) ensures that a marking-consistent, coherent, and internally-complete partial cover remains so; the requirements of this item correspond to the assumption of sequential independence in Theorem 4.16, which guarantees language-preserving applications of generalized concurrent rules. They also correspond to the additional visual annotations that we use in the rules: we annotate the required elements of  $\rho^F$  with  $\boxplus$  and the marking elements that stem from  $\rho^F$  (without being additionally preserved) with  $\square \rightarrow \boxplus$ . The annotation  $\dagger \rightarrow \boxplus$  is used for the marking elements that are additionally preserved. In our example, *CollapseHierarchyFwd* has no “free” required elements. The match for its required elements (Package p1, Folder f1, and the correspondence node in between) is already determined by the match for the rule application that is to be replaced. The same holds true for the other (relaxed) repair rules we saw so far. In example rules, which we later discuss in more detail, such elements occur. For instance, in the repair rule *MovePackageFwd* from Fig. 6.31, Package p1, Folder f1, and their correspondence node are required elements that are already required by the rule application that is to be revoked (and whose match is therefore fixed). Package p3, Folder f3, and their correspondence node, however, are required elements that are required by the replacing rule. For these elements, the restrictions apply, i.e., they have to be matched to elements that are already marked by a valid marking. The *subPackages*-edge in *CollapseHierarchyFwd* is a “free” marking element, i.e., not a marking element that is additionally preserved. It is only allowed to be matched to a yet unmarked edge of that type.

**Lemma 6.14** (Preservation of consistency conditions for repair-induced partial covers). *Let a partial triple graph  $G$ , a partial cover  $PC_G$  of  $G$ , and a (relaxed) repair rule  $\rho^{\text{RP}} = (L^{\text{RP}} \leftarrow K^{\text{RP}} \hookrightarrow R^{\text{RP}}, \text{nac}^{\text{RP}} \wedge \text{fnac}^{\text{RP}})$  be given. Let*

$$t : G \Rightarrow_{\rho^{\text{RP}}, m^{\text{RP}}}^{\text{SqPO}} G'$$

*be a transformation step where  $m^{\text{RP}}$  is a match that is legal with respect to  $PC_G$ . Then, if  $PC_G$  is NAC-consistent, filter-NAC-consistent, coherent, or internally complete, so is the repair-induced partial cover  $PC_{G'}^{\text{RP}}$ . If  $PC_G$  is marking-consistent and internally complete, also  $PC_{G'}^{\text{RP}}$  is marking-consistent (and internally complete).*

Moreover, for coherent  $PC_G$ , when  $1 \leq i_0 \leq t$  is the index of the invalid marking  $\varphi_{i_0}$  that exists according to Condition (2) in the definition of a legal match for a (relaxed) repair rule, there is a topological sorting

$$\{\varphi'_1, \dots, \varphi'_{i_0-1}, \dots, n^{\text{RP}} \circ y \circ \iota_j, \dots\}$$

of  $PC_{G'}^{\text{RP}}$ , where  $j$  ranges over the rules contributing to  $\rho'$ , and the first invalid marking, if any, appears after the markings  $n^{\text{RP}} \circ y \circ \iota_j$  (in case  $i_0 = 1$ , we interpret  $\{\varphi'_1, \dots, \varphi'_{i_0-1}\}$  as the empty set).

## 6.4 Information-Preserving Model Synchronization—Our Algorithm

In this section, we present our approach to incremental bidirectional model synchronization. We first present our actual algorithm and then discuss its central formal properties.

### 6.4.1 The Basic Setup

We assume an integrity-preserving TGG  $GG$  to be given (Definition 6.2). This means, the rules of  $GG$  are monotonic, might be equipped with NACs that are derived from a given set of negative constraints, and are allowed to set attribute values of newly created elements. Note that formally, changing an attribute value of an existing element involves the deletion of the old value. Hence, it cannot be performed with a monotonic rule anyhow. The language of the TGG defines consistency, i.e., a triple graph  $G = (G_S \leftarrow G_C \rightarrow G_T)$  is consistent if and only if  $G \in \mathcal{L}(GG)$ .

**The problem** We assume a consistent triple graph  $G = (G_S \leftarrow G_C \rightarrow G_T) \in \mathcal{L}(GG)$  to be given; by Lemma 6.8 and Theorem 6.9 there exists a precedence graph  $PG_G$  for  $G$ . Strictly speaking, there might be several different ones; we assume to know one of them. Furthermore, we assume that a user somehow edited the source graph of  $G$ , i.e., changed the graph  $G_S$  resulting in a graph  $H_S$ . We do not assume anything about how this edit was performed. We merely assume that the edit can be expressed as a span of morphisms and that we know the derived source-delta  $\delta = (G \xleftarrow{\text{del}} D \xrightarrow{\text{add}} G')$  (cf. Definition 6.3), where  $G' = (H_S \leftarrow G_C \rightarrow G_T)$ .

Generally, the result  $G'$  is a partial triple graph and does not belong to  $\mathcal{L}(GG)$ . The derived source-delta  $\delta$  provides a delta-induced partial cover  $PC_{G'}^\delta$  for  $G'$  (cf. Definition 6.17). Our goal is to provide a *model synchronization algorithm* that, given the partial triple graph  $G'$  and its delta-induced partial cover  $PC_{G'}^\delta$  as input, computes a triple graph  $H = (H_S \leftarrow H_C \rightarrow H_T) \in \mathcal{L}(GG)$ . As a side condition, we want to minimize the amount of elements of  $G_C$  and  $G_T$  that are deleted and recreated during that synchronization.

**Ingredients of our algorithm** We provide a rule-based model synchronization algorithm. During that algorithm, rules are applied to transform the partial triple graph  $G' = (H_S \leftarrow G_C \rightarrow G_T)$  to a triple graph  $H = (H_S \leftarrow H_C \rightarrow H_T) \in \mathcal{L}(GG)$ . We apply the three different kinds of synchronization operations introduced in Sect. 6.2.3, which can be derived from  $GG$ , namely

- (1) forward rules (cf. Definition 6.7),
- (2) (relaxed) revocation rules, i.e., inverses of forward rules (cf. Definition 6.8), and
- (3) (relaxed) repair rules, i.e., operationalized short-cut rules (cf. Definition 6.9).

Their individual purposes have already been explained in Sect. 6.2.3.

During the synchronization process, the synchronization operations are applied reacting to the state of a partial cover of the currently given partial triple graph (compare Fig. 6.1). In particular, its invalid markings and the unmarked elements of the source graph inform about locations where changes of the source graph still need to be propagated to the target graph. In the implementation of our approach as provided by Lars Fritsche [79, 80, 77, 76], the update of partial covers and computation of (at least some of) the legal matches is performed based on an *incremental pattern matcher*. In the presentation of our approach we therefore sometimes use terminology that assumes such an incremental pattern matcher to be given. This is a terminological choice however; it certainly suggests itself to employ incremental pattern matching as a technological basis for our approach but it is not to necessary do so. Other technological means to provide the

required information could be used as well. For a general discussion of the use of incremental pattern matching in TGG-based model synchronization we refer to [150, 151].

For our approach, we require an incremental pattern matcher (or some other technological basis) to provide the following information:

- (1) Given a (partial) cover  $PC_G$  for a (partial) triple graph  $G$ , the incremental pattern matcher needs to be able to determine invalid markings (if there are any).
- (2) The incremental pattern matcher needs to be able to determine legal matches for forward, (relaxed) revocation, and (relaxed) repair rules.
- (3) Upon the application of one of these kinds of rules (at a legal match), the incremental pattern matcher needs to provide the information that is necessary to determine the accordingly induced partial cover.

In the implementation of our approach though [79, 80, 77], instead of determining legal matches for repair rules with the incremental pattern matcher, we developed a dedicated pattern matcher for them. It is called whenever a match for a repair rule is needed. In this way, the incremental pattern matcher is more efficient as it has to keep track of less rules. For presentation purposes, however, it seems more economical to treat all kinds of rules in a uniform manner.

#### 6.4.2 The Synchronization Process

In this section, we formally present our algorithm for TGG-based model synchronization. Generally, our synchronization process applies forward rules to translate elements and, before revoking an invalid rule application, tries to possibly replace it through a suitable repair rule application. In that, it extends and refines the synchronization process suggested in [151, 150]. There, invalid rule applications are revoked as long as there exist any. Subsequently, forward rules are applied as long as possible. By trying to apply a suitable repair rule instead of revoking an invalid rule application, we are able to avoid deletion and recreation of elements. Note that we present an algorithm for synchronizing in forward direction (from source to target) while synchronizing backwards is performed analogously. For presentation purposes, namely, to not hide too many details behind the workings of

an incremental pattern matcher, we present certain computations as a sequence of instructions that such an incremental pattern matcher can perform concurrently.

---

**Algorithm 6.1** Structure of our synchronization process

---

```

1: function SYNCHRONIZE(tripleGraph, partialCover)
2:   (invalidMrks, unmarkedElts) ← validate(partialCover)
3:   fwdMatches ← updateMatches(tripleGraph)
4:
5:   if isFinished(invalidMrks, unmarkedElts, fwdMatches) then
6:     return
7:   end if
8:
9:   if translate(tripleGraph, partialCover, fwdMatches) then
10:    return
11:  end if
12:
13:  (invalidMrk, suc) ← repair(tripleGraph, partialCover, invalidMrks)
14:  if !suc then
15:    revoke(tripleGraph, partialCover, invalidMrk)
16:    return
17:  end if
18:  return
19: end function

```

---

Algorithm 6.1 presents the general structure of the algorithm, Algorithm 6.2 comprises the check of our termination condition, and Algorithms 6.3 to 6.5 detail the central methods, namely the concrete ways in which the three kinds of synchronization operations are applied. At the beginning, function SYNCHRONIZE is called on the current partial triple graph that is to be synchronized (Algorithm 6.1, line 1). The second input for the function is a partial cover for the partial triple graph. The assumption is that the partial cover is a precedence graph for the original triple graph whose source graph has been edited (and which therefore in general only constitutes a partial cover for the input partial triple graph). The algorithm begins with validating the given partial cover for the given partial triple graph, which means that the markings of the partial cover are

checked for validity (with the invalid markings being stored in `invalidMrks`), unmarked (i.e., newly added source elements) are collected (being stored in `unmarkedElts`), and new matches for forward rules are determined.<sup>7</sup>

---

**Algorithm 6.2** Synchronization process—termination criteria

---

```

1: function ISFINISHED(invalidMrks, unmarkedElts, fwdMatches)
2:   if isEmpty(invalidMrks) && isEmpty(fwdMatches) then
3:     if isEmpty(unmarkedElts) then
4:       return true
5:     else
6:       throw InconsistentStateException
7:     end if
8:   end if
9:   return false
10: end function

```

---

By calling the function `ISFINISHED` (Algorithm 6.1, line 5), termination criteria for the synchronization algorithm are checked. If the sets of invalid markings and of forward TGG rule matches are both empty and all elements of the source graph are marked as translated, the synchronization algorithm terminates (see Algorithm 6.2 and Algorithm 6.1, line 6). Yet, if both those sets are empty but there are still untranslated elements in the source graph, an exception is thrown in Algorithm 6.2, line 6, signaling that the (partial) triple graph is in an inconsistent state but that synchronization cannot continue.

Subsequently, function `TRANSLATE` is used (see Algorithm 6.1, line 9 and Algorithm 6.3) to propagate the creation of elements: If the set of legal forward TGG rule matches is non-empty (Algorithm 6.3, line 2), we (randomly) choose one of these matches, apply the corresponding rule, and continue the synchronization process with the partial triple graph and the partial cover that result from that transformation. In particular, the partial cover returned by `APPLYFWDRULE` is understood to be the

---

<sup>7</sup>This is one situation where, for clarity, we opted for a sequential description while in fact, an incremental pattern matcher continuously monitors the current partial triple graph, i.e., automatically updates the sets of available matches for forward rules and currently invalid markings at every change. It can also easily be extended to continuously keep track of the yet unmarked elements.

**Algorithm 6.3** Synchronization process—function TRANSLATE

---

```

1: function TRANSLATE(tripleGraph, partialCover, fwdMatches)
2:   if !isEmpty(fwdMatches) then
3:     fwdMatch ← chooseMatch(fwdMatches)
4:     (tripleGraph, partialCover) ← applyFWDRule(
       tripleGraph,
       partialCover,
       fwdMatch,
       getFWDRule(fwdMatch))
5:     synchronize(tripleGraph, partialCover)
6:     return true
7:   end if
8:   return false
9: end function

```

---

*forward-induced partial cover* as introduced in Definition 6.19. This step is done prior to any repair. The purpose is to create the context which may be needed to make repair rules applicable.

If the overall synchronization process does not yet terminate but the set of forward matches is empty, there must be at least one invalid marking whose corresponding rule application has to be repaired or revoked. We first try to REPAIR (Algorithm 6.1, line 13): We take the first invalid marking (Algorithm 6.4, line 2), where we assume that the set of invalid markings is topologically sorted, and apply a suitable (relaxed) repair rule that replaces the invalid marking if possible. As in the case of forward rules, if there are several possibilities, we randomly choose one of them. And, again, the synchronization process is recursively called for the resulting partial triple graph and partial cover; here, the partial cover is understood to be the *repair-induced partial cover* as introduced in Definition 6.21.

If there is no legal match for a suitable repair rule, however, the synchronization process proceeds by revoking the invalid marking (see Algorithm 6.5 and Algorithm 6.1, line 15). This means, the (unique) (relaxed) repair rule induced by the invalid marking is applied at its legal match. Again, the synchronization process is recursively called thereafter for the resulting partial triple graph and partial cover (Algorithm 6.5, line 4); here, the partial cover is understood to be the *revocation-induced partial cover* as

---

**Algorithm 6.4** Synchronization process—function REPAIR

---

```

1: function REPAIR(tripleGraph, partialCover, invalidMrks)
2:   invalidMrk ← first(invalidMrks)
3:   repairMatch ← getLegalRepairMatch(invalidMrk)
4:   if !isVoid(repairMatch) then
5:     (tripleGraph, partialCover) ← applyRepairRule(
       tripleGraph,
       partialCover,
       repairMatch,
       getRepairRule(repairMatch))
6:     synchronize(tripleGraph, partialCover)
7:     return (invalidMrk, true)
8:   end if
9:   return (invalidMrk, false)
10: end function

```

---



---

**Algorithm 6.5** Synchronization process—function REVOKE

---

```

1: function REVOKE(tripleGraph, partialCover, invalidMrk)
2:   revokeMatch ← getLegalRevocationMatch(invalidMrk)
3:   (tripleGraph, partialCover) ← applyRevocationRule(
       tripleGraph,
       partialCover,
       revokeMatch,
       getRevocationRule(revokeMatch))
4:   synchronize(tripleGraph, partialCover)
5:   return
6: end function

```

---

introduced in Definition 6.20.

### 6.4.3 Example Run of the Algorithm

In this section, we illustrate how our algorithm works in more detail. The example extends the one from Sect. 2.2.

First, Fig. 6.15 shows the original triple graph that shall be edited; Fig. 6.16 displays its corresponding precedence graph. The triple graph is very similar to the one in Fig. 2.3; only the Package  $sP$  contains two Packages and the Package  $ssP'$  contains two Classes. For brevity, we abbreviate the names of the nodes as well as of the types and omit the types of the edges and correspondence nodes completely. They can be inferred. In the precedence graph, we name the nodes after the name of their underlying rule. The number  $i$  in the index shares the number with the value of the `name`-attribute of the elements that the rule application created. For instance,  $CreateSub_4$  designates the application of the rule  $CreateSub$  that created the Package  $ssP'$  (with name "n4") and the related elements in correspondence and target graph.

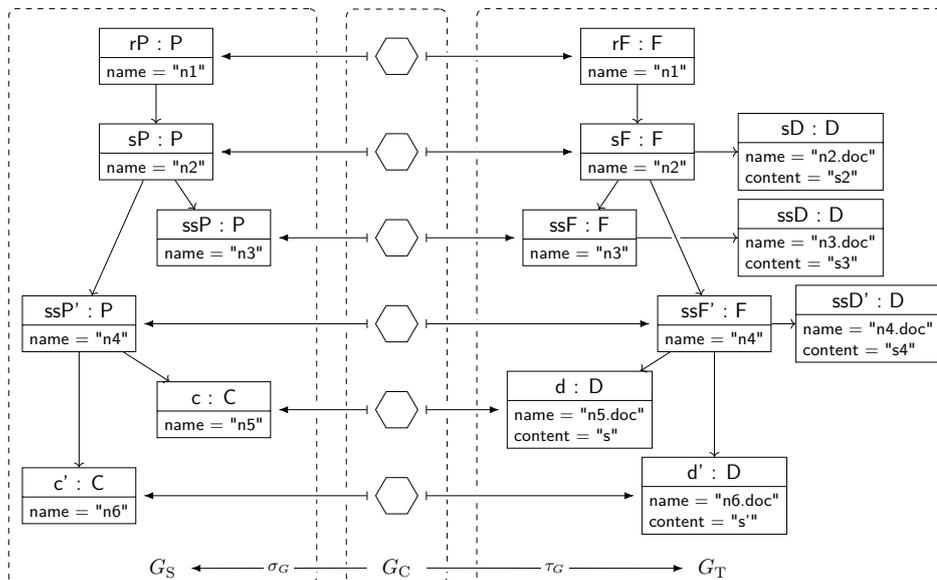


Figure 6.15: Original triple graph

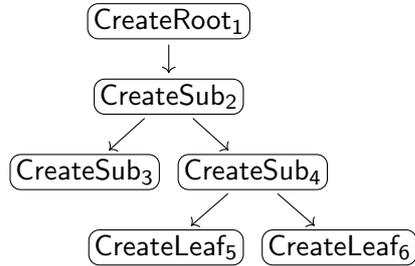


Figure 6.16: Precedence graph for the triple graph from Fig. 6.15

Figures 6.17 and 6.18 display an edit of the source graph and the resulting induced partial cover. The user created a new Package  $rP'$  as root and moved the former root-Package  $rP$  into it as well as the Package  $ssP'$ . The Package  $sP$  is deleted and Package  $ssP$  moved to Package  $rP$  instead. Furthermore, the user created a new Class  $c'$  in Package  $ssP$  and changed the name of Class  $c'$  (from "n6" to "m6"). Correspondence graph  $G_C$  and target graph  $G_T$  remain unchanged; the induced correspondence morphism  $\sigma_0 : G_C \dashrightarrow H_S$  becomes partial. In the induced partial cover, most markings become invalid by that edit; we signify this using red color. First,  $CreateRoot_1$  becomes invalid because its filter-NAC is violated by the newly created incoming edge. All three markings that signify applications of the rule  $CreateSub$  become invalid because underlying elements have been deleted such that they now merely represent partial morphisms.  $CreateLeaf_6$  is invalid for the same reason; the missing element is the attribution edge that the user deleted to change the value of the `name`-attribute. Only  $CreateLeaf_5$  is not invalid; however, it is not valid either because it depends on invalid markings. We signify this by the dashed border of the node. For better orientation, we also annotate the elements in the source graph accordingly. Source elements that are not in the image of the partial cover, i.e., elements that are unmarked, are denoted with  $\square$ . In contrast,  $\checkmark$  signifies that the element is marked by a valid marking. Finally,  $\cancel{\checkmark}$  signifies that the element is marked by an invalid marking. Elements that are not annotated are elements that are marked by a marking that is neither valid nor invalid.

In Fig. 6.17, for instance, all newly added elements are annotated with  $\square$ ; naturally they are outside of the image of the partial cover. The edge from  $ssP'$  to  $c'$ , for example, is annotated with  $\cancel{\checkmark}$ . It is marked, but its

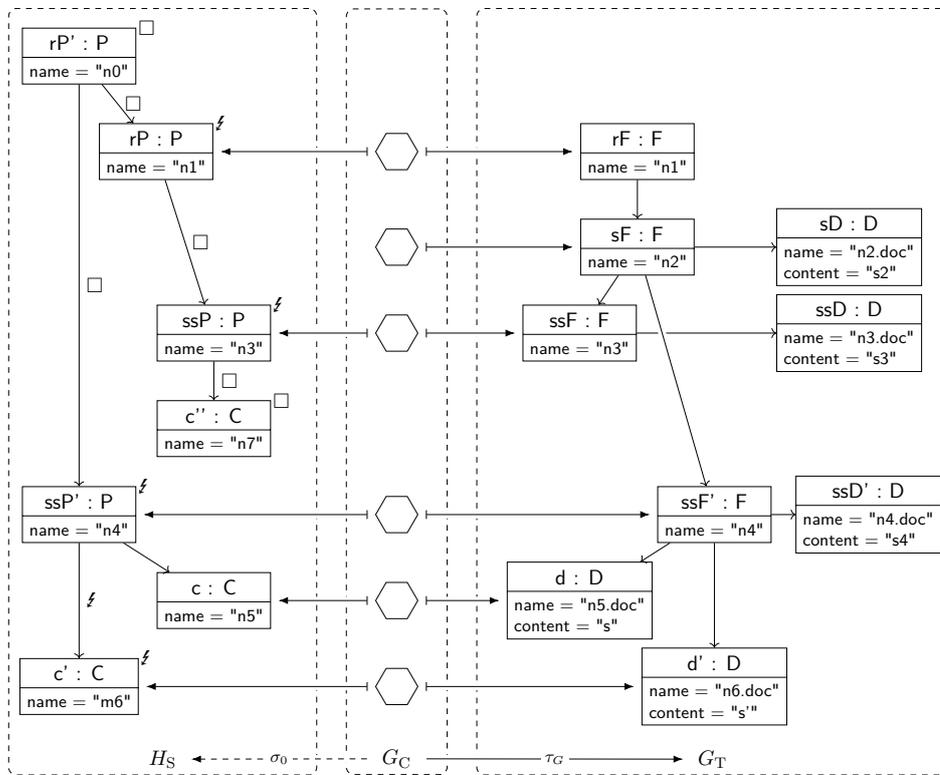


Figure 6.17: Source edit of the original triple graph

marking has become invalid by the edit.

Our algorithm starts with applying forward rules as long as possible. In principle, there are two matches for forward rules in the partial triple graph displayed in Fig. 6.17: one for *CreateRootFwd* (at Package  $rP'$ ) and one for *CreateLeafFwd* (at Class  $c''$ ). The second match, however, is not legal: the marking that marks Package  $ssP$  is invalid. Therefore, only *CreateRootFwd* is applied, translating the newly added Package  $rP'$ . The result of this application and its forward induced partial cover are shown in Figs. 6.19 and 6.20.

Because the algorithm is not finished (there are invalid markings as well as untranslated elements) but no legal forward matches are available, the algorithm tries to repair the first invalid marking (where “first” refers to

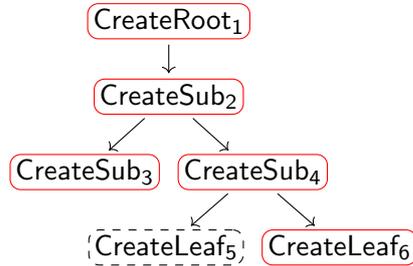


Figure 6.18: Induced partial cover after the edit

some topological sorting). In our example, this is the marking  $\text{CreateRoot}_1$  in every possible such sorting. Indeed, a suitable repair rule (that revokes the application of  $\text{CreateRoot}$  and replaces it by one of  $\text{CreateSub}$  while preserving  $\text{Package } rP$  and its corresponding  $\text{Folder } rF$ ) with a legal match is available; the repair rule is basically an inverse to the repair rule  $\text{MakeRootFwd}$  in Fig. 2.10. Applying it connects  $\text{Folder } rF'$  to  $\text{Folder } rF$  and creates the now necessary  $\text{Doc-File } rD$ . For this, we need to assume that a user provides a value for the  $\text{content}$ -attribute of the  $\text{Doc-File}$  or that some default value exists. In the repair induced partial cover, the marking  $\text{CreateRoot}_1$  is replaced by  $\text{CreateSub}_1$ , which, moreover, becomes dependent on  $\text{CreateRoot}_0$ . Both partial triple graph and partial cover are shown in Figs. 6.21 and 6.22.

Still, the algorithm does not terminate but no new legal match for a forward rule became available. Therefore, the algorithm again tries to repair the next invalid marking, the marking  $\text{CreateSub}_2$ . Here, depending on the kind of derived repair rules, several situations are possible. If one only constructs repair rules that stem directly from rules of the grammar, it might be that no rule to repair that marking is available. After all, all source-elements that the underlying rule created have been deleted by the edit. In that case, marking  $\text{CreateSub}_2$  had to be revoked. But if repair rules have also been derived from concurrent rules of the grammar, rule  $\text{CollapseHierarchyFwd}$  might be available; we assume this to be the case. For this rule, there are two legal matches in the graph that is depicted in Fig. 6.21: the  $\text{Package } p3$  from the rule can either be matched to  $\text{Package } ssP$  or to  $\text{Package } ssP'$  (and  $\text{Folder}$  and  $\text{Doc-File}$  on the target side accordingly). The match is chosen randomly; for this example, we assume

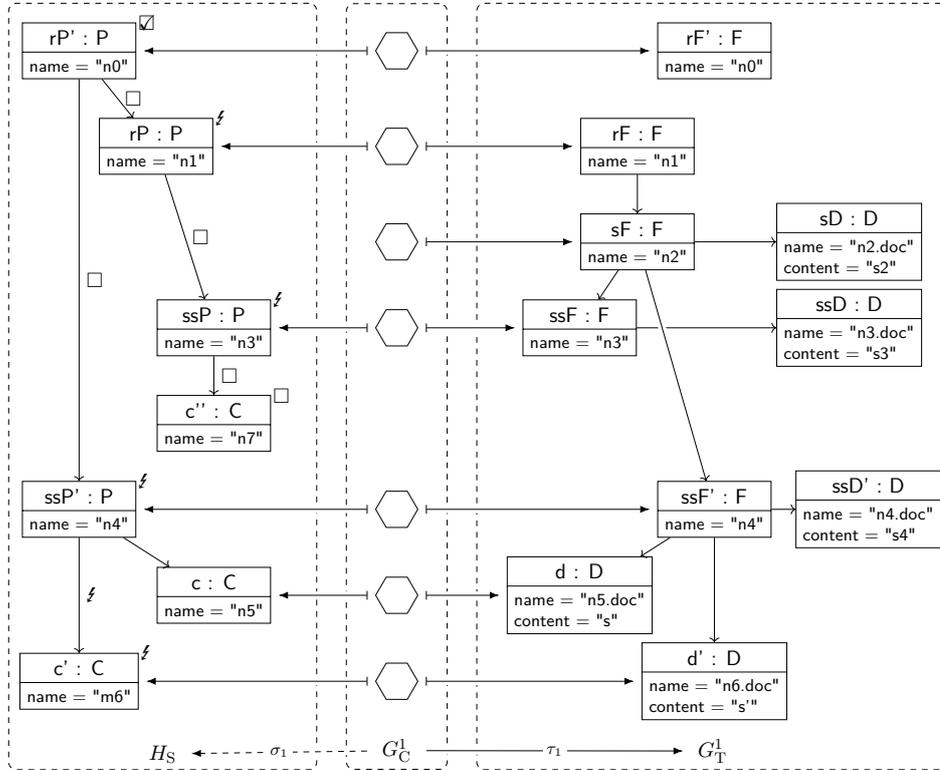


Figure 6.19: Partial triple graph after first repair step

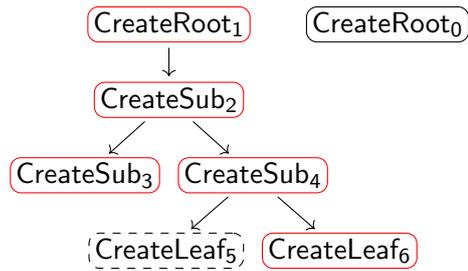


Figure 6.20: Forward-induced partial cover of first repair step

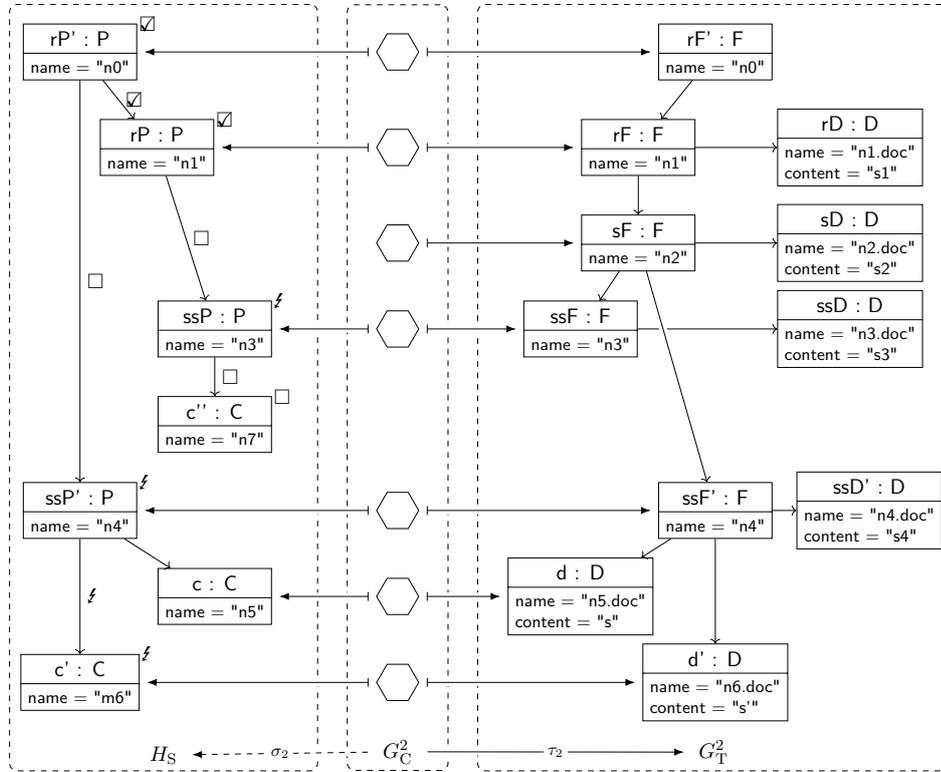


Figure 6.21: Partial triple graph after second repair step

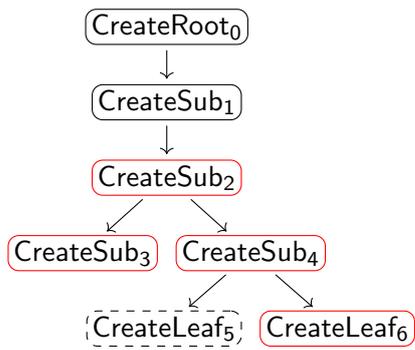


Figure 6.22: Repair-induced partial cover after second repair step

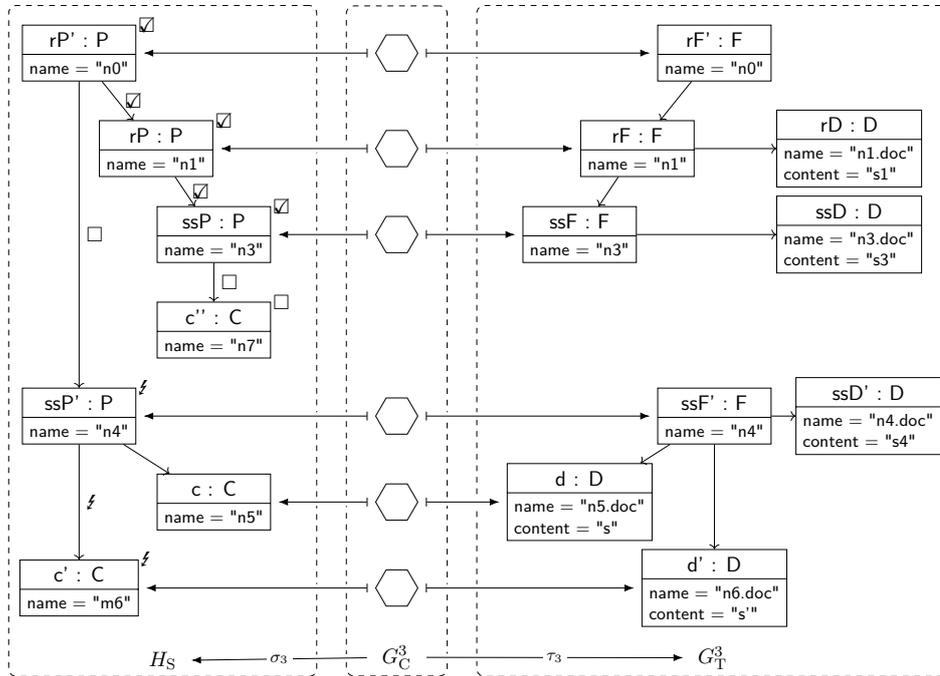


Figure 6.23: Triple graph after third repair step

Package `ssP` to be matched. The triple graph resulting from applying *CollapseHierarchyFwd* at that match is shown in Fig. 6.23; Fig. 6.24 shows the repair-induced partial cover. Folder `sF` and its Doc-File are deleted and Folder `ssF` is connected to the Folder `rF` instead. Because repair rules are applied according to the sesqui-pushout semantics, the application also (implicitly) deletes the edge that connected Folder `sF` to Folder `ssF'`. On the repair-induced partial cover that has the following effects: Marking `CreateSub2` is completely removed (because the repair rule actually completely revokes that marking) and `CreateSub3` is replaced by another marking `CreateSub3`, namely one that depends on `CreateSub1`. Furthermore, by the implicit deletions, the marking `CreateSub4` does not depend on any other marking any longer. After that repair, the partial triple graph is even a triple graph again.

When the algorithm continues, the match for *CreateLeafFwd* that translates Class `c''` finally became legal (because marking `CreateSub3` is valid

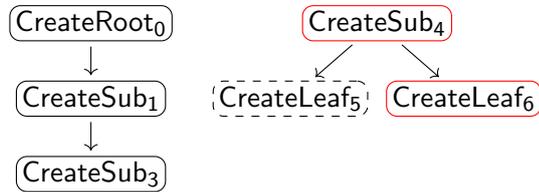


Figure 6.24: Repair-induced partial cover after third repair step

now). Thus, that rule is applied and the corresponding marking `CreateLeaf7` is appended in the forward-induced partial cover; the results are depicted in Figs. 6.25 and 6.26.

In the following iteration of the algorithm, the invalid marking `CreateSub4` is tried to be repaired. This can be done using the (relaxed) repair rule *CollapseHierarchyFwdRel* (Fig. 6.10) that covers the situation where the incoming `subFolders`-edge of the to-be-preserved `Folder` has already (implicitly) been deleted. Applying that rule at its unique legal match results in the graph that is depicted in Fig. 6.27; the repair-induced partial cover of that application is shown in Fig. 6.28. The `Folder ssF'` gets connected to the `Folder rF'`, and, in the partial cover, the invalid marking `CreateSub4` is replaced by a valid one that depends on `CreateRoot0`. Furthermore, the marking `CreateLeaf5` becomes valid because the marking it depends on became valid.

In its final iteration, the algorithm has to try to repair the invalid marking `CreateLeaf6`. This can be done using a repair rule that merely propagates the change of the `name`-attribute to the target side (i.e., by a repair rule similar to the rule *ChangeAttributeRootFwd* from Fig. 2.10). The resulting graph and its partial cover are shown in Figs. 6.29 and 6.30. The algorithm terminates because no legal matches for forward rules are available any longer and all markings are valid. Because all source-elements are marked, the algorithm returns the computed triple graph (and does not need to throw an exception). The result is a triple graph that belongs to the given language and the final partial cover is actually a precedence graph for it. Importantly, even though the considered source edit is not trivial, the number of rule applications that have been used in the synchronization process depended on the size of the edit and not on the size of the original triple graph. This becomes even more apparent when one recognizes that

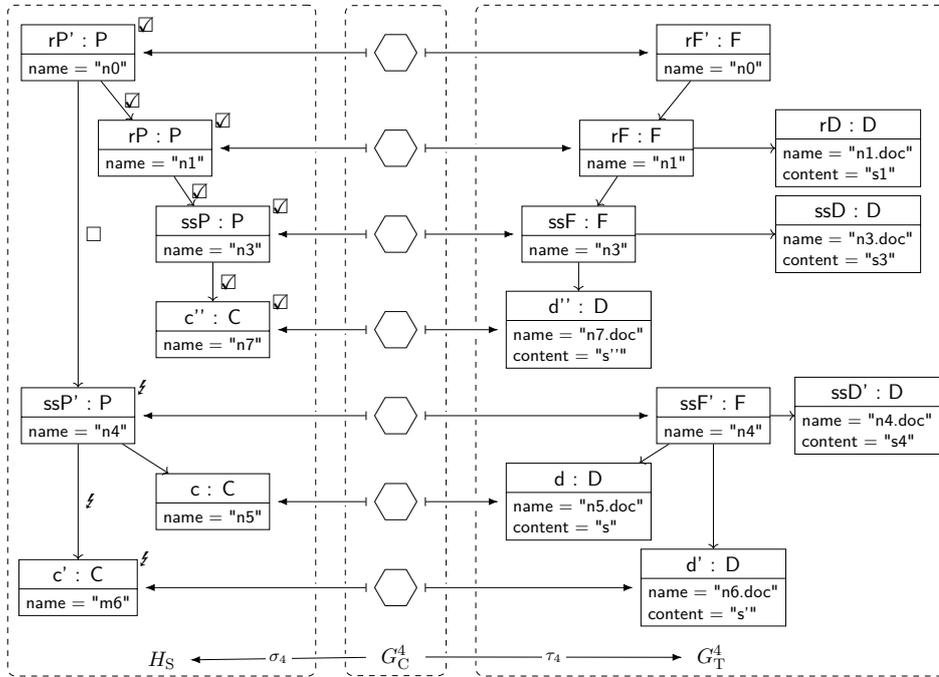


Figure 6.25: Triple graph after fourth repair step

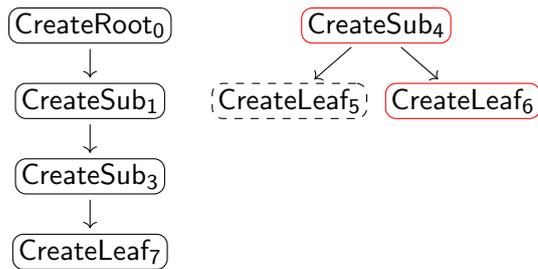


Figure 6.26: Forward-induced partial cover after fourth repair step

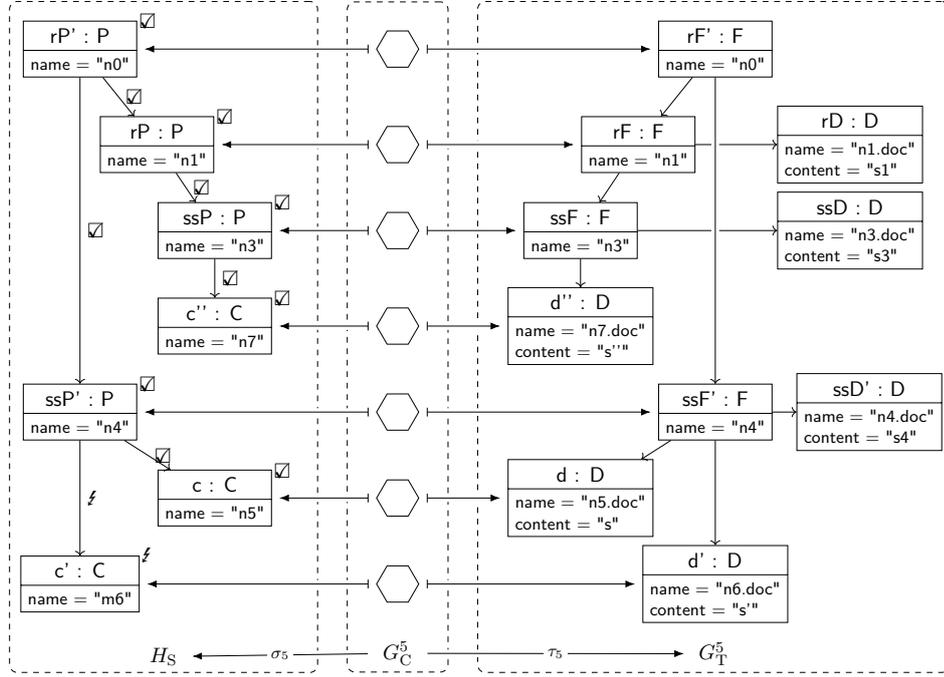


Figure 6.27: Triple graph after fifth repair step

the synchronization process would remain exactly the same if in the original model there were arbitrary long hierarchies of consistent pairs of Packages and Folders starting at Packages  $rP$ ,  $ssP$ , and/or  $ssP'$ .

We close this section with a short discussion on the availability of repair rules. The (relaxed) repair rule *CollapseHierarchyFwdRel* (Fig. 6.10) can be derived in more than one way. In Example 6.4, we described its derivation as relaxed repair rule from the repair rule *CollapseHierarchyFwd*, where the first input rule is actually a concurrent rule. Alternatively, that rule could be derived as relaxed repair rule from a repair rule *MovePackageFwd*. The rule is depicted in Fig. 6.31; for brevity, we omit all attributes. We used this rule as an important example in [80]. Its underlying short-cut rule (or GCR) is computed from *CreateSub* as first and second input (with common kernel preserving the Package  $p2$  and its corresponding Folder  $f2$ ). In particular, *CollapseHierarchyFwdRel* can be derived even in situations where one does not derive repair rules with concurrent rules as input. This

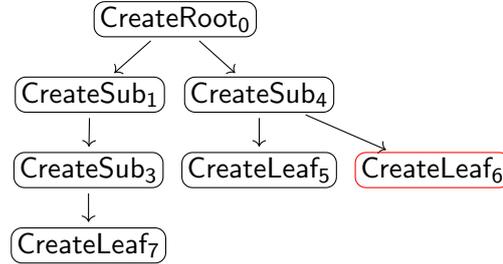


Figure 6.28: Repair-induced partial cover after fifth repair step

means, in our example run of the algorithm, even if *CollapseHierarchyFwd* is not available and *CreateSub<sub>2</sub>* has to be revoked, the two invalid markings *CreateSub<sub>3</sub>* and *CreateSub<sub>4</sub>* can still both be repaired. Instead of using *CollapseHierarchyFwd* and *CollapseHierarchyFwdRel* once, we revoke *CreateSub<sub>2</sub>* and use *CollapseHierarchyFwdRel* twice. The resulting triple graph is exactly the same and the same elements are preserved in both variants.<sup>8</sup> This illustrates an important open problem for our approach: For efficiency reasons, it would be desirable to be able to derive minimal sets of repair rules without losing expressivity. Finding such sets is a topic for future research.

#### 6.4.4 Central Formal Properties

In this section, we present important formal properties of our algorithm. We discuss termination, correctness, completeness, runtime, incrementality, and information loss. We start with introducing and discussing the preconditions under which our formal results will hold.

