TECHNISCHE
UNIVERSITÄT
DARMSTADT

Static Verification Techniques for Attributed Graph Transformations

DEM FACHBEREICH ELEKTROTECHNIK UND INFORMATIONSTECHNIK
DER TECHNISCHEN UNIVERSITÄT DARMSTADT
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
EINES DOKTOR-INGENIEURS (DR.-ING.)
GENEHMIGTE DISSERTATION

VON

DIPL.-ING. FREDERIK DECKWERTH

GEBOREN AM

10. MÄRZ 1982 in KRONBERG IM TAUNUS

REFERENT: PROF. DR. ANDREAS SCHÜRR
KORREFERENT: PROF. DR. ANNEGRET HABEL

TAG DER EINREICHUNG: 07.07.2016
TAG DER DISPUTATION: 02.11.2016

D17

DARMSTADT 2017

ABSTRACT

Graph transformation with its formal foundations and its broad range of theoretical results, on the one hand, and competitive tool support, on the other hand, constitutes an effective framework for model-driven software development. Within the last decade, the theory of algebraic graph transformations has been developed towards a comprehensive formal framework including several sophisticated results on modeling, analysing, and verifying graph transformation systems. Prominent theoretical results are the static verification of consistency constraints as well as static conflict detection and conflict resolution techniques. Consistency constraints provide means to declaratively define global assertions that must remain true. Conflict detection and resolution techniques provide means to statically discover potential unintended interactions of graph transformations.

Based on the framework for algebraic graph transformations several model transformation tools were developed over the last years. However, in order to become suitable for the practical needs in every-day software engineering, these tool oriented graph transformation approaches integrate language concepts that go beyond the simple manipulation of plain graphs. An important concept is the treatment of data values such as integers, booleans, and strings. The integration of primitive data attributes within the graph structure is indispensable to model almost all realistic systems, since they combine the structural aspects of a system with data aspects such as computations of values. While in the last years, many advanced language concepts were adapted from the tool oriented approaches and integrated within the theory of algebraic graph transformations, there is currently no theoretical approach that appropriately reflects the de-facto data attribute handling approach of practical implementations. Thus, the main body of theoretical results does not immediately apply to those implemented approaches. As a result, current tool support for analysis and verification techniques of attributed graph transformation systems is rather limited.

This thesis attempts to close this gap. To this end, a framework for attributed graph transformation systems is proposed. In contrast to existing approaches, the proposed framework reflects more closely the attribute handling of current state of the art graph transformation implementations. We show that our proposed approach preserves the fundamental theoretical results of the algebraic approach for graph transformations. Additionally, we verify the well-known results for the static verification of consistency constraints, conflict detection, and conflict resolution by confluence analysis within our framework. Finally, a prototypical implementation is provided to show that the theoretical concepts can be realized. Moreover, to assess its potential for analyzing real world applications, the prototype is applied to analyze a case study from the enterprise modeling domain.

ZUSAMMENFASSUNG

Aufgrund der Vielzahl an formal fundierten theoretischen Resultaten bezüglich der Analyse und Verifikation von Softwaresystemen sowie der ausgereiften Werkzeugunterstützung, bieten Graphtransformationen ein effektives Werkzeug zur modellgetriebenen Entwicklung von Software. Innerhalb der letzten Jahre hat sich der algebraische Ansatz zur Formalisierung von Graphtransformationssystemen zu einem umfassenden Rahmenwerk zur Modellierung, Analyse und Verifikation entwickelt. Prominente Beispiele dafür sind die statische Verifikation von Konsistenzbedingungen sowie Techniken zur Konfliktdetektion und zur automatischen Konfliktauflösung. Konsistenzbedingungen sind ein Mittel zur deklarativen Beschreibung von Bedingungen, welche immer erfüllt sein müssen. Techniken für die Konfliktdetektion und Konfliktauflösung ermöglichen es, statisch die Interaktionen von einzelnen Graphtransformationsschritten zu analysieren um unbeabsichtigte Wechselbeziehungen aufzuspüren.