First, we introduce *source-progressiveness* [150, Definition 36], meaning that every forward rule has at least one marking element (or, equivalently, that every rule from the given TGG creates at least one source element). In our application, this will guarantee a finite number of legal matches for forward rules.

**Definition 6.22** (Source-progressive TGG). A forward rule  $p^F$  of a TGG is called *source-progressive* if it has at least one marking source element.

<sup>8</sup>The deeper reason for this is that both procedures can be understood as different decompositions of the same *amalgamated rule* into a subrule and complement rule(s).

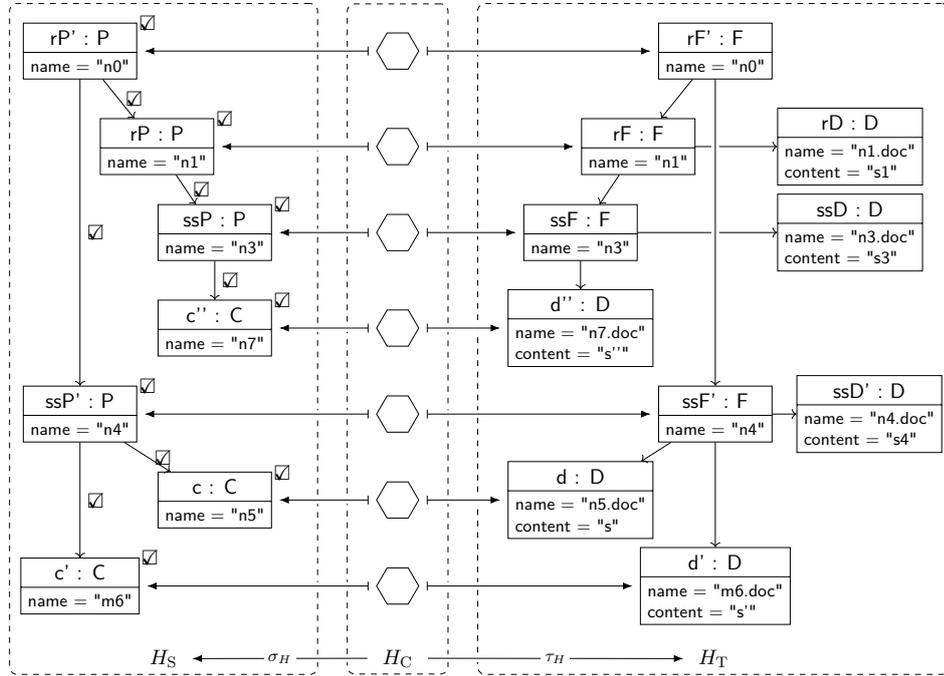


Figure 6.29: Resulting triple graph after termination of algorithm

An (integrity-preserving) TGG  $GG$  is *source-progressive* if all of its derived forward rules are.

The requirement on a TGG to be source-progressive has repeatedly been used to be able to guarantee the termination of a synchronization procedure [86, 150, 54]. Ehrig et al. provide a static analysis technique to check whether, when reducing a TGG to its source-progressive part (by omitting the rules that do not have a marking element), the model synchronization process remains complete [54, Sect. 8 and Theorem 9.25]. They check if any of the remaining rules is sequentially dependent on one of the omitted ones. If that is not the case, the forward rules which are not source-progressive can be safely omitted. The same kind of analysis can be used for our synchronization process.

The second precondition, *marking-completeness*, ensures that the synchronization process cannot get stuck because of applying rules in the

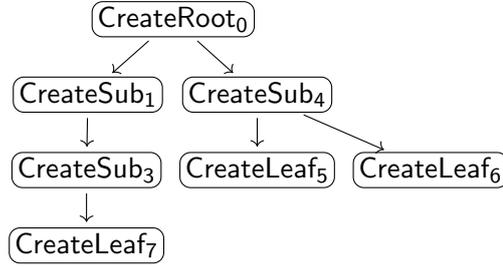


Figure 6.30: Repair-induced partial cover after sixth repair step. The partial cover is even a precedence graph for the triple graph  $H$  from Fig. 6.29.

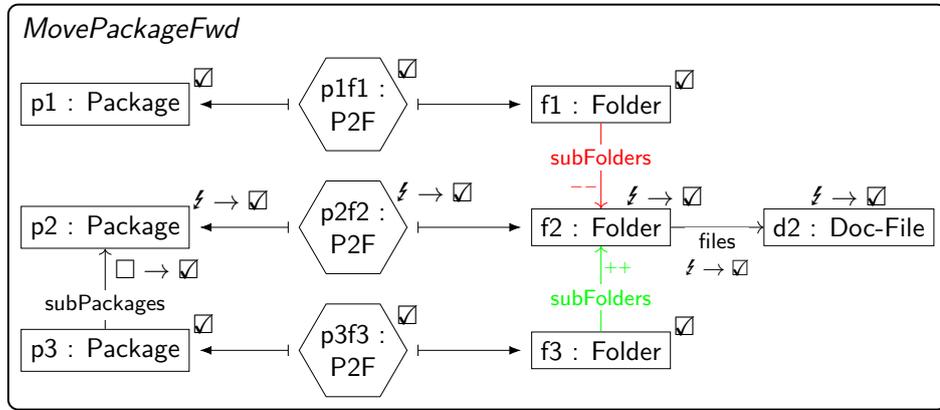


Figure 6.31: The repair rule *MovePackageFwd*

wrong order. It requires that partial precedence graphs that are not already precedence graphs can always be extended with further valid markings that correspond to forward rule applications at legal matches. Marking-completeness has been introduced by Leblebici [150, Definition 38] and we restate it in our terminology.

**Definition 6.23** (Marking-complete forward rules). The set of forward rules of an (integrity-preserving) TGG  $GG$  is *marking-complete* if for every partial triple graph  $G' = (H_S \leftarrow G'_C \rightarrow G'_T)$  where  $H_S$  belongs to the source language of  $GG$  (i.e., where  $H_S \in \mathcal{L}^S(GG)$ ) and every partial precedence graph  $PG_{G'}$  for  $G'$ , with  $\text{Im } PG_{G'} = (U_S \xleftarrow{\sigma_U} U_C \xrightarrow{\tau_U} U_T) \in \mathcal{L}(GG)$  being the image of  $PG_{G'}$ , whenever  $H_S$  is not isomorphic to  $U_S$ ,

there exists a legal match for one of the forward rules of  $GG$  in  $H_S \xleftarrow{i_{PG_{G'}}^S \circ \sigma_U} U_C \xrightarrow{\tau_U} U_T$ , the partial synchronization graph represented by  $PG_{G'}$ .

Leblebici presented the above definition in terms of forward transformation sequences. Because certain kinds of forward transformation sequences correspond to consistent triple graphs [150, Lemma 4], as do partial precedence graphs (Theorem 6.9), our definition is equivalent to the one of Leblebici. He furthermore discusses how *confluence* as well as *local-completeness* of the forward rules of the given TGG both imply marking-completeness.

Before proving termination, completeness, and correctness of our algorithm, we first present an important invariant of it that serves as basis at least for the proofs of completeness and correctness. Namely, the subset of valid markings of the partial cover always even constitutes a partial precedence graph, i.e., its image is a triple graph from the language of the given TGG.

**Proposition 6.15** (Valid markings as partial precedence graph). *Let  $GG$  be an integrity-preserving TGG  $GG$ ,  $G = (G_S \leftarrow G_C \rightarrow G_T) \in \mathcal{L}(GG)$  some consistent triple graph with a precedence graph  $PG_G$  for it, and  $\delta = (G \leftarrow D \rightarrow G')$  a source-delta, where  $G' = (H_S \leftarrow G_C \rightarrow G_T)$  and  $PC_{G'}^\delta$  is the delta-induced partial cover. Then, when calling the synchronization process with  $G', PC_{G'}^\delta$  as input, the subset of valid markings of the current partial cover remains a partial precedence graph throughout the algorithm.*

*Proof.* First, if the subset is empty, i.e., if every marking is (potentially) invalid, the assertion is trivially true (with the empty partial precedence graph representing the empty start graph). Therefore, in the following, we always assume the set to not be empty.

We have to show that the subset of valid markings is marking-consistent, NAC-consistent, coherent, internally complete, and that every such marking is total (compare Definition 6.14). By definition of validity (cf. Definition 6.16), the subset of valid markings consists of total markings. Furthermore, by definition of a precedence graph,  $PG_G$  is marking-consistent, NAC-consistent, coherent, and, in particular, internally complete (as it is even complete). These are properties that are preserved throughout the synchronization process because they are preserved by delta-induced,

forward-induced, revocation-induced and repair-induced partial covers (cf. Lemmas 6.10, 6.11, 6.13, and 6.14). Thus, at every time of the synchronization process, the currently given partial cover is marking-consistent, NAC-consistent, coherent, and internally complete. As a subset, the set of currently valid markings is immediately seen to also be marking-consistent, NAC-consistent, and coherent. It only remains to show that it is also internally complete.

Thus, at any point in the algorithm, let  $PC_{G'}$  be the currently given partial cover and  $x$  be some element in the image of its valid markings. We have to show that  $x$  is also marked by some valid marking. Because  $x$  is in the image of the subset of valid markings, there is a valid marking  $\varphi_i \in PC_{G'}$  such that  $x$  is in the image of  $\varphi_i$ . Either,  $\varphi_i$  itself marks  $x$ . Or, by internal completeness and coherence of  $PC_{G'}$ , there exists a unique marking  $\varphi_j \in PC_{G'}$  such that  $\varphi_j$  marks  $x$ . But this means  $\varphi_j <_d \varphi_i$ , which, by validity of  $\varphi_i$ , implies validity of  $\varphi_j$ .  $\square$

**Termination** We first show that our synchronization process terminates. Here, we initially understand termination in a rather weak sense, namely only meaning that the synchronization process halts, maybe with an exception. Later, we discuss *completeness*, meaning that for every sensible input, we are able to compute an output (Theorem 6.17).

**Proposition 6.16** (Termination). *Given an integrity-preserving TGG that is source-progressive, our synchronization process terminates.*

*Proof.* First, because the given TGG is source-progressive, the algorithm can only finitely many times consecutively call the function TRANSLATE: Since the source graph is not changed by the application of forward rules and no marking is removed from the partial cover, the number of initially unmarked source elements provides an upper bound on the number of forward rule applications: By source-progressiveness, each such application at a legal match reduces that number.

Furthermore, for every application of a (relaxed) repair or revocation rule, the *total number of markings behind the first invalid one strictly decreases*. This is with regard to the topological sorting of the respective induced partial covers that we discussed in Lemmas 6.13 and 6.14.

In summary, this means that—even though the total number of markings might grow and an application of a synchronization operation might cause

further markings to become invalid—the recursive call of the synchronization process can always only finitely many times consecutively call the function `TRANSLATE`, before the algorithm terminates (maybe with an exception) when calling `ISFINISHED` or one of the functions `REPAIR` or `REVOKE` has to be called. However, the number of markings behind the first invalid one (in the initially given topological sorting of the partial cover) constitutes a global upper bound on how often the functions `REPAIR` and `REVOKE` can be called (before no invalid markings are left). Therefore, after finitely many recursive calls, both the sets of invalid markings and of forward matches are empty which implies termination of the the overall synchronization process (maybe with an exception).  $\square$

**Completeness** Whereas we only needed the assumption of source-progressiveness to show the termination of our synchronization process, we additionally need to assume marking-completeness to show its completeness. This assumption is needed to avoid the throwing of exceptions that we tolerated in the proof of termination.

**Theorem 6.17** (Completeness). *Given an integrity-preserving TGG  $GG$  that is source-progressive and marking-complete, our synchronization process is complete.*

*Proof.* First of all, it is necessary to note that the synchronization procedure cannot get “stuck”: The application of forward rules is always (recursively) performed as long as possible using the function `TRANSLATE`. And if some invalid marking cannot be repaired, this is recognized by the check in Algorithm 6.4, line 4, whereupon the function `REVOKE` is called for the respective invalid marking (see Algorithm 6.1, line 15). Lemma 6.12 ensures that revocation is always possible. By Proposition 6.16, the overall process terminates, and by the above considerations, the computation will end with a call of the function `ISFINISHED`, i.e., with the termination check where at least the sets of invalid markings and of forward matches are both empty. For the following, let  $PC_{G'}$  be the currently given partial cover. As all its markings are valid, Proposition 6.15 guarantees that it even is a partial precedence graph. By Lemma 6.7 and Theorem 6.9 that partial precedence graph represents a valid triple graph that is furthermore a sub-triple-graph of the currently given partial triple graph. Therefore, if we show the image

of  $PC_{G'}$  to be the currently given (partial) given graph, i.e.,  $PC_{G'}$  to be even a precedence graph, the currently given graph is a valid triple graph.

Let  $\text{Im } PC_{G'} = (U_S \leftarrow U_C \rightarrow U_T)$  be the image of that partial precedence graph,  $H_S \leftarrow U_C \rightarrow U_T$  be the partial synchronization graph represented by it (cf. Definition 6.15), and  $H_S \leftarrow G''_C \rightarrow G''_T$  be the currently given partial triple graph. First, we can assume that  $U_C = G''_C$  and  $U_T = G''_T$  as the partial cover always completely covers the currently given correspondence and target graphs. This holds because the original precedence graph is complete and during the run of the algorithm only correspondence and target elements that are deleted by a (relaxed) repair or revocation rule get “demarked”. Likewise, forward- and repair-induced partial covers include the new comatches such that every newly added target or correspondence element is directly marked. This is evident from (the proofs of) Lemmas 6.10, 6.11, 6.13, and 6.14. This means, the currently given partial triple graph is the partial synchronization graph represented by the partial precedence graph  $PC_{G'}$ . If the set of unmarked elements were not empty, i.e., if  $U_S$  is not isomorphic to  $H_S$ , by  $H_S \in \mathcal{L}(GG)^S$  and marking completeness there would exist a legal match for a forward rule in  $H_S \leftarrow G''_C \rightarrow G''_T$ . However, the set of forward matches is empty by assumption. Therefore, we also have  $U_S = H_S$ , and  $PC_{G'}$  is even a precedence graph showing  $H_S \leftarrow G''_C \rightarrow G''_T \in \mathcal{L}(GG)$  as in the proof of Theorem 6.18.  $\square$

**Correctness** We can prove the correctness of our synchronization process without any further assumptions. This means, while we cannot guarantee termination and completeness for arbitrary TGGs, whenever our synchronization process computes a solution the solution is correct (independently from any further assumptions on the TGG besides being integrity-preserving).

**Theorem 6.18** (Correctness). *For every integrity-preserving TGG  $GG$ , our synchronization process is correct, i.e., whenever it terminates without exception, the output is a valid triple graph.*

*Proof.* When the synchronization process terminates without exception, i.e., when the process ends, as desired, with a call of Algorithm 6.2, line 4, we are in the situation that the sets of forward matches, of invalid markings, and of unmarked source elements are all empty. First, that the set of invalid

markings is empty means that all of the current markings are valid. Exactly as in the proof of completeness, the currently given partial cover  $PC_{G'}$  constitutes a partial precedence graph according to Proposition 6.15, and if we show its partial synchronization graph to be the currently given partial triple graph, we are finished. To see this, let  $\text{Im } PC_{G'} = (U_S \leftarrow U_C \rightarrow U_T)$  be the image of that partial precedence graph and  $H_S \leftarrow G''_C \rightarrow G''_T$  be the currently given partial triple graph. As above, we can assume that  $U_C = G''_C$  and  $U_T = G''_T$  as the partial cover always completely covers the currently given correspondence and target graphs. Secondly, the set of unmarked elements being empty exactly means that  $U_S = H_S$ .  $\square$

**Runtime Complexity** The central factor for the runtime behavior of our synchronization process is the incremental pattern matcher; the cost of the actual application of the synchronization operations or the update of the precedence graph (including topological re-sorting of parts of it) are comparatively small. Generally, incremental pattern matching, based on a Rete network, is known to be in  $\mathcal{O}(n^{2k-1})$ , where  $n$  counts the number of elements that have to be processed (i.e., the size of the change) and  $k$  the denotes the size of a rule [72, Table 1]. In practical applications, however, the actual runtime heavily depends on how well the chosen network structure fits to the problem at hand; see, e.g., [22] for recent work on how to adapt the network structure to evolving graphs. Therefore, we use the incremental pattern matcher as a black box and state our complexity in terms of calls of it.

**Proposition 6.19.** *Given a triple graph  $G = (G_S \leftarrow G_C \rightarrow G_T) \in \mathcal{L}(GG)$ , where  $GG$  is a source-progressive and integrity-preserving TGG, and a source edit  $\delta = G_S \leftarrow D_S \rightarrow H_S$ , the synchronization process terminates with maximally*

$$|G_S| + |H_S|$$

*calls of the incremental pattern matcher, where  $|X|$  denotes the number of elements in  $X$  (without attribute values) for  $X \in \{G_S, H_S\}$ . Under favorable circumstances, the process terminates with*

$$|G_S \Delta H_S|$$

*calls of the incremental pattern matcher, where  $\Delta$  denotes the symmetric difference. In particular, in that case the runtime only depends on the size of the delta and not on the size of the model.*

*Proof.* We already proved termination of the synchronization procedure (Proposition 6.16) and, thus, only have to count the number of calls of the incremental pattern matcher. For this, the central insight is that Lemmas 6.10, 6.11, 6.13, and 6.14 ensure that a once valid marking is never replaced (revoked or repaired) by our process, only (potentially) invalid ones are. Therefore, the upper bound on the number of repair- and revocation-rule applications is the number of markings in the precedence graph of the original triple graph  $G$ . Since the TGG is source-progressive,  $|G_S|$ , the number of source elements of the original triple graph (without attribute values), provides an upper bound on this number. Furthermore, again by source-progressiveness,  $|H_S|$ , the number of source elements of the edited triple graph (without attribute values), provides an upper bound on the maximally necessary number of forward rule applications. (Intuitively, this worst case happens if the complete original precedence graph has to be revoked, then  $H_S$  has to be translated from scratch, and every rule revokes or translates a single source element).

As a favorable situation consider the case where

- for every invalid marking a suitable repair rule with legal match exists,
- no such application triggers a potentially invalid marking to become invalid, and where
- no newly added element triggered a filter NAC violation.

In that case, the number of deleted elements provides an upper bound on the number of necessary repair rule applications, and the number of newly added elements an upper bound on the number of necessary forward rule applications, whereas revocation is not necessary at all.  $\square$

**Incrementality and Re-Use of Elements** The discussion about incrementality circles around two related but distinct concepts: runtime and information loss. Intuitively, if we are given two large, interrelated models and the source model is slightly edited, it is desirable that the effort of

restoring consistency depends on the size of the edit and not on the size of the model(s). This serves as definition for *full incrementality* in [86]. Secondly, one wants to avoid unnecessary changes to the target model to retain as much information as possible. Obviously, both ideas are closely related: Large changes to the target model (caused, for example, by completely reparsing the source model) implicate a runtime that depends on the size of the model(s). However, finding a minimal change to the target model that restores consistency after an edit might involve a costly analysis; thus, the concepts are clearly not identical.

Furthermore, both concepts, namely runtime dependent on the size of the edit or minimal change to restore consistency, come with severe practical problems. Concerning the runtime, it is easy to construct examples where a small, local change of a source model necessitates a large, global change to the target model to restore consistency. Hence, proving *full incrementality* in the sense of [86] will only ever be possible in very restricted settings. Minimality of change, also called the *principle of least change* in the literature [158, 42], has the problem that it is not at all clear which metric to use to measure the size of changes. We provide a small example for this (in [185] there is an example that is similar in spirit) and refer to [42] for an in-depth discussion. Assume, in the context of our running example, a consistent triple graph with at least one Class to be given, and assume a user to edit the source graph by deleting one Class and creating another. Now, the minimal change (in terms of the *graph edit distance*, i.e., in terms of the number of added and deleted elements) to the target graph that restores consistency would be to re-use the Doc-File of the deleted Class for the new one. For this, one merely has to suitably move its incoming files-edge and change its name-attribute. In particular, the content-attribute of the Doc-File would not change. While this leads to a valid triple graph and the change is minimal, the output would probably violate user expectations and, thus, be undesirable: Likely, the content-attribute contains a description that is completely inapt for the new Class. As such implicit semantics and user expectations differ between application scenarios, one would need to find tailored metrics for different applications and then examine model synchronization processes with respect to at least the most important ones.

Orejas and Pino suggest yet another way to define incrementality, namely as incrementality over a graph [185]. Basically, given a triple graph  $G$ , a delta, and a triple graph  $G'$  that results from applying a model synchro-

nization process to that data, the synchronization process is said to be *incremental over a triple graph  $H$*  if  $H$  is a common sub-triple-graph of  $G$  and  $G'$  such that a transformation sequence constructing  $H$  exists that can be extended to derive  $G$  and extended to derive  $G'$ . This means, the synchronization process is guaranteed to preserve  $H$ . First, our results, namely Proposition 6.15 and Lemmas 6.10, 6.11, 6.13, and 6.14, immediately imply that our model synchronization approach (if it terminates without exception) is incremental over the triple graph that is represented by the subset of valid markings of the delta-induced partial cover of the user edit. However, our approach (and, actually, also the one of Orejas and Pino as we will further discuss in Sect. 6.5) offers even more: It is able to preserve elements that have to appear in a new context in  $G'$ , i.e., elements that cannot any longer be parsed in the same way in which they had been parsed in  $G$ .

Instead of suggesting yet another definition of incrementality, we use the ones from above and appeal to intuition to argue why our approach should be considered to be reasonably incremental:

- (1) As just stated, it is incremental in the sense introduced by Orejas and Pino [185].
- (2) As seen in Proposition 6.19, we can provide (restricted but not completely unreasonable—compare Sect. 6.4.3) circumstances under which the runtime of our process only depends on the size of the change and not on the size of the models. This result is not merely theoretical but the approach is implemented and evaluation on synthetic and realistic input confirms the theoretical considerations [79, 80].
- (3) Proposition 4.2 formally ensures that every short-cut rule application preserves elements that would have been deleted if its first underlying rule had to be revoked, instead. As every repair rule (i.e., forward short-cut rule) application in our synchronization process signifies a situation in which the synchronization process of Lelebici in [150] has to perform exactly such a revocation, our process preserves at least as many elements as there had been repair rule applications in comparison. Usually, that effect even amplifies since those revocations trigger further revocations of dependent markings.

This does not say anything about how often repair rule applications will be possible. But in our evaluations in [79, 80] they regularly were. Beyond our practical application there, we were able to argue how, for a TGG that is similar to the one of the running example in this thesis (but more realistic and complex), using repair rules we are able to incrementally propagate typical code refactorings from the class diagram to the documentation model without the need to resort to revocations.

Since in our approach, all reuse of elements is encoded in the repair rules, undesired reuse can be avoided by excluding repair rules that implement such undesired behavior. For a given grammar this can be done once so that this does not raise the need for user interaction during the synchronization processes. Even more effectively, sometimes it might be possible to determine types of elements that should never be reused when specifying the grammar. Then one can directly avoid to construct such repair rules at all. We illustrate this with the next example.

**Example 6.8.** The rule *ReuseDocFile* is a repair rule that implements the kind of undesired reuse we already shortly mentioned above: the reuse of a Doc-File without changing its `content`-attribute. It is just one of several similar rules that can be obtained as repair rules from the grammar in our example. The undesired reuse can be avoided by once culling this and similar rules from the set of derived repair rules. It would be more effective though, to offer the option to directly adjust the derivation process of the repair rules. In our case, it would be meaningful to only derive repair rules that, if they preserve a Doc-File, either change its `content`-attribute or also preserve a corresponding Class. (Strictly speaking, in the second case the repair rule would re-parse that Class but its underlying short-cut rule would preserve it.)

**Hippocraticness** Following the QVT-R standard, Stevens introduces *hippocraticness* as an important property of a bx [199]. Hippocraticness means that synchronizing a pair of already consistent models results in the same pair of models again (i.e., neither model is changed). Because TGGs explicitly consider the correspondence structure, there are different ways in which hippocraticness could be defined in that setting. The strictest definition

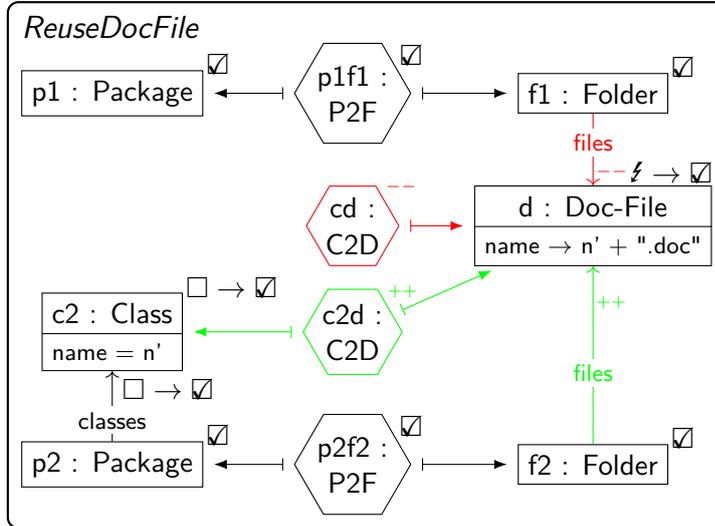


Figure 6.32: Example for a repair rule that implements undesired reuse

would probably be that a TGG-based synchronization process, whenever given some (partial) triple graph  $H_S \leftarrow G_C \rightarrow G_T$  to synchronize where  $H_S$  and  $G_T$  are consistent, should change neither  $H_S$  nor  $G_T$  but compute a correspondence graph  $G'_C$  (and the according morphisms) such that  $H_S \leftarrow G'_C \rightarrow G_T \in \mathcal{L}(GG)$ . A more lenient interpretation would be that such a process is allowed to return some  $H_S \leftarrow G'_C \rightarrow G'_T \in \mathcal{L}(GG)$  as long as  $H_S \leftarrow G_C \rightarrow G_T \notin \mathcal{L}(GG)$  (even if there exists a correspondence graph  $G''_C$  such that  $H_S \leftarrow G''_C \rightarrow G_T \in \mathcal{L}(GG)$ ).

Our approach is not guaranteed to be hippocratic, not even in the more lenient interpretation. In principle it can happen that in our approach, an already consistent model is re-parsed in such a way that correspondence and target model are changed even if the input had already been consistent. Whenever hippocraticness is desired, our approach can be combined with approaches that check for consistency (and compute a suitable correspondence structure to witness that). Then, synchronization only begins if the input is not already consistent. Technically, this is directly realizable: the TGG-based approach of Lelebici (et al.) [153, 150, 216] that checks triple graphs for consistency and computes a (maximally possible) correspondence structure between two models from the two given domains is also imple-

mented in eMoflon. Moreover, the approach has been extended to also deal with NACs [214] such that our setting is covered by the implementation. Of course, always first checking for consistency in that way significantly reduces the scalability of our approach.

**Treatment of attribute values** In the presentation of our algorithm, we did not explicitly discuss how to handle attributes. Instead, they are implicitly dealt with as the setting of attribute values amounts to the deletion and creation of attribution edges. That hides the fact, however, that in many cases there will be infinitely many matches for a rule that sets an attribute value (namely, infinitely many ways to evaluate a variable). Even if, in model synchronization processes, the match on the source part is fixed, it is not necessarily determined how to evaluate variables on the target side. In particular, their values might depend on non-reversible computations. This issue can be addressed by either resorting to user-interaction during the synchronization process or by using default values for attributes [54].

**Treatment of invalid input** The results we obtained so far ensure that for a source-progressive and marking-complete TGG, given valid input (i.e., a partial triple graph whose source graph belongs to the source language), our synchronization process produces valid output (i.e., a triple graph from the given TGG). The currently given subset of valid markings always encodes a valid triple graph (Proposition 6.15); however, its source graph only coincides with the one of the partial triple graph that is to be synchronized if all source elements have been marked. Furthermore, the proof of termination shows that our synchronization procedure is always able to get rid of all invalid markings. It can only stop (with an exception) in a situation where no legal forward match is available but unmarked source elements remain. In particular, if the algorithm stops with an exception, either the grammar cannot be marking-complete or the source graph of the input does not belong to the source language of the given TGG. Thus, given a grammar that is known to be marking-complete, our synchronization process can also be used to at least partially synchronize invalid input.

Of course, the question arises of whether the computed solution is of satisfying quality in that situation. To be able to synchronize a maximally possible subgraph, we would need to refine our treatment of NACs. Currently, we rely on the fact that the partial synchronization graphs never violate one of the given constraints; the NACs of the forward and repair rules ensure that such violations cannot be introduced during synchronization. For synchronizing a maximally possible subgraph, however, it might be necessary to drop valid markings to allow for the application of rules, whose NACs otherwise prevent this. Using our example of forbidding equal names for different Packages, a user might create a new Package with the same name as an existing one. In our current approach, the new Package is never propagated to the target side (at least if the marking that marks the older Package is not rendered invalid for another reason). However, if the old Package is empty but the user created a whole hierarchy starting with the new one, it would clearly be beneficial to include the new Package in the partial solution and drop the old one. We leave the exploration of that problem as an interesting topic for future work.

**User interaction** As a rule-based and randomized approach, in principle, our approach lends itself to allowing for user interaction. Whenever the algorithm randomly decides, e.g., which forward or repair rule to apply, the available options could be provided to the user, who can then select and thus direct the synchronization process towards a solution of high quality. Since our proofs do not depend on the random choices that are made, even all of the formal results obtained in this section are still valid when one allows for that kind of user interaction. For large models, however, the number of available options might easily overwhelm the user. Thus, to be of practical importance, this kind of user interaction would need to be accompanied by techniques to visualize and prioritize the available options to support the user. We, therefore, did not further pursue this topic.

As already discussed above, a further point for possible user interaction is the choice of the employed repair rules. The complete set of the automatically constructed ones can be provided to the user, who then decides which are to be used in the synchronization processes. In this way, undesired reuse can be prevented.

## 6.5 Related Work

We already gave an overview on bidirectional transformations in general and TGG-based model synchronization in particular in Sect. 2.1. In this section, we relate our work to other works from the area of graph transformation whose focus is not on model synchronization. With regard to other TGG-based approaches to model synchronization, we discuss some important details that were out of reach in Sect. 2.1.

**Processes and Subobject Transformation Systems** *Processes* are a means that has been introduced to understand the concurrent semantics of a grammar. Originally stemming from research on Petri nets, they have been generalized to graph grammars and later to categorical approaches to rewriting in variants of adhesive categories [19]. Also the slightly more general variant of a *subobject transformation system* (STS) has been introduced [47, 100]. What we call a *precedence graph* is actually (almost) an instance of an STS.<sup>9</sup> The central difference is that we do not require matches to be  $\mathcal{M}$ -morphisms but allow for quasi-injective matches. However, in [100], Hermann et al. also discuss how to construct a suitable STS (with  $\mathcal{M}$ -matches) to analyze derivations of the underlying grammar where the rules are applied at arbitrary matches. Both, a process or an STS characterize a derivation of their underlying grammar up to switch equivalence. In [100], the authors also consider negative application conditions. A central result is that in the presence of NACs, an STS characterizes a derivation sequence up to *permutation equivalence*, a relaxed notion of switch equivalence [100, Theorem 3.18]. Therefore, our result that precedence graphs characterize derivations up to switch equivalence (Theorem 6.9) is basically a known result. We obtain switch equivalence (instead of permutation equivalence) because we do not consider general NACs but only NACs derived from a negative constraint. We used the term “precedence graph” (instead of process or STS) to remain consistent with existing terminology in research on TGGs, where “precedence graph” has been used to denote similar structures [149, 4] (as mentioned in Sect. 2.1). Formally introducing STSs and discussing why and how—despite subtle differences—the results transfer

---

<sup>9</sup>A precedence graph is not a process in the sense of [19] because there, processes are introduced for grammars without monotonic rules.

to our precedence graphs seemed to amount to as much work as directly proving the desired results for our restricted setting.

**Model Synchronization, constraints, and application conditions** Support for additional constraints and application conditions is an important feature for model synchronization processes. In applications in the context of MDE, beyond the TGG there are often two meta-models defining the individual models of the two related domains (preferably, the TGG is *complete* in the sense that it relates each valid model of one domain to (at least) one valid model of the second). Meta-models are typically equipped with additional well-formedness constraints, for instance expressed in OCL [181]. This goes beyond the expressiveness of plain TGGs.

One general method to deal with additional constraints is to equip the rules of a TGG with application conditions in such a way that all graphs in the language of the grammar satisfy the additional constraints. In principle, this is possible. For example, a large and relevant portion of OCL can be translated into semantically equivalent graph constraints; in turn, these can be transformed into suitable application conditions for the rules of the grammar [188, 94]. While TGGs with such general application conditions have been considered [87, 54], we are not aware of any concrete model transformation or model synchronization process that has been worked out in that setting. In contrast, (formal) approaches with support for application conditions normally restrict themselves to the case of NACs [86, 126, 104, 54]; [4] extends this, but it remains unclear how the expressiveness of his *dynamic conditions* relates to the full first-order logic that is supported by nested constraints and conditions. Furthermore, support for NACs is regularly restricted to NACs that can be split into two sets such that one set of NACs only restricts the source graphs and the other one only the target graphs [126, 4, 104, 54]. The reason for this seems to be that the correctness (and other formal properties) of these approaches is based on the fact that each triple rule from the grammar is the concurrent rule of its source and its forward rule. For this to hold in the presence of NACs, one needs to be able to suitably partition the NACs of the triple rules into NACs for source and forward rules. That one part of the NACs only concerns the source graphs and a second part only the target graphs is a sufficient condition for this to be possible. Because we only consider NACs

that prevent the introduction of constraint violations—as, for instance, also done in [126, 4]—we are able to avoid that restriction. We only need to ensure that the applications of forward (and repair) rules cannot introduce constraints violations.

Furthermore, Hildebrandt et al. provide two complementary methods to deal with additional constraints in [107]. First, a static invariant-checker can be used to detect whether rules are able to introduce violations of constraints. For example, this can be used to detect whether the forward rules of a TGG are able to introduce violations of constraints that are formulated for the target model. If such violations can occur, the invariant-checker provides a counterexample. A user might use such a counterexample to manually adapt the grammar to avoid that situation. The invariant-checker is restricted to specific kinds of constraints (but covers negative constraints). To handle cases that are not covered by the invariant-checker, Hildebrandt et al. implement an additional validity check in their model synchronization process. This means, the output obtained from their TGG-based model synchronization process is also checked to conform to the additional constraints and discarded if that is not the case.

A specific kind of conditions are *attribute conditions* that allow one to prescribe complex relations between the attribute values occurring in valid triple graphs. Various approaches (and implementations) allow additionally equipping the rules of a TGG with such attribute conditions [86, 84, 91, 4, 150, 217]. While suggestions have been made how such attribute conditions might be formalized [13, 145], support in model synchronization processes either relies on solvers (e.g., [217]) or the user needs to provide a suitable operationalization of the conditions (e.g., [13]). While we do not discuss attribute conditions in this thesis, the implementation of our approach [79, 80, 77] supports them, based on the theory developed in [13] and its implementation in eMoflon. This means, the user needs to provide an operationalization of the attribute conditions (as Java code) for the forward rules.

**Preconditions for desirable behavior and static analyses** Ehrig, Hermann et al. show that their approach to model synchronization has desirable properties (correctness, completeness, and functional behavior) in case the given grammar is *deterministic* [104, 54]. This can be checked for statically

using *critical pair analysis*; confluence or even absence of (significant) critical pairs guarantees that the grammar is deterministic. To be able to support grammars that are not confluent, Klar et al. suggested the criterion of *source local completeness* [126], which non-confluent grammars might still satisfy. Intuitively, the criterion states that every match for a source rule can be extended to a match for a forward rule (not necessarily from the same underlying rule); in this way, (re-)translation processes cannot get stuck. While Klar et al. developed an efficient batch transformation process on that basis [126], Anjorin [4] used the criterion to show correctness and completeness of his incremental approach to the basic model synchronization problem. Furthermore, with co-workers, he developed a static analysis method to also check a grammar for source local completeness [10]. *Marking completeness*, as used in our approach, has been suggested by Leblebici [150] and used to show his reactive approach to be correct and complete. He also discusses how both confluence and source local completeness imply marking completeness (even if he does not provide a formal proof, the connections are rather obvious). This means that the conditions under which we have proven our approach to work as desired are exactly the conditions that other formal approaches rely on. Furthermore, static analyses are available that can be used to check a grammar for marking completeness. However, while critical pair analysis has also been developed for rules with NACs [144], the suggested analysis for source local completeness considers plain rules only [10]. It is future work to extend this analysis to rules with NACs.

**Combining TGGs with ILP solving** Starting with work by Leblebici (et al.) [153, 150], TGGs have been combined with ILP-techniques to address various problems in the realm of consistency management of models [216, 214, 217]; the research in [217] is most closely related to ours as it suggests a solution to the concurrent model synchronization problem and therefore can also be used to synchronize models after only one of them has been changed. The general idea behind all these approaches is that potential (partial) matches of rules in a triple graph constitute a search space. The kind of considered rules and matches and properties of the underlying triple graph vary between the different applications scenarios. In each application, however, the dependencies between the matches and mutual exclusions are translated into a 0-1 ILP problem and a (maximal) solution

for the problem is computed using some solver. The solution is then interpreted as a sequence of rule applications that constructs a solution to the original problem. While large parts of the approaches are formalized, a mathematical concept for the constructed search spaces themselves has not been suggested. We believe that the partial covers we introduce in this thesis could be used to do so. In contrast to the partial covers that occur in this thesis, these would be partial covers that are not coherent. The ILP problems then could be interpreted as obtaining a certain maximal precedence graph from the partial cover. This perspective might serve as a starting point to combine the new ILP-based approach with more classical approaches like ours in the future.

**Further related works** *Change-preserving model repair* as presented in [206, 182] is related to our approach. Assuming a set of consistency-preserving rules and a set of edit rules to be given, each edit rule is accompanied by one or more repair rules completing the edit step if possible. Such a complement rule is considered as repair rule of an edit rule w.r.t. an overarching consistency-preserving rule. Operationalized TGG rules fit into that approach but provide more structure: As graphs and rules are structured in triples, a source rule is also an edit rule being complemented by a forward rule. In contrast to that approach, source and forward rules can be automatically deduced from a given TGG rule. By our use of short-cut rules, we introduce a pre-processing step to first enlarge the sets of consistency-preserving rules and edit rules.

Boronat [34] presents an incremental uni-directional transformation approach. When re-translating a model after a change, affected elements of the old model are marked first and then, if possible, re-used instead of deleted and re-created (similar to the approaches suggested in [91, 185] for TGGs). Again, the same effects can be obtained by constructing and applying short-cut rules but there, for plain graph transformation. A correctness proof for that approach is still missing.

## 6.6 Conclusion

In this chapter, we developed our TGG-based approach to the basic model synchronization problem. The key technical contribution is the introduction

of partial covers and the investigation how controlled application of forward, repair, and revocation rules preserves important properties of a partial cover. This serves as a formal basis to show our synchronization process to terminate and to be correct and complete (under reasonable assumptions). We explicitly support attribution and additional negative constraints that further restrict the language of the given TGG. The use of repair rules results in the preservation of information that gets lost in several former approaches. Because we sum up the whole thesis immediately in the next chapter, we use this place to explicitly recapitulate how this chapter uses the two previous ones and to state smaller, more technical ideas for future work.

Originally, we set out to prove properties of our synchronization process based on the following idea: A consistent triple graph  $G$  corresponds to a sequence of rule applications

$$\emptyset \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow_{\rho_2, m_2} \cdots \Rightarrow_{\rho_t, m_t} G$$

that creates  $G$  (via rules of the grammar). The editing of the source graph  $G_S$  of  $G$  by a user can also be understood as a sequence of applications of source rules of the grammar and their inverses (at least, when the edited graph still belongs to the source language of the grammar). Sometimes, a combination of such applications can also be considered as the application of the source rule of a short-cut rule. Finally, our synchronization process is a sequence of rule applications, namely of forward, (relaxed) revocation, and (relaxed) repair rules. Obviously, the model synchronization process is correct when the composition of all three transformation sequences is equivalent to a sequence of applications of triple rules from the given TGG. In principle, such an equivalence can be argued for in the following way: Given suitable independence relations between the rule applications, the Local Church–Rosser Theorem (Theorem 3.7) allows switching the rule applications in such a way that a source rule application is immediately followed by an application of a corresponding forward rule (consistently matched). In particular, an ordinary source rule is followed by its forward rule, a source rule of a short-cut rule by its corresponding repair rule, and inverses of source rules by the corresponding revocation rule. Then, the Concurrency Theorem (Theorem 3.8) and the fact that triple rules are concurrent rules of their source and forward rules (Theorem 5.27) allow one to equivalently replace the transformation sequence by one that consists

of applications of triple rules, their inverses, and short-cut rules. In the presence of relaxed variants of revocation and repair rules, presumably the Amalgamation Theorem [54, Theorem 6.17] can be invoked to still obtain this result. The analysis case of the Generalized Concurrency Theorem (Theorem 4.14) then guarantees that the applications of short-cut rules can be replaced by applications of their underlying triple rules from the grammar. Finally, rule applications that are (immediately) followed by an application of their inverse rule cancel out and, hopefully, a sequence of applications of triple rules from the grammar remains, witnessing that the result of the synchronization process stems from the given language. Our theorem on language-preserving applications of GCRs (Theorem 4.16) presents sufficient conditions such that the necessary switching of rules is possible and related matches (e.g., for source and forward rule) will be consistent to each other. The just sketched argumentation is partially realized in [80] but only for very specific kinds of edits.

In this chapter, we introduced and used partial covers instead to prove the correctness (and other properties) of our approach. We did so for the following reasons (with the last two being the more important ones):

- It allows us to support more general NACs, namely NACs concerning the whole triple graph instead of NACs that only concern the source or the target part (because we do not depend on the fact that a triple rule is the concurrent rule of its source and its forward rule).
- In this way, the reasoning is closer to what actually happens in the implementation (because in the implementation, it is not explicitly analyzed to which kind of source rule applications a user edit might correspond).
- Finding conditions under which such a model synchronization process is complete seems to be more difficult.
- We are convinced that it is easier to lift our current approach to the concurrent model synchronization problem than it would be with the above sketched argument using equivalences of transformation sequences. This is because in the concurrent setting, when both sides have been edited, it will not suffice to consider the rather straightforward decomposition of rules in source and forward (or target and backward) rules. Instead, more complex variants have to

be found that, e.g., decompose a rule in a rule that performs a part of the source actions, a rule that performs a part of the target actions, and a rule that performs the rest (compare [186]).

But taking this approach also means that, while we naturally depend on the construction of GCRs (or short-cut rules) in Chapter 4, the introduction of typed attributed partial triple graphs as an adhesive HLR category in Chapter 5, and a bulk of technical lemmas, we actually do not directly need some of the main results from these chapters to prove the correctness and other properties of our approach. This concerns the Generalized Concurrency Theorem (Theorem 4.14), the conditions on language-preserving applications of GCRs (Theorem 4.16), or the preservation of *all* additional HLR properties when constructing (typed) attributed partial triple graphs from attributed graphs (Theorem 5.20). We decided to present these results nonetheless. First, these are the most important and interesting results when viewed from the perspective of extending the general theory of algebraic graph transformation or categorical rewriting more generally. Secondly, according to the just sketched procedure of proof, these results provide an additional intuition, why, at least “morally”, our approach should be a correct approach to the model synchronization problem.

In future work, it would be important to find static analysis methods to detect marking-completeness also for sets of rules that are equipped with NACs. Furthermore, instead of performing bookkeeping for a large set of repair rules, it would probably more efficient to restrict this to a small set of regularly needed rules and to derive missing ones on-the-fly during synchronization process. Both, identifying the most important repair rules and constructing missing ones when needed, are open problems.



## 7 Conclusion and Outlook

In this thesis, we develop a TGG-based solution to the basic model synchronization problem. Notable characteristics of our solution are that it

- explicitly supports advanced features that extend basic TGGs, namely attributes and negative constraints,
- avoids the kind of information loss that accompanies many former TGG-based solutions,
- is fully formalized and its central properties are proven, and
- is implemented and evaluated.<sup>1</sup>

The formal properties (like correctness, completeness, and termination) are proven under the same (or similar) assumptions as in previous approaches. Still, we are able to avoid information loss and, in especially favorable but by no means completely unrealistic circumstances, to guarantee for an efficient synchronization (in the sense that runtime depends on the size of the user edit and not the size of the models). The theory of *partial covers*, which we develop to prove these formal properties, provides a framework that might also further the understanding of recent approaches that combine TGGs with ILP-techniques [214, 216, 217]. As a prerequisite for our model synchronization algorithm, we develop repair rules and partial triple graphs. We do so on a very abstract level, namely in the setting of  $\mathcal{M}$ -adhesive categories; in this way, we contribute to the theory of algebraic graph transformation beyond the concrete setting in which we apply these developments. For the computation of repair rules, we introduce generalized concurrent rules as a new variant of sequential rule composition. Compared to concurrent rules, GCRs can preserve elements

---

<sup>1</sup>As already stated, the implementation is due to Lars Fritsche and documented in our publications [79, 80, 77] and, in particular, in his PhD thesis [76].

that the first underlying rule deletes but that can be considered as re-created by the second underlying rule. To introduce partial triple graphs, we work out the construction of  $S$ -cartesian functor categories as certain (non-full) subcategories of functor categories and show how under this construction variants of adhesiveness and additional HLR properties are preserved. The upshot is that  $S$ -cartesian functor categories, including partial triple graphs, are a suitable setting for double-pushout rewriting.

The results obtained and techniques developed in this thesis raise new questions or offer new approaches to existing problems. We close this thesis with presenting some possibilities to pursue the work of this thesis.

- The probably most obvious next step is the extension of our model synchronization algorithm to also be able to deal with the *concurrent model synchronization problem*, i.e., the situation where both source and target model have been changed and consistency shall be restored. The central new challenge that arises in this situation is that changes to source and target side might contradict each other and cannot both be maintained in a consistent triple graph. Moreover, generally, there is no single “right” procedure of solving such a conflict but different ways to solve one differ in their consequences; which outcome is desirable depends on the situation and the preferences of the users.

We actually developed a concept to also deal with the concurrent model synchronization problem; it is implemented and evaluated [77]. The central idea is that we use the precedence graph (or partial cover) to detect such conflicts (and the “areas” of the models that might be affected by such a conflict) and resolve them according to user-specified policies. The operations that constitute the policies are again forward, (relaxed) revocation, and (relaxed) repair rules. The work in this thesis provides a foundation that could be used to present our approach to the concurrent model synchronization problem in a more formal way and to prove its properties.

- Also for the basic model synchronization problem, it would be interesting to extend the theory of partial covers with the notion of *conflicts*. In this thesis, all partial covers that we considered were always *coherent*, i.e., there was only ever maximally one rule to translate a given source element. Furthermore, we only applied the

synchronization operations in such a way that we did not introduce new (filter-)NAC violations.

Allowing for such conflicts (which could be formally characterized via the parallel dependence of the underlying forward rule applications) and developing methods for their effective resolution, could serve as a basis to evade the precondition of marking-completeness that was necessary in our proofs.

- In Sect. 6.5 and above, we mentioned how our formal work provides a framework that also helps to better understand recent work that integrates TGGs with ILP-techniques [214, 216, 217]. More interestingly though, it would be to combine that approach with ours to benefit from their individual strengths. Using ILP-solving, Weidmann et al. can guarantee a certain optimality of the solutions that they compute, which we cannot. However, their technique is too slow to be able to deal with really large models. Thus, it would be desirable to use our approach to propagate uncontroversial changes but their ILP-based approach to compute optimal solutions for “important parts” of the model. The central challenges for this are finding conditions under which (i) a partial precedence graph for some part of the model that was obtained as a solution to an ILP-problem combines with a partial precedence graph for the rest of the model into a precedence graph for the whole model and (ii) the ILP-solution remains optimal (or at least “good enough”) in that larger context.
- Intuitively, we use partial covers and repair rules to incrementally re-parse the source graph that has been edited by a user and to adapt the correspondence and target graph in the process in such a way that they become consistent with the source graph. Therefore, it should be rather straightforward to adapt the theory we have developed to just incrementally re-parse a single graph that has been edited.

This could also serve as a starting point to efficiently detect if an edited graph still belongs to the given language if that language is defined by means of a grammar. It might even be possible that partial covers could serve as a basis to develop further methods for graph repair, a problem that received quite some research attention in recent years [178, 179, 206, 193, 191].

- The short-cut rules from which we derive our repair rules constitute (language-preserving) editing rules for the whole triple graph. Similarly, generalized concurrent rules can serve as such editing rules for more general grammars. While this thesis elaborates the theory of GCRs rather comprehensively—including a characterization for language-preserving applications (Theorem 4.16)—there are just too many GCRs that can be derived from even a small set of rules (cf. Corollary 4.8). The task therefore is to identify classes of common kernels for rules that lead to practically relevant editing rules. A promising starting point for such an endeavor is to learn from existing editing histories: Elements that regularly occur in immediate proximity of such an edit (e.g., nodes for which incident edges are created or deleted) are regularly put into new contexts by users and therefore promising candidates for being included into common kernels.

Based on a relevant and comprehensive set of such editing rules, it should be possible to develop *projectional editors* for modeling domains [197, 194], i.e., editors that only allow for but also support the editing of models in ways that preserve consistency (here: the language). In the case of TGGs, this could even be combined with on-the-fly propagation of changes: While the editor is an editor for only one of the domains and the possible edits stem from source rules (of short-cut rules), internally, with each edit, the corresponding forward rule (repair rule) is applied and directly propagates the edit to the second domain.

As already mentioned in the introduction, change propagation between and consistency management for models is still one of the key challenges in MDE [75, 36]. This is not new. For years, time and again immature tool support, in particular with an eye toward change propagation and consistency management, is mentioned as one important factor that hampers the adoption of MDE (in industry) [170, 38, 140, 171, 209]. We envision a single integrated modeling environment (similar to modern IDEs for code) in which all needed models can be maintained. It shall be possible to declaratively define what it means for pairs of models from different domains to be in a consistent state; either directly via means of a TGG or by some other mechanism that is translated into a TGG that defines the same consistency relation. Based on these TGGs for pairs of domains, sophisticated means

for consistency management shall be available. It shall be possible to restore consistency between a given pair of models or even between models of a selected network on demand. The inconsistencies may exist because several models have been edited locally or because of edits, performed by other users, that stem from a repository.<sup>2</sup> Consistency restoration shall be based on operations that the tool environment automatically derives from the relevant TGGs. Besides, consistency restoration shall be configurable and offer high formal guarantees. With regard to formal guarantees, we naturally require those synchronization processes to be correct, i.e., to actually restore consistency between all selected models. Beyond that, we envision support for (i) additional constraints for the individual modeling domains, which serve to restrict the kind of computed models beyond the given TGG (i.e., support for constraints that annotate the individual meta-models, e.g., in OCL [181]) and for (ii) comprehensive static analyses based on the derived synchronization operations. A user should be warned if properties that guarantee desirable behavior of the synchronization process are not met, i.e., when it cannot be guaranteed that the process computes a correct result or terminates (within reasonable time); TGGs are known to be an approach that allows for exactly such analyses [10, 54]. In such a complex setting, there will usually be more than one way to restore consistency. In different situations, different ways to deal with that indeterminism might be appropriate. It should therefore be possible that a user chooses between different modes of consistency restoration. Ideas for such modes are full automation (where choices are at least documented but may not meet the user's expectation), the computation of several different solutions, resolving indeterminism according to predefined user-policies (like "preference of creation over deletion" or "preference of the changes of one model over the ones of others"), or user interaction. One could also think about enriching this modeling environment with projectional editors.

While this thesis naturally does not achieve that ultimate goal, we hope that our research contributes to come to grips with this complex problem in the long run. We provide an improved TGG-based approach to the basic model synchronization problem that already respects an important class of additional constraints (namely negative constraints) that can be used

---

<sup>2</sup>For the version management of models of the same domain, other approaches, as for example developed by Timo Kehrer [123], can be used.

to restrict the kinds of models that are computed as solutions. We are convinced that the formal foundations that we developed in the process are apt to completely formalize (and prove correct) also a TGG-based approach which we contributed and that addresses the concurrent model synchronization problem [77]. Respecting more kinds of constraints and extending our approach to networks of models are interesting and challenging next steps to take.

## Bibliography

- [1] Faris Abou-Saleh, James Cheney, Jeremy Gibbons, James McKinna, and Perdita Stevens. “Introduction to Bidirectional Transformations”. In: *Bidirectional Transformations – International Summer School, Oxford, UK, July 25–29, 2016, Tutorial Lectures*. Ed. by Jeremy Gibbons and Perdita Stevens. Vol. 9715. Lecture Notes in Computer Science. Springer, 2018, pp. 1–28. ISBN: 978-3-319-79107-4. URL: [https://doi.org/10.1007/978-3-319-79108-1\\_1](https://doi.org/10.1007/978-3-319-79108-1_1) (cit. on p. 14).
- [2] Jiří Adámek, Horst Herrlich, and George E. Strecker. *Abstract and Concrete Categories. The Joy of Cats*. Online Edition. 2004. URL: <http://katmat.math.uni-bremen.de/acc/> (visited on 10/19/2020) (cit. on pp. 10, 69, 70, 332, 333, 338).
- [3] Abdullah Alqahtani and Reiko Heckel. “Model Based Development of Data Integration in Graph Databases Using Triple Graph Grammars”. In: *Software Technologies: Applications and Foundations – STAF 2018 Collocated Workshops, Toulouse, France, June 25–29, 2018, Revised Selected Papers*. Ed. by Manuel Mazzara, Iulian Ober, and Gwen Salaün. Vol. 11176. Lecture Notes in Computer Science. Springer, 2018, pp. 399–414. URL: [https://doi.org/10.1007/978-3-030-04771-9\\_29](https://doi.org/10.1007/978-3-030-04771-9_29) (cit. on p. 20).
- [4] Anthony Anjorin. “Synchronization of Models on Different Abstraction Levels using Triple Graph Grammars”. PhD thesis. Darmstadt University of Technology, Germany, 2014. URL: <http://tuprints.ulb.tu-darmstadt.de/4399/> (cit. on pp. 23, 25, 26, 28, 37, 179, 262–265).
- [5] Anthony Anjorin. “An Introduction to Triple Graph Grammars as an Implementation of the Delta-Lens Framework”. In: *Bidirectional Transformations – International Summer School, Oxford, UK, July 25–29, 2016, Tutorial Lectures*. Ed. by Jeremy Gibbons and Perdita Stevens. Vol. 9715. Lecture Notes in Computer Science. Springer, 2018, pp. 29–72. URL: [https://doi.org/10.1007/978-3-319-79108-1\\_2](https://doi.org/10.1007/978-3-319-79108-1_2) (cit. on p. 18).
- [6] Anthony Anjorin, Thomas Buchmann, Bernhard Westfechtel, Zinovy Diskin, Hsiang-Shang Ko, Romina Eramo, Georg Hinkel, Leila Samimi-Dehkordi, and Albert Zündorf. “Benchmarking bidirectional transformations: theory, implementation, application, and assessment”. In: *Softw. Syst. Model.* 19.3

- (2020), pp. 647–691. URL: <https://doi.org/10.1007/s10270-019-00752-x> (cit. on pp. 2, 4, 14, 19).
- [7] Anthony Anjorin, Zinovy Diskin, Frédéric Jouault, Hsiang-Shang Ko, Erhan Leblebici, and Bernhard Westfechtel. “BenchmarX Reloaded: A Practical Benchmark Framework for Bidirectional Transformations”. In: *Proceedings of the 6th International Workshop on Bidirectional Transformations co-located with The European Joint Conferences on Theory and Practice of Software, BX@ETAPS 2017, Uppsala, Sweden, April 29, 2017*. Ed. by Romina Eramo and Michael Johnson. Vol. 1827. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 15–30. URL: <http://ceur-ws.org/Vol-1827/paper6.pdf> (cit. on p. 4).
- [8] Anthony Anjorin, Erhan Leblebici, Roland Kluge, Andy Schürr, and Perdita Stevens. “A Systematic Approach and Guidelines to Developing a Triple Graph Grammar”. In: *Proceedings of the 4th International Workshop on Bidirectional Transformations co-located with Software Technologies: Applications and Foundations, STAF 2015, L’Aquila, Italy, July 24, 2015*. Ed. by Alcino Cunha and Ekkart Kindler. Vol. 1396. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 81–95. URL: <http://ceur-ws.org/Vol-1396/p81-anjorin.pdf> (cit. on pp. 4, 27).
- [9] Anthony Anjorin, Erhan Leblebici, and Andy Schürr. “20 Years of Triple Graph Grammars: A Roadmap for Future Research”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 73 (2015). URL: <https://doi.org/10.14279/tuj.eceasst.73.1031> (cit. on pp. 20, 21).
- [10] Anthony Anjorin, Erhan Leblebici, Andy Schürr, and Gabriele Taentzer. “A Static Analysis of Non-confluent Triple Graph Grammars for Efficient Model Transformation”. In: *Graph Transformation – 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22–24, 2014. Proceedings*. Ed. by Holger Giese and Barbara König. Vol. 8571. Lecture Notes in Computer Science. Springer, 2014, pp. 130–145. ISBN: 978-3-319-09107-5. URL: [https://doi.org/10.1007/978-3-319-09108-2\\_9](https://doi.org/10.1007/978-3-319-09108-2_9) (cit. on pp. 3, 265, 275).
- [11] Anthony Anjorin, Sebastian Rose, Frederik Deckwerth, and Andy Schürr. “Efficient Model Synchronization with View Triple Graph Grammars”. In: *Modelling Foundations and Applications – 10th European Conference, ECMFA@STAF 2014, York, UK, July 21–25, 2014. Proceedings*. Ed. by Jordi Cabot and Julia Rubin. Vol. 8569. Lecture Notes in Computer Science. Springer, 2014, pp. 1–17. URL: [https://doi.org/10.1007/978-3-319-09195-2\\_1](https://doi.org/10.1007/978-3-319-09195-2_1) (cit. on p. 18).

- [12] Anthony Anjorin, Andy Schürr, and Gabriele Taentzer. “Construction of Integrity Preserving Triple Graph Grammars”. In: *Graph Transformations – 6th International Conference, ICGT 2012, Bremen, Germany, September 24–29, 2012. Proceedings*. Ed. by Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Vol. 7562. Lecture Notes in Computer Science. Springer, 2012, pp. 356–370. URL: [https://doi.org/10.1007/978-3-642-33654-6\\_24](https://doi.org/10.1007/978-3-642-33654-6_24) (cit. on pp. 182, 184, 185).
- [13] Anthony Anjorin, Gergely Varró, and Andy Schürr. “Complex Attribute Manipulation in TGGs with Constraint-Based Programming Techniques”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 49 (2012). URL: <https://doi.org/10.14279/tuj.eceasst.49.707> (cit. on pp. 5, 28, 264).
- [14] Anthony Anjorin, Nils Weidmann, Robin Oppermann, Lars Fritsche, and Andy Schürr. “Automating test schedule generation with domain-specific languages: a configurable, model-driven approach”. In: *MoDELS ’20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18–23 October, 2020*. Ed. by Eugene Syriani, Houari A. Sahraoui, Juan de Lara, and Silvia Abrahão. ACM, 2020, pp. 320–331. URL: <https://doi.org/10.1145/3365438.3410991> (cit. on p. 20).
- [15] Michal Antkiewicz and Krzysztof Czarnecki. “Design Space of Heterogeneous Synchronization”. In: *Generative and Transformational Techniques in Software Engineering II, International Summer School, GTTSE 2007, Braga, Portugal, July 2–7, 2007. Revised Papers*. Ed. by Ralf Lämmel, Joost Visser, and João Saraiva. Vol. 5235. Lecture Notes in Computer Science. Springer, 2008, pp. 3–46. URL: [https://doi.org/10.1007/978-3-540-88643-3\\_1](https://doi.org/10.1007/978-3-540-88643-3_1) (cit. on p. 13).
- [16] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. “Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations”. In: *Model Driven Engineering Languages and Systems – 13th International Conference, MODELS 2010, Oslo, Norway, October 3–8, 2010, Proceedings, Part I*. Ed. by Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen. Vol. 6394. Lecture Notes in Computer Science. Springer, 2010, pp. 121–135. ISBN: 978-3-642-16144-5. URL: [https://doi.org/10.1007/978-3-642-16145-2\\_9](https://doi.org/10.1007/978-3-642-16145-2_9) (cit. on p. 187).
- [17] Steve Awodey. *Category Theory*. 2nd. Vol. 52. Oxford Logic Guides. New York, NY, USA: Oxford University Press, 2010 (cit. on pp. 10, 47, 140).

- [18] Guilherme Grochau Azzi, Andrea Corradini, and Leila Ribeiro. “On the essence and initiality of conflicts in M-adhesive transformation systems”. In: *J. Log. Algebraic Methods Program.* 109 (2019). URL: <https://doi.org/10.1016/j.jlamp.2019.100482> (cit. on pp. 49, 72, 74, 151–153, 337).
- [19] Paolo Baldan, Andrea Corradini, Tobias Heindel, Barbara König, and Paweł Sobociński. “Processes and unfoldings: concurrent computations in adhesive categories”. In: *Math. Struct. Comput. Sci.* 24.4 (2014). URL: <https://doi.org/10.1017/S096012951200031X> (cit. on pp. 127, 262).
- [20] François Bancilhon and Nicolas Spyratos. “Update Semantics of Relational Views”. In: *ACM Trans. Database Syst.* 6.4 (1981), pp. 557–575. URL: <https://doi.org/10.1145/319628.319634> (cit. on p. 15).
- [21] Davi M. J. Barbosa, Julien Cretin, Nate Foster, Michael Greenberg, and Benjamin C. Pierce. “Matching lenses: alignment and view update”. In: *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27–29, 2010*. Ed. by Paul Hudak and Stephanie Weirich. ACM, 2010, pp. 193–204. URL: <https://doi.org/10.1145/1863543.1863572> (cit. on p. 18).
- [22] Matthias Barkowsky and Holger Giese. “Host-Graph-Sensitive RETE Nets for Incremental Graph Pattern Matching”. In: *Graph Transformation – 14th International Conference, ICGT 2021, Held as Part of STAF 2021, Virtual Event, June 24–25, 2021, Proceedings*. Ed. by Fabio Gadducci and Timo Kehrer. Vol. 12741. Lecture Notes in Computer Science. Springer, 2021, pp. 145–163. URL: [https://doi.org/10.1007/978-3-030-78946-6\\_8](https://doi.org/10.1007/978-3-030-78946-6_8) (cit. on p. 254).
- [23] Michael Barr and Charles Wells. *Category theory for computing science (2. ed.)* Prentice Hall international series in computer science. Prentice Hall, 1995. ISBN: 978-0-13-323809-9 (cit. on p. 174).
- [24] Nicolas Behr. “Sesqui-Pushout Rewriting: Concurrency, Associativity and Rule Algebra Framework”. In: *Proceedings Tenth International Workshop on Graph Computation Models, GCM@STAF 2019, Eindhoven, The Netherlands, 17th July 2019*. Ed. by Rachid Echahed and Detlef Plump. Vol. 309. EPTCS. 2019, pp. 23–52. URL: <https://doi.org/10.4204/EPTCS.309.2> (cit. on pp. 59, 81, 128).
- [25] Nicolas Behr, Vincent Danos, and Ilias Garnier. “Combinatorial Conversion and Moment Bisimulation for Stochastic Rewriting Systems”. In: *Log. Methods Comput. Sci.* 16.3 (2020). URL: <https://lmcs.episciences.org/6628> (cit. on p. 128).

- [26] Nicolas Behr, Russ Harmer, and Jean Krivine. “Concurrency Theorems for Non-linear Rewriting Theories”. In: *Graph Transformation – 14th International Conference, ICGT 2021, Held as Part of STAF 2021, Virtual Event, June 24–25, 2021, Proceedings*. Ed. by Fabio Gadducci and Timo Kehrer. Vol. 12741. Lecture Notes in Computer Science. Springer, 2021, pp. 3–21. URL: [https://doi.org/10.1007/978-3-030-78946-6\\_1](https://doi.org/10.1007/978-3-030-78946-6_1) (cit. on p. 81).
- [27] Nicolas Behr and Jean Krivine. “Rewriting Theory for the Life Sciences: A Unifying Theory of CTMC Semantics”. In: *Graph Transformation – 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25–26, 2020, Proceedings*. Ed. by Fabio Gadducci and Timo Kehrer. Vol. 12150. Lecture Notes in Computer Science. Springer, 2020, pp. 185–202. ISBN: 978-3-030-51371-9. URL: [https://doi.org/10.1007/978-3-030-51372-6\\_11](https://doi.org/10.1007/978-3-030-51372-6_11) (cit. on p. 128).
- [28] Nicolas Behr, Maryam Ghaffari Saadat, and Reiko Heckel. “Commutators for Stochastic Rewriting Systems: Theory and Implementation in Z3”. In: *Proceedings of the Eleventh International Workshop on Graph Computation Models, GCM@STAF 2020, Online-Workshop, 24th June 2020*. Ed. by Berthold Hoffmann and Mark Minas. Vol. 330. EPTCS. 2020, pp. 126–144. URL: <https://doi.org/10.4204/EPTCS.330.8> (cit. on p. 187).
- [29] Nicolas Behr and Paweł Sobociński. “Rule Algebras for Adhesive Categories”. In: *Log. Methods Comput. Sci.* 16.3 (2020). URL: <https://lmcs.episciences.org/6615> (cit. on pp. 74, 110, 127, 128).
- [30] Enrico Biermann, Claudia Ermel, and Gabriele Taentzer. “Formal foundation of consistent EMF model transformations by algebraic graph transformation”. In: *Softw. Syst. Model.* 11.2 (2012), pp. 227–250. URL: <https://doi.org/10.1007/s10270-011-0199-7> (cit. on p. 29).
- [31] Dominique Blouin, Alain Plantec, Pierre Dissaux, Frank Singhoff, and Jean-Philippe Diguët. “Synchronization of Models of Rich Languages with Triple Graph Grammars: An Experience Report”. In: *Theory and Practice of Model Transformations – 7th International Conference, ICMT@STAF 2014, York, UK, July 21–22, 2014. Proceedings*. Ed. by Davide Di Ruscio and Dániel Varró. Vol. 8568. Lecture Notes in Computer Science. Springer, 2014, pp. 106–121. ISBN: 978-3-319-08788-7. URL: [https://doi.org/10.1007/978-3-319-08789-4\\_8](https://doi.org/10.1007/978-3-319-08789-4_8) (cit. on pp. 5, 193).
- [32] Paul Boehm, Harald-Reto Fonio, and Annegret Habel. “Amalgamation of Graph Transformations with Applications to Synchronization”. In: *Mathematical Foundations of Software Development, Proceedings of the International Joint Conference on Theory and Practice of Software Development*

- (TAPSOFT), Berlin, Germany, March 25–29, 1985, Volume 1: Colloquium on Trees in Algebra and Programming (CAAP’85). Ed. by Hartmut Ehrig, Christiane Floyd, Maurice Nivat, and James W. Thatcher. Vol. 185. Lecture Notes in Computer Science. Springer, 1985, pp. 267–283. ISBN: 3-540-15198-2. URL: [https://doi.org/10.1007/3-540-15198-2\\_17](https://doi.org/10.1007/3-540-15198-2_17) (cit. on pp. 81, 98).
- [33] Francis Borceux. *Handbook of Categorical Algebra*. Vol. 3: *Sheaf Theory*. Encyclopedia of Mathematics and its Applications 52. Cambridge: Cambridge University Press, 1994. ISBN: 9780511525872. DOI: [10.1017/CB09780511525872](https://doi.org/10.1017/CB09780511525872) (cit. on p. 151).
- [34] Artur Boronat. “Offline Delta-Driven Model Transformation with Dependency Injection”. In: *Fundamental Approaches to Software Engineering – 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings*. Ed. by Reiner Hähnle and Wil M. P. van der Aalst. Vol. 11424. Lecture Notes in Computer Science. Springer, 2019, pp. 134–150. ISBN: 978-3-030-16721-9. URL: [https://doi.org/10.1007/978-3-030-16722-6\\_8](https://doi.org/10.1007/978-3-030-16722-6_8) (cit. on p. 266).
- [35] Graham R. Brightwell and Peter Winkler. “Counting Linear Extensions is #P-Complete”. In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5–8, 1991, New Orleans, Louisiana, USA*. Ed. by Cris Koutsougeras and Jeffrey Scott Vitter. ACM, 1991, pp. 175–181. URL: <https://doi.org/10.1145/103418.103441> (cit. on p. 217).
- [36] Antonio Bucchiarone, Federico Ciccozzi, Leen Lambers, Alfonso Pierantonio, Matthias Tichy, Massimo Tisi, Andreas Wortmann, and Vadim Zaytsev. “What Is the Future of Modeling?” In: *IEEE Softw.* 38.2 (2021), pp. 119–127. URL: <https://doi.org/10.1109/MS.2020.3041522> (cit. on pp. 1, 274).
- [37] Thomas Buchmann. “BXtend – A Framework for (Bidirectional) Incremental Model Transformations”. In: *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2018, Funchal, Madeira – Portugal, January 22–24, 2018*. Ed. by Slimane Hammoudi, Luís Ferreira Pires, and Bran Selic. SciTePress, 2018, pp. 336–345. ISBN: 978-989-758-283-7. URL: <https://doi.org/10.5220/0006563503360345> (cit. on p. 19).
- [38] David Budgen, Andy J. Burn, O. Pearl Brereton, Barbara A. Kitchenham, and Rialette Pretorius. “Empirical evidence about the UML: a systematic literature review”. In: *Softw. Pract. Exp.* 41.4 (2011), pp. 363–392. URL: <https://doi.org/10.1002/spe.1009> (cit. on p. 274).

- [39] Boris Cadish and Zinovy Diskin. “Heterogeneous View Integration via Sketches and Equations”. In: *Foundations of Intelligent Systems, 9th International Symposium, ISMIS '96, Zakopane, Poland, June 9–13, 1996, Proceedings*. Ed. by Zbigniew W. Ras and Maciej Michalewicz. Vol. 1079. Lecture Notes in Computer Science. Springer, 1996, pp. 603–612. URL: [https://doi.org/10.1007/3-540-61286-6\\_184](https://doi.org/10.1007/3-540-61286-6_184) (cit. on p. 174).
- [40] Aurelio Carboni and Peter T. Johnstone. “Connected Limits, Familial Representability and Artin Glueing”. In: *Math. Struct. Comput. Sci.* 5.4 (1995), pp. 441–459. URL: <https://doi.org/10.1017/S0960129500001183> (cit. on p. 173).
- [41] Aurelio Carboni, Stephen Lack, and R.F.C. Walters. “Introduction to extensive and distributive categories”. In: *Journal of Pure and Applied Algebra* 84.2 (1993), pp. 145–158. DOI: [https://doi.org/10.1016/0022-4049\(93\)90035-R](https://doi.org/10.1016/0022-4049(93)90035-R). URL: <http://www.sciencedirect.com/science/article/pii/002240499390035R> (cit. on pp. 150, 338).
- [42] James Cheney, Jeremy Gibbons, James McKinna, and Perdita Stevens. “On principles of Least Change and Least Surprise for bidirectional transformations”. In: *J. Object Technol.* 16.1 (2017), 3:1–31. URL: <https://doi.org/10.5381/jot.2017.16.1.a3> (cit. on p. 256).
- [43] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. “JTL: A Bidirectional and Change Propagating Transformation Language”. In: *Software Language Engineering – Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12–13, 2010, Revised Selected Papers*. Ed. by Brian A. Malloy, Steffen Staab, and Mark van den Brand. Vol. 6563. Lecture Notes in Computer Science. Springer, 2010, pp. 183–202. URL: [https://doi.org/10.1007/978-3-642-19440-5\\_11](https://doi.org/10.1007/978-3-642-19440-5_11) (cit. on p. 19).
- [44] Anthony Cleve, Ekkart Kindler, Perdita Stevens, and Vadim Zaytsev. “Multidirectional Transformations and Synchronisations (Dagstuhl Seminar 18491)”. In: *Dagstuhl Reports* 8.12 (2018), pp. 1–48. URL: <https://doi.org/10.4230/DagRep.8.12.1> (cit. on p. 2).
- [45] Andrea Corradini, Dominique Duval, Michael Löwe, Leila Ribeiro, Rodrigo Machado, Andrei Costa, Guilherme Grochau Azzi, Jonas Santos Bezerra, and Leonardo Marques Rodrigues. “On the Essence of Parallel Independence for the Double-Pushout and Sesqui-Pushout Approaches”. In: *Graph Transformation, Specifications, and Nets – In Memory of Hartmut Ehrig*. Ed. by Reiko Heckel and Gabriele Taentzer. Vol. 10800. Lecture Notes in Computer Science. Springer, 2018, pp. 1–18. ISBN: 978-3-319-75395-9. URL: [https://doi.org/10.1007/978-3-319-75396-6\\_1](https://doi.org/10.1007/978-3-319-75396-6_1) (cit. on p. 59).

- [46] Andrea Corradini, Tobias Heindel, Frank Hermann, and Barbara König. “Sesqui-Pushout Rewriting”. In: *Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17–23, 2006, Proceedings*. Ed. by Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg. Vol. 4178. Lecture Notes in Computer Science. Springer, 2006, pp. 30–45. ISBN: 3-540-38870-2. URL: [https://doi.org/10.1007/11841883\\_4](https://doi.org/10.1007/11841883_4) (cit. on pp. 58, 59, 128, 170).
- [47] Andrea Corradini, Frank Hermann, and Paweł Sobociński. “Subobject Transformation Systems”. In: *Appl. Categorical Struct.* 16.3 (2008), pp. 389–419. URL: <https://doi.org/10.1007/s10485-008-9127-6> (cit. on p. 262).
- [48] Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. “Bidirectional Transformations: A Cross-Discipline Perspective”. In: *Theory and Practice of Model Transformations – 2nd International Conference, ICMT@TOOLS 2009, Zurich, Switzerland, June 29–30, 2009. Proceedings*. Ed. by Richard F. Paige. Vol. 5563. Lecture Notes in Computer Science. Springer, 2009, pp. 260–283. ISBN: 978-3-642-02407-8. URL: [https://doi.org/10.1007/978-3-642-02408-5\\_19](https://doi.org/10.1007/978-3-642-02408-5_19) (cit. on pp. 1, 2, 13).
- [49] Zinovy Diskin, Hamid Gholizadeh, Arif Wider, and Krzysztof Czarnecki. “A three-dimensional taxonomy for bidirectional model synchronization”. In: *J. Syst. Softw.* 111 (2016), pp. 298–322. URL: <https://doi.org/10.1016/j.jss.2015.06.003> (cit. on p. 14).
- [50] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. “From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case”. In: *Journal of Object Technology* 10 (2011), 6:1–25. DOI: 10.5381/jot.2011.10.1.a6. URL: [http://www.jot.fm/contents/issue\\_2011\\_01/article6.html](http://www.jot.fm/contents/issue_2011_01/article6.html) (cit. on p. 16).
- [51] Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki, Hartmut Ehrig, Frank Hermann, and Fernando Orejas. “From State- to Delta-Based Bidirectional Model Transformations: The Symmetric Case”. In: *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16–21, 2011. Proceedings*. Ed. by Jon Whittle, Tony Clark, and Thomas Kühne. Vol. 6981. Lecture Notes in Computer Science. Springer, 2011, pp. 304–318. URL: [https://doi.org/10.1007/978-3-642-24485-8\\_22](https://doi.org/10.1007/978-3-642-24485-8_22) (cit. on pp. 16, 192, 193).

- [52] Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, Frank Hermann, and Gabriele Taentzer. “Information Preserving Bidirectional Model Transformations”. In: *Fundamental Approaches to Software Engineering, 10th International Conference, FASE 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24 – April 1, 2007, Proceedings*. Ed. by Matthew B. Dwyer and Antónia Lopes. Vol. 4422. Lecture Notes in Computer Science. Springer, 2007, pp. 72–86. URL: [https://doi.org/10.1007/978-3-540-71289-3\\_7](https://doi.org/10.1007/978-3-540-71289-3_7) (cit. on pp. 21, 22, 80).
- [53] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Berlin, Heidelberg, and New York: Springer, 2006. ISBN: 978-3-540-31187-4. URL: <https://doi.org/10.1007/3-540-31188-2> (cit. on pp. 3, 5, 10, 22, 35, 46, 49, 50, 55, 62, 72, 75, 77, 78, 81, 82, 108, 113, 124, 128, 131, 153, 154, 168, 185, 214, 337, 338).
- [54] Hartmut Ehrig, Claudia Ermel, Ulrike Golas, and Frank Hermann. *Graph and Model Transformation – General Framework and Applications*. Monographs in Theoretical Computer Science. An EATCS Series. Heidelberg et al.: Springer, 2015. ISBN: 978-3-662-47979-7. URL: <https://doi.org/10.1007/978-3-662-47980-3> (cit. on pp. 2, 3, 5–8, 29, 46, 47, 50, 63, 67, 70, 76, 78, 93, 128, 131, 149, 154, 182, 184, 190, 192, 193, 196, 248, 260, 263, 264, 268, 275, 339).
- [55] Hartmut Ehrig, Claudia Ermel, Frank Hermann, and Ulrike Prange. “On-the-Fly Construction, Correctness and Completeness of Model Transformations Based on Triple Graph Grammars”. In: *Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, CO, USA, October 4–9, 2009. Proceedings*. Ed. by Andy Schürr and Bran Selic. Vol. 5795. Lecture Notes in Computer Science. Springer, 2009, pp. 241–255. URL: [https://doi.org/10.1007/978-3-642-04425-0\\_18](https://doi.org/10.1007/978-3-642-04425-0_18) (cit. on pp. 22, 192).
- [56] Hartmut Ehrig, Claudia Ermel, and Gabriele Taentzer. “A Formal Resolution Strategy for Operation-Based Conflicts in Model Versioning Using Graph Modifications”. In: *Fundamental Approaches to Software Engineering – 14th International Conference, FASE 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26–April 3, 2011. Proceedings*. Ed. by Dimitra Giannakopoulou and Fernando Orejas. Vol. 6603. Lecture Notes in Computer Science. Springer, 2011, pp. 202–216. URL: [https://doi.org/10.1007/978-3-642-19811-3\\_15](https://doi.org/10.1007/978-3-642-19811-3_15) (cit. on p. 190).

- [57] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, and Fernando Orejas. “ $\mathcal{M}$ -Adhesive Transformation Systems with Nested Application Conditions. Part 2: Embedding, Critical Pairs and Local Confluence”. In: *Fundam. Informaticae* 118.1-2 (2012), pp. 35–63. URL: <https://doi.org/10.3233/FI-2012-705> (cit. on pp. 46, 67, 131, 154).
- [58] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, and Fernando Orejas. “ $\mathcal{M}$ -adhesive transformation systems with nested application conditions. Part 1: parallelism, concurrency and amalgamation”. In: *Math. Struct. Comput. Sci.* 24.4 (2014). URL: <https://doi.org/10.1017/S0960129512000357> (cit. on pp. 46, 56, 57, 60, 67, 68, 81, 98, 110, 112, 126, 128, 131, 154, 327).
- [59] Hartmut Ehrig, Ulrike Golas, and Frank Hermann. “Categorical Frameworks for Graph Transformation and HLR Systems Based on the DPO Approach”. In: *Bull. EATCS* 102 (2010), pp. 111–121. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/158> (cit. on pp. 49, 128).
- [60] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, and Francesco Parisi-Presicce. “Parallelism and Concurrency in High-Level Replacement Systems”. In: *Math. Struct. Comput. Sci.* 1.3 (1991), pp. 361–404. URL: <https://doi.org/10.1017/S096012950001353> (cit. on pp. 49, 127).
- [61] Hartmut Ehrig, Annegret Habel, and Leen Lambers. “Parallelism and Concurrency Theorems for Rules with Nested Application Conditions”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 26 (2010). URL: <https://doi.org/10.14279/tuj.eceasst.26.363> (cit. on pp. 81, 128).
- [62] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner, and Andrea Corradini. “Algebraic Approaches to Graph Transformation – Part II: Single Pushout Approach and Comparison with Double Pushout Approach”. In: *Handbook of Graph Grammars and Computing by Graph Transformations*. Vol. 1: *Foundations*. Ed. by Grzegorz Rozenberg. World Scientific, 1997, pp. 247–312. ISBN: 9810228848 (cit. on p. 177).
- [63] Hartmut Ehrig, Frank Hermann, and Christoph Sartorius. “Completeness and Correctness of Model Transformations based on Triple Graph Grammars with Negative Application Conditions”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 18 (2009). URL: <https://doi.org/10.14279/tuj.eceasst.18.270> (cit. on p. 21).
- [64] Hartmut Ehrig and Michael Löwe. “Categorical principles, techniques and results for high-level-replacement systems in computer science”. In: *Appl. Categorical Struct.* 1.1 (1993), pp. 21–50. URL: <https://doi.org/10.1007/BF00872984> (cit. on p. 49).

- [65] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Vol. 6. EATCS Monographs on Theoretical Computer Science. Springer, 1985. ISBN: 3-540-13718-1. URL: <https://doi.org/10.1007/978-3-642-69962-7> (cit. on pp. 75, 313).
- [66] Hartmut Ehrig, Fernando Orejas, and Ulrike Prange. “Categorical Foundations of Distributed Graph Transformation”. In: *Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17–23, 2006, Proceedings*. Ed. by Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg. Vol. 4178. Lecture Notes in Computer Science. Springer, 2006, pp. 215–229. ISBN: 3-540-38870-2. URL: [https://doi.org/10.1007/11841883\\_16](https://doi.org/10.1007/11841883_16) (cit. on p. 177).
- [67] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. “Fundamental Theory for Typed Attributed Graph Transformation”. In: *Graph Transformations, Second International Conference, ICGT 2004, Rome, Italy, September 28–October 2, 2004, Proceedings*. Ed. by Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg. Vol. 3256. Lecture Notes in Computer Science. Springer, 2004, pp. 161–177. ISBN: 3-540-23207-9. URL: [https://doi.org/10.1007/978-3-540-30203-2\\_13](https://doi.org/10.1007/978-3-540-30203-2_13) (cit. on p. 75).
- [68] Hartmut Ehrig and Barry K. Rosen. “Parallelism and concurrency of graph manipulations”. In: *Theoretical Computer Science* 11.3 (1980), pp. 247–275. DOI: [https://doi.org/10.1016/0304-3975\(80\)90016-X](https://doi.org/10.1016/0304-3975(80)90016-X). URL: <http://www.sciencedirect.com/science/article/pii/030439758090016X> (cit. on pp. 81, 127).
- [69] Romina Eramo, Alfonso Pierantonio, and Michele Tucci. “Enhancing the JTL tool for bidirectional transformations”. In: *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Nice, France, April 09–12, 2018*. Ed. by Stefan Marr and Jennifer B. Sartor. ACM, 2018, pp. 36–41. URL: <https://doi.org/10.1145/3191697.3191720> (cit. on p. 19).
- [70] Claudia Ermel, Frank Hermann, Jürgen Gall, and Daniel Binander. “Visual Modeling and Analysis of EMF Model Transformations Based on Triple Graph Grammars”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 54 (2012). URL: <https://doi.org/10.14279/tuj.eceasst.54.771> (cit. on p. 26).
- [71] Sebastian Fischer, Zhenjiang Hu, and Hugo Pacheco. “The essence of bidirectional programming”. In: *Sci. China Inf. Sci.* 58.5 (2015), pp. 1–21. URL: <https://doi.org/10.1007/s11432-015-5316-8> (cit. on p. 17).

- [72] Charles Forgy. “Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem”. In: *Artif. Intell.* 19.1 (1982), pp. 17–37. URL: [https://doi.org/10.1016/0004-3702\(82\)90020-0](https://doi.org/10.1016/0004-3702(82)90020-0) (cit. on p. 254).
- [73] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. “Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem”. In: *ACM Trans. Program. Lang. Syst.* 29.3 (2007), p. 17. URL: <https://doi.org/10.1145/1232420.1232424> (cit. on pp. 15, 192).
- [74] Martin Fowler. *Refactoring – Improving the Design of Existing Code*. Addison Wesley object technology series. Addison-Wesley, 1999. ISBN: 978-0-201-48567-7. URL: <http://martinfowler.com/books/refactoring.html> (cit. on p. 84).
- [75] Mirco Franzago, Davide Di Ruscio, Ivano Malavolta, and Henry Muccini. “Collaborative Model-Driven Software Engineering: A Classification Framework and a Research Map”. In: *IEEE Trans. Software Eng.* 44.12 (2018), pp. 1146–1175. URL: <https://doi.org/10.1109/TSE.2017.2755039> (cit. on pp. 1, 274).
- [76] Lars Fritsche. “Local Consistency Restoration Methods for Triple Graph Grammars”. PhD thesis. Technische Universität Darmstadt, Germany, 2022. URL: <https://tuprints.ulb.tu-darmstadt.de/21443/> (cit. on pp. 9, 181, 231, 271).
- [77] Lars Fritsche, Jens Kosiol, Adrian Möller, Andy Schürr, and Gabriele Taentzer. “A precedence-driven approach for concurrent model synchronization scenarios using triple graph grammars”. In: *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2020, Virtual Event, USA, November 16–17, 2020*. Ed. by Ralf Lämmel, Laurence Tratt, and Juan de Lara. ACM, 2020, pp. 39–55. ISBN: 978-1-4503-8176-5. URL: <https://doi.org/10.1145/3426425.3426931> (cit. on pp. 9, 29, 30, 181, 189, 231, 232, 264, 271, 272, 276).
- [78] Lars Fritsche, Jens Kosiol, Andy Schürr, and Gabriele Taentzer. “Short-Cut Rules – Sequential Composition of Rules Avoiding Unnecessary Deletions”. In: *Software Technologies: Applications and Foundations – STAF 2018 Collocated Workshops, Toulouse, France, June 25–29, 2018, Revised Selected Papers*. Ed. by Manuel Mazzara, Iulian Ober, and Gwen Salaün. Vol. 11176. Lecture Notes in Computer Science. Springer, 2018, pp. 415–430. ISBN: 978-3-030-04770-2. URL: [https://doi.org/10.1007/978-3-030-04771-9\\_30](https://doi.org/10.1007/978-3-030-04771-9_30) (cit. on pp. 7, 10, 81–83, 85, 86, 88, 98, 124, 127).