Darüber hinaus hat sich auf dieser Grundlage eine vielseitige Landschaft ausgereifter Graphtransformationswerkzeuge etabliert. Um jedoch den Anforderungen, welche sich bei der modellbasierten Entwicklung realer Systeme ergeben, gerecht zu werden, bieten alle modernen Graphtransformationswerkzeuge Sprachkonzepte, die über die reine Manipulation einfacher Graphstrukturen hinausgehen. In dieser Hinsicht ist die Einbettung von Datenattributen in Form von Zahlen oder Zeichenketten in die Graphstruktur sowie deren Manipulation in Form von Berechnungen eine der grundlegenden Erweiterungen für die Modellierung realer Softwaresysteme. Während es in den letzten Jahren gelang, viele dieser innovativen Sprachkonzepte aus der Welt der Graphtransformationswerkzeuge in das theoretische Rahmenwerk der algebraischen Graphtransformationen zu überführen, bildet die Einbettung von Datenattributen hier eine Ausnahme. So ist es bisher nicht gelungen, eine formale Repräsentation zu entwickeln, welche die praktische Behandlung von Datenattributen hinreichend widerspiegelt, um die theoretischen Resultate auf die Werkzeuge zu übertragen. Somit stellt diese Lücke zwischen den theoretischen Konzepten und der praktischen Umsetzung eine der Hauptursachen dar für die bis dato eingeschränkte Werkzeugunterstützung für die Analyse und Verifikation attributierter Graphtransformationssysteme. Im Rahmen dieser Dissertation soll diese Lücke geschlossen werden. Dazu wird ein Rahmenwerk zur Transformation attributierter Graphen beschrieben. Im Gegensatz zu existierenden Ansätzen bildet die vorgestellte Lösung die Bedingungen, wie sie für reale Implementierungen gelten, wesentlich genauer ab. Es wird gezeigt, dass die grundlegenden theoretischen Resultate des algebraischen Ansatzes auch in dem neuen Rahmenwerk gültig sind. Des Weiteren wird die Gültigkeit der bekannten Resultate zur Konsistenzverifikation sowie zur Konfliktdetektion und Konfluenz-Analyse in diesem Rahmenwerk gezeigt. Zum Schluss wird eine prototypische Implementierung vorgestellt und anhand einer Fallstudie gezeigt, dass die praktische Umsetzung der gezeigten theoretischen Resultate sich für die Analyse und Verifikation realer Systeme eignet.

CONTENTS

ACRONYMS

API	Application programming interface
ATS	Adhesive transformation system
CMS	Campus management system
FOL	First order logic
GTS	Graph transformation system
HLR	High-level replacement
LHS	Left-hand side
MDE	Model-driven engineering
NC	Negative constraint
OCL	Object Constraint Language
PB	Pullback
PO	Pushout
RHS	Right-hand side
SMT	Satisfiablity modulo theories
TPGTS	Typed projective graph transformation system
TSGTS	Typed symbolic graph transformation system
UML	Unified Modeling Language
VK-square	Van Kampen square

1 INTRODUCTION

Software influences nowadays nearly every aspect of our daily life, as an integral part of almost every modern electronic device. Moreover, also our economy is massively influenced by software. While in the early days, computers were brought into service in order to deal with relatively simple data management tasks, such as managing inventories, payrolls, etc.; nowadays enterprise applications are complex systems able to manage almost every business function. These functionalities include, for example, order processing, procurement, production scheduling, customer information management, energy management, and accounting. To cope with the increasing complexity of software systems, the design, development, and maintenance of software is more and more considered as an engineering discipline on its own. Model-driven engineering (MDE) is a branch of software engineering that considers models as central artifacts in a semi-automated software production process [BCW12].

In the context of enterprise application engineering model-driven techniques have been successfully applied, recently [Día13]. The aim of enterprise modelling is to support and improve the design, documentation, analysis and administration of business objects and operations using modelling languages [FG98]. Enterprise models provide representations of processes, resources, structures, goals, and constraints relevant for the modeled enterprise. If equipped with precise execution semantics, these models can serve as a basis for automating the coordination of work. Moreover, to determine in advance whether a model exhibits no undesirable behaviour static analysis and verification of these models can be performed, which can greatly improve the reliability of such systems [WVvdA^{+}09].