- [79] Lars Fritsche, Jens Kosiol, Andy Schürr, and Gabriele Taentzer. “Efficient Model Synchronization by Automatically Constructed Repair Processes”. In: *Fundamental Approaches to Software Engineering – 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings*. Ed. by Reiner Hähnle and Wil M. P. van der Aalst. Vol. 11424. Lecture Notes in Computer Science. Springer, 2019, pp. 116–133. ISBN: 978-3-030-16721-9. URL: [https://doi.org/10.1007/978-3-030-16722-6\\_7](https://doi.org/10.1007/978-3-030-16722-6_7) (cit. on pp. 9, 28, 30, 88, 125, 127, 181, 182, 231, 232, 257, 258, 264, 271).
- [80] Lars Fritsche, Jens Kosiol, Andy Schürr, and Gabriele Taentzer. “Avoiding unnecessary information loss: correct and efficient model synchronization based on triple graph grammars”. In: *Int. J. Softw. Tools Technol. Transf.* 23.3 (2021), pp. 335–368. DOI: 10.1007/s10009-020-00588-7. URL: <https://link.springer.com/article/10.1007/s10009-020-00588-7> (cit. on pp. 9, 13, 28–30, 181, 182, 231, 232, 246, 257, 258, 264, 268, 271).
- [81] Karsten Gabriel, Benjamin Braatz, Hartmut Ehrig, and Ulrike Golas. “Finitary  $\mathcal{M}$ -adhesive categories”. In: *Math. Struct. Comput. Sci.* 24.4 (2014). URL: <https://doi.org/10.1017/S0960129512000321> (cit. on pp. 182, 209, 353).
- [82] Richard Garner and Stephen Lack. “On the axioms for adhesive and quasiadhesive categories”. In: *Theory and Applications of Categories* 27.3 (2012), pp. 27–46. URL: <https://www.emis.de/journals/TAC/volumes/27/3/27-03abs.html> (cit. on pp. 51, 144, 318).
- [83] Jeremy Gibbons and Perdita Stevens, eds. *Bidirectional Transformations – International Summer School, Oxford, UK, July 25–29, 2016, Tutorial Lectures*. Vol. 9715. Lecture Notes in Computer Science. Springer, 2018. ISBN: 978-3-319-79107-4. URL: <https://doi.org/10.1007/978-3-319-79108-1> (cit. on p. 2).
- [84] Holger Giese and Stephan Hildebrandt. *Efficient Model Synchronization of Large-Scale Models*. Tech. rep. 28. Potsdam: Hasso-Plattner-Institut, 2009. URL: <https://publishup.uni-potsdam.de/frontdoor/index/index/docId/2883> (visited on 10/29/2020) (cit. on pp. 5, 23–25, 28, 193, 264).
- [85] Holger Giese, Stephan Hildebrandt, and Leen Lambers. “Bridging the gap between formal semantics and implementation of triple graph grammars – Ensuring conformance of relational model transformation specifications and implementations”. In: *Softw. Syst. Model.* 13.1 (2014), pp. 273–299. URL: <https://doi.org/10.1007/s10270-012-0247-y> (cit. on pp. 5, 24).

- [86] Holger Giese and Robert Wagner. “From model transformation to incremental bidirectional model synchronization”. In: *Softw. Syst. Model.* 8.1 (2009), pp. 21–43. URL: <https://doi.org/10.1007/s10270-008-0089-9> (cit. on pp. 5, 8, 23–25, 28, 37, 248, 256, 263, 264).
- [87] Ulrike Golas, Hartmut Ehrig, and Frank Hermann. “Formal Specification of Model Transformations by Triple Graph Grammars with Application Conditions”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 39 (2011). URL: <https://doi.org/10.14279/tuj.eceasst.39.646> (cit. on pp. 5, 263).
- [88] Ulrike Golas, Annegret Habel, and Hartmut Ehrig. “Multi-amalgamation of rules with application conditions in  $M$ -adhesive categories”. In: *Math. Struct. Comput. Sci.* 24.4 (2014). URL: <https://doi.org/10.1017/S0960129512000345> (cit. on pp. 51, 71, 73, 81, 98).
- [89] Ulrike Golas, Leen Lambers, Hartmut Ehrig, and Holger Giese. “Toward Bridging the Gap between Formal Foundations and Current Practice for Triple Graph Grammars – Flexible Relations between Source and Target Elements”. In: *Graph Transformations – 6th International Conference, ICGT 2012, Bremen, Germany, September 24–29, 2012. Proceedings.* Ed. by Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Vol. 7562. Lecture Notes in Computer Science. Springer, 2012, pp. 141–155. ISBN: 978-3-642-33653-9. URL: [https://doi.org/10.1007/978-3-642-33654-6\\_10](https://doi.org/10.1007/978-3-642-33654-6_10) (cit. on p. 176).
- [90] Joel Greenyer and Ekkart Kindler. “Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars”. In: *Softw. Syst. Model.* 9.1 (2010), pp. 21–46. URL: <https://doi.org/10.1007/s10270-009-0121-8> (cit. on p. 15).
- [91] Joel Greenyer, Sebastian Pook, and Jan Rieke. “Preventing Information Loss in Incremental Model Synchronization by Reusing Elements”. In: *Modelling Foundations and Applications – 7th European Conference, ECMFA 2011, Birmingham, UK, June 6 – 9, 2011 Proceedings.* Ed. by Robert B. France, Jochen Malte Küster, Behzad Bordbar, and Richard F. Paige. Vol. 6698. Lecture Notes in Computer Science. Springer, 2011, pp. 144–159. ISBN: 978-3-642-21469-1. URL: [https://doi.org/10.1007/978-3-642-21470-7\\_11](https://doi.org/10.1007/978-3-642-21470-7_11) (cit. on pp. 24, 28, 264, 266).
- [92] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. “Graph Grammars with Negative Application Conditions”. In: *Fundam. Informaticae* 26.3/4 (1996), pp. 287–313. URL: <https://doi.org/10.3233/FI-1996-263404> (cit. on p. 183).

- [93] Annegret Habel, Jürgen Müller, and Detlef Plump. “Double-pushout graph transformation revisited”. In: *Math. Struct. Comput. Sci.* 11.5 (2001), pp. 637–688. URL: <https://doi.org/10.1017/S0960129501003425> (cit. on pp. 54, 183).
- [94] Annegret Habel and Karl-Heinz Pennemann. “Correctness of high-level transformation systems relative to nested conditions”. In: *Math. Struct. Comput. Sci.* 19.2 (2009), pp. 245–296. URL: <https://doi.org/10.1017/S0960129508007202> (cit. on pp. 52, 53, 133, 178, 183, 187, 263).
- [95] Jonathan Hayman and Tobias Heindel. “On Pushouts of Partial Maps”. In: *Graph Transformation – 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22–24, 2014. Proceedings.* Ed. by Holger Giese and Barbara König. Vol. 8571. Lecture Notes in Computer Science. Springer, 2014, pp. 177–191. ISBN: 978-3-319-09107-5. URL: [https://doi.org/10.1007/978-3-319-09108-2\\_12](https://doi.org/10.1007/978-3-319-09108-2_12) (cit. on p. 177).
- [96] Reiko Heckel and Annika Wagner. “Ensuring consistency of conditional graph rewriting – a constructive approach”. In: *Electron. Notes Theor. Comput. Sci.* 2 (1995), pp. 118–126. URL: [https://doi.org/10.1016/S1571-0661\(05\)80188-4](https://doi.org/10.1016/S1571-0661(05)80188-4) (cit. on p. 187).
- [97] Tobias Heindel. “A category theoretical approach to the concurrent semantics of rewriting: adhesive categories and related concepts”. PhD. University of Duisburg-Essen, 2009. URL: <http://duepublico.uni-duisburg-essen.de/servlets/DocumentServlet/Document-24329/diss.pdf> (visited on 10/19/2020) (cit. on p. 49).
- [98] Tobias Heindel. “Adhesivity with Partial Maps instead of Spans”. In: *Fundam. Informaticae* 118.1-2 (2012), pp. 1–33. URL: <https://doi.org/10.3233/FI-2012-704> (cit. on pp. 46, 49, 50, 66, 74, 102, 177, 337).
- [99] Tobias Heindel and Paweł Sobociński. “Van Kampen Colimits as Bicolimits in Span”. In: *Algebra and Coalgebra in Computer Science, Third International Conference, CALCO 2009, Udine, Italy, September 7–10, 2009. Proceedings.* Ed. by Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki. Vol. 5728. Lecture Notes in Computer Science. Springer, 2009, pp. 335–349. ISBN: 978-3-642-03740-5. URL: [https://doi.org/10.1007/978-3-642-03741-2\\_23](https://doi.org/10.1007/978-3-642-03741-2_23) (cit. on p. 173).
- [100] Frank Hermann, Andrea Corradini, and Hartmut Ehrig. “Analysis of permutation equivalence in  $\mathcal{M}$ -adhesive transformation systems with negative application conditions”. In: *Math. Struct. Comput. Sci.* 24.4 (2014). URL: <https://doi.org/10.1017/S0960129512000382> (cit. on pp. 182, 184, 262).

- [101] Frank Hermann, Hartmut Ehrig, Claudia Ermel, and Fernando Orejas. “Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars”. In: *Fundamental Approaches to Software Engineering – 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 – April 1, 2012. Proceedings*. Ed. by Juan de Lara and Andrea Zisman. Vol. 7212. Lecture Notes in Computer Science. Springer, 2012, pp. 178–193. ISBN: 978-3-642-28871-5. URL: [https://doi.org/10.1007/978-3-642-28872-2\\_13](https://doi.org/10.1007/978-3-642-28872-2_13) (cit. on p. 29).
- [102] Frank Hermann, Hartmut Ehrig, Ulrike Golas, and Fernando Orejas. “Efficient analysis and execution of correct and complete model transformations based on triple graph grammars”. In: *Proceedings of the First International Workshop on Model-Driven Interoperability, MDI@MoDELS 2010, Oslo, Norway, October 3–5, 2010*. Ed. by Jean Bézivin, Richard Mark Soley, and Antonio Vallecillo. ACM, 2010, pp. 22–31. ISBN: 978-1-4503-0292-0. URL: <https://doi.org/10.1145/1866272.1866277> (cit. on pp. 21, 22, 34, 35).
- [103] Frank Hermann, Hartmut Ehrig, Ulrike Golas, and Fernando Orejas. “Formal analysis of model transformations based on triple graph grammars”. In: *Math. Struct. Comput. Sci.* 24.4 (2014). URL: <https://doi.org/10.1017/S0960129512000370> (cit. on pp. 21, 22, 184, 195, 196).
- [104] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, Yingfei Xiong, Susann Gottmann, and Thomas Engel. “Model synchronization based on triple graph grammars: correctness, completeness and invertibility”. In: *Softw. Syst. Model.* 14.1 (2015), pp. 241–269. URL: <https://doi.org/10.1007/s10270-012-0309-1> (cit. on pp. 5, 8, 18, 26–28, 37, 78, 191, 213, 263, 264).
- [105] Frank Hermann, Hartmut Ehrig, Fernando Orejas, and Ulrike Golas. “Formal Analysis of Functional Behaviour for Model Transformations Based on Triple Graph Grammars”. In: *Graph Transformations - 5th International Conference, ICGT 2010, Enschede, The Netherlands, September 27 – October 2, 2010. Proceedings*. Ed. by Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr. Vol. 6372. Lecture Notes in Computer Science. Springer, 2010, pp. 155–170. ISBN: 978-3-642-15927-5. URL: [https://doi.org/10.1007/978-3-642-15928-2\\_11](https://doi.org/10.1007/978-3-642-15928-2_11) (cit. on pp. 21, 22).
- [106] Soichiro Hidaka, Massimo Tisi, Jordi Cabot, and Zhenjiang Hu. “Feature-based classification of bidirectional transformation approaches”. In: *Softw. Syst. Model.* 15.3 (2016), pp. 907–928. URL: <https://doi.org/10.1007/s10270-014-0450-0> (cit. on p. 14).

- [107] Stephan Hildebrandt, Leen Lambers, Basil Becker, and Holger Giese. “Integration of Triple Graph Grammars and Constraints”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 54 (2012). URL: <https://doi.org/10.14279/tuj.eceasst.54.770> (cit. on pp. 5, 264).
- [108] Stephan Hildebrandt, Leen Lambers, Holger Giese, Jan Rieke, Joel Greenyer, Wilhelm Schäfer, Marius Lauder, Anthony Anjorin, and Andy Schürr. “A Survey of Triple Graph Grammar Tools”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 57 (2013). URL: <https://doi.org/10.14279/tuj.eceasst.57.865> (cit. on p. 20).
- [109] Georg Hinkel and Erik Burger. “Change propagation and bidirectionality in internal transformation DSLs”. In: *Softw. Syst. Model.* 18.1 (2019), pp. 249–278. URL: <https://doi.org/10.1007/s10270-017-0617-6> (cit. on p. 19).
- [110] Martin Hofmann, Benjamin C. Pierce, and Daniel Wagner. “Symmetric lenses”. In: *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26–28, 2011*. Ed. by Thomas Ball and Mooly Sagiv. ACM, 2011, pp. 371–384. URL: <https://doi.org/10.1145/1926385.1926428> (cit. on p. 16).
- [111] Martin Hofmann, Benjamin C. Pierce, and Daniel Wagner. “Edit lenses”. In: *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22–28, 2012*. Ed. by John Field and Michael Hicks. ACM, 2012, pp. 495–508. ISBN: 978-1-4503-1083-3. URL: <https://doi.org/10.1145/2103656.2103715> (cit. on p. 16).
- [112] Rudi Horn, Roly Perera, and James Cheney. “Incremental relational lenses”. In: *Proc. ACM Program. Lang.* 2.ICFP (2018), 74:1–74:30. URL: <https://doi.org/10.1145/3236769> (cit. on p. 18).
- [113] Zhenjiang Hu, Andy Schürr, Perdita Stevens, and James F. Terwilliger. “Bidirectional Transformation "bx" (Dagstuhl Seminar 11031)”. In: *Dagstuhl Reports* 1.1 (2011), pp. 42–67. URL: <https://doi.org/10.4230/DagRep.1.1.42> (cit. on pp. 2, 13).
- [114] Michael Johnson and Robert D. Rosebrugh. “Delta Lenses and Opfibrations”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 57 (2013). URL: <https://doi.org/10.14279/tuj.eceasst.57.875> (cit. on p. 17).

- [115] Michael Johnson and Robert D. Rosebrugh. “Spans of Delta Lenses”. In: *Proceedings of the 4th International Workshop on Bidirectional Transformations co-located with Software Technologies: Applications and Foundations, STAF 2015, L’Aquila, Italy, July 24, 2015*. Ed. by Alcino Cunha and Ekkart Kindler. Vol. 1396. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 1–15. URL: <http://ceur-ws.org/Vol-1396/p1-johnson.pdf> (cit. on p. 17).
- [116] Michael Johnson and Robert D. Rosebrugh. “Universal Updates for Symmetric Lenses”. In: *Proceedings of the 6th International Workshop on Bidirectional Transformations co-located with The European Joint Conferences on Theory and Practice of Software, BX@ETAPS 2017, Uppsala, Sweden, April 29, 2017*. Ed. by Romina Eramo and Michael Johnson. Vol. 1827. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 39–53. URL: <http://ceur-ws.org/Vol-1827/paper8.pdf> (cit. on p. 17).
- [117] Michael Johnson and Robert D. Rosebrugh. “Cospans and symmetric lenses”. In: *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Nice, France, April 09–12, 2018*. Ed. by Stefan Marr and Jennifer B. Sartor. ACM, 2018, pp. 21–29. URL: <https://doi.org/10.1145/3191697.3191717> (cit. on p. 17).
- [118] Michael Johnson, Robert D. Rosebrugh, and Richard J. Wood. “Algebras and Update Strategies”. In: *J. Univers. Comput. Sci.* 16.5 (2010), pp. 729–748. URL: <https://doi.org/10.3217/jucs-016-05-0729> (cit. on pp. 15, 17).
- [119] Michael Johnson, Robert D. Rosebrugh, and Richard J. Wood. “Lenses, fibrations and universal translations”. In: *Math. Struct. Comput. Sci.* 22.1 (2012), pp. 25–42. URL: <https://doi.org/10.1017/S0960129511000442> (cit. on pp. 15, 17, 174).
- [120] Peter T. Johnstone, Stephen Lack, and Paweł Sobociński. “Quasitoposes, Quasiadhesive Categories and Artin Glueing”. In: *Algebra and Coalgebra in Computer Science, Second International Conference, CALCO 2007, Bergen, Norway, August 20–24, 2007, Proceedings*. Ed. by Till Mossakowski, Ugo Montanari, and Magne Haveraaen. Vol. 4624. Lecture Notes in Computer Science. Springer, 2007, pp. 312–326. ISBN: 978-3-540-73857-2. URL: [https://doi.org/10.1007/978-3-540-73859-6\\_21](https://doi.org/10.1007/978-3-540-73859-6_21) (cit. on pp. 74, 132, 137, 173, 174).
- [121] Dieter Jungnickel. *Graphs, Networks and Algorithms*. 4th ed. Vol. 5. Algorithms and Computation in Mathematics. Heidelberg: Springer-Verlag Berlin Heidelberg, 2013. DOI: <https://doi.org/10.1007/978-3-642-32278-5> (cit. on p. 212).

- [122] Harmen Kastenbergh and Arend Rensink. “Graph Attribution Through Sub-Graphs”. In: *Graph Transformation, Specifications, and Nets – In Memory of Hartmut Ehrig*. Ed. by Reiko Heckel and Gabriele Taentzer. Vol. 10800. Lecture Notes in Computer Science. Springer, 2018, pp. 245–265. ISBN: 978-3-319-75395-9. URL: [https://doi.org/10.1007/978-3-319-75396-6\\_14](https://doi.org/10.1007/978-3-319-75396-6_14) (cit. on pp. 132, 175).
- [123] Timo Kehrer. “Calculation and propagation of model changes based on user-level edit operations: a foundation for version and variant management in model-driven engineering”. PhD thesis. University of Siegen, 2015. URL: <http://dokumentix.ub.uni-siegen.de/opus/volltexte/2015/963/> (cit. on p. 275).
- [124] Timo Kehrer, Gabriele Taentzer, Michaela Rindt, and Udo Kelter. “Automatically Deriving the Specification of Model Editing Operations from Meta-Models”. In: *Theory and Practice of Model Transformations – 9th International Conference, ICMT@STAF 2016, Vienna, Austria, July 4–5, 2016, Proceedings*. Ed. by Pieter Van Gorp and Gregor Engels. Vol. 9765. Lecture Notes in Computer Science. Springer, 2016, pp. 173–188. ISBN: 978-3-319-42063-9. URL: [https://doi.org/10.1007/978-3-319-42064-6\\_12](https://doi.org/10.1007/978-3-319-42064-6_12) (cit. on p. 128).
- [125] Richard Kennaway. “Graph Rewriting in Some Categories of Partial Morphisms”. In: *Graph-Grammars and Their Application to Computer Science, 4th International Workshop, Bremen, Germany, March 5–9, 1990, Proceedings*. Ed. by Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Vol. 532. Lecture Notes in Computer Science. Springer, 1991, pp. 490–504. ISBN: 3-540-54478-X. URL: <https://doi.org/10.1007/BFb0017408> (cit. on pp. 49, 177).
- [126] Felix Klar, Marius Lauder, Alexander Königs, and Andy Schürr. “Extended Triple Graph Grammars with Efficient and Compatible Graph Translators”. In: *Graph Transformations and Model-Driven Engineering – Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*. Ed. by Gregor Engels, Claus Lewerentz, Wilhelm Schäfer, Andy Schürr, and Bernhard Westfechtel. Vol. 5765. Lecture Notes in Computer Science. Springer, 2010, pp. 141–174. URL: [https://doi.org/10.1007/978-3-642-17322-6\\_8](https://doi.org/10.1007/978-3-642-17322-6_8) (cit. on pp. 21, 22, 182, 184, 185, 195, 196, 263–265).
- [127] Hsiang-Shang Ko and Zhenjiang Hu. “An axiomatic basis for bidirectional programming”. In: *Proc. ACM Program. Lang.* 2.POPL (2018), 41:1–41:29. URL: <https://doi.org/10.1145/3158129> (cit. on p. 17).

- [128] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu. “BiGUL: a formally verified core language for putback-based bidirectional programming”. In: *Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2016, St. Petersburg, FL, USA, January 20 – 22, 2016*. Ed. by Martin Erwig and Tiark Rompf. ACM, 2016, pp. 61–72. ISBN: 978-1-4503-4097-7. URL: <https://doi.org/10.1145/2847538.2847544> (cit. on p. 17).
- [129] Harald König and Zinovy Diskin. “Efficient Consistency Checking of Interrelated Models”. In: *Modelling Foundations and Applications – 13th European Conference, ECMFA@STAF 2017, Marburg, Germany, July 19–20, 2017, Proceedings*. Ed. by Anthony Anjorin and Huáscar Espinoza. Vol. 10376. Lecture Notes in Computer Science. Springer, 2017, pp. 161–178. ISBN: 978-3-319-61481-6. URL: [https://doi.org/10.1007/978-3-319-61482-3\\_10](https://doi.org/10.1007/978-3-319-61482-3_10) (cit. on pp. 133, 177).
- [130] Harald König and Patrick Stünkel. “Single Pushout Rewriting in Comprehensive Systems”. In: *Graph Transformation – 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25–26, 2020, Proceedings*. Ed. by Fabio Gadducci and Timo Kehrer. Vol. 12150. Lecture Notes in Computer Science. Springer, 2020, pp. 91–108. ISBN: 978-3-030-51371-9. URL: [https://doi.org/10.1007/978-3-030-51372-6\\_6](https://doi.org/10.1007/978-3-030-51372-6_6) (cit. on pp. 133, 177).
- [131] Alexander Königs. “Model Integration and Transformation: A Triple Graph Grammar-based QVT Implementation”. PhD thesis. Darmstadt University of Technology, 2009. URL: <http://tuprints.ulb.tu-darmstadt.de/1194/> (cit. on p. 15).
- [132] Jens Kosiol, Lars Fritsche, Nebras Nassar, Andy Schürr, and Gabriele Taentzer. “Constructing Constraint-Preserving Interaction Schemes in Adhesive Categories”. In: *Recent Trends in Algebraic Development Techniques – 24th IFIP WG 1.3 International Workshop, WADT 2018, Egham, UK, July 2–5, 2018, Revised Selected Papers*. Ed. by José Luiz Fiadeiro and Ionut Tutu. Vol. 11563. Lecture Notes in Computer Science. Springer, 2018, pp. 139–153. URL: [https://doi.org/10.1007/978-3-030-23220-7\\_8](https://doi.org/10.1007/978-3-030-23220-7_8) (cit. on p. 10).
- [133] Jens Kosiol, Lars Fritsche, Andy Schürr, and Gabriele Taentzer. “Adhesive Subcategories of Functor Categories with Instantiation to Partial Triple Graphs”. In: *Graph Transformation – 12th International Conference, ICGT 2019, Held as Part of STAF 2019, Eindhoven, The Netherlands, July 15–16, 2019, Proceedings*. Ed. by Esther Guerra and Fernando Orejas. Vol. 11629. Lecture Notes in Computer Science. Springer, 2019, pp. 38–54. ISBN: 978-

- 3-030-23610-6. URL: [https://doi.org/10.1007/978-3-030-23611-3\\_3](https://doi.org/10.1007/978-3-030-23611-3_3) (cit. on pp. 7, 134, 136).
- [134] Jens Kosiol, Lars Fritsche, Andy Schürr, and Gabriele Taentzer. “Double-pushout-rewriting in  $S$ -Cartesian functor categories: Rewriting theory and application to partial triple graphs”. In: *J. Log. Algebraic Methods Program.* 115 (2020), p. 100565. URL: <https://doi.org/10.1016/j.jlamp.2020.100565> (cit. on pp. 7, 134).
- [135] Jens Kosiol, Daniel Strüber, Gabriele Taentzer, and Steffen Zschaler. “Graph Consistency as a Graduated Property – Consistency-Sustaining and -Improving Graph Transformations”. In: *Graph Transformation – 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25–26, 2020, Proceedings*. Ed. by Fabio Gadducci and Timo Kehrer. Vol. 12150. Lecture Notes in Computer Science. Springer, 2020, pp. 239–256. ISBN: 978-3-030-51371-9. URL: [https://doi.org/10.1007/978-3-030-51372-6\\_14](https://doi.org/10.1007/978-3-030-51372-6_14) (cit. on pp. 10, 187).
- [136] Jens Kosiol, Daniel Strüber, Gabriele Taentzer, and Steffen Zschaler. “Sustaining and improving graduated graph consistency: A static analysis of graph transformations”. In: *Sci. Comput. Program.* 214 (2022), p. 102729. URL: <https://doi.org/10.1016/j.scico.2021.102729> (cit. on p. 10).
- [137] Jens Kosiol and Gabriele Taentzer. “A Generalized Concurrent Rule Construction for Double-Pushout Rewriting”. In: *Graph Transformation – 14th International Conference, ICGT 2021, Held as Part of STAF 2021, Virtual Event, June 24–25, 2021, Proceedings*. Ed. by Fabio Gadducci and Timo Kehrer. Vol. 12741. Lecture Notes in Computer Science. Springer, 2021, pp. 22–39. ISBN: 978-3-030-78945-9. URL: [https://doi.org/10.1007/978-3-030-78946-6\\_2](https://doi.org/10.1007/978-3-030-78946-6_2) (cit. on pp. 7, 83).
- [138] Jens Kosiol and Gabriele Taentzer. *A Generalized Concurrent Rule Construction for Double-Pushout Rewriting. Generalized Concurrency Theorem and Language-Preserving Rule Applications*. Under review. 2022 (cit. on pp. 7, 83, 114).
- [139] Hans-Jörg Kreowski. “Is parallelism already concurrency? Part 1: Derivations in graph grammars”. In: *Graph-Grammars and Their Application to Computer Science, 3rd International Workshop, Warrenton, Virginia, USA, December 2–6, 1986*. Ed. by Hartmut Ehrig, Manfred Nagl, Grzegorz Rozenberg, and Azriel Rosenfeld. Vol. 291. Lecture Notes in Computer Science. Springer, 1986, pp. 343–360. ISBN: 3-540-18771-5. URL: [https://doi.org/10.1007/3-540-18771-5\\_63](https://doi.org/10.1007/3-540-18771-5_63) (cit. on pp. 61, 127).

- [140] Adrian Kuhn, Gail C. Murphy, and C. Albert Thompson. “An Exploratory Study of Forces and Frictions Affecting Large-Scale Model-Driven Development”. In: *Model Driven Engineering Languages and Systems – 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30–October 5, 2012. Proceedings*. Ed. by Robert B. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson. Vol. 7590. Lecture Notes in Computer Science. Springer, 2012, pp. 352–367. URL: [https://doi.org/10.1007/978-3-642-33666-9\\_23](https://doi.org/10.1007/978-3-642-33666-9_23) (cit. on p. 274).
- [141] Stephen Lack and Paweł Sobociński. “Adhesive and quasiadhesive categories”. In: *RAIRO Theor. Informatics Appl.* 39.3 (2005), pp. 511–545. URL: <https://doi.org/10.1051/ita:2005028> (cit. on pp. 5, 46, 49–51, 74, 82, 128, 131).
- [142] Stephen Lack and Paweł Sobociński. “Toposes Are Adhesive”. In: *Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17–23, 2006, Proceedings*. Ed. by Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg. Vol. 4178. Lecture Notes in Computer Science. Springer, 2006, pp. 184–198. ISBN: 3-540-38870-2. URL: [https://doi.org/10.1007/11841883\\_14](https://doi.org/10.1007/11841883_14) (cit. on p. 151).
- [143] Leen Lambers, Kristopher Born, Jens Kosiol, Daniel Strüber, and Gabriele Taentzer. “Granularity of conflicts and dependencies in graph transformation systems: A two-dimensional approach”. In: *J. Log. Algebraic Methods Program.* 103 (2019), pp. 105–129. URL: <https://doi.org/10.1016/j.jlamp.2018.11.004> (cit. on p. 10).
- [144] Leen Lambers, Hartmut Ehrig, Ulrike Prange, and Fernando Orejas. “Parallelism and Concurrency in Adhesive High-Level Replacement Systems with Negative Application Conditions”. In: *Electronic Notes in Theoretical Computer Science* 203.6 (2008). Proceedings of the Second Workshop on Applied and Computational Category Theory (ACCAT 2007), pp. 43–66. DOI: <https://doi.org/10.1016/j.entcs.2008.10.042>. URL: <http://www.sciencedirect.com/science/article/pii/S1571066108004362> (cit. on pp. 128, 265).
- [145] Leen Lambers, Stephan Hildebrandt, Holger Giese, and Fernando Orejas. “Attribute Handling for Bidirectional Model Transformations: The Triple Graph Grammar Case”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 49 (2012). URL: <https://doi.org/10.14279/tuj.eceasst.49.706> (cit. on pp. 5, 186, 264).

- [146] Leen Lambers, Jens Kosiol, Daniel Strüber, and Gabriele Taentzer. “Exploring Conflict Reasons for Graph Transformation Systems”. In: *Graph Transformation – 12th International Conference, ICGT 2019, Held as Part of STAF 2019, Eindhoven, The Netherlands, July 15–16, 2019, Proceedings*. Ed. by Esther Guerra and Fernando Orejas. Vol. 11629. Lecture Notes in Computer Science. Springer, 2019, pp. 75–92. URL: [https://doi.org/10.1007/978-3-030-23611-3\\_5](https://doi.org/10.1007/978-3-030-23611-3_5) (cit. on p. 10).
- [147] Saunders Mac Lane. *Categories for the Working Mathematician*. 2nd ed. Vol. 5. Graduate Texts in Mathematics. New York, 1997. DOI: [10.1007/978-1-4757-4721-8](https://doi.org/10.1007/978-1-4757-4721-8) (cit. on p. 10).
- [148] Kevin Lano and Shekoufeh Kolahdouz Rahimi. “Implementing QVT-R via semantic interpretation in UML-RSDS”. In: *Softw. Syst. Model.* 20.3 (2021), pp. 725–766. URL: <https://doi.org/10.1007/s10270-020-00824-3> (cit. on p. 15).
- [149] Marius Lauder, Anthony Anjorin, Gergely Varró, and Andy Schürr. “Efficient Model Synchronization with Precedence Triple Graph Grammars”. In: *Graph Transformations – 6th International Conference, ICGT 2012, Bremen, Germany, September 24–29, 2012. Proceedings*. Ed. by Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Vol. 7562. Lecture Notes in Computer Science. Springer, 2012, pp. 401–415. ISBN: 978-3-642-33653-9. URL: [https://doi.org/10.1007/978-3-642-33654-6\\_27](https://doi.org/10.1007/978-3-642-33654-6_27) (cit. on pp. 5, 8, 23, 25, 28, 37, 179, 262).
- [150] Erhan Leblebici. “Inter-Model Consistency Checking and Restoration with Triple Graph Grammars”. PhD. Darmstadt University of Technology, Germany, 2018. URL: <http://tuprints.ulb.tu-darmstadt.de/7426/> (cit. on pp. 5, 8, 9, 19, 23, 26–28, 36, 37, 179, 207, 213, 215, 232, 247–250, 257, 259, 264, 265).
- [151] Erhan Leblebici, Anthony Anjorin, Lars Fritsche, Gergely Varró, and Andy Schürr. “Leveraging Incremental Pattern Matching Techniques for Model Synchronisation”. In: *Graph Transformation – 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18–19, 2017, Proceedings*. Ed. by Juan de Lara and Detlef Plump. Vol. 10373. Lecture Notes in Computer Science. Springer, 2017, pp. 179–195. ISBN: 978-3-319-61469-4. URL: [https://doi.org/10.1007/978-3-319-61470-0\\_11](https://doi.org/10.1007/978-3-319-61470-0_11) (cit. on pp. 5, 8, 26, 28, 34, 36, 37, 179, 232).
- [152] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. “Developing eMoflon with eMoflon”. In: *Theory and Practice of Model Transformations – 7th International Conference, ICMT@STAF 2014, York, UK, July 21–22, 2014. Proceedings*. Ed. by Davide Di Ruscio and Dániel Varró. Vol. 8568. Lecture

- Notes in Computer Science. Springer, 2014, pp. 138–145. ISBN: 978-3-319-08788-7. URL: [https://doi.org/10.1007/978-3-319-08789-4\\_10](https://doi.org/10.1007/978-3-319-08789-4_10) (cit. on p. 20).
- [153] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. “Inter-model Consistency Checking Using Triple Graph Grammars and Linear Optimization Techniques”. In: *Fundamental Approaches to Software Engineering – 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings*. Ed. by Marieke Huisman and Julia Rubin. Vol. 10202. Lecture Notes in Computer Science. Springer, 2017, pp. 191–207. URL: [https://doi.org/10.1007/978-3-662-54494-5\\_11](https://doi.org/10.1007/978-3-662-54494-5_11) (cit. on pp. 19, 27, 259, 265).
- [154] Erhan Leblebici, Anthony Anjorin, Andy Schürr, Stephan Hildebrandt, Jan Rieke, and Joel Greenyer. “A Comparison of Incremental Triple Graph Grammar Tools”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 67 (2014). URL: <https://doi.org/10.14279/tuj.eceasst.67.939> (cit. on p. 20).
- [155] Erhan Leblebici, Anthony Anjorin, Andy Schürr, and Gabriele Taentzer. “Multi-amalgamated triple graph grammars: Formal foundation and application to visual language translation”. In: *J. Vis. Lang. Comput.* 42 (2017), pp. 99–121. URL: <https://doi.org/10.1016/j.jvlc.2016.03.001> (cit. on pp. 5, 30).
- [156] Michael Löwe. “Algebraic Approach to Single-Pushout Graph Transformation”. In: *Theor. Comput. Sci.* 109.1&2 (1993), pp. 181–224. URL: [https://doi.org/10.1016/0304-3975\(93\)90068-5](https://doi.org/10.1016/0304-3975(93)90068-5) (cit. on pp. 81, 177).
- [157] Michael Löwe. “Polymorphic Sesqui-Pushout Graph Rewriting”. In: *Graph Transformation – 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L’Aquila, Italy, July 21–23, 2015. Proceedings*. Ed. by Francesco Parisi-Presicce and Bernhard Westfechtel. Vol. 9151. Lecture Notes in Computer Science. Springer, 2015, pp. 3–18. ISBN: 978-3-319-21144-2. URL: [https://doi.org/10.1007/978-3-319-21145-9\\_1](https://doi.org/10.1007/978-3-319-21145-9_1) (cit. on pp. 58, 59, 81, 205).
- [158] Nuno Macedo and Alcino Cunha. “Least-change bidirectional model transformation with QVT-R and ATL”. In: *Softw. Syst. Model.* 15.3 (2016), pp. 783–810. URL: <https://doi.org/10.1007/s10270-014-0437-x> (cit. on pp. 19, 256).

- [159] Nuno Macedo, Tiago Guimarães, and Alcino Cunha. “Model repair and transformation with Echo”. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11–15, 2013*. Ed. by Ewen Denney, Tevfik Bultan, and Andreas Zeller. IEEE, 2013, pp. 694–697. URL: <https://doi.org/10.1109/ASE.2013.6693135> (cit. on p. 19).
- [160] Rodrigo Machado. “Higher-order graph rewriting systems”. PhD. Universidade Federal do Rio Grande do Sul, 2012. URL: <https://lume.ufrgs.br/handle/10183/54887> (visited on 10/21/2020) (cit. on pp. 132, 133, 175, 178).
- [161] Rodrigo Machado, Leila Ribeiro, and Reiko Heckel. “Rule-based transformation of graph rewriting rules: Towards higher-order graph grammars”. In: *Theor. Comput. Sci.* 594 (2015), pp. 1–23. URL: <https://doi.org/10.1016/j.tcs.2015.01.034> (cit. on pp. 132, 133, 136, 175, 178).
- [162] Solomon Maina, Anders Miltner, Kathleen Fisher, Benjamin C. Pierce, David Walker, and Steve Zdancewic. “Synthesizing quotient lenses”. In: *Proc. ACM Program. Lang.* 2.ICFP (2018), 80:1–80:29. URL: <https://doi.org/10.1145/3236775> (cit. on p. 17).
- [163] Ernest G. Manes. “Implementing Collection Classes with Monads”. In: *Math. Struct. Comput. Sci.* 8.3 (1998), pp. 231–276. DOI: [10.1017/S0960129598002515](https://doi.org/10.1017/S0960129598002515). URL: <http://journals.cambridge.org/action/displayAbstract?aid=44747> (cit. on p. 353).
- [164] Kazutaka Matsuda, Zhenjiang Hu, Keisuke Nakano, Makoto Hamana, and Masato Takeichi. “Bidirectionalization transformation based on automatic derivation of view complement functions”. In: *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP 2007, Freiburg, Germany, October 1–3, 2007*. Ed. by Ralf Hinze and Norman Ramsey. ACM, 2007, pp. 47–58. URL: <https://doi.org/10.1145/1291151.1291162> (cit. on p. 17).
- [165] Maria Maximova, Hartmut Ehrig, and Claudia Ermel. “Formal Relationship between Petri Net and Graph Transformation Systems based on Functors between M-adhesive Categories”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 40 (2010). URL: <https://doi.org/10.14279/tuj.eceasst.40.628> (cit. on p. 174).
- [166] Maria Maximova, Hartmut Ehrig, and Claudia Ermel. “Local confluence analysis of hypergraph transformation systems with application conditions based on M-functors and AGG”. In: *Sci. Comput. Program.* 104 (2015), pp. 44–70. URL: <https://doi.org/10.1016/j.scico.2014.10.002> (cit. on pp. 174, 175).

- [167] Anders Miltner, Kathleen Fisher, Benjamin C. Pierce, David Walker, and Steve Zdancewic. “Synthesizing bijective lenses”. In: *Proc. ACM Program. Lang.* 2.POPL (2018), 1:1–1:30. URL: <https://doi.org/10.1145/3158089> (cit. on p. 17).
- [168] Anders Miltner, Solomon Maina, Kathleen Fisher, Benjamin C. Pierce, David Walker, and Steve Zdancewic. “Synthesizing symmetric lenses”. In: *Proc. ACM Program. Lang.* 3.ICFP (2019), 95:1–95:28. URL: <https://doi.org/10.1145/3341699> (cit. on p. 17).
- [169] *MOF Query/View/Transformation. Version 1.3*. OMG. June 2016. URL: <https://www.omg.org/spec/QVT> (visited on 07/31/2021) (cit. on p. 14).
- [170] Parastoo Mohagheghi, Miguel A. Fernández, Juan A. Martell, Mathias Fritzsche, and Wasif Gilani. “MDE Adoption in Industry: Challenges and Success Criteria”. In: *Models in Software Engineering, Workshops and Symposia at MODELS 2008, Toulouse, France, September 28 – October 3, 2008. Reports and Revised Selected Papers*. Ed. by Michel R. V. Chaudron. Vol. 5421. Lecture Notes in Computer Science. Springer, 2008, pp. 54–59. URL: [https://doi.org/10.1007/978-3-642-01648-6\\_6](https://doi.org/10.1007/978-3-642-01648-6_6) (cit. on p. 274).
- [171] Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, and Miguel A. Fernández. “An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases”. In: *Empir. Softw. Eng.* 18.1 (2013), pp. 89–116. URL: <https://doi.org/10.1007/s10664-012-9196-x> (cit. on p. 274).
- [172] Ugo Montanari and Leila Ribeiro. “Linear Ordered Graph Grammars and Their Algebraic Foundations”. In: *Graph Transformation, First International Conference, ICGT 2002, Barcelona, Spain, October 7–12, 2002, Proceedings*. Ed. by Andrea Corradini, Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Vol. 2505. Lecture Notes in Computer Science. Springer, 2002, pp. 317–333. ISBN: 3-540-44310-X. URL: [https://doi.org/10.1007/3-540-45832-8\\_24](https://doi.org/10.1007/3-540-45832-8_24) (cit. on p. 156).
- [173] A. R. Shir Ali Nasab and S. N. Hosseini. “Pullback in Partial Morphism Categories”. In: *Appl. Categorical Struct.* 25.2 (2017), pp. 197–225. URL: <https://doi.org/10.1007/s10485-015-9420-0> (cit. on p. 177).
- [174] Nebras Nassar, Jens Kosiol, Thorsten Arendt, and Gabriele Taentzer. “OCL2AC: Automatic Translation of OCL Constraints to Graph Constraints and Application Conditions for Transformation Rules”. In: *Graph Transformation – 11th International Conference, ICGT 2018, Held as Part of STAF 2018, Toulouse, France, June 25–26, 2018, Proceedings*. Ed. by Leen Lambers and Jens H. Weber. Vol. 10887. Lecture Notes in Computer

- Science. Springer, 2018, pp. 171–177. URL: [https://doi.org/10.1007/978-3-319-92991-0\\_11](https://doi.org/10.1007/978-3-319-92991-0_11) (cit. on pp. 10, 187).
- [175] Nebras Nassar, Jens Kosiol, Thorsten Arendt, and Gabriele Taentzer. “Constructing Optimized Validity-Preserving Application Conditions for Graph Transformation Rules”. In: *Graph Transformation – 12th International Conference, ICGT 2019, Held as Part of STAF 2019, Eindhoven, The Netherlands, July 15–16, 2019, Proceedings*. Ed. by Esther Guerra and Fernando Orejas. Vol. 11629. Lecture Notes in Computer Science. Springer, 2019, pp. 177–194. URL: [https://doi.org/10.1007/978-3-030-23611-3\\_11](https://doi.org/10.1007/978-3-030-23611-3_11) (cit. on pp. 10, 187).
- [176] Nebras Nassar, Jens Kosiol, Thorsten Arendt, and Gabriele Taentzer. “Constructing optimized constraint-preserving application conditions for model transformation rules”. In: *J. Log. Algebraic Methods Program.* 114 (2020), p. 100564. URL: <https://doi.org/10.1016/j.jlamp.2020.100564> (cit. on pp. 10, 187).
- [177] Nebras Nassar, Jens Kosiol, Timo Kehrer, and Gabriele Taentzer. “Generating Large EMF Models Efficiently – A Rule-Based, Configurable Approach”. In: *Fundamental Approaches to Software Engineering – 23rd International Conference, FASE 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings*. Ed. by Heike Wehrheim and Jordi Cabot. Vol. 12076. Lecture Notes in Computer Science. Springer, 2020, pp. 224–244. URL: [https://doi.org/10.1007/978-3-030-45234-6\\_11](https://doi.org/10.1007/978-3-030-45234-6_11) (cit. on p. 10).
- [178] Nebras Nassar, Jens Kosiol, and Hendrik Radke. “Rule-based Repair of EMF Models: Formalization and Correctness Proof”. In: *Eighth International Workshop on Graph Computation Models – Electronic Pre-proceedings*. Ed. by Andrea Corradini. 2017. URL: <http://pages.di.unipi.it/corradini/Workshops/GCM2017/papers/Nassar-Kosiol-Radke-GCM2017.pdf> (cit. on pp. 10, 273).
- [179] Nebras Nassar, Hendrik Radke, and Thorsten Arendt. “Rule-Based Repair of EMF Models: An Automated Interactive Approach”. In: *Theory and Practice of Model Transformation – 10th International Conference, ICMT@STAF 2017, Marburg, Germany, July 17–18, 2017, Proceedings*. Ed. by Esther Guerra and Mark van den Brand. Vol. 10374. Lecture Notes in Computer Science. Springer, 2017, pp. 171–181. URL: [https://doi.org/10.1007/978-3-319-61473-1\\_12](https://doi.org/10.1007/978-3-319-61473-1_12) (cit. on p. 273).
- [180] Russell O’Connor. *Functor is to Lens as Applicative is to Biplate: Introducing Multiplate*. 2011. arXiv: 1103.2841 [cs.PL]. URL: <http://arxiv.org/abs/1103.2841> (cit. on p. 17).

- [181] *Object Constraint Language*. OMG. Feb. 2014. URL: <http://www.omg.org/spec/OCL/> (cit. on pp. 263, 275).
- [182] Manuel Ohrndorf, Christopher Pietsch, Udo Kelter, and Timo Kehrer. “ReVision: a tool for history-based model repair recommendations”. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ICSE 2018, Gothenburg, Sweden, May 27 – June 03, 2018*. Ed. by Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman. ACM, 2018, pp. 105–108. ISBN: 978-1-4503-5663-3. URL: <https://doi.org/10.1145/3183440.3183498> (cit. on p. 266).
- [183] Fernando Orejas, Artur Boronat, Hartmut Ehrig, Frank Hermann, and Hanna Schölzel. “On Propagation-Based Concurrent Model Synchronization”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 57 (2013). URL: <https://doi.org/10.14279/tuj.eceasst.57.871> (cit. on pp. 2, 190).
- [184] Fernando Orejas and Leen Lambers. “Symbolic Attributed Graphs for Attributed Graph Transformation”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 30 (2010). URL: <https://doi.org/10.14279/tuj.eceasst.30.405> (cit. on pp. 49, 153, 186).
- [185] Fernando Orejas and Elvira Pino. “Correctness of Incremental Model Synchronization with Triple Graph Grammars”. In: *Theory and Practice of Model Transformations – 7th International Conference, ICMT@STAF 2014, York, UK, July 21–22, 2014. Proceedings*. Ed. by Davide Di Ruscio and Dániel Varró. Vol. 8568. Lecture Notes in Computer Science. Springer, 2014, pp. 74–90. ISBN: 978-3-319-08788-7. URL: [https://doi.org/10.1007/978-3-319-08789-4\\_6](https://doi.org/10.1007/978-3-319-08789-4_6) (cit. on pp. 5, 23, 25, 27, 28, 181, 190, 256, 257, 266).
- [186] Fernando Orejas, Elvira Pino, and Marisa Navarro. “Incremental Concurrent Model Synchronization using Triple Graph Grammars”. In: *Fundamental Approaches to Software Engineering – 23rd International Conference, FASE 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings*. Ed. by Heike Wehrheim and Jordi Cabot. Vol. 12076. Lecture Notes in Computer Science. Springer, 2020, pp. 273–293. URL: [https://doi.org/10.1007/978-3-030-45234-6\\_14](https://doi.org/10.1007/978-3-030-45234-6_14) (cit. on pp. 2, 29, 269).
- [187] Ulrike Prange, Hartmut Ehrig, and Leen Lambers. “Construction and Properties of Adhesive and Weak Adhesive High-Level Replacement Categories”. In: *Appl. Categorical Struct.* 16.3 (2008), pp. 365–388. URL: <https://doi.org/10.1007/s10485-007-9106-3> (cit. on pp. 72, 149, 152, 336).

- [188] Hendrik Radke, Thorsten Arendt, Jan Steffen Becker, Annegret Habel, and Gabriele Taentzer. “Translating essential OCL invariants to nested graph constraints for generating instances of meta-models”. In: *Sci. Comput. Program.* 152 (2018), pp. 38–62. URL: <https://doi.org/10.1016/j.scico.2017.08.006> (cit. on p. 263).
- [189] Arend Rensink. “Representing First-Order Logic Using Graphs”. In: *Graph Transformations, Second International Conference, ICGT 2004, Rome, Italy, September 28–October 2, 2004, Proceedings*. Ed. by Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg. Vol. 3256. Lecture Notes in Computer Science. Springer, 2004, pp. 319–335. ISBN: 3-540-23207-9. URL: [https://doi.org/10.1007/978-3-540-30203-2\\_23](https://doi.org/10.1007/978-3-540-30203-2_23) (cit. on p. 53).
- [190] Edmund Robinson and Giuseppe Rosolini. “Categories of Partial Maps”. In: *Inf. Comput.* 79.2 (1988), pp. 95–130. URL: [https://doi.org/10.1016/0890-5401\(88\)90034-X](https://doi.org/10.1016/0890-5401(88)90034-X) (cit. on p. 66).
- [191] Christian Sandmann and Annegret Habel. “Rule-based Graph Repair”. In: *Proceedings Tenth International Workshop on Graph Computation Models, GCM@STAF 2019, Eindhoven, The Netherlands, 17th July 2019*. Ed. by Rachid Echahed and Detlef Plump. Vol. 309. EPTCS. 2019, pp. 87–104. URL: <https://doi.org/10.4204/EPTCS.309.5> (cit. on p. 273).
- [192] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012. ISBN: 978-3-642-17335-6. URL: <https://doi.org/10.1007/978-3-642-17336-3> (cit. on pp. 71, 339).
- [193] Sven Schneider, Leen Lambers, and Fernando Orejas. “A Logic-Based Incremental Approach to Graph Repair”. In: *Fundamental Approaches to Software Engineering – 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings*. Ed. by Reiner Hähnle and Wil M. P. van der Aalst. Vol. 11424. Lecture Notes in Computer Science. Springer, 2019, pp. 151–167. ISBN: 978-3-030-16721-9. URL: [https://doi.org/10.1007/978-3-030-16722-6\\_9](https://doi.org/10.1007/978-3-030-16722-6_9) (cit. on pp. 190, 273).
- [194] Johannes Schröpfer, Thomas Buchmann, and Bernhard Westfechtel. “A Generic Projectional Editor for EMF Models”. In: *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2020, Valletta, Malta, February 25–27, 2020*. Ed. by Slimane Hammoudi, Luís Ferreira Pires, and Bran Selic.

- SCITEPRESS, 2020, pp. 381–392. ISBN: 978-989-758-400-8. URL: <https://doi.org/10.5220/0008971003810392> (cit. on p. 274).
- [195] Andy Schürr. “Specification of Graph Translators with Triple Graph Grammars”. In: *Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG ’94, Herrsching, Germany, June 16–18, 1994, Proceedings*. Ed. by Ernst W. Mayr, Gunther Schmidt, and Gottfried Tinhofer. Vol. 903. Lecture Notes in Computer Science. Springer, 1994, pp. 151–163. ISBN: 3-540-59071-4. URL: [https://doi.org/10.1007/3-540-59071-4\\_45](https://doi.org/10.1007/3-540-59071-4_45) (cit. on pp. 3, 8, 20, 21, 30, 78, 80, 134, 165).
- [196] Andy Schürr and Felix Klar. “15 Years of Triple Graph Grammars”. In: *Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7–13, 2008. Proceedings*. Ed. by Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, and Gabriele Taentzer. Vol. 5214. Lecture Notes in Computer Science. Springer, 2008, pp. 411–425. URL: [https://doi.org/10.1007/978-3-540-87405-8\\_28](https://doi.org/10.1007/978-3-540-87405-8_28) (cit. on pp. 3, 5, 20, 21, 192).
- [197] Friedrich Steimann, Marcus Frenkel, and Markus Voelter. “Robust projectional editing”. In: *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2017, Vancouver, BC, Canada, October 23–24, 2017*. Ed. by Benoît Combemale, Marjan Mernik, and Bernhard Rumpe. ACM, 2017, pp. 79–90. ISBN: 978-1-4503-5525-4. URL: <https://doi.org/10.1145/3136014.3136034> (cit. on p. 274).
- [198] Perdita Stevens. “A Landscape of Bidirectional Model Transformations”. In: *Generative and Transformational Techniques in Software Engineering II, International Summer School, GTTSE 2007, Braga, Portugal, July 2–7, 2007. Revised Papers*. Ed. by Ralf Lämmel, Joost Visser, and João Saraiva. Vol. 5235. Lecture Notes in Computer Science. Springer, 2008, pp. 408–424. URL: [https://doi.org/10.1007/978-3-540-88643-3\\_10](https://doi.org/10.1007/978-3-540-88643-3_10) (cit. on p. 13).
- [199] Perdita Stevens. “Bidirectional model transformations in QVT: semantic issues and open questions”. In: *Softw. Syst. Model.* 9.1 (2010), pp. 7–20. URL: <https://doi.org/10.1007/s10270-008-0109-9> (cit. on pp. 14, 15, 192, 258).
- [200] Perdita Stevens. “Observations relating to the equivalences induced on model sets by bidirectional transformations”. In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 49 (2012). URL: <https://doi.org/10.14279/tuj.eceasst.49.714> (cit. on p. 14).

- [201] Perdita Stevens. “Maintaining consistency in networks of models: bidirectional transformations in the large”. In: *Softw. Syst. Model.* 19.1 (2020), pp. 39–65. URL: <https://doi.org/10.1007/s10270-019-00736-x> (cit. on p. 2).
- [202] Milica Stojkovic, Sven Laux, and Anthony Anjorin. “Existing and New Ideas on Least Change Triple Graph Grammars”. In: *Proceedings of the 6th International Workshop on Bidirectional Transformations co-located with The European Joint Conferences on Theory and Practice of Software, BX@ETAPS 2017, Uppsala, Sweden, April 29, 2017*. Ed. by Romina Eramo and Michael Johnson. Vol. 1827. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 1–5. URL: <http://ceur-ws.org/Vol-1827/paper2.pdf> (cit. on pp. 22, 26).
- [203] Daniel Strüber. “Generating Efficient Mutation Operators for Search-Based Model-Driven Engineering”. In: *Theory and Practice of Model Transformation – 10th International Conference, ICMT@STAF 2017, Marburg, Germany, July 17–18, 2017, Proceedings*. Ed. by Esther Guerra and Mark van den Brand. Vol. 10374. Lecture Notes in Computer Science. Springer, 2017, pp. 121–137. ISBN: 978-3-319-61472-4. URL: [https://doi.org/10.1007/978-3-319-61473-1\\_9](https://doi.org/10.1007/978-3-319-61473-1_9) (cit. on p. 132).
- [204] Patrick Stünkel, Harald König, Yngve Lamo, and Adrian Rutle. “Multi-model correspondence through inter-model constraints”. In: *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Nice, France, April 09–12, 2018*. Ed. by Stefan Marr and Jennifer B. Sartor. ACM, 2018, pp. 9–17. URL: <https://doi.org/10.1145/3191697.3191715> (cit. on pp. 133, 177).
- [205] Gabriele Taentzer, Claudia Ermel, Philip Langer, and Manuel Wimmer. “Conflict Detection for Model Versioning Based on Graph Modifications”. In: *Graph Transformations – 5th International Conference, ICGT 2010, Enschede, The Netherlands, September 27 – October 2, 2010, Proceedings*. Ed. by Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr. Vol. 6372. Lecture Notes in Computer Science. Springer, 2010, pp. 171–186. ISBN: 978-3-642-15927-5. URL: [https://doi.org/10.1007/978-3-642-15928-2\\_12](https://doi.org/10.1007/978-3-642-15928-2_12) (cit. on p. 190).
- [206] Gabriele Taentzer, Manuel Ohrndorf, Yngve Lamo, and Adrian Rutle. “Change-Preserving Model Repair”. In: *Fundamental Approaches to Software Engineering – 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings*. Ed. by Marieke Huisman and Julia Rubin. Vol. 10202. Lecture Notes in Computer Science. Springer, 2017, pp. 283–299. ISBN: 978-3-662-54493-8. URL:

- [https://doi.org/10.1007/978-3-662-54494-5\\_16](https://doi.org/10.1007/978-3-662-54494-5_16) (cit. on pp. 266, 273).
- [207] Stefan Tomaszek, Erhan Leblebici, Lin Wang, and Andy Schürr. “Model-driven Development of Virtual Network Embedding Algorithms with Model Transformation and Linear Optimization Techniques”. In: *Modellierung 2018, 21.–23. Februar 2018, Braunschweig, Germany*. Ed. by Ina Schaefer, Dimitris Karagiannis, Andreas Vogelsang, Daniel Méndez, and Christoph Seidl. Vol. P-280. LNI. Gesellschaft für Informatik e.V., 2018, pp. 39–54. URL: <https://dl.gi.de/20.500.12116/14957> (cit. on p. 20).
- [208] Javier Troya, Alexander Bergmayr, Loli Burgueño, and Manuel Wimmer. “Towards systematic mutations for and with ATL model transformations”. In: *Eighth IEEE International Conference on Software Testing, Verification and Validation, ICST 2015 Workshops, Graz, Austria, April 13–17, 2015*. IEEE Computer Society, 2015, pp. 1–10. ISBN: 978-1-4799-1885-0. URL: <https://doi.org/10.1109/ICSTW.2015.7107455> (cit. on p. 132).
- [209] Antonio Vetrò, Wolfgang Böhm, and Marco Torchiano. “On the Benefits and Barriers When Adopting Software Modelling and Model Driven Techniques – An External, Differentiated Replication”. In: *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2015, Beijing, China, October 22–23, 2015*. IEEE Computer Society, 2015, pp. 168–171. URL: <https://doi.org/10.1109/ESEM.2015.7321210> (cit. on p. 274).
- [210] Janis Voigtländer. “Bidirectionalization for free! (Pearl)”. In: *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21–23, 2009*. Ed. by Zhong Shao and Benjamin C. Pierce. ACM, 2009, pp. 165–176. URL: <https://doi.org/10.1145/1480881.1480904> (cit. on p. 17).
- [211] Janis Voigtländer, Zhenjiang Hu, Kazutaka Matsuda, and Meng Wang. “Combining syntactic and semantic bidirectionalization”. In: *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27–29, 2010*. Ed. by Paul Hudak and Stephanie Weirich. ACM, 2010, pp. 181–192. URL: <https://doi.org/10.1145/1863543.1863571> (cit. on p. 17).
- [212] Annika Wagner. “On the Expressive Power of Algebraic Graph Grammars with Application Conditions”. In: *TAPSOFT’95: Theory and Practice of Software Development, 6th International Joint Conference CAAP/FASE, Aarhus, Denmark, May 22–26, 1995, Proceedings*. Ed. by Peter D. Mosses, Mogens Nielsen, and Michael I. Schwartzbach. Vol. 915. Lecture Notes in Computer Science. Springer, 1995, pp. 409–423. ISBN: 3-540-59293-8. URL: [https://doi.org/10.1007/3-540-59293-8\\_210](https://doi.org/10.1007/3-540-59293-8_210) (cit. on p. 22).

- [213] Meng Wang, Jeremy Gibbons, and Nicolas Wu. “Incremental updates for efficient bidirectional transformations”. In: *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19–21, 2011*. Ed. by Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy. ACM, 2011, pp. 392–403. ISBN: 978-1-4503-0865-6. URL: <https://doi.org/10.1145/2034773.2034825> (cit. on pp. 17, 18).
- [214] Nils Weidmann and Anthony Anjorin. “Schema Compliant Consistency Management via Triple Graph Grammars and Integer Linear Programming”. In: *Fundamental Approaches to Software Engineering – 23rd International Conference, FASE 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings*. Ed. by Heike Wehrheim and Jordi Cabot. Vol. 12076. Lecture Notes in Computer Science. Springer, 2020, pp. 315–334. ISBN: 978-3-030-45233-9. URL: [https://doi.org/10.1007/978-3-030-45234-6\\_16](https://doi.org/10.1007/978-3-030-45234-6_16) (cit. on pp. 19, 27, 260, 265, 271, 273).
- [215] Nils Weidmann, Anthony Anjorin, Lars Fritsche, Gergely Varró, Andy Schürr, and Erhan Leblebici. “Incremental Bidirectional Model Transformation with eMoflon:IBeX”. In: *Proceedings of the 8th International Workshop on Bidirectional Transformations co-located with the Philadelphia Logic Week, Bx@PLW 2019, Philadelphia, PA, USA, June 4, 2019*. Ed. by James Cheney and Hsiang-Shang Ko. Vol. 2355. CEUR Workshop Proceedings. CEUR-WS.org, 2019, pp. 45–55. URL: <http://ceur-ws.org/Vol-2355/paper4.pdf> (cit. on pp. 4, 20).
- [216] Nils Weidmann, Anthony Anjorin, Erhan Leblebici, and Andy Schürr. “Consistency management via a combination of triple graph grammars and linear programming”. In: *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2019, Athens, Greece, October 20–22, 2019*. Ed. by Oscar Nierstrasz, Jeff Gray, and Bruno C. d. S. Oliveira. ACM, 2019, pp. 29–41. ISBN: 978-1-4503-6981-7. URL: <https://doi.org/10.1145/3357766.3359544> (cit. on pp. 19, 27, 259, 265, 271, 273).
- [217] Nils Weidmann, Lars Fritsche, and Anthony Anjorin. “A search-based and fault-tolerant approach to concurrent model synchronisation”. In: *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2020, Virtual Event, USA, November 16–17, 2020*. Ed. by Ralf Lämmel, Laurence Tratt, and Juan de Lara. ACM, 2020, pp. 56–71. ISBN: 978-1-4503-8176-5. URL: <https://doi.org/10.1145/3426425.3426932> (cit. on pp. 19, 26–29, 181, 192, 264, 265, 271, 273).

- [218] Nils Weidmann, Shubhangi Salunkhe, Anthony Anjorin, Enes Yigitbas, and Gregor Engels. “Automating Model Transformations for Railway Systems Engineering”. In: *Journal of Object Technology* 20.3 (June 2021). The 17th European Conference on Modelling Foundations and Applications (ECMFA 2021), 10:1–14. DOI: [10.5381/jot.2021.20.3.a10](https://doi.org/10.5381/jot.2021.20.3.a10). URL: [http://www.jot.fm/contents/issue\\_2021\\_03/article10.html](http://www.jot.fm/contents/issue_2021_03/article10.html) (cit. on p. 20).
- [219] Bernhard Westfechtel. “Case-based exploration of bidirectional transformations in QVT Relations”. In: *Softw. Syst. Model.* 17.3 (2018), pp. 989–1029. URL: <https://doi.org/10.1007/s10270-016-0527-z> (cit. on p. 14).
- [220] Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, and Hong Mei. “Towards automatic model synchronization from model transformations”. In: *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5–9, 2007, Atlanta, Georgia, USA*. Ed. by R. E. Kurt Stirewalt, Alexander Egyed, and Bernd Fischer. ACM, 2007, pp. 164–173. URL: <https://doi.org/10.1145/1321631.1321657> (cit. on p. 17).
- [221] Yingfei Xiong, Hui Song, Zhenjiang Hu, and Masato Takeichi. “Synchronizing concurrent model updates based on bidirectional transformation”. In: *Softw. Syst. Model.* 12.1 (2013), pp. 89–104. URL: <https://doi.org/10.1007/s10270-010-0187-3> (cit. on p. 29).

## A $\mathcal{E}'$ - $\mathcal{M}'$ Pair Factorization of Morphisms in $\mathbf{ATrG}_{\mathbf{ATG}}$

In this section, we present the  $\mathcal{E}'$ -quasi-injective factorization of pairs of quasi-injective morphisms that we have mentioned in Sect. 6.2.1. The intuition behind the factorization is that, since its second part is a quasi-injective morphism (and not, for instance, an  $\mathcal{M}$ -morphism), possible identifications of attribute values can still take place in that morphism and do not need to be encoded in the class  $\mathcal{E}'$  (that intuitively encodes the possibilities to identify structural elements). We only have to encode identifications of attribute values in  $\mathcal{E}'$  that are enforced by identifications of attribution edges. Since, for graph with finite structural part, there are only finitely many ways to identify attribution edges, this results also in only finitely many necessities to identify attribute values. We present the factorization for typed attributed triple graphs. In exactly the same way, it can be developed for typed attributed partial triple graphs, however.

**Definition A.1** ( $\mathcal{E}'$  with restricted identification of attribute values). In  $\mathbf{ATrG}_{\mathbf{ATG}}$ , let  $\mathcal{E}'$  be the class of pairs of morphisms  $(a' : A \rightarrow C, b' : B \rightarrow C)$  that have the following properties:

- Both  $a'$  and  $b'$  are quasi-injective and they share the same codomain.
- The morphisms  $a'$  and  $b'$  are jointly surjective on the structural part (in every component).
- For both domains, for every component  $X$ , with  $X \in \{\mathbf{S}, \mathbf{C}, \mathbf{T}\}$ ,  $A^X$  and  $B^X$  are attributed with term algebras.
- When  $T(Y)^X$  is the term algebra that attributes  $A^X$  and  $T(Z)^X$  is the term algebra that attributes  $B^X$ , then  $C^X$  is attributed with the term algebra

$$T(Y + Z)^X /_{\equiv} ,$$

where  $\equiv$  is generated by by all equations  $t_1 = t_2$  such that

- term  $t_1$  serves as target of an attribution edge in  $A^X$  and  $t_2$  as one in  $B^X$ , i.e., there exist edges  $e_1 \in NA_A^X \cup EA_A^X$  and  $e_2 \in NA_B^X \cup EA_B^X$  such that  $tar_j^X(e_i) = t_i$  for  $i = 1, 2$  and  $j \in \{NA_A, EA_A, NA_B, EA_B\}$  chosen appropriately, and
- morphisms  $a'$  and  $b'$  identify these attribution edges in  $C$ , i.e.,  $a'^X(e_1) = b'^X(e_2)$ .

The algebra parts of  $a'$  and  $b'$  then, i.e., the morphisms  $a'_A{}^X$  and  $b'_A{}^X$ , are the projections onto the quotient algebra  $T(Y + Z)^X / \equiv$ .

In the statement of the following lemma and in its proof we use the notation of the setting in which we apply this factorization.

**Lemma A.1** ( *$\mathcal{E}'$ -quasi-injective factorization of pairs of quasi-injective morphisms*). *In  $\mathbf{ATrG}_{ATG}$ , let  $\mathcal{E}'$  be the just defined class of pairs of morphisms and let  $\mathcal{M}'$  be the class of quasi-injective morphisms. Let  $(n : R_i \rightarrow H, q : C_j \rightarrow H)$  be a pair of quasi-injective morphisms with the same codomain whose domains are both attributed with term algebras. Then  $(n, q)$  has a  $\mathcal{E}'$ - $\mathcal{M}'$  factorization, i.e., there exists  $(a', b') \in \mathcal{E}'$  and  $q'$  quasi-injective such that  $n = q' \circ a'$  and  $q = q' \circ b'$ .*

*Furthermore, given triple graphs  $R_i$  and  $C_j$  attributed with term algebras, there are, up to isomorphism, only finitely many pairs of morphisms  $(n, q)$  such that  $(n, q) \in \mathcal{E}'$  and  $R_i$  and  $C_j$  are the domains of  $n$  and  $q$ , respectively.*

*Proof.* Let  $n : R_i \rightarrow H$  and  $q : C_j \rightarrow H$  be such a pair of quasi-injective morphisms with the same codomain. For the structural parts of  $n$  and  $q$  we can compute the ordinary jointly surjective-injective factorization and obtain the structural part  $(V_{C'}^X, E_{C'}^X, NA_{C'}^X, EA_{C'}^X)$ , where  $X \in \{S, C, T\}$ , of an typed attributed triple graph  $C'$ , together with the respective parts of morphisms  $a' : R_i \rightarrow C'$ ,  $b' : C_j \rightarrow C'$  and  $q' : C' \rightarrow H$ . We construct a suitable term algebra that completes  $C'$  into an attributed triple graph in such a way that  $a'$ ,  $b'$ , and  $q'$  have the desired properties.

For the construction, compare Fig. A.1. First, notice that  $T(Y + Z)^X$  is the coproduct of  $T(Y)^X$  and  $T(Z)^X$  (since building a term algebra is a free construction and free constructions preserve coproducts); let  $\iota_Y^X : T(Y)^X \rightarrow T(Y + Z)^X$  and  $\iota_Z^X : T(Z)^X \rightarrow T(Y + Z)^X$  be the coproduct

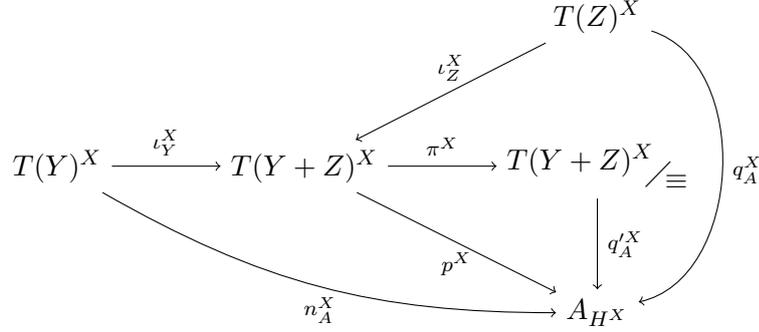


Figure A.1: The algebra homomorphism  $q_A'^X : T(Y+Z)^X / \equiv \rightarrow A_{H^X}$

injections. Let  $\equiv$  be generated by  $n$  and  $q$  as defined in the definition above, and let  $\pi^X : T(Y+Z)^X \rightarrow T(Y+Z)^X / \equiv$  be the resulting projection. Given the algebra  $A_{H^X}$  of  $H^X$  and the according algebra homomorphisms  $n_A^X : T(Y)^X \rightarrow A_{H^X}$  and  $q_A^X : T(Z)^X \rightarrow A_{H^X}$ , by the universal property of  $T(Y+Z)^X$  we obtain a morphism  $p^X : T(Y+Z)^X \rightarrow A_{H^X}$  such that  $p \circ \iota_Y^X = n_A^X$  and  $p \circ \iota_Z^X = q_A^X$ . This then ensures that the precondition of the Homomorphism Theorem [65, 3.13 Theorem] is met such that we obtain a homomorphism

$$q_A'^X : T(Y+Z)^X / \equiv \rightarrow A_{H^X}$$

with  $q_A'^X \circ \pi^X = p^X$ . We set  $a_A'^X := \pi^X \circ \iota_Y^X$  and  $b_A'^X := \pi^X \circ \iota_Z^X$ .

We then set  $A_{C'^X} := T(Y+Z)^X / \equiv$ , i.e., we take  $T(Y+Z)^X / \equiv$  as the algebras for  $C'$ . For  $C'$  to become an attributed triple graph, we still have to define the target functions  $(tar_j^X)_{j \in \{NA_{C'}, EA_{C'}\}}$  (the source functions are induced by  $n$  and  $q$ ). Since  $NA_{C'}^X$  and  $EA_{C'}^X$  have been obtained via the jointly-surjective-injective factorizations of  $n_{NA_{R_j}}^X$  and  $q_{NA_{C_i}}^X$  (or  $n_{EA_{R_j}}^X$  and  $q_{EA_{C_i}}^X$ , respectively), each such attribution edge has a preimage under one of these morphisms. For any  $e \in NA_{C'}^X$  we therefore define

$$tar_{NA_{C'}}^X(e) := a_A'^X(tar_{NA_{R_j}}^X((n_{NA_{R_j}}^X)^{-1}(e)))$$

or

$$tar_{NA_{C'}}^X(e) := b_A'^X(tar_{NA_{C_i}}^X((q_{NA_{C_i}}^X)^{-1}(e))) ,$$

depending on which of these cases applies (and analogously for each  $\text{tar}_{EA_{C'}}^X$ ). If this is well-defined, it is exactly the condition we need for  $a'$  and  $b'$  to become morphisms between attributed triple graphs. And it is because (i)  $n_{NA_{R_j}}^X$  and  $q_{NA_{C_i}}^X$  are injective by assumption (i.e., each of the functions has maximally one preimage for  $e$ ), and (ii) if both cases apply, we have

$$a'_A{}^X(\text{tar}_{NA_{R_j}}^X((n_{NA_{R_j}}^X)^{-1}(e))) = b'_A{}^X(\text{tar}_{NA_{C_i}}^X((q_{NA_{C_i}}^X)^{-1}(e)))$$

by definition of  $\equiv$ . Also  $(a', b') \in \mathcal{E}'$  and  $q' \circ a' = n$ ,  $q' \circ b' = q$  by construction. Moreover,  $q'$  is quasi-injective because its structural part is injective by construction.

Thus, it remains to show the claim about the finiteness. Given triple graphs  $R_i$  and  $C_j$  attributed with term algebras and with finite sets of attribution edges, there are only finitely many ways to overlap the attribution edges from  $R_i$  with those from  $C_j$ . Each of these ways fixes (up to isomorphism) a single term algebra  $T(Y + Z)^X / \equiv$  we have to consider in this case. Therefore, as long as the structural parts of  $R_i$  and  $C_j$  are finite, the claim holds.  $\square$

## B Further Proofs

### B.1 Proofs from Chapter 4

*Proof of Lemma 4.5.* First, compute the cube that is depicted in Fig. B.1 by computing the missing side faces as pullbacks: The pullbacks in the front and back exist since  $a_4, a_3 \in \mathcal{M}$ ; the resulting morphisms  $d, e, x, y$  are  $\mathcal{M}$ -morphisms as they result from pullbacks along  $\mathcal{M}$ -morphisms. The morphism  $\bar{x}_{f_1} : Y \rightarrow X$  such that the whole cube commutes is then induced by the universal property of  $X$  as pullback object; moreover, by pullback decomposition, the induced right side face is a pullback, as well. Since  $(\mathcal{C}, \mathcal{M})$  is  $\mathcal{M}$ -adhesive and  $a_3 \in \mathcal{M}$ , the pushout at the bottom of the cube has the weak vertical van Kampen property. Since  $a_2 \circ c_{f_1}, a_1 \circ b_{f_1}, e, d \in \mathcal{M}$ , this implies that the top face is a pushout as well. (If  $(\mathcal{C}, \mathcal{M})$  is even adhesive HLR and we drop the assumption  $a_2$  (and hence  $a_2 \circ c_{f_1}) \in \mathcal{M}$ , we cannot conclude  $e, d \in \mathcal{M}$ . However, the top square still is a pushout because the bottom pushout then even has the van Kampen property.) We prove the statement by proving that, without loss of generality,  $x$  and  $y$  are the identity morphisms of  $C_{f_1}$  and  $B_{f_1}$ , respectively.

Composing the top pushout with the initial pushout (1) results in a second pushout over  $f_1 : A_1 \rightarrow A_2$ . Since  $x, y \in \mathcal{M}$  also  $b_{f_1} \circ y, c_{f_1} \circ x \in \mathcal{M}$ . Then, by initiality of (1), we obtain  $\mathcal{M}$ -morphisms  $b_{f_1}^* : B_{f_1} \hookrightarrow Y$  and  $c_{f_1}^* : C_{f_1} \hookrightarrow X$  such that

$$b_{f_1} \circ y \circ b_{f_1}^* = b_{f_1} \text{ and } c_{f_1} \circ x \circ c_{f_1}^* = c_{f_1} ;$$

compare the top of Fig. B.1. Canceling the monomorphisms  $b_{f_1}$  and  $c_{f_1}$ , respectively, shows that  $x$  and  $y$  are split epi, and hence isomorphisms. Thus, without loss of generality,  $X = C_{f_1}$ ,  $Y = B_{f_1}$ ,  $x = id_{C_{f_1}}$ ,  $y = id_{B_{f_1}}$ , and  $\bar{x}_{f_1} = x_{f_1}$ . In particular,  $a_3 \circ d = a_1 \circ b_{f_1}$ ,  $a_4 \circ e = a_2 \circ c_{f_1}$ , and the desired square is a pullback.  $\square$

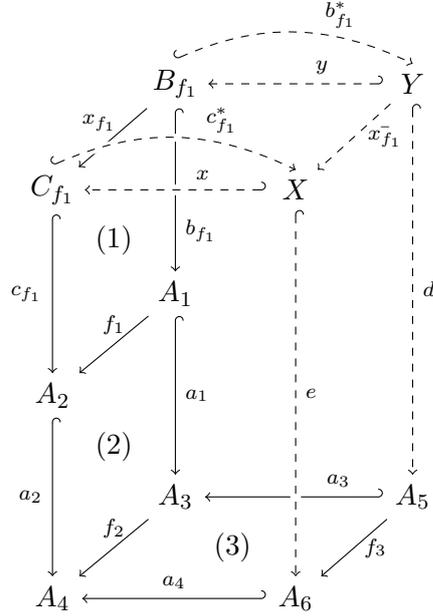


Figure B.1: Proving the interaction of initial pushouts with pullbacks

*Proof of Proposition 4.10.* First, under the given assumptions, both constructions are always applicable: The pushout complements for the first step of the computation exist since  $E$  is an  $E$ -dependency relation for the given pair of rules. All other steps only involve taking pushouts or pullbacks along  $\mathcal{M}$ -morphisms, which always exist in  $\mathcal{M}$ -adhesive categories.

For the rest of the proof, compare Fig. B.2. The figure basically depicts the bottom of Fig. 4.25 that illustrates Construction 4.7. Additionally, the pullback computing the interface  $K$  of the concurrent rule  $\rho_1 *_{E} \rho_2$  constitutes the top of the central cube. The morphism  $k'$  is the unique morphism induced by the universal property of  $K'$  such that the whole cube commutes. Furthermore, as both the top and the bottom square of the cube are pullbacks and the outer horizontal squares commute, we obtain unique morphisms  $p, p'$  with  $k_i \circ p = e''_i \circ u_i$  and  $k'_i \circ p' = s_i$  for  $i = 1, 2$ . In particular,  $p$  is the same morphism whose existence is guaranteed by Lemma 4.1. Furthermore, Construction 4.7 implies that the vertical squares

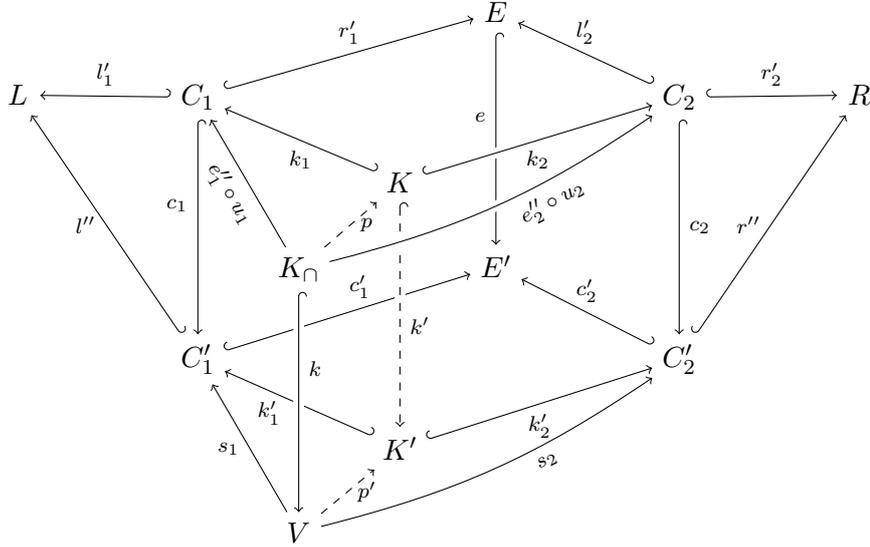


Figure B.2: Obtaining the morphism  $k'$  as pushout along  $k$

in the background are pushouts. Therefore, the cube pushout-pullback property implies that also the vertical squares in the front of the cube are.

Furthermore, applying  $\mathcal{M}$  pushout-pullback decomposition to the square  $K_\cap \rightarrow C_2 \hookrightarrow C'_2 \leftarrow V \leftarrow K_\cap$  (vertical front square of outer cube) implies that the square  $K_\cap \rightarrow K \hookrightarrow K' \leftarrow V \leftarrow K_\cap$  is a pushout.  $\mathcal{M}$  pushout-pullback decomposition is applicable since the outer square is a pushout by assumption, and the front square is in particular a pullback (as pushout along an  $\mathcal{M}$ -morphism); moreover, all necessary morphisms stem from  $\mathcal{M}$ .

This means that both constructions compute the same interface  $K'$  (up to unique isomorphism), namely the pushout of  $k$  and  $p$ . It remains to show that also the computed morphisms coincide. We show this for  $l'^1$  and  $l'^2$ ; the case of  $r'^1$  and  $r'^2$  is completely analogous. According to Construction 4.4,  $l'^1 : K' \rightarrow L$  is the unique morphism with  $l'^1 \circ p' = e'_1 \circ v_1$  and  $l'^1 \circ k' = l'_1 \circ k_1$ . We show that whenever  $l'^2 := l'' \circ k'_1$  is computed according to Construction 4.7, it also satisfies this property. This is the

case because

$$\begin{aligned} e'_1 \circ v_1 &= l'' \circ s_1 \\ &= l'' \circ k'_1 \circ p' \\ &= l'^2 \circ p' \end{aligned}$$

and

$$\begin{aligned} l'_1 \circ k_1 &= l'' \circ c_1 \circ k_1 \\ &= l'' \circ k'_1 \circ k' \\ &= l'^2 \circ k' . \end{aligned} \quad \square$$

In the following, we prove the results from Sect. 4.5.2. For this, we first introduce a technical lemma that states a useful property of Construction 4.7, which is needed in the following proofs. In the presence of  $\mathcal{M}$ -effective unions, the lemma reduces the question of whether  $l' \in \mathcal{M}$  to the question whether  $l'' \in \mathcal{M}$  in the construction of GCRs.

**Lemma B.1** (Reduction of linearity). *Let  $(\mathcal{C}, \mathcal{M})$  be an  $\mathcal{M}$ -adhesive category with  $\mathcal{M}$ -effective unions,  $\rho_1, \rho_2$  two rules, and  $E$  and  $k$  an  $E$ -dependency relation and common kernel for them that are compatible. Then  $\rho_1 *_{E,k} \rho_2$  is a generalized concurrent rule if and only if  $l'', r'' \in \mathcal{M}$ . In particular, in the notation of Fig. 4.25,  $l' := l'' \circ k'_1 \in \mathcal{M}$  if and only if  $l'' \in \mathcal{M}$  (and analogously for  $r''$ ).*

*Proof.* We prove  $l'' \in \mathcal{M} \iff l' \in \mathcal{M}$ ; the proof for  $r'' \in \mathcal{M} \iff r' \in \mathcal{M}$  is completely analogous.

First,  $k'_1 \in \mathcal{M}$  always holds by closedness of  $\mathcal{M}$  under pushouts and pullbacks. Therefore, by closedness of  $\mathcal{M}$  under composition,  $l'' \in \mathcal{M}$  implies  $l' = l'' \circ k'_1 \in \mathcal{M}$ .

For the second direction, i.e., given  $l' \in \mathcal{M}$ , we show that the outer square in Fig. B.3 is a pullback. As the inner square is a pushout (see the proof Proposition 4.10),  $l'' \circ c_1 = l'_1$  and  $l'' \circ k'_1 = l'$ , and  $l'_1 \in \mathcal{M}$ , in categories with  $\mathcal{M}$ -effective unions this means that  $l' \in \mathcal{M}$  implies  $l'' \in \mathcal{M}$ .

To see that this square is a pullback, we apply Lemma 2.2. from [82] (recalled as Lemma 3.4 in Sect. 3.1) to the diagram depicted in Fig. B.4. Since  $k', l'_1, k \in \mathcal{M}$ , this means we have to show that square (2) is a pushout and squares (3) – (6) are pullbacks. To see that square (6) (the

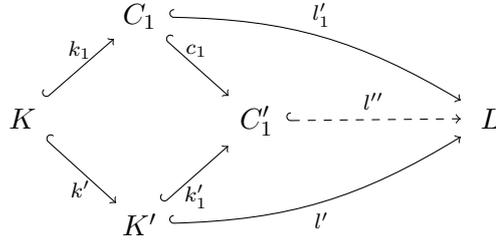


Figure B.3: Obtaining  $l'' \in \mathcal{M}$  by  $\mathcal{M}$ -effective unions

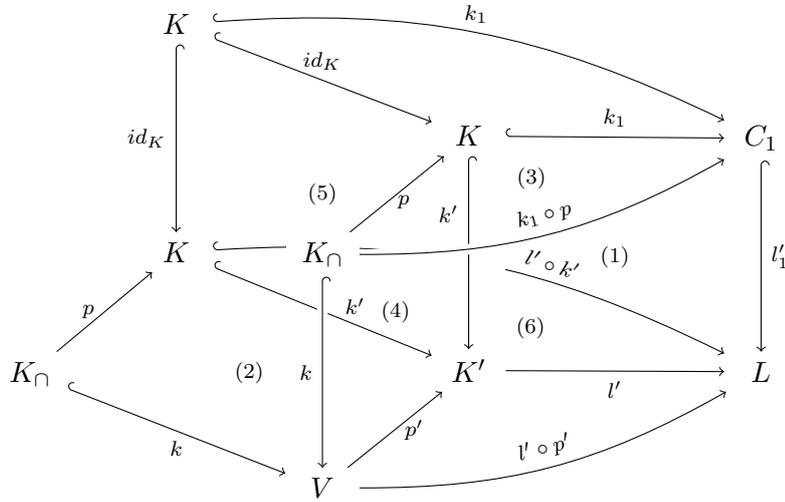


Figure B.4: Proving (1) (the outer square from Fig. B.3) to be a pullback

bent square in the front) is a pullback, observe that it coincides with the composition of the two pullback squares at the very left in the back of Fig. 4.25:  $l' \circ p' = l'' \circ k'_1 \circ p' = e'_1 \circ v_1$  and  $k_1 \circ p = e''_1 \circ u_1$  as in the proof of Proposition 4.10. Square (3) (the bent square in the back) is a pullback by Fact 3.9 (5.), using  $l'_1 \circ k_1 = l = l' \circ k'$  and the fact that  $l'_1$  is mono. Square (2) = (4) is a pushout (along the  $\mathcal{M}$ -morphism  $k$ ) according to the proof of Proposition 4.10; and thus, in particular, also a pullback. Square (5) is a pullback according to Fact 3.9 (3.) since  $k'$  is mono.  $\square$

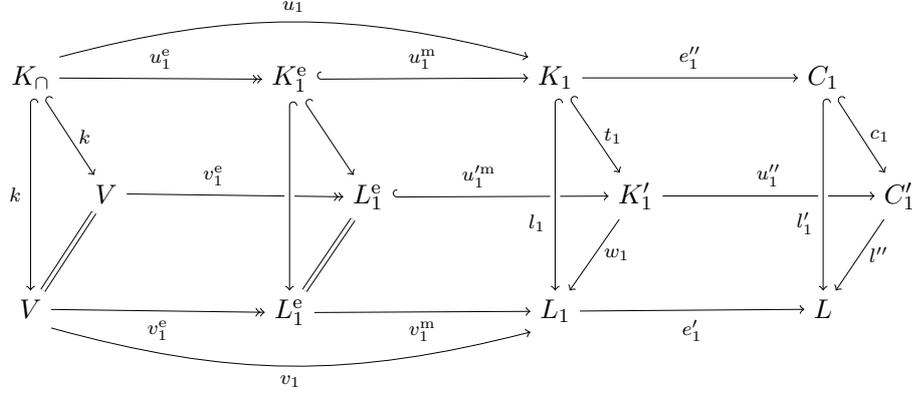


Figure B.5: Characterizing generalized concurrent rules

*Proof of Proposition 4.11.* We prove the statement for  $u_1, v_1$  as the case  $u_2, v_2$  is completely analogous. For both directions of the proof compare Fig. B.5. The diagram is obtained in the following way: Basically, it shows the left prism of Fig. 4.25. However, the pushout of  $k$  and  $e_1'' \circ u_1$  computing  $C'_1$  is split into three pushouts where  $u_1 = u_1^m \circ u_1^e$  is the epi- $\mathcal{M}$  factorization of  $u_1$ . This is possible since pushouts along  $\mathcal{M}$ -morphisms exist and  $\mathcal{M}$ -morphisms are closed under pushout. The morphisms  $w_1 : K'_1 \rightarrow L_1$  and  $v_1^m : L_1^e \rightarrow L_1$  are both obtained by the universal property of pushouts.