Graph transformation systems provide a rule-based declarative approach to specify the manipulation of graph-like models. Basically, a graph transformation rule consists of a precondition and a postcondition that describes the effect of applying a rule. Graph transformation has shown its value for enterprise and business process modeling, e. g., in [EGSW07], where graph transformation rules are used to abstractly specify business tasks. Furthermore, graph transformation is a formal technique with a rich body of theoretical results. Prominent examples are the static verification of consistency constraints as well as conflict detection and resolution.

Verification of consistency constraints. Basically, consistency constraints provide means to declaratively define global assertions that must remain true. In the context of business process modeling constraints usually reflect certain business concerns. These concerns may range from external regulations such as legal requirements and standards up to internal directives on the procedures to guarantee seamless business operation. Moreover, the consistency of a system can be enforced constructively by translating the constraints to application conditions for rules. In this way graph transformation rules can only be applied if this does not lead to the violation of a constraint.

Conflict analysis Conflict analysis techniques provide means to statically discover potential unintended interactions of graph transformation rules. For example, in the context of business process modeling we may assume that dozens of tasks (represented as graph transformation rules) operate concurrently on the same data. By conflict detection and analysis techniques we can statically identify and analyse problematic interactions that arise if two tasks modify concurrently the same datum, leading to inconsistencies.

An important aspect in order to be able to handle real world problems is the integration of primitive data *attributes* (e. g., integers, booleans, strings, etc.) within the structural (i. e. graph-like) part of a model. For pure transformation approaches it is usually assumed that the attributes have concrete values; however, this assumption does not apply for the verification of graph transformation systems, in general. More specifically, to verify graph transformation systems, we have to consider the rules themselves, which usually contain constraints and arithmetic expressions over attribute variables. Hence, in order to capture the behaviour of rules for all possible attribute values (especially for unbounded attribute domains) we have to symbolically reason about unevaluated expressions on attribute values.

From a practical point of view, various graph transformation tools exist to actually execute graph transformation rules (including attributes). However, the majority of graph transformation tools aims at transforming large graphs [ABJ$^+$10, BDH$^+$15, GBG$^+$06, LAS14]; there are presently only three tools supporting static verification of consistency constraints and conflict analysis for model transformations [RET11, ABJ$^+$10, AHPZ07]. The situation becomes even worse when considering the verification of attributed graph transformation systems. Tool support for conflict resolution (by confluence analysis) of attributed graph transformation systems is still an open issue.

From a theoretical point of view several approaches exist, to formalize attributed graphs and their transformation. The most prominent approach combines graphs with algebras to specify computations on attribute values [HKT02]. To this end, graphs are extended by an extra kind of nodes reserved for carrying attributes values. Although this representation is theoretically satisfactory, problems arise if the attribute domains are unbounded (e. g., for integer numbers). In this case, including the algebra in the graph structure leads to infinite graphs. This assumption is especially problematic with respect to an implementation, as on a real system the underlying data structures need to be finite. Hence, to provide an implementation these infinite graph data structures have to be projected to appropriate finite data structures. However, in this case the theoretical results need not apply to the implementation anymore, as they are obtained with the assumption of infinite graph data structures. Hence, one has to show that the theoretical results remain valid although the implementation does not comply with all assumption. However, if such a proof for the equivalence of a graph transformation theory operating on infinite graphs and a graph transformation implementation operating on finite data structures exists, it seems more reasonable to directly formalize attributed graphs and transformations without the need for infinite graph data structures. These considerations lead us to the following research questions:

1. *Is it possible to formalize attributed graphs and transformations without the need for infinite graph data structures?*
2. *Do the theoretical results for consistency constraint verification and conflict analysis remain valid in such a formalism?*
3. *To which extent can real world applications benefit from these static verification techniques?*

While the first two items relate to theoretical considerations, the last item requires an implementation to actually perform experiments in order to evaluate the practical impact of the theoretical results.

1.1 Objectives

The objective of this thesis is to develop a theoretical framework to support the static verification of attributed graph transformation systems, with the aim to provide an implementation that is capable to extend current state of the art graph transformation tools by static verification techniques.