First, observe that  $v_1^e$  is an epi since epis are closed under pushout. Thus,  $v_1^m \circ v_1^e$  constitutes an epi- $\mathcal{M}$  factorization of  $v_1$  if and only if  $v_1^m \in \mathcal{M}$ . Moreover, by Lemma B.1,  $\rho_1 *_{E,k} \rho_2$  is a generalized concurrent rule if and only if  $l'' \in \mathcal{M}$ . Thus, we have to prove  $v_1^m \in \mathcal{M} \iff l'' \in \mathcal{M}$ . For both directions we will make use of the fact that  $u_1^m, t_1 \in \mathcal{M}$  (which holds by closedness of  $\mathcal{M}$ -morphisms under pushouts).

Assuming  $l'' \in \mathcal{M}$ , we obtain  $w_1 \in \mathcal{M}$  by closedness of  $\mathcal{M}$ -morphisms under pullback. That the right bottom square is indeed always a pullback (and not only a pushout) in adhesive HLR categories, holds by Lemma B.2, which we state and prove subsequently. And by  $v_1^m = w_1 \circ u_1^m$ ,  $w_1, u_1^m \in \mathcal{M}$  and closedness of  $\mathcal{M}$  under composition, we obtain  $v_1^m \in \mathcal{M}$ .

Assuming  $v_1^m \in \mathcal{M}$ ,  $w_1 \in \mathcal{M}$  follows by the existence of  $\mathcal{M}$ -effective unions. For this, we only need that the central square in the front is a pullback which holds by  $\mathcal{M}$  pullback-pushout decomposition (Lemma 3.3).

Thus,  $l'' \in \mathcal{M}$  by closedness of  $\mathcal{M}$  under pushouts.

The statement on the necessity of the existence of  $\mathcal{M}$ -effective unions follows directly from the analogous statement for the special case of  $\mathcal{M}$ -matching.  $\square$

**Lemma B.2.** *Let  $(\mathcal{C}, \mathcal{M})$  be an adhesive HLR category,  $\rho_i$ , where  $i = 1, 2$ , two rules, and  $E$  and  $k$  an  $E$ -dependency relation and a common kernel for them that are compatible. Then, the square  $K'_1 \xrightarrow{w_1} L_1 \xrightarrow{e'_1} L \xleftarrow{l''} C'_1 \xleftarrow{u'_1} K'_1$  (see Fig. B.5), occurring when applying the construction of generalized concurrent rules to that data, is always a pullback square. The same holds for the according square on the right side of the construction.*

*Proof.* First, square (2) in Fig. B.6 is a pullback. This follows applying  $\mathcal{M}$  pullback-pushout decomposition (Lemma 3.3) since (1) + (2) is a pullback, (1) a pushout,  $l_1 \in \mathcal{M}$ , and (2) commutes (all by assumption or construction).

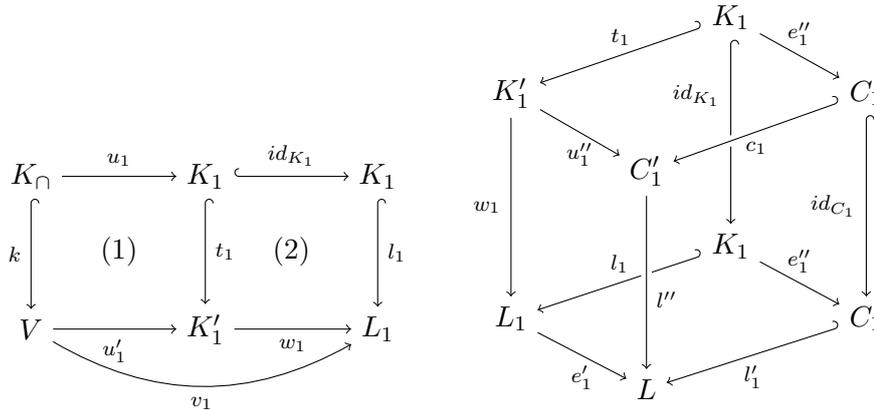


Figure B.6: Applying  $\mathcal{M}$  pullback-pushout decomposition

Figure B.7: Showing the front faces to be pullbacks

But this means that the bottom and the top square in Fig. B.7 are pushouts and the back faces pullbacks. Hence, the two front faces are pullbacks by the van Kampen property of the bottom pushout. In particular, as desired, the left front square is a pullback.  $\square$

*Proof of Corollary 4.12.* Compare Fig. B.8 for the following proof. Let square (1) constitute the initial pushout over  $k$ ; in particular, the square is also a pullback (as  $b_k \in \mathcal{M}$ ). Again,  $u_1 = u_1^m \circ u_1^e$  is an epi- $\mathcal{M}$  factorization of  $u_1$ , square (2) is a pushout, and  $v_1^m$  is obtained by its universal property.

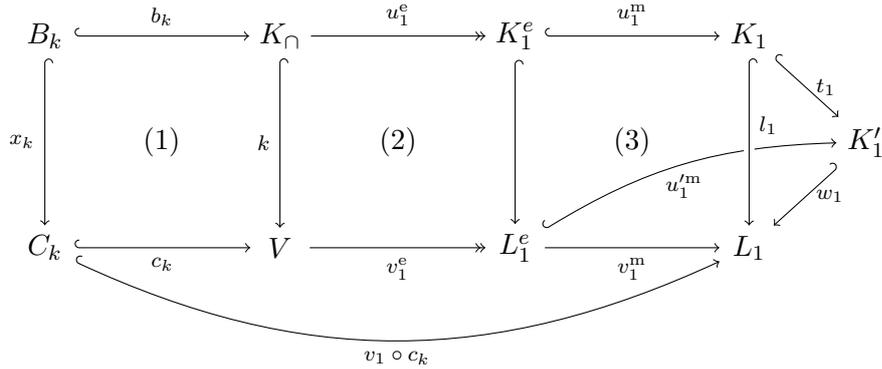


Figure B.8: Providing a sufficient criterion for Construction 4.7 to result in a generalized concurrent rule

First, the outer square (1)+(2)+(3) is a pullback by pullback composition: Both (1) and (2) + (3) are pullbacks by assumption. Moreover, (1) + (2) is a pushout by pushout composition. Thus, pulling back  $l_1$  and  $v_1 \circ c_k$  results in the span  $C_k \leftarrow B_k \hookrightarrow K_1$  and pushing out again in the co-span  $C_k \hookrightarrow K'_1 \leftarrow K_1$  (see also Fig. B.5). This means,  $v_1 \circ c_k \in \mathcal{M}$  implies  $w_1 \in \mathcal{M}$  (by  $\mathcal{M}$ -effective unions) which, in turn, implies  $v_1^m \in \mathcal{M}$  (by composition of  $\mathcal{M}$ -morphisms). In summary,  $v_1 \circ c_k \in \mathcal{M}$  implies that an epi- $\mathcal{M}$  factorization of  $v_1$  might be computed by pushing out  $u_1^e$  along  $k$ . Then, Proposition 4.11 guarantees a generalized concurrent rule to result from applying Construction 4.4 (or, equivalently, Construction 4.7).  $\square$

*Proof of Proposition 4.13.* First, assume  $(L \xleftarrow{l'} K' \xrightarrow{r'} R, ac)$  to be a generalized concurrent rule. Figure B.9 depicts the initial pushouts over  $k'$  and  $l'_1$  as squares (1) and (4), respectively. Since  $l' \in \mathcal{M}$  by assumption,  $l'' \in \mathcal{M}$  by Lemma B.1. This, in turn, implies that (3) is a pullback by application of Fact 3.9 (5.). Since (2) is also a pullback, (2) + (3) is one, and Lemma 4.5 implies the existence of  $\mathcal{M}$ -morphisms  $s_L$  and  $t_L$  such that the whole diagram commutes (in particular,  $s_L$  satisfies Eq. (4.1)) and the

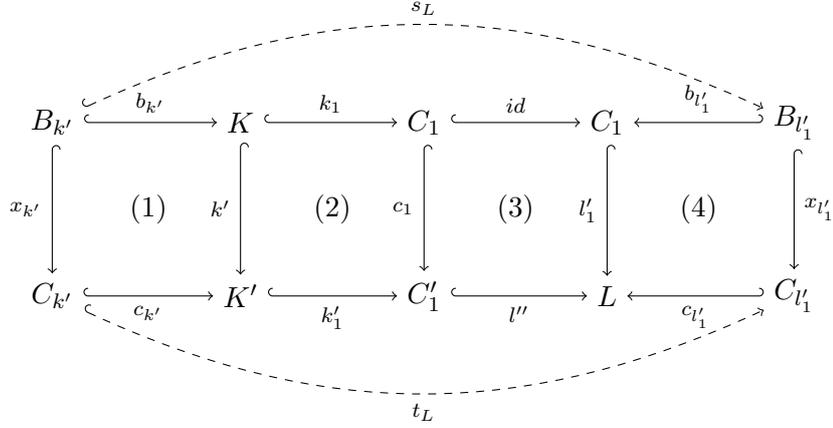


Figure B.9: Obtaining the factorization of the initial pushout over  $k'$  through the one over  $l'_1$

induced square  $B_{k'} \xrightarrow{s_L} B_{l'_1} \xrightarrow{x_{l'_1}} C_{l'_1} \xleftarrow{t_L} C_{k'} \xleftarrow{x_{k'}} B_{k'}$  is a pullback. The morphisms  $s_R, t_R$  are obtained completely analogously.

In the other direction, assume the initial pushout over  $k'$  to factor through the ones over  $l'_1$  and  $r'_2$  such that Eq. (4.1) holds. We have to construct a common kernel  $k : K_\cap \hookrightarrow V$  that is compatible with  $E$  and computes  $\rho_1 *_{E,k} \rho_2$ . First,  $K_\cap, u_1, u_2$  are computed by pulling back  $e_1 \circ r_1$  along  $e_2 \circ l_2$ ; thus, compatibility is satisfied. Next, since  $k'$  is given,  $C'_1, E', C'_2$  are determined as well as they arise by (iteratively) pushing out along  $k'$  (see, e.g., Fig. B.2). Thus, we have to find  $V, k, v_1, v_2$  such that (i) pushing out  $e''_1 \circ u_1, e_1 \circ r_1 \circ u_1$ , and  $e''_2 \circ u_2$  along  $k$  computes  $C'_1, E'$ , and  $C'_2$  (and the appropriate morphisms), respectively, and (ii) the squares induced by  $v_1, v_2$  constitute pullback squares (such that, with the morphisms  $u_i$  and  $v_i$ ,  $k : K_\cap \hookrightarrow V$  indeed constitutes a common kernel for  $\rho_1, \rho_2$ ).

For this, we are going to compute a span  $C'_{k'} \xleftarrow{x'_{k'}} B'_{k'} \xleftarrow{b'_{k'}} K_\cap$  such that we obtain a suitable common kernel  $k : K_\cap \hookrightarrow V$  (satisfying the above requirements) by pushing out that span. For the first part of that computation, compare Fig. B.10. The factorization of  $x_{k'}$  (as pullback)

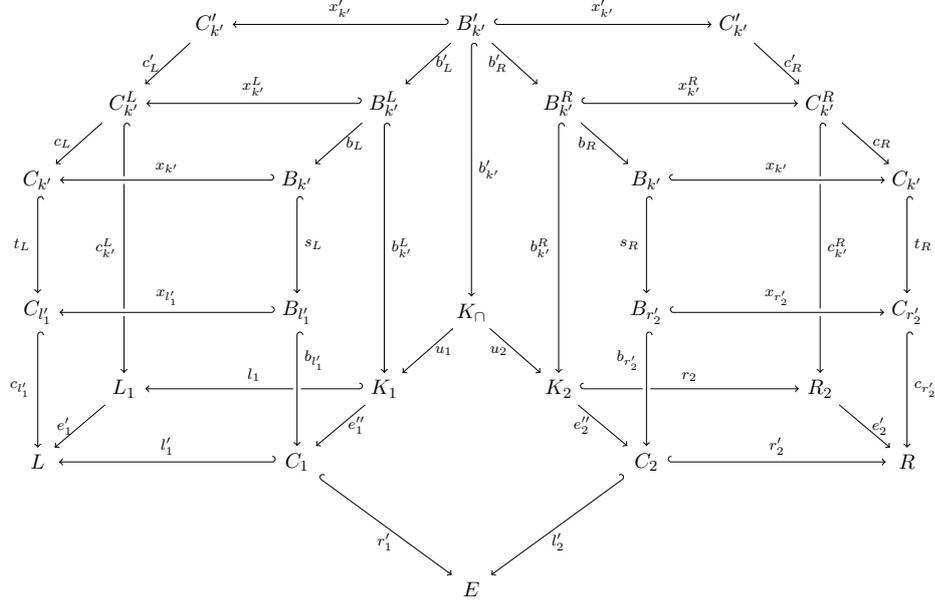


Figure B.10: Computing the span  $C'_{k'} \xleftarrow{x'_{k'}} B'_{k'} \xrightarrow{b'_{k'}} K_{\cap}$  needed to compute  $V$

through  $x'_{l'_1}$  and  $x'_{r'_2}$  is depicted at its front. Equation (4.1) implies that

$$\begin{aligned}
 r'_1 \circ b'_{l'_1} \circ s_L &= r'_1 \circ k_1 \circ b_{k'} \\
 &= l'_2 \circ k_2 \circ b_{k'} \\
 &= l'_2 \circ b'_{r'_2} \circ s_R
 \end{aligned}$$

(where we also use that  $r'_1 \circ k_1 = l'_2 \circ k_2$  by construction of the concurrent rule; cf. Fig. 3.11). Furthermore, we constructed  $u_1$  and  $u_2$  in such a way that also  $r'_1 \circ e''_1 \circ u_1 = l'_2 \circ e''_2 \circ u_2$ . Together with  $r'_1$  and  $l'_2$  being monic, these two equalities imply that pulling back  $b'_{l'_1} \circ s_L$  along  $e''_1 \circ u_1$  and  $b'_{r'_2} \circ s_R$  along  $e''_2 \circ u_2$  result in the same pullback object (up to unique isomorphism): One can instead compute the pullback of  $r'_1 \circ b'_{l'_1} \circ s_L = l'_2 \circ b'_{r'_2} \circ s_R$  along  $r'_1 \circ e''_1 \circ u_1 = l'_2 \circ e''_2 \circ u_2$ . The computation of that pullback (split into two pullbacks each) is depicted at the center of Fig. B.10 and results in an  $\mathcal{M}$ -morphism (since  $b'_{l'_1} \circ s_L \in \mathcal{M}$ )  $b'_{k'} : B'_{k'} \hookrightarrow K_{\cap}$ . The objects  $C'_{l'_1}$  and  $C'_{r'_2}$  (and the according morphisms) are also obtained via pullback;  $x'_{l'_1}$  and

$x_{k'}^R$  are induced by their universal properties. Finally,  $C_{k'}^L \xleftarrow{c'_L} C'_{k'} \xrightarrow{c'_R} C_{k'}^R$  is obtained by pulling back  $C_{k'}^L \xrightarrow{c_L} C_{k'} \xleftarrow{c_R} C_{k'}^R$ . Via the universal property of this pullback and

$$\begin{aligned} c_L \circ x_{k'}^L \circ b'_L &= x_{k'} \circ b_L \circ b'_L \\ &= x_{k'} \circ b_R \circ b'_R \\ &= c_R \circ x_{k'}^R \circ b'_R \end{aligned}$$

we obtain the morphism  $x'_{k'} : B'_{k'} \rightarrow C'_{k'}$  such that the two top squares in the back commute. Next, consider the cube depicted in Fig. B.11, which is

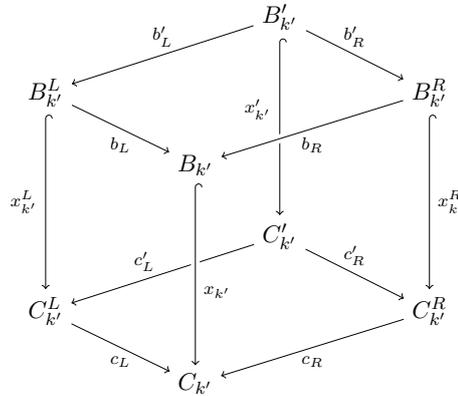


Figure B.11: Cube formed by the top squares from Fig. B.10

formed by the top squares from Fig. B.10. Its bottom square is a pullback by construction, its top square is also a pullback since it arises by pulling back the central bottom pullback from Fig. B.10 (computing  $K_{\cap}$ ) along the morphism  $r'_1 \circ b_{l'_1} \circ s_L = l'_2 \circ b_{r'_2} \circ s_R$ . Both front faces are pushouts by the van Kampen property of the left and right bottom pushouts in Fig. B.10; as  $x_{k'}^L, x_{k'}^R \in \mathcal{M}$ , they are pullbacks, as well. Together, this implies that both back faces are pullbacks (by pullback composition). Using that, the van Kampen property of the pushouts in the front implies that the faces in the back are also pushouts. Summarizing, the cube from Fig. B.11 has top and bottom pullback squares and all side squares are pushouts as well as pullbacks.

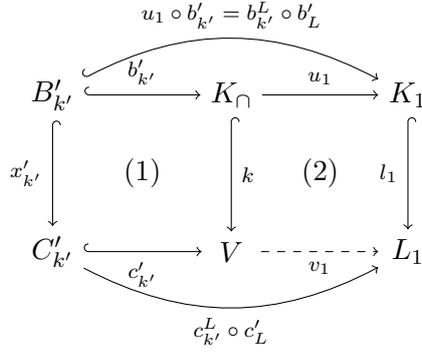


Figure B.12: Computing the common kernel  $k$

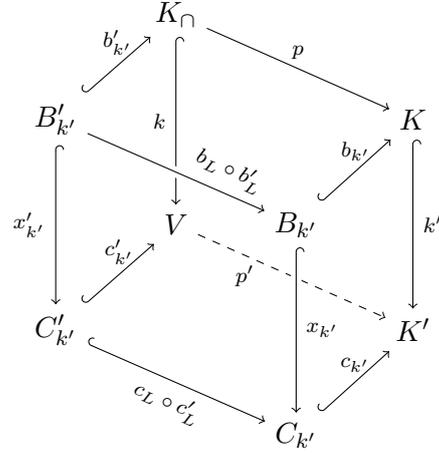


Figure B.13: Ensuring  $k$  to compute  $\rho_1 *_{E,k} \rho_2$

Now, we compute  $V$  (and the morphism  $k$ ) as pushout (1) in Fig. B.12. Since  $x'_{k'} \in \mathcal{M}$ , also  $k \in \mathcal{M}$ . Moreover, a morphism  $v_1 : V \rightarrow L_1$  is obtained such that (2) commutes via the universal property of that pushout. By pullback composition, the outer square (1)+(2) is a pullback (it is composed of the central vertical and the back top squares on the left of Fig. B.10 which are both pullbacks by the above considerations). Altogether, (2) is a pullback by  $\mathcal{M}$  pullback-pushout decomposition (Lemma 3.3). Likewise, a morphism  $v_2 : V \rightarrow R_2$  can be obtained such that  $k$  embeds as pullback into  $r_2$  via  $u_2$  and  $v_2$ .

Finally, we ensure that  $k$  indeed computes the desired generalized concurrent rule; compare Fig. B.13 for this: The morphism  $p : K_{\cap} \rightarrow K$  exists by the universal property of  $K$  as a pullback object such that  $k_i \circ p = e''_i \circ u_i$

for  $i = 1, 2$ . Using this and Eq. (4.1), we compute

$$\begin{aligned}
l \circ p \circ b'_{k'} &= l'_1 \circ k_1 \circ p \circ b'_{k'} \\
&= l'_1 \circ e''_1 \circ u_1 \circ b'_{k'} \\
&= l'_1 \circ b'_{l'_1} \circ s_L \circ b_L \circ b'_L \\
&= l'_1 \circ k_1 \circ b_{k'} \circ b_L \circ b'_L \\
&= l \circ b_{k'} \circ b_L \circ b'_L
\end{aligned}$$

(compare Figs. B.2 and B.10). In particular,  $p \circ b'_{k'} = b_{k'} \circ b_L \circ b'_L$  since  $l$  is monic such that the top square of the cube in Fig. B.13 commutes. By commutativity of the other squares and the universal property of the pushout square in the left back, we obtain the morphism  $p' : V \rightarrow K'$  such that the whole cube commutes. Using pushout composition and decomposition and the fact that the remaining three side squares are pushouts, the right square in the back is also a pushout. Thus, the pushouts of  $e''_i$ , where  $i = 1, 2$ , and  $e_1 \circ r_1 \circ u_1 = e_2 \circ l_2 \circ u_2$  along  $k$  during the computation of the generalized concurrent rule can all be split into first computing the pushout of  $p$  along  $k$  and then the pushouts of  $k_i$ , where  $i = 1, 2$ , and  $r'_1 \circ k_1 = l'_2 \circ k_2$  along  $k'$ . This means, the constructed common kernel  $k : K_\cap \hookrightarrow V$  indeed computes  $\rho_1 *_{E,k} \rho_2$  as generalized concurrent rule.  $\square$

*Proof of Theorem 4.14.* The *synthesis case* holds by the synthesis case of the Concurrency Theorem (see, e.g., [58, Theorem 4.17] for its statement in the context of  $\mathcal{M}$ -adhesive categories and rules with application conditions) and Proposition 4.2: Whenever such an  $E$ -related sequence of applications of  $\rho_1$  and  $\rho_2$  is given, the transformation  $G_0 \Rightarrow_{\rho_1 *_{E} \rho_2, m} G_2$  exists by the Concurrency Theorem, and, hence, the transformation  $G_0 \Rightarrow_{\rho_1 *_{E,k} \rho_2, m} G_2$  by Proposition 4.2.

For the first statement of the *analysis case*, it suffices to show that  $\rho_1 *_{E} \rho_2$  is applicable at match  $m$  if and only if a morphism  $q_2^* : Q \rightarrow K_\cap$  as required by  $m$  to have enhancement-free boundary exists. This then reduces the statement to the analysis case of the Concurrency Theorem. By applicability of  $\rho_1 *_{E,k} \rho_2$  at  $m$  we already know  $m \models ac$ . Thus, in the presence of initial pushouts,  $\rho_1 *_{E} \rho_2$  is applicable at  $m$  if and only if there exists a morphism  $b_m^{**} : B_m \hookrightarrow K$  such that  $l \circ b_m^{**} = b_m$  (see Fact 3.11). For both directions of the following proof, compare Fig. B.14. There, (2)



that  $k \circ q_2^* = q_2$  because the bent pushout square (along the  $\mathcal{M}$ -morphism  $k$ ) is in particular also a pullback.

In the other direction, assume  $q_2^*$  to exist. Since,  $l' \circ b_m^* = b_m$  and  $l'$  is mono, pulling back  $b_m^*$  along  $p'$  (which is always possible since  $b_m^* \in \mathcal{M}$ ) results in the span  $(B_m \xleftarrow{q_1} Q \xrightarrow{q_2} V)$ . Pulling back  $q_2$  along  $k$  subsequently gives the span  $(Q \xleftarrow{id_Q} Q \xrightarrow{q_2^*} K_\cap)$  since  $k \circ q_2^* = q_2$  and  $k$  is mono. This

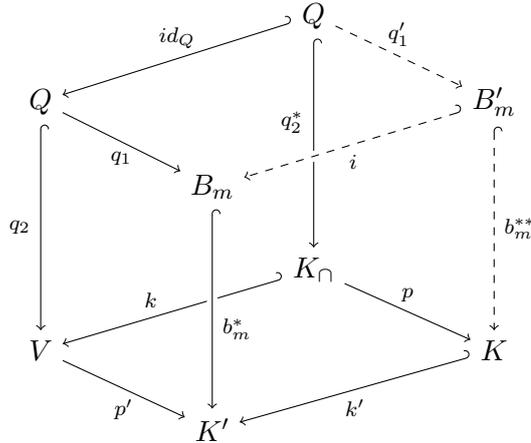


Figure B.15: Obtaining the morphism  $b_m^{**} : B_m \hookrightarrow K$

computation results in the solid part of Fig. B.15. Now, pulling back to construct the right side results in a cube with bottom a pushout along  $\mathcal{M}$ -morphism  $k$ , all side faces pullbacks, and all vertical morphisms from  $\mathcal{M}$ . Hence, by the weak vertical van Kampen property, the top square is a pushout. By  $id_Q \in \mathcal{M}$  and uniqueness of pushout complements, we can assume  $B'_m = B_m, i = id_{B_m}$ , and  $q'_1 = q_1$  without loss of generality. In particular, we obtain a morphism  $b_m^{**} : B_m \hookrightarrow K$  such that  $k' \circ b_m^{**} = b_m^*$ . Using that one computes

$$\begin{aligned} l \circ b_m^{**} &= l' \circ k' \circ b_m^{**} \\ &= l' \circ b_m^* \\ &= b_m \end{aligned}$$

which implies that  $\rho_1 *_E \rho_2$  is applicable at  $m$ .

The second statement of the *analysis case*, i.e., the additional statement under the assumption  $m \in \mathcal{M}$ , is proved exactly as the analysis case of the Generalized Concurrency Theorem for  $\mathcal{M}$ -matching (Theorem 4.9). There, beyond  $m \in \mathcal{M}$  we additionally assumed  $e_1, e_2 \in \mathcal{M}$ , which we do not assume here. However, checking that proof one sees that the assumption  $m \in \mathcal{M}$  is enough to prove that  $\rho_1 *_E \rho_2$  is applicable at  $m$ . The additional assumption  $e_1, e_2 \in \mathcal{M}$  is only needed to show that also  $m_1, m_2 \in \mathcal{M}$  (which we do not claim here).

Application conditions are dealt with in exactly the same way as in the case of  $\mathcal{M}$ -matching.  $\square$

*Proof of Corollary 4.15.* This corollary is really just a translation of the analysis case of the Generalized Concurrency Theorem to the concrete case of graphs. It only uses the fact what kind of elements are included in the boundary graph  $G_m$  (namely boundary nodes and elements sharing their image with others) as recollected after Definition 3.20.  $\square$

## B.2 Proofs from Chapter 5

*Proof of Proposition 5.12.* Let  $[\mathcal{X}_S, \mathcal{C}_\mathcal{M}]_{\text{cart}}$  be an  $S$ -cartesian functor category and  $I$  the initial object of  $\mathcal{C}$ . Clearly, the constant functor  $\Delta I : \mathcal{X} \rightarrow \mathcal{C}$  (that maps every object to  $I$  and every morphism to  $id_I$ ) is an  $S$ - $\mathcal{M}$ -restricted functor since  $id_I \in \mathcal{M}$ . Let  $F$  be any  $S$ - $\mathcal{M}$ -restricted functor. We show that the natural transformation from  $\Delta I$  to  $F$  that has the respective unique morphisms as components is a morphism in  $[\mathcal{X}_S, \mathcal{C}_\mathcal{M}]_{\text{cart}}$ , i.e., at each  $m \in S$ , the induced naturality square is a pullback (it is a natural transformation since every square that starts at the initial object necessarily commutes). Figure B.16 shows such a component of the natural transformation; Fact 3.9 (5) implies that the square is a pullback whenever  $Fm$  is a monomorphism.

This also implies that  $\Delta I$  is  $\mathcal{M}^{\text{cart}}$ -initial in  $[\mathcal{X}_S, \mathcal{C}_\mathcal{M}]_{\text{cart}}$  if  $I$  is  $\mathcal{M}$ -initial in  $\mathcal{C}$ .  $\square$

*Proof of Proposition 5.13.* It suffices to show the statement in the case of binary coproducts. First, that the componentwise computed sum of two  $S$ - $\mathcal{M}$ -restricted functors  $F$  and  $G$  is an  $S$ - $\mathcal{M}$ -restricted functor again is a direct consequence of Lemma 3.10: For each  $m : x \rightarrow y \in S$ ,  $Fm, Gm \in \mathcal{M}$

$$\begin{array}{ccc}
 I & \xrightarrow{i_{Fx}} & Fx \\
 \downarrow id_I & & \downarrow Fm \\
 I & \xrightarrow{i_{Fy}} & Fy
 \end{array}$$

Figure B.16: Morphism from the initial object  $\Delta I$  in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$

$$\begin{array}{ccc}
 Fx & \xrightarrow{inc_{Fx}} & Fx + Gx \\
 \downarrow Fm & & \downarrow Fm + Gm \\
 Fy & \xrightarrow{inc_{Fy}} & Fy + Gy
 \end{array}$$

Figure B.17: Induced coprojections

and hence  $Fm + Gm : Fx + Gx \hookrightarrow Fy + Gy \in \mathcal{M}$ . It remains to show that (i) the componentwise coprojections and (ii) the induced mediating morphisms are indeed morphisms in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ .

For (i), let  $m : x \rightarrow y \in S$ ; the square in Fig. B.17 depicts the induced naturality square. To show that this is a pullback square, we decompose it into the two squares depicted in Fig. B.18. One checks (using standard arguments) that (1) is a pushout; since it is a pushout along the  $\mathcal{M}$ -morphism  $Fm$ , it is in particular a pullback. Moreover, Fact 3.9 (5) implies that square (2) is a pullback since  $id_{Fy} + Gm$  is a monomorphism (by Lemma 3.10). This implies that the outer square is a pullback, too.

For (ii), let two  $S$ -cartesian natural transformations  $\nu_1 : F \rightarrow Q$  and  $\nu_2 : G \rightarrow Q$  between  $S$ - $\mathcal{M}$ -restricted functors  $F, G, Q$  be given. By the universal property of the coproducts in  $\mathcal{C}$ , there are morphisms  $\epsilon_x : Fx + Gx \rightarrow Qx$  for every  $x \in \mathcal{C}$  with  $\epsilon_x \circ inc_{Fx} = \nu_{1,x}$  and  $\epsilon_x \circ inc_{Gx} = \nu_{2,x}$  (the components of a natural transformation in  $[\mathcal{X}, \mathcal{C}]$ ). Since this is the only possible choice for the required unique morphism  $\epsilon : F + G \rightarrow Q$ , we have to show that it is a morphism in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ . For this, let  $m : x \rightarrow y \in S$  and compare Fig. B.19: We have to show that the square

$$\begin{array}{ccccc}
Fx & \xrightarrow{inc_{Fx}} & Fx + Gx & \xleftarrow{id_{Fx+Gx}} & Fx + Gx \\
\downarrow Fm & & \downarrow Fm+ & & \downarrow Fm + Gm \\
& (1) & id_{Gx} & (2) & \\
Fy & \xrightarrow{inc_{Fy}^x} & Fy + Gx & \xleftarrow{id_{Fy} + Gm} & Fy + Gy
\end{array}$$

Figure B.18: Induced coprojections constitute pullback squares

$Fx + Gx \rightarrow Qx \hookrightarrow Qy \leftarrow Fy + Gy \leftarrow Fx + Gx$  is a pullback. For this, compute the pullback of  $(Qm, \epsilon_y)$  (which exists because  $Qm \in \mathcal{M}$ ). In particular  $s : D \hookrightarrow Fy + Gy \in \mathcal{M}$ ; compute the pullbacks of  $(s, inc_{Gy})$  and  $(s, inc_{Fy})$ , respectively. Since  $\nu_{1,y} = \epsilon_y \circ inc_{Fy}$  and  $\nu_{2,y} = \epsilon_y \circ inc_{Gy}$  and the outer, bent squares are pullbacks ( $\nu_1$  and  $\nu_2$  are  $S$ -cartesian), the resulting pullback objects are  $Fx$  and  $Gx$  again. But by assumption, the pullback of the coproduct diagram at the bottom along the  $\mathcal{M}$ -morphism  $s$  results in a coproduct diagram, which means that  $D \cong Fx + Gx$ .  $\square$

*Proof of Proposition 5.14.* All three statements immediate follow from the fact that pullbacks and pushouts along componentwise  $\mathcal{M}$ -morphisms in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  are computed componentwise if  $\mathcal{C}$  is PVK square adhesive (Propositions 5.3 and 5.6) and the statements are true for every component by assumption.  $\square$

*Proof of Proposition 5.15.* Let  $\phi : A \rightarrow B$  be a natural transformation in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ . Let  $\epsilon_x : Ax \rightarrow Kx \in \mathcal{E}$  and  $\mu_x : Kx \rightarrow Bx \in \mathcal{M}'$  denote the unique factorizations of the components  $\phi_x : Ax \rightarrow Bx$ .  $\mathcal{E}$ - $\mathcal{M}'$  factorization systems are known to be functorial; we spell this out a little bit (compare Fig. B.20 for all of the following): On objects,  $K$  is defined as indicated above by factorizing the components of the natural transformation  $\phi$ . For each morphism  $m : x \rightarrow y$  in  $\mathcal{X}$ , by the diagonalization property of the  $\mathcal{E}$ - $\mathcal{M}'$  factorization system (compare, e.g., [2, 14.7 Proposition]), since  $\epsilon_x \in \mathcal{E}$ ,  $\mu_y \in \mathcal{M}'$ , and  $(Bm \circ \mu_x) \circ \epsilon_x = \mu_y \circ (\epsilon_y \circ Am)$ , there is a unique diagonal morphism  $Km : Kx \rightarrow Ky$  such that  $Km \circ \epsilon_x = \epsilon_y \circ Am$  and  $\mu_y \circ Km = Bm \circ \mu_x$ . In particular,  $K$  becomes a functor such that

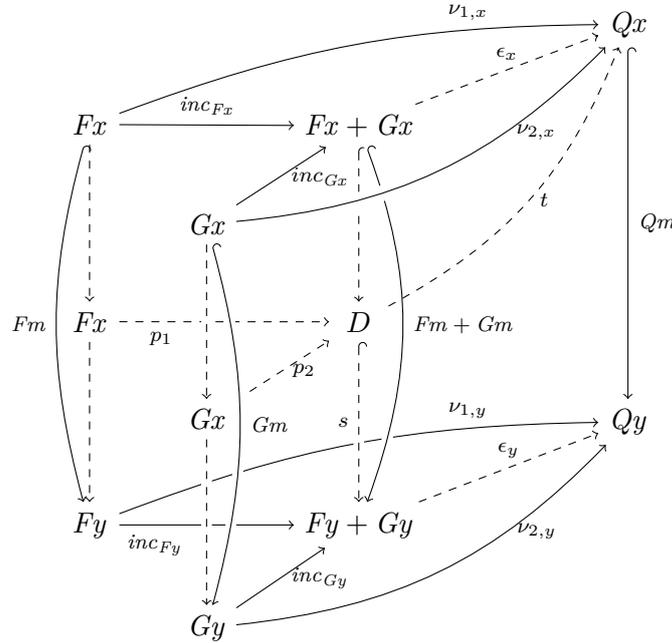


Figure B.19: Induced mediating morphism constitutes a pullback square

$\epsilon : A \rightarrow K$  and  $\mu : K \rightarrow B$  are natural transformations. We have to show that  $K$  is an  $S$ - $\mathcal{M}$ -restricted functor and that  $\mu$  and  $\epsilon$  are  $S$ -cartesian.

For this, let  $m \in S$ . Then the outer square depicted in Fig. B.20 is a pullback by assumption. Compute  $Bx \xleftarrow{\mu'_x} Kx' \xrightarrow{Km'} Ky$  as pullback of the morphisms  $\mu_y$  and  $Bm$  (which exists since  $Bm \in \mathcal{M}$ ). In particular,  $Km' \in \mathcal{M}$ . By pullback decomposition, the unique morphism  $\epsilon'_x : Ax \rightarrow Kx'$  that exists by the universal property of the right pullback square makes the left square into a pullback, too. This implies  $\epsilon'_x \in \mathcal{E}$  since  $\mathcal{E}$  is closed under pullbacks along  $\mathcal{M}$ -morphisms by assumption. Since  $\mathcal{M}'$  is closed under all pullbacks in any factorization system [2, 14.15 Proposition], also  $\mu'_x \in \mathcal{M}'$ . Then, the morphism  $i : Kx \rightarrow Kx'$  that exists by the universal property of the right pullback square is easily checked to satisfy  $i \circ \epsilon_x = \epsilon'_x$  (using that  $Km'$  is a monomorphism). Hence,  $i$  is an isomorphism (the unique isomorphism mediating between the two factorizations of  $\phi_x$ ),

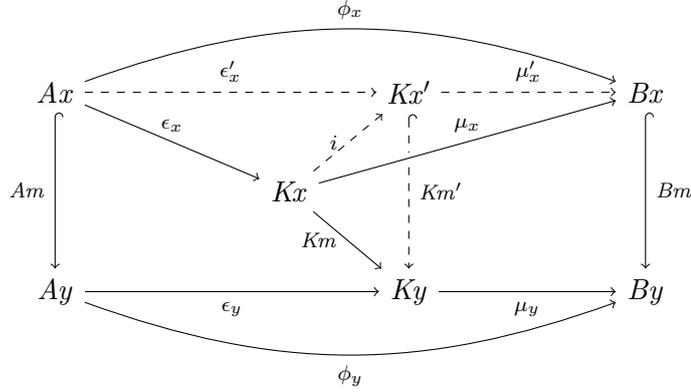


Figure B.20: Functoriality and preservation of  $\mathcal{E}$ - $\mathcal{M}'$  factorization systems

which implies that the two squares  $Kx \rightarrow Bx \rightarrow By \leftarrow Ky \leftarrow Kx$  and  $Ax \rightarrow Kx \rightarrow Ky \leftarrow Ay \leftarrow Ax$  are pullback squares and  $Km \in \mathcal{M}$ . In summary,  $K$  is an  $S$ - $\mathcal{M}$ -restricted functor and  $\epsilon$  and  $\mu$  are  $S$ -cartesian natural transformations.

The required uniqueness and composition properties follow by the componentwise definition of  $\mathcal{E}$ - and  $\mathcal{M}'$ -morphisms in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ .  $\square$

*Proof of Proposition 5.16.* Let  $\phi : A \rightarrow A'$  be a morphism in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  such that every component  $\phi_x \in \mathcal{M}'$ . Then, since  $\mathcal{C}$  has initial pushouts over  $\mathcal{M}'$ -morphisms, there is an initial pushout over every  $\phi_x : Ax \rightarrow A'x$ ; let  $C_0x$  denote the corresponding context objects and  $c_{0,x} : C_0x \hookrightarrow A'x \in \mathcal{M}$  the morphisms. These objects  $C_0x$  do not need to assemble into a functor from  $\mathcal{X}$  to  $\mathcal{C}$ ; however, we are using them to construct an  $S$ - $\mathcal{M}$ -restricted functor  $C$  and an  $\mathcal{M}^{\text{cart}}$ -morphism  $\chi : C \hookrightarrow A'$  as context object of the initial pushout over  $\phi$ . The intuitive idea behind the following is to construct the “smallest completion” of the  $C_0x$  into a functor  $C$ . This is achieved by constructing each  $Cx$  as the “universal”  $\mathcal{M}$ -subobject of  $A'x$  through which  $C_0x$  factors such that the  $Cx$  assemble into an  $S$ - $\mathcal{M}$ -restricted functor. For this, we define the class  $I$  to index all those  $S$ - $\mathcal{M}$ -restricted functors  $C_i$  with  $S$ -cartesian natural transformations  $\chi_i : C_i \hookrightarrow A'$  with the following properties: (i) the natural transformations  $\chi_i$  are elements of  $\mathcal{M}^{\text{cart}}$ , i.e.,  $\chi_{i,x} \in \mathcal{M}$  for all objects  $x \in C$  and (ii) for all such  $x$  the morphism  $c_{0,x}$

factorizes through  $\chi_{i,x}$ , i.e., there are (necessarily unique  $\mathcal{M}$ -) morphisms  $d_{i,x} : C_0x \hookrightarrow C_ix$  such that  $\chi_{i,x} \circ d_{i,x} = c_{0,x}$ ; note that the collection  $I$  is not empty because the identity natural transformation on  $A'$  belongs to it. Compute  $Cx$  as the limit of the sink of  $\mathcal{M}$ -morphisms  $(\chi_{i,x} : C_ix \hookrightarrow A'x)_{i \in I}$  (see Fig. B.21 for this and all of the following). By assumption, these limits exist in  $\mathcal{C}$  and  $\chi'_{i,x} \in \mathcal{M}$  for the morphisms  $\chi'_{i,x} : Cx \rightarrow C_ix$ . We show that  $C$  extends to an  $S$ - $\mathcal{M}$ -restricted functor and  $\chi_x := \chi_{i,x} \circ \chi'_{i,x} : Cx \rightarrow A'x$  (for any of the  $i \in I$ ) to an  $S$ -cartesian natural transformation; by construction, all of its components are in  $\mathcal{M}$ .