The proposed approach is based on symbolic graphs originally introduced by Orejas and Lambers in [OL10b]. Symbolic graphs combine graphs with the expressiveness of first-order logic to define data aspects without the need for infinite (graph) data structures. Orejas and Lambers propose two possible approaches to transform symbolic graphs, namely symbolic and lazy graph transformation [OL10b, OL12]. While for applying a symbolic graph transformation rule, all attribute expressions have to be evaluated when matching the precondition, for applying a lazy graph transformation the evaluation of attribute expressions may postponed after transforming the graph structure. In [KDL+15] we have shown that symbolic graph transformation is too pessimistic for conflict analysis as it produces many false positives (i. e., recognized conflicts which can never occur); in [DKL+16] we have shown that lazy graph transformation is too expressive for conflict detection, i. e., the approach allows for specifying transformations to which the results required to perform constraint verification and conflict analysis do not apply.

In order to overcome these limitations, I propose *projective graph transformation* in this thesis as a new approach for transforming symbolic graphs. Projective graph transformation can be considered as a reasonable compromise between symbolic and lazy graph transformations. In order to transfer the theoretical results for constraint verification and conflict analysis to projective graph transformations, we introduce the new concept of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories. Finally, a prototypical implementation is provided to show that the theoretical concepts can be realized. Moreover, we show that the implementation is sound, although we permit symbolic graphs to carry formulas over undecidable fragments of first-order logic. Towards showing that real world applications can benefit from the proposed techniques, we use the implementation to analyze a case study from the enterprise modeling domain.

1.2 Outline

Chapter 2. An informal introduction to graph transformation, constraint verification and conflict analysis is given. All concepts are motivated by a case study of a *campus management system* (CMS) that serves as running example throughout this thesis. A campus management system is a software system that facilitates various kinds of administrative processes of universities, which may range from student affairs, over course and program portfolio administration, up to facility management.

Chapter 3. An introduction to the formal foundations of modeling and transforming attributed graphs by symbolic graphs and symbolic graph transformation is given. To this end, the basic concepts of category theory required for the algebraic graph transformation approach are recapitulated, and a short introduction to first-order logic languages is given. Thereupon, both results are combined leading to the notion of symbolic graphs and symbolic graph transformation systems, whose application for model transformation is illustrated by means of the running example. The chapter concludes by showing that symbolic graph transformations are insufficient for our purposes.

Chapter 4. This section introduces the new concept of projective graph transformation. The main focus is to show that projective graph transformation is suitable to overcome the difficulties of symbolic graph transformations. To this end the application of projective graph transformations for model transformations is discussed by means of several examples. Moreover we discuss several technical aspects of projective graph transformations and compare this concept with symbolic graph transformations.

Chapter 5. In this chapter the new concept of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems is introduced, which provides the categorical foundation for the remainder of this thesis. Basically, $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems allow for formalizing transformation systems that require distinguished classes for left and right production morphisms as well as for match morphisms. The main contribution of this chapter is to show that the fundamental results of the double pushout approach remain valid for $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems, including the results required for consistency constraint verification and conflict analysis.

Chapter 6. To show that the theoretical results from Chapter 5 apply for projective graph transformation systems, we show in this chapter that projective graph transformation systems are $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive.

Chapter 7. The main contribution of this chapter is the extension of the well known results for consistency constraint verification to projective graph transformation. To this end, the corresponding proofs of Chapter 5 are instantiated for projective graph transformation systems. Moreover, several technical aspects regarding the practical applicability of these techniques are discussed.

Chapter 8. The main contribution of this chapter is the extension of the results for conflict detection and resolution to projective graph transformation systems. To this end, the constructions and corresponding proofs are provided to show that the theoretical results for conflict detection and resolution remain valid in projective graph transformation.

Chapter 9. In this chapter presents the Symbolic Graph Analysis and Verification (SyGrAV) tool prototype, which encompasses implementations for all techniques presented in Chapter 7 and Chapter 8. We give an overview on our efforts and insights gathered during implementing the SyGrAV tool prototype and when analysing the campus management system case study. Additionally, we provide means for the soundness of the current implementation. The chapter concludes with discussing the measurement results.