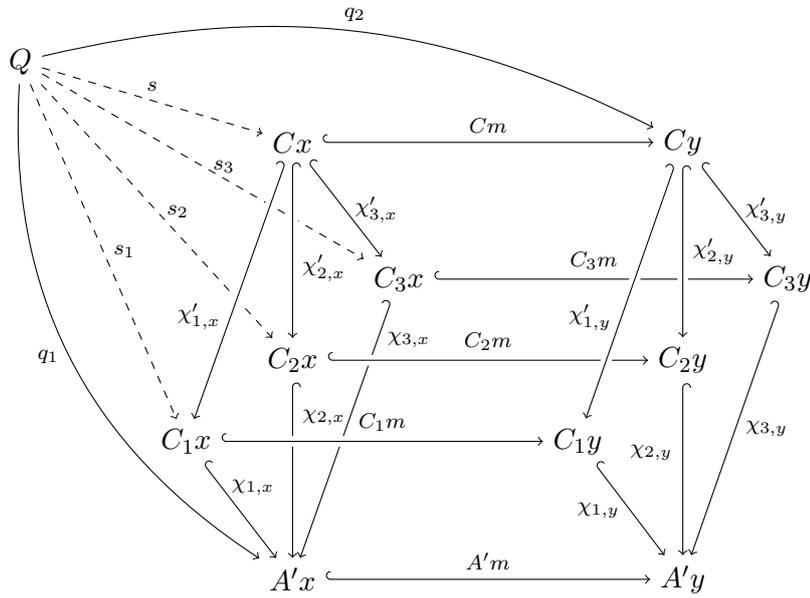


Figure B.21: Illustrating the construction and properties of the limit  $C$  (for  $|I| = 3$ )

First, for every morphism  $m : x \rightarrow y$  of  $\mathcal{X}$ , by the universal property of  $Cy$  as limit and the existence of morphisms  $C_im : C_ix \rightarrow C_iy$  there is a unique morphism  $Cm : Cx \rightarrow Cy$  such that  $\chi_y \circ Cm = A'm \circ \chi_x$ . In particular,  $C$  is a functor and the  $\chi_x$ s constitute the components of a natural transformation from  $C$  to  $A'$ .

If  $m \in S$ , by assumption  $C_im \in \mathcal{M}$  and hence  $Cm \in \mathcal{M}$  by decomposition of  $\mathcal{M}$ -morphisms. It remains to show that in this case  $Cx \hookrightarrow Cy \rightarrow A'y \hookleftarrow$

$A'x \leftarrow Cx$  is a pullback square which then implies that  $\chi : C \rightarrow A'$  is  $S$ -cartesian. If there is an object  $Q$  in  $\mathcal{C}$  with morphisms  $q_1 : Q \rightarrow Cy, q_2 : Q \rightarrow A'x$  such that  $A'm \circ q_1 = \chi_y \circ q_2$ , there are morphisms  $s_i : Q \rightarrow C_i x$  such that  $\chi_{i,x} \circ s_i = q_1$  and  $C_i m \circ s_i = \chi'_{i,y} \circ q_2$  for every  $i \in I$  (since every  $C_i$  is an  $S\mathcal{M}$ -restricted functor such that all squares  $C_i x \hookrightarrow A'x \hookrightarrow A'y \hookrightarrow C_i y \hookrightarrow C_i x$  are pullback squares). In particular,  $(s_i : Q \rightarrow C_i x)_{i \in I}$  is a commutative cone over  $(\chi_{i,x} : C_i x \rightarrow A'x)_{i \in I}$  (by  $\chi_{i,x} \circ s_i = q_1 = \chi_{j,x} \circ s_j$  for  $i, j \in I$ ) such that the universal property of  $Cx$  implies the existence of a unique morphism  $s : Q \rightarrow Cx$  with  $\chi'_{i,x} \circ s = s_i$  for all  $i \in I$ . Moreover, we have

$$\begin{aligned} q_1 &= \chi_{i,x} \circ s_i \\ &= \chi_{i,x} \circ \chi'_{i,x} \circ s \\ &= \chi_x \circ s \end{aligned} \tag{B.2}$$

and

$$\begin{aligned} \chi'_{i,y} \circ q_2 &= C_i m \circ s_i \\ &= C_i m \circ \chi'_{i,x} \circ s \\ &= \chi'_{i,y} \circ Cm \circ s \end{aligned} \tag{B.3}$$

for all  $i \in I$ , which implies  $q_2 = Cm \circ s$  since the family  $(\chi'_{i,y})_{i \in I}$  is jointly monomorphic. Finally,  $s$  is unique such that equalities computed in Eqs. (B.2) and (B.3) hold; this is directly implied by the monotonicity of  $Cm$ . In summary, the desired square is a pullback.

The just constructed  $C$  is the context object of the initial pushout over  $\phi$ . For this, compute the pullback of  $\chi : C \hookrightarrow A'$  and  $\phi : A \rightarrow A'$  which exists according to Proposition 5.3 and results in a span of natural transformations  $C \xleftarrow{\phi'} B \xrightarrow{\beta} A$  with  $\beta \in \mathcal{M}^{\text{cart}}$ . Showing that this pullback is even an initial pushout can be done in exactly the same way as in the according proof of [187, Theorem 2 (5.)] for general functor categories. The general ideas are as follows: For every component  $x \in \mathcal{C}$ , one uses the initial pushout over  $\phi_x$  (in  $\mathcal{C}$ ) and pushout-pullback decomposition to show that the square  $Bx \xrightarrow{\beta_x} Ax \xrightarrow{\phi_x} A'x \xleftarrow{\chi_x} Cx \xleftarrow{\phi'_x} Bx$  is a pushout (in  $\mathcal{C}$ ); as pushouts are computed componentwise in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$ , the desired square is a pushout. Initiality of that pushout in  $[\mathcal{X}_S, \mathcal{C}_M]_{\text{cart}}$  is then implied by its construction as a suitable limit.  $\square$

*Proof of Proposition 5.17.* This is an immediate consequence of the fact that pullbacks and pushouts along componentwise  $\mathcal{M}$ -morphisms in  $[\mathcal{X}_S, \mathcal{C}_{\mathcal{M}}]_{\text{cart}}$  are computed componentwise if  $\mathcal{C}$  is PVK square adhesive (Propositions 5.3 and 5.6).  $\square$

*Proof of Lemma 5.18.* We have to show that pushouts along  $\mathcal{M}$ -morphisms are  $\mathcal{M}$ -partial van Kampen squares. Let a pushout along an  $\mathcal{M}$ -morphism  $m$  and a commutative cube over it as depicted in Figs. 3.2 and 3.3 be given such that  $b, c \in \mathcal{M}$ .

First, if all side faces are pullbacks and  $d \in \mathcal{M}$ , the top square is a pushout since  $\mathcal{C}$  is adhesive HLR.

Given a cube where the back faces are pullbacks and the top square is a pushout, the two front faces are pullbacks since  $\mathcal{C}$  is adhesive HLR. In this case, it remains to show that  $d \in \mathcal{M}$ . Since  $c, n, n' \in \mathcal{M}$  and  $d \circ n' = n \circ c$ , the additional decomposition property implies that  $d \in \mathcal{M}$  if  $d$  is a monomorphism. A proof that this is the case if  $\mathcal{C}$  has all pullbacks can, e.g., be found in the proof of [98, Proposition 3.8.].  $\square$

*Proof of Theorem 5.20.* By Corollary 5.19 and Theorem 5.8, categories  $[\mathcal{X}_S, \mathbf{AGraph}_{\mathcal{M}}]_{\text{cart}}$  are PVK square adhesive as well as adhesive HLR. Categories  $[\mathcal{X}_S, \mathbf{Graph}_{\mathcal{M}}]_{\text{cart}}$  are even adhesive. We show the preservation of the additional HLR properties.

In the case of graphs (or even more generally, graph-like structures in the sense of [18]) all of the preconditions are known to hold. The maybe shortest way to prove that is noting that such structures are presheaves over **Set** and hence toposes. Thus, they have strict initial objects, epimono-factorization systems, etc. Therefore, we concentrate on the case of attributed graphs in the following.

*Initial objects:* **AGraph** are known to have an initial object which is not  $\mathcal{M}$ -initial, namely the empty graph with the initial algebra (i.e., the term algebra  $T_{\Sigma}$ ) as data part. Consequently, the data part of this  $E$ -graph is given by those terms that are of a type  $s \in S'$ . By Proposition 5.12, the category  $[\mathcal{X}_S, \mathbf{AGraph}_{\mathcal{M}}]_{\text{cart}}$  has the constant functor at this attributed graph as an initial object (which is not  $\mathcal{M}^{\text{cart}}$ -initial).

*Coproducts:* The category **AGraph** is known to have coproducts. Given two attributed graphs  $AG^i = (G^i, A^i)$ ,  $i = 1, 2$ , their coproduct is computed as follows (compare [53, Lemma 11.15]): First, the coproduct of

the  $E$ -graphs  $G^1$  and  $G^2$  and the  $\Sigma$ -algebras  $A^1$  and  $A^2$  are computed independently. The coproduct  $G^1 + G^2$  of the two  $E$ -graphs is just their disjoint union; the coproduct  $A^1 + A^2$  of the algebras can be computed by a generalization of the construction of free products of groups (compare [53, Example A.28]). Then, for each  $s \in S'$ , the data nodes of sort  $s$  in the  $E$ -graph  $G^1 + G^2$  have to be replaced by the corresponding elements of sort  $s$  in  $A^1 + A^2$ . Whenever such a data node  $x$  served as target node for some attribution edge, the new target for this attribution edge becomes the image of  $x$  (considered as element of one of the  $A^i$ 's) under the inclusion into the coproduct  $A^1 + A^2$ . Using Proposition 5.13, to show that  $[\mathcal{X}_S, \mathbf{AGraph}]$  has coproducts which are computed componentwise, it suffices to show that that construction is stable under pullback along  $\mathcal{M}$ -morphisms. Given a morphism  $m = (m_G, m_A) : AY \rightarrow AG^1 + AG^2 \in \mathcal{M}$ ,  $m_A$  and consequently also the component  $m_{AG}$  of  $m_G$  are isomorphisms. Pullbacks in  $\mathbf{AGraph}$  are computed componentwise [53, Fact A.23]. Hence, if  $AP^i = (P^i, B^i)$ , where  $i = 1, 2$ , arises as pullback of  $m$  along the coprojections  $AG^i \rightarrow AG^1 + AG^2$ , we obtain  $B_i \cong A_i$  and analogously  $A_P^i \cong A_G^i$  for the data nodes of the  $E$ -graphs. For the remaining sets that constitute the  $E$ -graph  $AY$ , again since the pullbacks are computed componentwise in  $\mathbf{Set}$  and  $\mathbf{Set}$  is extensive [41], each of these sets is a disjoint union of the according sets in  $AP^1$  and  $AP^2$  (it is also not difficult to check this directly). Summarizing,  $AP^1 \rightarrow AY \leftarrow AP^2$  is a coproduct diagram in  $\mathbf{AGraph}$ .

*Initial pushouts:* The category  $\mathbf{AGraph}$  has initial pushouts over every morphism [53, Fact 10.7]. To show that  $[\mathcal{X}_S, \mathbf{AGraph}]$  has initial pushouts over every morphism, using Proposition 5.16 it suffices to show that  $\mathbf{AGraph}$  has intersections of  $\mathcal{M}$ -subobjects. On the algebra part, all algebras of such a sink are isomorphic hence the same algebra can be chosen for the limit. On the part of the  $E$ -graph, the limit can be constructed componentwise in  $\mathbf{Set}$ . In each component we need to compute the intersection of a family of injective functions and even though this family does not need to constitute a set, its limit exists in  $\mathbf{Set}$  since  $\mathbf{Set}$  is complete and well-powered [2, 12.5 Proposition and 12.6 Examples].

*$\mathcal{E}$ - $\mathcal{M}'$  factorization:* In the category  $\mathbf{AGraph}$ , there are several ways to choose classes of morphisms  $\mathcal{E}$  and  $\mathcal{M}'$  to obtain an  $\mathcal{E}$ - $\mathcal{M}'$  factorization [53, Example 9.22]; we focus on one of them. To obtain an  $\mathcal{E}$ - $\mathcal{M}'$  factorization (even factorization system) in  $[\mathcal{X}_S, \mathbf{AGraph}]$ , according to Proposition 5.15

this choice should constitute a factorization system and  $\mathcal{E}$  should be closed under pullback. Define  $\mathcal{E}$  to be the class of attributed graph morphisms where every component is surjective and  $\mathcal{M}'$  to be the class where every component is injective; this is known to constitute a factorization system on the algebra part [192, Example 3.3.3.] as well as on the  $E$ -graph part (inherited componentwise from **Set**). Moreover as pullbacks are computed componentwise and pullbacks of surjective functions in **Set** are surjective again, this class  $\mathcal{E}$  is closed under pullback.

*$\mathcal{M}$ -effective unions:* The category **AGraph** is known to have  $\mathcal{M}$ -effective unions [54, Remark 5.57] and, according to Proposition 5.17, this property is preserved for the componentwise  $\mathcal{M}$ -morphisms in  $[\mathcal{X}_S, \mathbf{AGraph}]$ .  $\square$

*Proof of Proposition 5.23.* The morphisms in **PTrG<sub>TG</sub>** and **APTrG<sub>ATG</sub>** are morphisms in **PTrG** and **APTrG**, respectively. All used concepts (pullbacks, pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms, factorization of morphisms, etc.) are just computed as in **PTrG** (or **APTrG**). For example in the case of pushouts, since the inclusion functor  $I : [\bullet \leftarrow \bullet \rightarrow \bullet, \mathbf{Graph}] \rightarrow [\mathcal{A}, \mathbf{Graph}]$  creates pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms (Proposition 5.6), the necessary typing morphism of the result of such a pushout in **PTrG<sub>TG</sub>** (**APTrG<sub>ATG</sub>**) is obtained by the universal property of that result in  $[\mathcal{A}, \mathbf{Graph}]$ . Using the fact that the coprojections of a pushout are jointly epimorphic, correct typing of unique mediating morphisms is easily checked. The typing morphism of the result of a pullback is obtained by composition with the typing morphism of one of the other (attributed) partial triple graphs from the pullback diagram (where independence from the choice is checked via a simple diagram chase). In this case, the correct typing of unique mediating morphisms is directly implied from the typing morphism of the pullback object being defined by composition.

For the detailed proofs of these assertions, compare Fig. B.22. Concerning the case of pushouts, given typed partial triple graphs  $t_i : I(G_i) \rightarrow I(J(TG))$ , where  $i = 1, 2, 3$ , and typed morphisms  $p'_1 : G_1 \rightarrow G_3, p'_2 : G_1 \rightarrow G_2$  (where one of them is an  $\mathcal{M}$ -morphism), compute the cospan  $G_2 \xrightarrow{p_1} G_4 \xleftarrow{p_2} G_3$  as pushout; applying the functor  $I$  to the diagram results in a pushout, again, since  $I$  creates pushouts along  $\mathcal{M}^{\text{cart}}$ -morphisms. Since  $p'_1, p'_2$  are typed morphisms, by the universal property of  $G_4$  as pushout

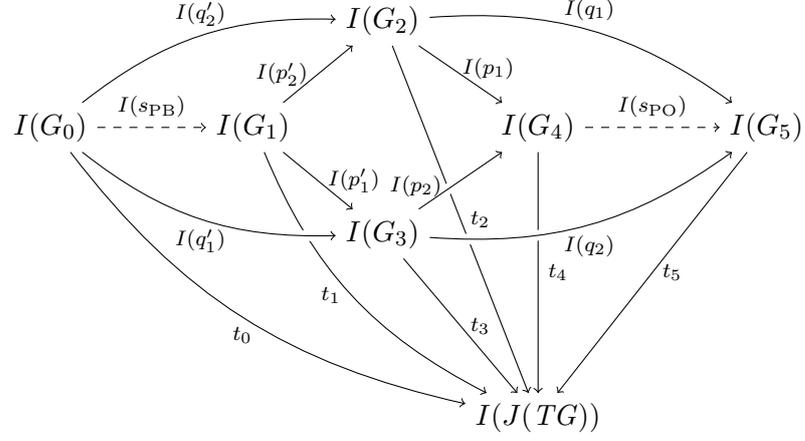


Figure B.22: Showing the typing of pushout and pullback objects and the mediating morphisms to be typed morphisms

object we obtain a (unique) morphism  $t_4 : I(G_4) \rightarrow I(J(TG))$  such that

$$t_4 \circ I(p_2) = t_3 \text{ and } t_4 \circ I(p_1) = t_2$$

in the functor category  $[\bullet \leftarrow \bullet \rightarrow \bullet, \mathbf{AGraph}]$ . This exhibits  $p_1, p_2$  as typed morphisms.

If  $q_1 : G_2 \rightarrow G_5, q_2 : G_3 \rightarrow G_5$  are any typed morphisms with  $q_1 \circ p'_2 = q_2 \circ p'_1$ , we have to show that the unique mediating morphism  $s_{PO} : G_4 \rightarrow G_5$  is a typed morphism. For this, we compute

$$\begin{aligned} t_4 \circ I(p_2) &= t_3 \\ &= t_5 \circ I(q_2) \\ &= t_5 \circ I(s_{PO}) \circ I(p_2) \end{aligned}$$

and, completely analogously,  $t_4 \circ I(p_1) = t_5 \circ I(s_{PO}) \circ I(p_1)$  which implies  $t_4 = t_5 \circ I(s_{PO})$  since  $I(p_1)$  and  $I(p_2)$  are jointly epimorphic. This shows  $s_{PO}$  to be a typed morphism.

If the central square in Fig. B.22 is a pullback of typed morphisms, one defines  $t_1 := t_3 \circ I(p'_1)$  which makes  $p'_1$  to a typed morphism. Moreover, we

have that  $p'_2$  is a typed morphism with that choice of  $t_1$  by computing

$$\begin{aligned} t_2 \circ I(p'_2) &= t_4 \circ I(p_1) \circ I(p'_2) \\ &= t_4 \circ I(p_2) \circ I(p'_1) \\ &= t_3 \circ I(p'_1) \\ &= t_1 . \end{aligned}$$

That a mediating morphism  $s_{\text{PB}}$  for a pullback of typed morphisms is a typed morphism is computed via

$$\begin{aligned} t_1 \circ I(s_{\text{PB}}) &= t_3 \circ I(p'_1) \circ I(s_{\text{PB}}) \\ &= t_3 \circ I(q'_1) \\ &= t_0 . \end{aligned}$$

The according computations for pushout complements, initial objects, coproducts,  $\mathcal{E}'$ - $\mathcal{M}'$  pair factorizations, initial pushouts, and  $\mathcal{M}$ -effective unions, showing the relevant objects and morphisms to be typed, are all very similar. Hence, the results obtained for **PTrG** and **APTrG** also hold in **PTrG<sub>TrG</sub>** and **APTrG<sub>ATG</sub>**.

Proving the statement about the subcategories and preservation and reflection of rule applications is analogous to the proof of Proposition 5.22.  $\square$

*Proof of Proposition 5.24.* By assumption, all squares in Fig. 5.18 are commuting. Moreover, the hooked morphisms are injective. Hence, all the statements are just translations of the requirement that the squares (1) – (6) are pullback squares into its set-theoretic meaning.  $\square$

*Proof of Lemma 5.25.* We only have to show that the four central squares in Fig. 5.20 are pullback squares. The two squares from the bottom row and the right square of the top row are pullback squares by assumption (as  $r$  and  $l$  are  $\mathcal{M}$ -morphisms). The remaining square can be decomposed into the two pullback squares depicted in Fig. B.23 and is therefore a pullback square itself. In Fig. B.23, the right square is a pullback by assumption ( $l \in \mathcal{M}$ ) and the left square is one since  $l_{\xi}$  is in particular a monomorphism.  $\square$

$$\begin{array}{ccccc}
K_{\tilde{S}} & \xleftarrow{l_{\tilde{S}}} & L_{\tilde{S}} & \xleftarrow{\iota_{L_S}} & L_C \\
\parallel & & \uparrow l_{\tilde{S}} & & \uparrow l_C \\
K_{\tilde{S}} & \xlongequal{\quad} & K_{\tilde{S}} & \xleftarrow{\iota_{K_S}} & K_C
\end{array}$$

Figure B.23: Showing the forward rule to stem from **APT<sub>r</sub>G<sub>ATG</sub>**

$$\begin{array}{ccccccc}
L = & (L_S \xleftarrow{\sigma_L} L_{\tilde{S}} \xleftarrow{\iota_{L_S}} L_C \xleftarrow{\iota_{L_T}} L_{\tilde{T}} \xrightarrow{\tau_L} L_T) \\
\uparrow & \uparrow l_S & \uparrow l_{\tilde{S}} & \uparrow id & \uparrow id & \uparrow id & \uparrow id \\
A = & (K_S \xleftarrow{\sigma_K} K_{\tilde{S}} \xleftarrow{\iota_{L_S} \circ l_{\tilde{S}}} L_C \xleftarrow{\iota_{L_T}} L_{\tilde{T}} \xrightarrow{\tau_L} L_T) \\
\downarrow & \downarrow r_S & \downarrow id \\
L^F = & (R_S \xleftarrow{r_S \circ \sigma_K} K_{\tilde{S}} \xleftarrow{\iota_{L_S} \circ l_{\tilde{S}}} L_C \xleftarrow{\iota_{L_T}} L_{\tilde{T}} \xrightarrow{\tau_L} L_T)
\end{array}$$

Figure B.24: Partial morphism from  $L$  to  $L^F$ 

*Proof of Lemma 5.26.* We just depict the morphism in Fig. B.24. Note that this is usually not a partial morphism in **PTrG<sub>TG</sub>** (**APT<sub>r</sub>G<sub>ATG</sub>**) because the second square in the top row does not need to be a pullback.  $\square$

*Proof of Lemma 5.28.* We have to check that (i)  $D$  is actually a partial triple graph, (ii) the morphisms  $g$  and  $d$  are well-defined (and, in particular, morphisms in **APT<sub>r</sub>G<sub>ATG</sub>**), (iii) the induced square is a pullback, and (iv) the finality of  $D$  as pullback complement.

For (i) and (ii), Fig. B.25 depicts the general situation. Recall that every object in the diagram is a typed attributed graph and every morphism a morphism between such. In particular, every object and morphism consists of several components. Figure B.26 offers a more local view of square (1) of Fig. B.25, namely two components and their connection via source and target morphisms.

To show (i), we just have to check that for all components of  $D$  the images

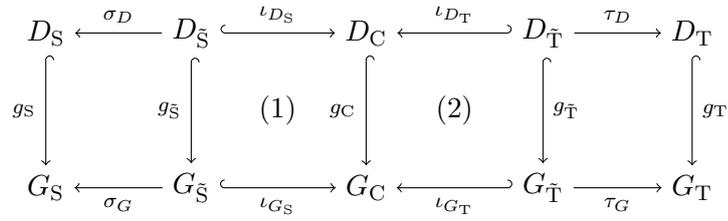


Figure B.25: Global view of the morphism  $g : D \hookrightarrow G$

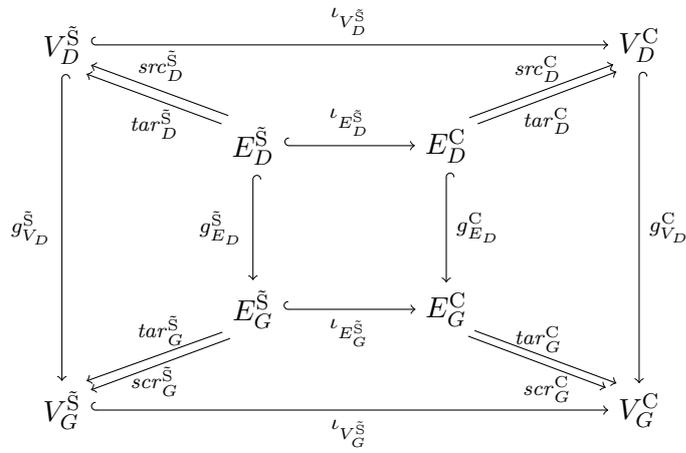


Figure B.26: A component of the morphism  $g : D \hookrightarrow G$

under all applicable source and target functions and morphisms  $\sigma_D, \tau_D, \iota_{D_S},$  and  $\iota_{D_T}$  are also elements of  $D$ . When this is the case, it also directly follows that  $\iota_{D_S}$  and  $\iota_{D_T}$  are  $\mathcal{M}$ -morphisms in  $\mathbf{AGraph}_{\mathbf{ATG}}$  because  $\iota_{G_S}$  and  $\iota_{G_T}$  are. For source and target functions this is immediate, as in the definitions of all the edge-sets  $E_D^X, NA_D^X, EA_D^X$ , where  $X \in \{S, \tilde{S}, C, \tilde{T}, T\}$ , edges from the respective sets of  $G$ , whose source or target node does not belong to  $D$ , are directly excluded (note that for attribution-edges we do not have to check the existence of target nodes as the algebras do not change). The same kind of argumentation applies to  $\sigma_D$  and  $\tau_D$ . We exemplarily argue for the well-definedness of the inclusions  $\iota_{D_S}$  and  $\iota_{D_T}$  using  $\iota_{V_G^{\tilde{S}}} : V_D^{\tilde{S}} \hookrightarrow V_D^C$  and  $\iota_{E_D^{\tilde{S}}} : E_D^{\tilde{S}} \hookrightarrow E_D^C$ . Thus, let  $n \in V_D^{\tilde{S}}$  be given; we show  $\iota_{V_G^{\tilde{S}}}(n) \in V_D^C$ . First, since both the morphisms  $m$  and  $l$  constitute pullbacks at this component, for any  $n' \in m_{V_L^C}^C(V_L^C \setminus V_K^C)$  it holds that either  $n'$  does not have a preimage  $n$  under  $\iota_{V_G^{\tilde{S}}}$  or  $n \in m_{V_L^{\tilde{S}}}^{\tilde{S}}(V_L^{\tilde{S}} \setminus V_K^{\tilde{S}})$  for such a preimage. Therefore, the first deletion in the definition of  $V_D^{\tilde{S}}$  implies  $\iota_{V_G^{\tilde{S}}}(n) \notin m_{V_L^C}^C(V_L^C \setminus V_K^C)$  for all  $n \in V_D^{\tilde{S}}$ . A second reason for  $\iota_{V_G^{\tilde{S}}}(n) \notin V_D^C$  would be the existence of  $n' \in V_G^{\tilde{S}}$  such that  $\iota_{V_G^{\tilde{S}}}(n) = \iota_{V_G^{\tilde{S}}}(n')$  but  $\sigma_{V_G}(n') \notin V_D^{\tilde{S}}$ . But  $\iota_{V_G^{\tilde{S}}}$  is injective such that  $n = n'$  and  $\sigma_{V_G}(n) \in V_D^{\tilde{S}}$  by definition of  $V_D^{\tilde{S}}$ . The final possibility for  $\iota_{V_G^{\tilde{S}}}(n)$  not being an element of  $V_D^C$  would be the existence of  $n' \in V_G^{\tilde{T}}$  such that  $\iota_{V_G^{\tilde{S}}}(n) = \iota_{V_G^{\tilde{T}}}(n')$  but  $\tau_{V_G}(n') \notin V_D^{\tilde{T}}$ . But also such elements do not belong to  $V_D^{\tilde{S}}$  by definition. Thus,  $\iota_{V_G^{\tilde{S}}}(V_D^{\tilde{S}}) \subseteq V_D^C$ .

Similarly, let  $e \in E_D^{\tilde{S}}$  be given. In that case, there are four possible reasons for  $\iota_{E_G^{\tilde{S}}}(e)$  not being an element of  $E_D^C$ . First, it could hold that  $\text{src}_G^C(\iota_{E_G^{\tilde{S}}}(e)) \notin V_D^C$  or  $\text{tar}_G^C(\iota_{E_G^{\tilde{S}}}(e)) \notin V_D^C$ . Since  $\iota_{G_S}$  is a morphism, this would imply  $\iota_{V_G^{\tilde{S}}}(\text{src}_G^{\tilde{S}}(e)) \notin V_D^C$  or  $\iota_{V_G^{\tilde{S}}}(\text{tar}_G^{\tilde{S}}(e)) \notin V_D^C$ . Now,  $\text{src}_G^{\tilde{S}}(e), \text{tar}_G^{\tilde{S}}(e) \in V_D^{\tilde{S}}$  by definition of  $E_D^{\tilde{S}}$ , which implies  $\iota_{V_G^{\tilde{S}}}(\text{src}_G^{\tilde{S}}(e)), \iota_{V_G^{\tilde{S}}}(\text{tar}_G^{\tilde{S}}(e)) \in V_D^C$  by the considerations above. The remaining three cases (and all the omitted ones) are completely analogous to the respective cases for nodes. Summarizing,  $D$  is a typed attributed partial triple graph.

To address (ii), we first show that  $g : D \rightarrow G$  is a morphism. As  $g$  is defined as an inclusion, all compatibility conditions on graph morphisms

are automatically true. We only have to show that  $g$  is a morphism also in **APTrG<sub>ATG</sub>**, i.e., that squares (1) and (2) from Fig. B.25 are pullback squares, which can be done componentwise. As all involved morphisms are injective, this means to show that in all components,  $D_{\tilde{S}}$  and  $D_{\tilde{T}}$  are intersections of  $D_C$  with  $G_{\tilde{S}}$  and  $G_{\tilde{T}}$ , respectively. Again, we show this using  $V_D^{\tilde{S}}$  and  $E_D^{\tilde{S}}$  as representative examples. We have to show that  $V_D^{\tilde{S}} \supseteq V_D^C \cap V_G^{\tilde{S}}$  and  $E_D^{\tilde{S}} \supseteq E_D^C \cap E_G^{\tilde{S}}$ . For this, let  $n \in V_D^C \cap V_G^{\tilde{S}}$ , i.e., let  $n \in V_G^{\tilde{S}}$  be such that  $\iota_{V_G^{\tilde{S}}}(n) \in V_D^C$ . Again, we check that none of the reasons to exclude  $n$  from  $V_D^{\tilde{S}}$  applies. First,  $\iota_{V_G^{\tilde{S}}}(n) \in V_D^C$  implies that there cannot exist  $n' \in V_G^{\tilde{S}}$  such that  $\iota_{V_G^{\tilde{S}}}(n') = \iota_{V_G^{\tilde{S}}}(n)$  and  $\sigma_{V_G}(n') \notin V_D^S$ . By injectivity of  $\iota_{V_G^{\tilde{S}}}$ ,  $n$  is the only possible candidate for this and  $\sigma_{V_G}(n) \in V_D^S$  since  $n \in V_D^{\tilde{S}}$ . Furthermore, as above,  $\iota_{V_G^{\tilde{S}}}(n) \in V_D^C$  implies  $\iota_{V_G^{\tilde{S}}}(n) \notin m_{V_L}^C(V_L^C \setminus V_K^C)$ , which in turn implies  $n \notin m_{V_L}^{\tilde{S}}(V_L^{\tilde{S}} \setminus V_K^{\tilde{S}})$ . Finally,  $n$  also cannot be such that there exists  $n' \in V_G^{\tilde{T}}$  with  $\iota_{V_G^{\tilde{T}}}(n') = \iota_{V_G^{\tilde{S}}}(n)$  and  $\tau_{V_G}(n') \notin V_D^T$ : By definition, this would exclude  $\iota_{V_G^{\tilde{S}}}(n)$  from  $V_D^C$ . Summarizing, it follows that  $n \in V_D^{\tilde{S}}$ .

Similarly, let  $e \in E_D^C \cap E_G^{\tilde{S}}$ , i.e., let  $e \in E_G^{\tilde{S}}$  be such that  $\iota_{E_G^{\tilde{S}}}(e) \in E_D^C$ . The only difference to the case of  $V_D^{\tilde{S}}$  is that  $e$  can additionally be excluded from  $E_D^{\tilde{S}}$  for its source or target node not belonging to  $V_D^{\tilde{S}}$ . However, obviously  $e \in E_G^{\tilde{S}}$  implies  $\text{src}_{E_G^{\tilde{S}}}(e) \in V_G^{\tilde{S}}$ , and  $\iota_{E_G^{\tilde{S}}}(e) \in E_D^C$  implies  $\text{scr}_G^C(\iota_{E_G^{\tilde{S}}}(e)) \in V_D^C$  by definition of  $E_D^C$ . Moreover, both these nodes have the same image in  $V_G^C$ , since  $g$  is just an inclusion (compare Fig. B.26). By the above consideration on nodes ( $V_D^{\tilde{S}}$  is the intersection of  $V_G^{\tilde{S}}$  and  $V_D^C$ ), this implies  $\text{src}_D^{\tilde{S}}(e) \in V_D^{\tilde{S}}$  as desired. Target nodes are treated completely analogously.

It remains to show that also  $d$  is well-defined and a morphism in **APTrG<sub>ATG</sub>**. For that, it suffices to show that for each  $x \in K$  the image  $m(l(x))$  is already an element of  $D$ . Then pullback decomposition shows that the required squares induced by  $d$  are pullbacks (since  $g \circ d = m \circ l$  by construction and the statement is true for  $g$  and for  $m \circ l$ ). Again, we show this exemplarily using  $V_K^{\tilde{S}}$  and  $E_K^{\tilde{S}}$ . First, for any  $n \in V_K^{\tilde{S}}$ , we have  $\sigma_{V_G}(m_{V_L}^{\tilde{S}}(l_{V_K}^{\tilde{S}}(n))) = m_{V_L}^{\tilde{S}}(l_{V_K}^S(\sigma_{V_K}(n))) \notin m_{V_L}^{\tilde{S}}(V_L^{\tilde{S}} \setminus V_K^{\tilde{S}})$ . Therefore,  $m_{V_L}^{\tilde{S}}(l_{V_K}^{\tilde{S}}(n))$  is not excluded from  $V_D^{\tilde{S}}$  for its image under  $\sigma_{V_G}$  not belonging to  $V_D^S$ . Obviously,  $m_{V_L}^{\tilde{S}}(l_{V_K}^{\tilde{S}}(n)) \notin m_{V_L}^{\tilde{S}}(V_L^{\tilde{S}} \setminus V_K^{\tilde{S}})$  (as

$$\begin{array}{ccccccc}
V_K^{\tilde{S}} & \xrightarrow{\iota_{V_K^{\tilde{S}}}} & V_K^C & \xleftarrow{\iota_{V_K^{\tilde{T}}}} & V_K^{\tilde{T}} & \xrightarrow{\tau_{V_K}} & V_K^T \\
\downarrow l_{V_K^{\tilde{S}}} & & \downarrow l_{V_K^C} & & \downarrow l_{V_K^{\tilde{T}}} & & \downarrow l_{V_K^T} \\
& \text{(PB)} & & \text{(PB)} & & & \\
V_L^{\tilde{S}} & \xrightarrow{\iota_{V_L^{\tilde{S}}}} & V_L^C & \xleftarrow{\iota_{V_L^{\tilde{T}}}} & V_L^{\tilde{T}} & \xrightarrow{\tau_{V_L}} & V_L^T \\
\downarrow m_{V_L^{\tilde{S}}} & & \downarrow m_{V_L^C} & & \downarrow m_{V_L^{\tilde{T}}} & & \downarrow m_{V_L^T} \\
& \text{(PB)} & & \text{(PB)} & & & \\
V_G^{\tilde{S}} & \xrightarrow{\iota_{V_G^{\tilde{S}}}} & V_G^C & \xleftarrow{\iota_{V_G^{\tilde{T}}}} & V_G^{\tilde{T}} & \xrightarrow{\tau_{V_G}} & V_G^T
\end{array}$$

Figure B.27: Proving  $m(l(K)) \subseteq D$ 

$m$  and  $l$  are injective). Finally, assume  $n' \in V_G^{\tilde{T}}$  to exist such that  $\iota_{V_G^{\tilde{T}}}(n') = \iota_{V_G^{\tilde{S}}}(m_{V_L^{\tilde{S}}}^{\tilde{S}}(l_{V_K^{\tilde{S}}}^{\tilde{S}}(n)))$  (compare Fig. B.27 for the following). This implies that also  $\iota_{V_G^{\tilde{T}}}(n') = m_{V_L^C}^C(l_{V_K^{\tilde{S}}}^{\tilde{S}}(n))$ . As  $V_K^{\tilde{T}}$  is the intersection of  $V_K^C$  and  $V_G^{\tilde{T}}$ , this in turn implies the existence of a node  $n'' \in V_K^{\tilde{T}}$  such that  $\iota_{V_K^{\tilde{S}}}(n) = \iota_{V_K^{\tilde{T}}}(n'')$  and  $n' = m_{V_L^{\tilde{T}}}^{\tilde{T}}(l_{V_K^{\tilde{T}}}^{\tilde{T}}(n''))$ . From that it follows that  $\tau_{V_G}(n') = m_{V_L^T}^T(l_{V_K^{\tilde{T}}}^{\tilde{T}}(\tau_{V_K}(n'')))$ , which finally implies  $\tau_{V_G}(n') \in V_D^T$  (as  $\tau_{V_G}(n')$  has a preimage in  $V_K^{\tilde{T}}$  under  $m_{V_L^{\tilde{T}}}^{\tilde{T}} \circ l_{V_K^{\tilde{T}}}^{\tilde{T}}$ ).

Considering  $E_K^{\tilde{S}}$ , again the only additional case we have to exclude is that there exists  $e \in E_K^{\tilde{S}}$  such that  $\text{src}_G^{\tilde{S}}(m_{E_L^{\tilde{S}}}^{\tilde{S}}(l_{E_K^{\tilde{S}}}^{\tilde{S}}(e))) \notin V_D^{\tilde{S}}$  (and likewise for the target function). But this follows directly from  $\text{src}_G^{\tilde{S}}(m_{E_L^{\tilde{S}}}^{\tilde{S}}(l_{E_K^{\tilde{S}}}^{\tilde{S}}(e))) = m_{V_L^{\tilde{S}}}^{\tilde{S}}(l_{V_K^{\tilde{S}}}^{\tilde{S}}(\text{src}_K^{\tilde{S}}(e)))$ , since  $\text{src}_K^{\tilde{S}}(e) \in V_K^{\tilde{S}}$  and  $n \in V_K^{\tilde{S}}$  implies  $m_{V_L^{\tilde{S}}}^{\tilde{S}}(l_{V_K^{\tilde{S}}}^{\tilde{S}}(n)) \in V_D^{\tilde{S}}$  by the above considerations.

Statement (iii) asserts that  $m \circ l = g \circ d$  is even a pullback; first note that equality holds by definition of  $g$  and  $d$ . On the data part, the statement follows from Fact 3.9 (4) as the algebra homomorphisms  $l_A$  and  $g_A$  are just identities and, therefore,  $d_A$  coincides with  $m_A$  by definition. On the structural part, as every involved morphism is injective, the question of being a pullback again reduces to showing  $K = L \cap D$  in  $G$ . Statement (ii) proves  $K \subseteq L \cap D$ . But  $K \supseteq L \cap D$  holds by definition: In every component,

elements of  $G$  with preimage under  $m(L \setminus K)$  are removed to obtain  $D$ .

Finally, to prove statement (iv), let there be a competing pullback, i.e., let there be morphisms  $f : K' \rightarrow L$ ,  $f' : K' \rightarrow K$ ,  $d' : K' \rightarrow D'$  and  $z : D' \rightarrow G$  such that  $m \circ f = z \circ d'$  is a pullback and  $f = l \circ f'$  (recall Fig. 3.8). We have to show that there exists a unique morphism  $z' : D' \rightarrow D$  such that  $g \circ z' = z$  and  $d \circ f' = z' \circ d'$ . Note that, since  $g$  is a monomorphism, it suffices to show the existence of a morphism  $z'$  that satisfies the desired equalities. In the rest of the proof, we write every morphism  $f$  as pair  $(f_G, f_A)$  with  $f_G$  being its  $E$ -graph morphism and  $f_A$  the algebra homomorphism.

First, on the data part, define  $z'_A := z_A$ . Since  $g_A$  is an identity morphism, this immediately implies  $g_A \circ z'_A = z_A$  but also  $z'_A \circ d'_A = d_A \circ f'_A$  by left-cancellability of  $g_A$ : We obtain

$$\begin{aligned} g_A \circ z'_A \circ d'_A &= z_A \circ d'_A \\ &= m_A \circ f_A \\ &= m_A \circ l_A \circ f'_A \\ &= g_A \circ d_A \circ f'_A, \end{aligned}$$

which implies  $z'_A \circ d'_A = d_A \circ f'_A$ .

For an arbitrary graph element  $x \in D'$ , we set  $z'_G(x) := g_G^{-1}(z_G(x))$ . First, since every component of  $g_G$  is injective, this is well-defined as long as  $z_G(D') \subseteq g_G(D)$ ; also  $z'$  being a morphism in  $\mathbf{APTTrG}_{\mathbf{ATG}}$  then follows from  $g$  and  $z$  being such. Furthermore, exactly as above, we immediately obtain  $g_G \circ z'_G = z_G$  from that definition and  $z'_G \circ d'_G = d_G \circ f'_G$  follows from left-cancellability of  $g_G$  with the same computation. This means, we only have to show that  $z_G(D') \subseteq g_G(D)$ .

To show this, first, let  $x \in D'$  be such that  $z_G(x) \in G$  has a preimage in  $L$  under  $m_G$ . As  $z_G \circ d'_G = m_G \circ f'_G$  defines a pullback (and this pullback is computed componentwise in  $\mathbf{Set}$ ), this implies that  $x$  has (at least) one preimage  $x' \in K'$  under  $d'_G$  and  $z_G(x) = m_G(l_G(f'_G(x')))$ . These considerations imply that  $z_G(D') \subseteq G \setminus (m_G(L \setminus K))$  (where  $\setminus$  is understood as componentwise set-theoretic difference). However,  $D$  is exactly constructed in such a way that for *any* morphism  $q : Q \rightarrow G$  between typed attributed partial triple graphs, the existence of an element  $x \in Q$  such that  $q_G(x) \in G \setminus D$  implies the existence of an element  $x' \in Q$

such that  $q_G(x') \in m_G(L \setminus K)$ . This can be seen by checking the definitions of the sets constituting  $D$ ; again, we do so exemplary.

As the statement is trivial when already  $q_G(x) \in m_G(L \setminus K)$ , we assume  $x \in Q$  such that  $q_G(x) \in G \setminus D$  but  $q_G(x) \notin m_G(L \setminus K)$  in the following. First, as  $V_D^X = V_G^X \setminus m_{V_L}^X(V_L^X \setminus V_K^X)$  for  $X \in \{S, T\}$ , it is not possible that  $x \in V_Q^X$ . Therefore, assume  $x \in E_Q^X$ . Then,  $q_{E_Q}^X(x) \in E_G^X \setminus E_D^X$  but  $q_{E_Q}^X(x) \notin m_{E_L}^X(E_L^X \setminus E_K^X)$  implies  $\text{src}_G^X(q_{E_Q}^X(x)) \notin V_D^X$  (or the analogous statement for the target-function). But this means  $\text{src}_G^X(q_{E_Q}^X(x)) = q_{V_Q}^X(\text{src}_Q^X(x)) \in m_{V_L}^X(V_L^X \setminus V_K^X)$ . Exactly the same argument applies to  $NA_D^X$ . For  $EA_D^X$ , one first obtains an edge  $e \notin E_G^X \setminus E_D^X$ , and either directly  $e \in m_{E_L}^X(E_L^X \setminus E_K^X)$  or its source or target stems from  $m_{V_L}^X(V_L^X \setminus V_K^X)$  as above.

Further, assuming  $q_{V_Q}^{\tilde{S}}(x) \in V_G^{\tilde{S}} \setminus (V_D^{\tilde{S}} \cup m_{V_L}^{\tilde{S}}(V_L^{\tilde{S}} \setminus V_K^{\tilde{S}}))$ , there are two possibilities for this to happen. First,  $\sigma_{V_G}(q_{V_Q}^{\tilde{S}}(x)) \notin V_D^{\tilde{S}}$ . But, exactly as for the source or target functions above, since  $\sigma_{V_G}(q_{V_Q}^{\tilde{S}}(x)) = q_{V_Q}^{\tilde{S}}(\sigma_{V_Q}(x)) \notin V_D^{\tilde{S}}$ , the node  $\sigma_{V_Q}(x) \in V_Q^{\tilde{S}}$  is the element with image in  $m_{V_L}^{\tilde{S}}(V_L^{\tilde{S}} \setminus V_K^{\tilde{S}})$ . Secondly, there could exist an element  $n' \in V_G^{\tilde{T}}$  such that  $\iota_{V_G^{\tilde{S}}}(q_{V_Q}^{\tilde{S}}(x)) = \iota_{V_G^{\tilde{T}}}(n')$  and  $\tau_{V_G}(n') \notin V_D^{\tilde{T}}$ . Then, since  $V_Q^{\tilde{T}}$  is a pullback of  $V_Q^{\tilde{S}}$  and  $V_G^{\tilde{T}}$  as  $q$  is a morphism (compare, e.g., the second row of Fig. B.27, replacing  $L$  by  $Q$  and  $m$  by  $q$ ), there exists  $x' \in V_Q^{\tilde{T}}$  such that  $\iota_{V_G^{\tilde{S}}}(x) = \iota_{V_G^{\tilde{T}}}(x')$  and  $q_{V_Q}^{\tilde{T}}(x') = n'$ . As already repeatedly used above, this then results in  $q_{V_G}^{\tilde{T}}(\tau_{V_Q}(x')) = \tau_{V_G}(n') \notin V_D^{\tilde{T}}$  and therefore  $q_{V_G}^{\tilde{T}}(\tau_{V_Q}(x')) \in m_{V_Q}^{\tilde{T}}(V_L^{\tilde{T}} \setminus V_K^{\tilde{T}})$ . All remaining cases are completely analogous to the ones we discussed.  $\square$

*Proof of Corollary 5.29.* By Lemma 5.28, a final pullback complement for  $m \circ l$  exists such that, if  $K \xrightarrow{d} D \xrightarrow{g} G$  is that FPBC, it has the property that  $d$  is quasi-injective and  $g \in \mathcal{M}$ . As the second step of the rule application just computes the pushout of  $(d, r)$ ,  $r \in \mathcal{M}$  implies  $h \in \mathcal{M}$  and  $d$  quasi-injective implies  $n$  quasi-injective.  $\square$

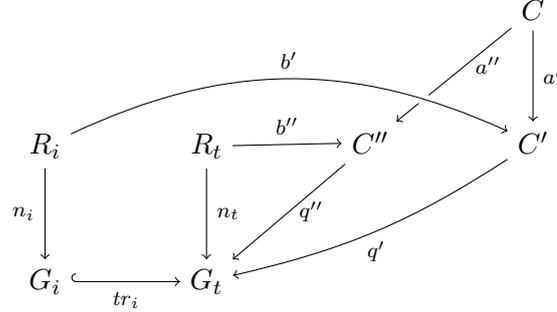


Figure B.28: Preservation of the validity of NACs

### B.3 Proofs from Chapter 6

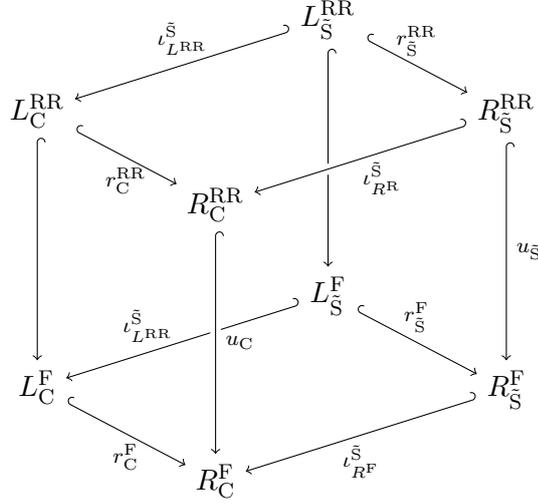
*Proof of Proposition 6.1.* Facts 3.5 and 3.6 immediately imply that  $G \in \mathcal{L}(GG^{NC})$  implies  $G \in \mathcal{L}(GG)$ . Conversely, assuming  $G \in \mathcal{L}(GG)$ , there exists a transformation sequence  $\emptyset \Rightarrow_{r_1, m_1} G_1 \Rightarrow \dots \Rightarrow_{r_s, m_s} G_s \Rightarrow_{r_{s+1}, m_{s+1}} G$  via rules from  $\mathcal{R}$  and  $G \models c$  for every constraint  $c \in NC$ . Since the rules from  $\mathcal{R}$  are monotonic, i.e., they only create structure, this means that  $G_i \models c$  for all  $1 \leq i \leq s$  and all  $c \in NC$ . Then, again Facts 3.5 and 3.6 imply that  $\emptyset \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow \dots \Rightarrow_{\rho_s, m_s} G_s \Rightarrow_{\rho_{s+1}, m_{s+1}} G$  is a transformation sequence, where  $\rho_i = (r_i, nac_i)$ : it holds that  $m_i \models nac_i$  for all  $1 \leq i \leq s + 1$ .  $\square$

*Proof of Lemma 6.2.* For the proof, let  $tr_i \circ n_i$  be such a morphism for an arbitrary  $1 \leq i \leq t - 1$  and assume

$$tr_i \circ n_i \not\models \text{Right}(r_i, nac_i) .$$

By construction of  $nac_i$  this means that there exists a pair of jointly epic quasi-injective morphisms  $b' : R_i \rightarrow C'$  and  $a' : C \rightarrow C'$  and a quasi-injective morphism  $q' : C' \rightarrow G_t$  such that  $q' \circ b' = tr_i \circ n_i$  (compare Fig. B.28), where  $\neg \exists C$  is one of the negative constraints of the underlying TGG from which the NACs have been derived; the morphism  $b' : R_i \rightarrow C'$  contributes one component in the computation of the NAC of  $\rho_i$ .

The  $\mathcal{E}'$ -quasi-injective factorization of the pair of quasi-injective morphisms  $(n_t, q' \circ a')$  then provides quasi-injective morphisms  $b'' : R_t \rightarrow C''$ ,  $a'' : C \rightarrow C''$ , and  $q'' : C'' \rightarrow G_t$  such that  $q'' \circ b'' = n_t$ . In particular,

Figure B.29: Proving revocation rules to belong to  $\mathbf{APTTrG}_{\text{ATG}}$ 

$n_t \not\equiv \neg \exists b'' : R_t \rightarrow C''$ . But by construction, this is one of the morphisms that contributes to the computation of  $nac_t$  (this also remains valid in the case of optimized constraints), which means that

$$n_t \not\equiv \neg \exists b'' : R_t \rightarrow C'' \implies m_t \not\equiv nac_t ;$$

this contradicts the assumption that  $\rho_t$  is applicable at  $m_t$ .  $\square$

*Proof of Lemma 6.3.* For a “standard” revocation rule  $\rho^R := R^F \leftrightarrow L^F : r^F$  this is immediate as  $r^F$  is a morphism between (non-partial) triple graphs.

For a relaxed revocation rule, to show that  $r^{\text{RR}}$  is an  $\mathcal{M}$ -morphism in  $\mathbf{APTTrG}_{\text{ATG}}$  we have to show that the two squares involving the inclusions of the domains of the correspondence morphisms constitute pullback squares. To see this, compare Fig. B.29 that depicts two components of the pullback computing  $r^{\text{RR}}$ . The square at the left front and its opposite square in the background are pullbacks by definition; they compute  $L^{\text{RR}}$  and  $r^{\text{RR}}$ . The bottom square is a pullback since  $r^F$  is an  $\mathcal{M}$ -morphism in  $\mathbf{APTTrG}_{\text{ATG}}$ . Then, pullback composition and decomposition implies that the top square is a pullback as well. The case for the target domain is completely analogous.

Moreover, since in any forward rule  $r^F : L^F \hookrightarrow R^F$  the source component  $r_S^F$  of that morphism is the identity  $id_{R_S}$  and pullbacks are computed componentwise in **APTrG<sub>ATG</sub>**, the source component  $r_S^{\text{RR}}$  of every (relaxed) revocation rule  $\rho^{\text{RR}}$  can be assumed to be an identity. Then, let  $m^{\text{RR}} : R^{\text{RR}} \rightarrow G$  be an arbitrary quasi-injective morphism. The rule  $\rho^{\text{RR}}$  only deletes, which means that applying it at  $m^{\text{RR}}$  in the sesqui-pushout approach only amounts to computing the FPC of  $m \circ r^{\text{RR}}$ . As the source component of  $r^{\text{RR}}$  is an identity, Lemma 5.28 ensures that the source graph of that FPBC coincides with the source graph  $G_S$  of  $G$ .  $\square$

*Proof of Lemma 6.4.* Obviously, whenever  $a_K$  exists, we can just define  $a_L := l_F^{\text{SC}} \circ a_K$ . We prove the existence of  $a_K$ .

The rule  $\rho'$  is a monotonic rule of (typed attributed) triple graphs. Therefore, when regarded as a partial triple graph,

$$L'_F = R'_S \leftarrow L'_C \hookrightarrow L'_C \hookrightarrow L'_C \rightarrow L'_T .$$

Since  $\rho^{-1} \times_k \rho'$  is a general rule in **ATrG<sub>ATG</sub>**, the interface of its forward rule is given by

$$K_F^{\text{SC}} = R_S^{\text{SC}} \leftarrow K_C^{\text{SC}} \hookrightarrow K_C^{\text{SC}} \hookrightarrow K_C^{\text{SC}} \rightarrow K_T^{\text{SC}}$$

according to Definition 5.6. Note that, compared to this definition, since  $\rho^{-1} \times_k \rho'$  is even a rule in **ATrG<sub>ATG</sub>**, we obtain  $K_S^{\text{SC}} = K_C^{\text{SC}}$  and likewise  $K_T^{\text{SC}} = K_C^{\text{SC}}$ . The desired morphism  $a_K : L'_F \rightarrow K_F^{\text{SC}}$  is depicted in Fig. B.30; all its components are obtained from the definition of a short-cut rule as depicted in Figs. 4.2 and 4.3. To show that  $a_K$  really defines a morphism (in **APTrG<sub>ATG</sub>**), we have to show that all four squares commute and that the two central squares are pullbacks. For the three right squares this is evident or follows from the fact that  $k' \circ e_2$  is a morphism.

For the leftmost square, we compute

$$\begin{aligned} \sigma_{R^{\text{SC}}} \circ r_C^{\text{SC}} \circ k'_C \circ e_2^C &= \sigma_{R^{\text{SC}}} \circ r_2'^C \circ e_2^C \\ &= \sigma_{R^{\text{SC}}} \circ e_2'^C \circ r'_C \\ &= e_2'^S \circ \sigma_{R'} \circ r'_C . \end{aligned} \quad \square$$

*Proof of Lemma 6.5.* For the plain repair rules, this is Lemma 5.25. Since  $a_L$  is a morphism in **APTrG<sub>ATG</sub>** (see Lemma 6.4), the computation of

$$\begin{array}{ccccccccc}
 L'_F = & (R'_S & \xleftarrow{\sigma_{R'} \circ r'_C} & L'_C & \xrightarrow{id} & L'_C & \xleftarrow{id} & L'_C & \xrightarrow{\tau_{L'}} & L'_T) \\
 a_K = & e_2^S \downarrow & & k'_C \circ e_2^C \downarrow & & k'_C \circ e_2^C \downarrow & & k'_C \circ e_2^C \downarrow & & k'_T \circ e_2^T \downarrow \\
 K_F^{SC} = & (R_S^{SC} & \xleftarrow{\sigma_{R^{SC}} \circ r_C^{SC}} & K_C^{SC} & \xrightarrow{id} & K_C^{SC} & \xleftarrow{id} & K_C^{SC} & \xrightarrow{\tau_{K^{SC}}} & K_T^{SC})
 \end{array}$$

Figure B.30: Embedding the LHS of a forward rule into the LHS of a forward rule of a derived short-cut rule

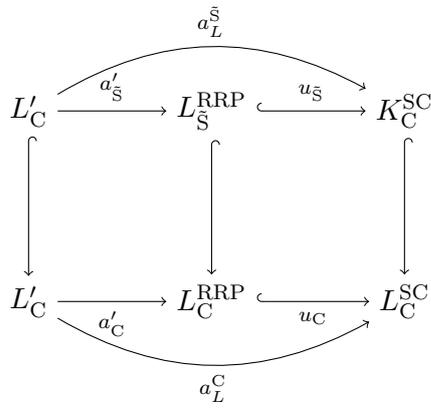


Figure B.31: Proving  $a'$  to be a morphism in  $\mathbf{APTrG}_{ATG}$

the rule's NAC is also performed in that category such that it consists of morphisms in  $\mathbf{APTTrG}_{\text{ATG}}$ .

For the morphisms  $l^{\text{RRP}}$  and  $r^{\text{RRP}}$  of a relaxed repair rule, the claim follows exactly as for relaxed revocation rules, i.e., exactly as in the proof of Lemma 6.3. Thus, we only need to show that in this case the morphisms that define the negative application conditions remain morphisms in  $\mathbf{APTTrG}_{\text{ATG}}$ . For that, it suffices to show that  $a'$  is a morphism in  $\mathbf{APTTrG}_{\text{ATG}}$  (because then, the whole computation of the NAC again takes place in  $\mathbf{APTTrG}_{\text{ATG}}$ ), i.e., it has to induce pullbacks in the usual two places. Figure B.31 depicts the situation for the source side; the argument for the target side is completely analogous. In this diagram, the outer square is a pullback (since  $a_L$  is a morphism in  $\mathbf{APTTrG}_{\text{ATG}}$ ) and  $u_{\bar{s}}$  is a monomorphism (since every component of  $u$  is by definition). Then, a slightly stronger form of pullback decomposition (as, for example, stated as Lemma 1.8. (2) in [163]), implies that the front square is a pullback, as desired.  $\square$

*Proof of Lemma 6.6.* The morphisms  $e_i : A'_i \rightarrow \text{Im } \Phi$  are obtained in the following way: The coproduct  $\coprod_{i \in I} A'_i$  induces a unique morphism  $f : \coprod_{i \in I} A'_i \rightarrow B$  such that  $f \circ i_{A'_i} = \varphi_i$  for all  $i \in I$ , where  $i_{A'_i}$  are the respective coproduct injections. Applying the extremal  $\mathcal{E}$ - $\mathcal{M}$  factorization to  $f$  results in  $\coprod_{i \in I} A'_i \xrightarrow{e} \text{Im } \Phi \xrightarrow{\iota_\Phi} B$ . We define  $e_i := e \circ i_{A'_i}$ .

The uniqueness of the underlying extremal  $\mathcal{E}$ - $\mathcal{M}$  factorization of single morphisms guarantees uniqueness of the such arising factorization of a family of morphisms with the same codomain into an extremal sink of morphisms followed by an  $\mathcal{M}$ -morphism (see, e.g., [81, Fact 3.7] for the binary case). Therefore, given an  $\mathcal{M}$ -morphism  $m : Y \hookrightarrow B$  and a family of morphisms  $e'_i : A_i \rightarrow Y$  such that  $m \circ e'_i = \varphi_i$  for all  $i \in I$ , we can first apply that kind of factorization to the family  $\{e'_i\}_{i \in I}$  to obtain an  $\mathcal{M}$ -morphism  $m' : I \hookrightarrow Y$  and morphisms  $e''_i : A'_i \rightarrow I$  such that  $m' \circ e''_i = e'_i$ . But then,  $\varphi_i = \iota_\Phi \circ e_i$  and  $\varphi_i = (m \circ m') \circ e''_i$  constitute two factorizations of  $\Phi$  into an extremal sink followed by an  $\mathcal{M}$ -morphism. By uniqueness of such a factorization, there exists a unique isomorphism  $j : \text{Im } \Phi \rightarrow I$  such that  $j \circ e_i = e''_i$  and  $m \circ m' \circ j = \iota_\Phi$  for all  $i \in I$ . In particular  $v := m' \circ j$  is the desired  $\mathcal{M}$ -morphism.  $\square$

*Proof of Lemma 6.7.* We prove this statement using induction on  $t$ . For

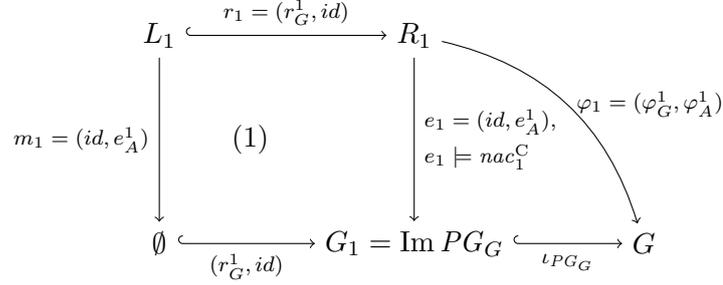


Figure B.32: Proving a one-element partial precedence graph  $PG_G$  to correspond to a one-step transformation

$t = 1$ , we split  $\varphi_1 = \iota_{PG_G} \circ e_1$  as enabled by Lemma 6.6; for this and the following compare Fig. B.32. Since  $PG_G$  is internally complete,  $\text{Im } PG_G \subseteq G$  is completely marked (except for possible attribute values in this image). This first implies that the structural part of  $\text{Im } PG_G$  coincides with (or: is isomorphic to) the one of  $R_1$  because  $e_1$  is quasi-injective. Moreover, it means that every element of  $R_1$  (except for attribute values) is a marking element, i.e., the structural part of  $L_1$  is empty. (Formally, for every  $X \in \{S, \tilde{S}, C, \tilde{T}, T\}$ ,  $V_{L_1}^X = E_{L_1}^X = NA_{L_1}^X = EA_{L_1}^X = \emptyset$ .) Therefore, there exists a quasi-injective morphism  $m_1 = (id, e_A^1) : L_1 \rightarrow \emptyset$ , where  $\emptyset$  is the (structurally empty) start graph of the TGG and  $e_A^1$  is the attribute component of  $e_1$ . Figure B.32 depicts this situation for  $e_1$  being chosen in such a way that the structural part of  $\text{Im } PG_G$  indeed coincides with the one of  $R_1$ . Furthermore, the algebras of  $G_1$  coincide with the ones of  $\emptyset$  because the algebras of  $\emptyset$  coincide with the ones of  $G$  by assumption. Together this means that square (1) is a pushout, i.e., it constitutes an application of  $\rho_1$  at  $m_1$  in case  $m_1 \models nac_1$ . But  $PG_G$  is assumed to be NAC-consistent, i.e.,  $e_1 \models nac_1^C$  which is equivalent to  $m_1 \models nac_1$  by construction of  $nac_1^C$  (since  $e_1$  is the comatch of the transformation). Finally, since  $tr_t = id_{G_t}$  for any  $t$  and we have  $n_1 = e_1$  in our constructed transformation, we also obtain the desired equation  $\iota_{PG_G} \circ tr_1 \circ n_1 = \varphi_1$ .

For the inductive step, assume the assertion to hold for partial precedence graphs with  $t - 1$  markings and let  $PG_G = \{\varphi_1, \dots, \varphi_t\}$  be topologically sorted. First,  $PG'_G = \{\varphi_1, \dots, \varphi_{t-1}\}$  is a (topologically sorted) partial precedence graph again: Consistence, coherence, and totality of each mark-

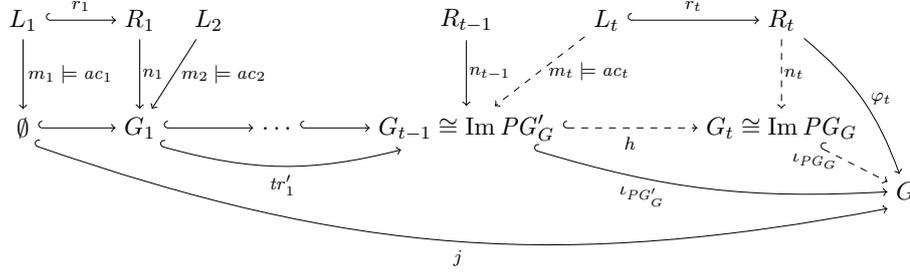


Figure B.33: Obtaining a transformation sequence from a partial precedence graph

ing for  $PG'_G$  are inherited from  $PG_G$ . Moreover, since  $PG_G$  is topologically sorted,  $\varphi_t$  cannot mark an element of  $\text{Im } PG'_G$ . Thus, internal completeness of  $PG_G$  also implies internal completeness of  $PG'_G$ . In particular, applying the induction hypothesis to  $PG'_G$ , we obtain a sequence of transformations

$$t' = (\emptyset \Rightarrow_{\rho_1, m_1} G_1 \Rightarrow \dots \Rightarrow_{\rho_{t-1}, m_{t-1}} G_{t-1})$$

such that  $G_{t-1} \cong \text{Im } PG'_G$  and  $\iota_{PG'_G} \circ tr'_i \circ n_i = \varphi_i$  for all  $1 \leq i \leq t - 1$ . Here,  $tr'_i : G_i \hookrightarrow G_{t-1}$  denotes the track morphism with respect to the transformation sequence  $t'$ . The situation is depicted in Fig. B.33.

Since  $PG_G$  is marking-consistent and internally complete, all elements of  $\varphi_t(r_t(L_t))$  are already marked (in  $G$ ) by one of the markings  $\varphi_1, \dots, \varphi_{t-1}$ . This implies that the (quasi-injective) morphism  $\varphi_t : R_t \rightarrow G$  restricts to a (quasi-injective) morphism  $m_t : L_t \rightarrow G_{t-1} \cong \text{Im } PG'_G$ . Moreover, coherence implies that the span  $G_{t-1} \xleftarrow{m_t} L_t \xrightarrow{r_t} R_t$  is a pullback of  $G_{t-1} \xrightarrow{\iota_{PG'_G}} G \xleftarrow{\varphi_t} R_t$ : As none of the elements from  $\varphi_t(R_t \setminus r_t(L_t))$  is already marked,  $G_{t-1}$  and  $R_t$  intersect in  $L_t$ . This means that  $r_t$  (the plain version of  $\rho_t$ ) is applicable at  $m_t$  such that the resulting graph  $G_t$  is isomorphic to  $PG_G$ . The morphism  $u : G_t \hookrightarrow G$  that exists by the universal property of  $G_t$  as a pushout is necessarily the morphism  $\iota_{PG_G}$  that includes  $G_t$  (as image of  $PG_G$ ) into  $G$ . This ensures that the desired equations hold and implies  $n_t \models nac_t^C$  by NAC-consistency. As above, this ensures  $m_t \models nac_t$  which means that the application of  $r_t$  at  $m_t$  is even an application of  $\rho_t$ .  $\square$

*Proof of Lemma 6.8.* First, as composition of a total quasi-injective mor-

phism  $n_i$  with  $\mathcal{M}$ -morphisms  $tr_i$  and  $u$ , every  $\varphi_i$  is a total, quasi-injective morphism from  $R_i$  to the (potentially partial) triple graph  $G$ .

The other properties (consistency, coherence, and internal completeness) follow immediately from  $t$  being a sequence of transformations starting at  $\emptyset$  via an easy induction. We only sketch the induction step for NAC-consistency: By the inductive hypothesis, we obtain a partial precedence graph  $PG'_G = \{\varphi_1, \dots, \varphi_{t-1}\}$  from the first  $t-1$  transformation steps such that  $e'_i \models nac_i^C$  for all  $1 \leq i \leq t-1$  where the morphisms  $e'_i$  arise by the image factorization of  $PG'_G$ . When  $h : \text{Im } PG'_G \cong G_{t-1} \hookrightarrow G_t \cong \text{Im } PG_G$  denotes the final transformation step, on the one hand, the morphisms  $e_i := h \circ e'_i$  are the  $\mathcal{E}$ -components of the image factorization of  $PG_G = \{\varphi_1, \dots, \varphi_{t-1}, \varphi_t\}$  where  $\varphi_t := \iota_{PG_G} \circ n_t$  (compare Fig. B.33 also for this argument). This means, these are the morphisms for which we have to show that they satisfy the NACs  $nac_i^C$ . On the other hand, each  $e_i$  is the composition of the comatch  $n_i$  with the track morphism  $tr_i : G_i \rightarrow G_t$  (by the inductive assumption). Therefore, Lemma 6.2 implies  $e_i \models nac_i^C$  (because  $nac_i^C$  is defined as  $\text{Right}(r_i, nac_i)$ ).  $\square$

*Proof of Theorem 6.9.* Since sub-triple-graphs  $U$  of  $G$  with  $U \in \mathcal{L}(GG)$  correspond to the transformation sequences that construct them, and these, in turn, correspond to partial precedence graphs of  $G$ , the first statement immediately follows from Lemmas 6.7 and 6.8.

The stated equations follow by inspecting the proofs of Lemmas 6.7 and 6.8: The decomposition of markings  $\varphi_i$  into matches  $m_i$  (as provided by operation *Trafo* in Lemma 6.7) and the composition of comatches  $n_i$  to markings  $\varphi_i$  (as provided by operation *PG* in Lemma 6.8) are inverse operations. This is, because in the case of monotonic rules, the comatch of a rule application determines the match. To prove the final statement, first notice that it suffices to consider the underlying plain rules and ignore the NACs of the rules. We assume the NACs of all rules to be derived from the same set of negative constraints and to be invalid if and only if an application would trigger a violation of one of the constraints. Therefore, a reordering of rule applications (resulting in an isomorphic graph) cannot ever introduce violations of application conditions (this is proved in the same way as Lemma 6.2).

For plain, monotonic rules  $r_1, r_2$ , however, a transformation sequence  $t = (G_0 \Rightarrow_{r_1, m_1} G_1 \Rightarrow_{r_2, m_2} G_2)$  is sequentially independent if and only

if the first transformation does not create an element that the second one matches. Switching the transformations to obtain the transformation sequence according to the Local Church–Rosser Theorem results in the transformation sequence  $t' = (G_0 \Rightarrow_{r_2, m'_2} G'_1 \Rightarrow_{r_1, m'_1} G_2)$  as depicted in Fig. B.34. Here,  $m'_2$  exists by the definition of sequential independence and can directly serve as match (because the rules are monotonic) and  $m'_1 = h'_2 \circ m_1$ . Pushout composition and decomposition guarantees that applying  $r_1$  at this match indeed results in  $G_2$  again with comatch  $n'_1 = h_2 \circ n_1$ . Moreover, the equalities  $n'_1 = h_2 \circ n_1$  and  $n_2 = h'_1 \circ n'_2$  ensure that when applying the operation PG to transformation sequences  $t$  and  $t'$  the switch of independent steps only affects the order of the markings  $\varphi_1$  and  $\varphi_2$  but not the underlying mappings. Inductively, then, switching independent transformations merely reorders partial precedence graphs.

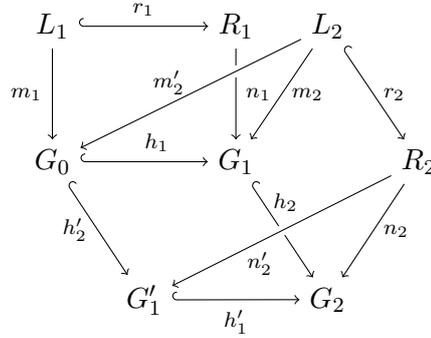


Figure B.34: Switching independent transformations of monotonic rules

Conversely, given two topological sortings  $PG_G = \{\varphi_1, \dots, \varphi_t\}$  and  $PG'_G = \{\varphi_{\pi(1)}, \dots, \varphi_{\pi(t)}\}$  of the same partial precedence graph, where  $\pi : \{1, \dots, t\} \rightarrow \{1, \dots, t\}$  is some bijection,  $\varphi_1$  is necessarily independent of all other markings (i.e.,  $\varphi_j \not\prec_d \varphi_1$  for all  $1 \leq j \leq t$ ). Let

$$PG''_G = \{\varphi_1, \varphi_{\pi(1)}, \dots, \widehat{\varphi_{\pi(j_0)}}, \dots, \varphi_{\pi(t)}\}$$

be the sequence that results from switching  $\varphi_1$  to the front of the second sequence (i.e.,  $\pi(j_0) = 1$  and  $\widehat{\varphi_{\pi(j_0)}}$  denotes omission). On the level of transformation sequences then, the considerations on independence mean that  $\text{Trafo}(PG'_G)$  is switch equivalent to  $\text{Trafo}(PG''_G)$ . Therefore, recursively,  $\text{Trafo}(PG_G)$  and  $\text{Trafo}(PG'_G)$  are.  $\square$

*Proof of Lemma 6.10.* As  $PC_{G'}^\delta$  does not contain any new marking, the preservation of coherence is clear.

For marking-consistency, it suffices to note that the dependency relation induced by the markings on  $PC_{G'}^\delta$  is a subrelation of the one on  $PC_G$  as no new dependencies are added. In particular, if the relation is acyclic on  $PC_G$ , it is so on  $PC_{G'}^\delta$ . The same argument shows that topological sortings are preserved.

NAC-consistency is preserved because the ordinary NACs are evaluated at the image of  $PC_{G'}^\delta$  and, by construction (for each marking  $\varphi'_i$  the domain is a sub-partial-triple-graph of the domain of  $\varphi_i$ ), the image of  $PC_{G'}^\delta$  is a sub-partial-triple-graph of the image of  $PC_G$ . This implies that a NAC violation of one of the markings of  $PC_{G'}^\delta$  could only stem from a NAC violation of the corresponding marking in  $PC_G$ .

Finally, if an element  $x \in G$  is marked by  $\varphi_i$ , it is either deleted by the delta or marked by  $\varphi'_i$  in  $G'$  (where  $\varphi'_i$  is obtained from  $\varphi_i$  as defined above). In particular if  $PC_G$  marks  $\text{Im } PC_G$  completely,  $PC_{G'}^\delta$  marks  $\text{Im } PC_{G'}^\delta$  completely.  $\square$

*Proof of Lemma 6.11.* First, except for filter-NAC-consistency and the statement about partial precedence graphs, Lemma 6.10 shows the preservation of consistency properties for  $PC_{G'}^t$  (because a transformation step is a special case of a delta). Thus, we have to specifically argue for filter-NAC-consistency and the statement about partial precedence graphs. For the other properties, we only have to show that they are also preserved under union with the comatch  $n^F$ .

Since an application of a forward rule does not change the source graph of the (partial) triple graph to which it is applied ( $r_S^F = id$  for every forward rule  $\rho^F$ ),

$$\varphi_i \models fnac_i^C$$

immediately implies

$$\varphi'_i \models fnac_i^C$$

for every  $1 \leq i \leq t$  (because we assume filter NACs to only forbid source elements). Moreover,  $m^F \models fnac \iff n^F \models fnac^C$ . Therefore  $PC_{G'}^F$  (and  $PC_{G'}^t$ ) are also filter-NAC-consistent.

Since  $m^F$  is legal, required elements of  $r^F$  are matched to elements that are marked by valid markings from  $PC_G$ . This means, if  $\varphi_i \in PC_G$  marks

an element  $x \in G$  that is matched by a required element of  $r^F$ , the partial cover  $PC_{G'}^F$  contains a dependency  $\varphi'_i <_d n^F$ . However,  $m^F$  being legal also means that marking elements of  $r^F$  are matched to elements in  $G$  that are not yet marked by  $PC_G$ . This means  $n^F \not<_d \varphi'_i$  for all  $1 \leq i \leq t$  (and  $n^F \not< n^F$  by definition). In particular, adding  $n^F$  to  $PC_{G'}^t$  cannot introduce a new cycle with regard to  $<$ .

Because  $PC_{G'}^t$  is NAC-consistent and  $m_{\text{Im}}^F \models nac^F$ , where  $m_{\text{Im}}^F$  is the image-restriction of  $m^F$ , Lemma 6.2 shows that no marking  $\varphi'_i$  can violate its NAC (evaluated in the image of  $PC_{G'}^F$ ). Furthermore,  $m_{\text{Im}}^F \models nac^F \iff e^F \models nac^C$ , where  $e^F : R \rightarrow \text{Im } PC_{G'}$  is the morphism with  $\iota_{\text{Im } PC_{G'}} \circ e^F = n^F$  that exists according to Lemma 6.6. Lemma 6.2 is applicable and the just used equivalence holds by considering the according transformations in  $\text{Trafo}(PC_{G'}^F)$ . Thus,  $PC_{G'}^F$  is NAC-consistent.

As already mentioned above,  $m^F$  being legal by definition means that marking elements of  $r^F$  are matched to elements in  $G$  that are not yet marked by  $PC_G$ . This, in turn, implies that  $n^F$  only marks elements that are not yet marked by  $PC_{G'}^t$ . In particular, coherence is preserved.

Regarding internal completeness, it suffices to note that, because all required elements of  $r^F$  are matched to already marked ones, all elements of  $\text{Im } PC_{G'}^F \setminus \text{Im } PC_{G'}^t$  are elements that are newly marked by  $n^F$ . Thus, also internal completeness is preserved.

Furthermore, as the transformation  $t : G \Rightarrow_{\rho^F, m^F} G'$  does not delete elements, if every marking  $\varphi_i \in PC_G$  is total, so is every  $\varphi'_i \in PC_{G'}^t$ . As also  $n^F$  is total,  $PC_{G'}^F$  is a partial precedence graph if  $PC_G$  is.

Finally, if  $1 \leq j_0 \leq t$  is the smallest index such that  $\varphi_{j_0}$  is an invalid marking, legality of the match implies

$$\varphi'_j < n^F \implies \varphi'_{j_0} \not< \varphi'_j$$

for every  $1 \leq j \leq t$ . This means,  $n^F$  and every marking on which it depends can be sorted in front of  $\varphi'_{j_0}$ , the first invalid marking.  $\square$

*Proof of Lemma 6.12.* By definition,  $\text{dom } \varphi_i$  is an  $\mathcal{M}$ -subobject of  $R_i$  in  $[\mathcal{A}, \mathbf{AGraph}]/ATG$ . Hence, the desired revocation rule  $r_i^{\text{RR}}$  is obtained by just pulling back  $r^F$  along the inclusion of that domain, according to Definition 6.8. The rest is immediate by this definition and Corollary 5.29.  $\square$

*Proof of Lemma 6.13.* First, since  $r^R$  only deletes, its application does not create or mark any elements such that marking-consistence and coherence of a partial cover are not affected by it. Internal completeness follows because the application of a revocation rule, at a legal match, is such that it deletes all elements whose marking would be lost in the revocation-induced partial cover.

Moreover, let  $\varphi_i$  be the marking that is revoked by  $t$ . Since  $m^R$  is a legal match for  $r^R$ , all  $\varphi_j \in PC_G$  with  $\varphi_j < \varphi_i$  are valid. As the application of  $r^R$  only deletes elements, it cannot introduce any new violations of NACs or filter NACs. First, this means that the revocation also preserves NAC- and filter-NAC-consistency. Furthermore, because the introduction of filter NAC violations is excluded, a delta-induced partial marking  $\varphi'_j$  such that  $\varphi_j$  is valid and  $\varphi_i \not\prec \varphi_j$  could only be invalid if the transformation  $t$  deleted an element from its domain. However,  $\varphi_i \not\prec \varphi_j$  in particular entails  $\varphi_i \not\prec_d \varphi_j$  which means

$$\varphi_i(R_i \setminus r_i(L_i)) \cap \varphi_j(L_i) = \emptyset .$$

Furthermore,  $PC_G$  being coherent implies

$$\varphi_i(R_i \setminus r_i(L_i)) \cap \varphi_j(R_j \setminus r_j(L_j)) = \emptyset$$

as no element is marked twice. Summarizing, we obtain

$$\varphi_i(R_i \setminus r_i(L_i)) \cap \varphi_j(R_j) = \emptyset ,$$

which, in particular, implies

$$\varphi_i(R^R \setminus r^R(L^R)) \cap \varphi_j(R_j) = \emptyset , \quad (\text{B.4})$$

as  $R^R$  and  $L^R$  are sub-triple graphs of  $R_i$  and  $L_i$ , respectively. In particular, the application of  $r^R$  at  $m^R$  does not specify the explicit deletion of any element in the image of the marking  $\varphi_j$ . Thus, it remains to exclude that such a deletion happens implicitly. But also implicit deletions are excluded by the above equation: According to Lemma 5.28 (and the discussion in front of its statement), an implicitly deleted element of  $\varphi_j(R_j)$  would either be a dangling edge, or an element from the domain of a correspondence morphism whose image under that correspondence morphism is deleted by the transformation, or an element from the correspondence graph whose preimage in one of the domains of the correspondence morphisms is deleted.

In either case, the intersection in Eq. (B.4) cannot be empty for this to happen.

Finally, to show that  $\varphi'_j$  is valid, it remains to show that it is not potentially invalid. For this, assume  $\varphi'_j$  to be potentially invalid. By definition, this means that there is an invalid marking  $\varphi'_q$  such that  $\varphi'_q < \varphi'_j$ . But because  $\varphi_j$  is valid and  $\varphi'_q < \varphi'_j$  implies  $\varphi_q < \varphi_j$ , the transformation  $t$  must have invalidated that marking. By the above considerations, this means  $\varphi_i <_d \varphi_q$ . But this contradicts the assumption  $\varphi_i \not\prec \varphi_j$ .  $\square$

*Proof of Lemma 6.14.* Before starting the actual proof, in preliminary considerations we state the ramifications of the assumptions from Remark 6.3, namely the assumption that  $\rho$  is computed in a sequence of rule applications such that all further rule applications (transitively) depend on the first. By condition (2) of the definition of a legal match (necessity of revocation), the underlying marking  $\varphi_{i_0}$  of the first rule that contributes to  $\rho$  is invalid. By our assumption, all rules that contribute to  $\rho$  depend on the first rule, i.e., all markings  $\varphi_i$  that correspond to such a rule depend on  $\varphi_{i_0}$  and are at least potentially invalid. This means, no valid marking  $\varphi_j$  can depend on such a marking  $\varphi_i$ .

For marking-consistency, we have to check that the newly added markings, i.e., the ones that stem from the concurrent rule  $\rho'$  do not introduce cycles in the dependency relation. Therefore, let  $\varphi'_q$  be a marking that already stems from a marking  $\varphi_q \in PC_G$  and  $\varphi'_j$  one of the newly added ones such that  $\varphi'_q < \varphi'_j$ . We show that  $\varphi'_j \not\prec \varphi'_q$ ; this means we have to show that  $\varphi'_j$  does not mark any element that  $\varphi'_q$  (transitively) requires. First, by definition of a legal match for a relaxed repair rule,  $m^{\text{RP}}$  is such that  $\rho^{\text{F}}$  (and therefore  $\rho^{\text{RP}}$ ) only marks elements that (i) stem from the common kernel of  $\rho^{-1} \times_k \rho'$  (i.e., elements that had been marked by one of the rules from  $\rho$ ) or (ii) had not yet been marked. As every marking element of a forward rule  $r_j^{\text{F}}$ , where  $r_j$  is one of the underlying rules of  $\rho'$ , is also a marking element of  $\rho^{\text{F}}$ , the same holds true for the newly obtained markings  $\varphi'_j$ . Consider (i): If  $\varphi'_q$  became (transitively) dependent on  $\varphi'_j$  because  $\varphi'_j$  marks such an element, i.e., if  $\varphi'_j$  preserves an element that  $\varphi_q$  (and  $\varphi'_q$ ) transitively require,  $\varphi_q$  would depend on (at least) one marking  $\varphi_{i_u}$  that stems from  $\rho$ , namely the marking that originally marked that element. By the preliminary considerations this would imply that  $\varphi_q$  is potentially invalid. However, required elements of  $\varphi'_j$  that are free to be

matched are only matched to validly marked elements by definition of a legal match. This means,  $\varphi'_j$  could only (transitively) depend on  $\varphi'_q$  because that marks an element that  $\varphi'_j$  transitively needs for one of its required elements whose match is already fixed by the match for  $\rho$ . Such an element needs to be a required element for some marking  $\varphi_{i_v}$  from  $\rho$  and we obtain  $\varphi_q < \varphi_{i_v}$ . Thus, we have markings  $\varphi_{i_u}$  and  $\varphi_{i_v}$  from  $\rho$  such that  $\varphi_{i_u} < \varphi_q < \varphi_{i_v}$ , which is excluded by the definition of a legal match.

In case (ii),  $\varphi_q$  cannot depend on an unmarked element by internal completeness. In summary, we obtain marking-consistency of  $PC_{G'}^{\text{RP}}$ , provided marking-consistency and internal completeness of  $PC_G$ .

NAC- and filter-NAC-consistency and coherence are proved exactly as in the case of forward rules. For internal completeness, it suffices to observe that every element that had been marked by one of the underlying rules of  $\rho$  but that does not get marked by one of the underlying rules of  $\rho'$  is an element that gets deleted by the application of the repair rule. Furthermore, every element that gets newly created is also marked by one of the newly added markings that stem from  $\rho'$ .

For the final statements about the existence of topological orderings, the considerations for marking consistency also imply that  $n^{\text{RP}} \circ y \circ \iota_j \not\prec \varphi'_i$  for all  $1 \leq i \leq i_0 - 1$  and all  $n^{\text{RP}} \circ y \circ \iota_j$  that are newly added to  $PC_{G'}^{\text{RP}}$ . Therefore, the markings  $\varphi'_1, \dots, \varphi'_{i_0-1}$  can constitute the start of a topological sorting of  $PC_{G'}^{\text{RP}}$ . Furthermore, as in the case of the forward rules, the markings on which the newly added markings  $n^{\text{RP}} \circ y \circ \iota_j$  (transitively) depend cannot themselves depend on one of these markings. Therefore, those of them that do not already belong to  $\varphi'_1, \dots, \varphi'_{i_0-1}$  can be sorted behind these but in front of the newly added markings  $n^{\text{RP}} \circ y \circ \iota_j$ . Thus, the desired topological sorting exists. Furthermore, by preservation of NAC- and filter-NAC-consistency, the only possibility for one of the markings in front of the markings  $n^{\text{RP}} \circ y \circ \iota_j$  to become invalid is that the transformation  $t$  (implicitly) deletes an element that such a marking matches. That this is not possible is showed exactly as in the case of revocation rules (see Lemma 6.13).  $\square$

## **C Scientific CV**

This page contains personal data. It therefore is not part of the digital publication.