Chapter 10. This chapter provides an overview on the relevant related work.

Chapter 11. The thesis concludes with a summary of the key contributions, the observations and lessons learned during the elaboration of the approaches, as well a survey on directions for future work.

1.3 Hints for Reading This Thesis

For those readers who are interested mainly in the concepts and results for attributed graph transformation systems, but not so much in the general theory and in the proofs, we advise to skip Chapter 5 and Chapter 6.

2

MOTIVATION AND CONTRIBUTIONS

In this section we introduce a case study of a campus management system. A campus management system (CMS) is a software for organizing the daily business operations of universities. Following [AA10], a CMS typically includes the following components:

1. *Student administrations*, including the administration of student related data such as enrollments, academic progress, transcripts and degrees.

2. *Program portfolio administration*, including the maintenance of degree programs, modules and catalogues, as well as the organization of program specific curricula.

3. *Examination and lecture administration*, including the scheduling of dates and the allocation of rooms for examination and lectures, the processing of registrations for courses and exams, and the documentation of examination results.

The CMS case study serves as a running example throughout this thesis.

In the following, we show how such a system can be modeled by using graph transformations. To this end, we informally introduce in Section 2.1 the concepts of metamodels, models, and graph transformation by means of the case study. Finally, in Section 2.2 we provide a first overview on the techniques for static verification of consistency constraints and conflict detection for typed attributed graph transformations.

2.1 A Graph Transformation Based Data-Centric Workflow Model

A campus management system (CMS) is an enterprise application software tailored to organize the daily business of universities. According to Martin Fowler, “Enterprise applications are about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data” [Fow02]. In the following, we focus on modeling the business processes. Usually, a business process is not monolithic, in fact a business process is usually composed from smaller activities called *tasks*. For example, the process of conducting an examination from its creation to its completion (from an organisational perspective) comprises tasks such as reserving a room, determining a date for the examination, and documenting the results. To realize such a process, tasks have to be usually applied in a specific order. For example, a date has to be fixed for an examination before a room can be reserved for that date. Hence, a process description includes in addition (to the involved tasks) a workflow that prescribes in which sequence and under which conditions tasks have to be conducted.

We use a data-centric approach for workflow modeling. The data-centric approach was first proposed in [NC03] and formed the basis of a substantial effort at IBM Research in the field of business process modeling. The key idea of data-centric workflow modeling is to shift the focus from the actions taken, to the data that are acted upon [NC03]. Hence, the key entities of the data-centric approach are data records called *artifacts*. A useful metaphor to think of an artifact, is a piece of paper (or a file containing a collection of papers). For example, an examination artifact can be considered as a paper form containing fields for exam related data such as the date and the reserved room, as well as the number of students currently registered for that exam. However, not every data record is an artifact. In addition to the (business) relevant data, an artifact stores information about its macro-level life cycle; that is, information about its *key processing stages* and their sequencing.

We follow closely the artifact centric workflow model presented in [BGH$^+$07]. Accordingly, artifacts are classified according to their stored data and characteristic processing stages. The processing stages of an artifact are determined by a set of states. Artifacts are processed by *tasks*; that is, tasks interact with the artifacts by (a) instantiating new artifacts, (b) updating the contents of artifacts, and (c) by triggering state transitions.

In the following, we introduce a graph transformation based artifact-centric workflow model for a campus management system. To this end, we present in Section 2.1.1 a metamodel of the campus management system to characterize the data domain for the artifacts. The artifacts life-cycle is presented in Section 2.1.2. Finally, in Section 2.1.3 the tasks are modeled by graph transformations.

2.1.1 *Modeling the Domain for Artifacts*

A *metamodel* defines the core concepts of a domain. Basically, a metamodel consists of *classes* to specify the entities of a domain and *associations* to define their relations. Figure 2.1 shows the metamodel of our campus management system. The metamodel is denoted using the UML class diagram notation; that is, a metamodel is depicted as a graph whose nodes correspond to classes and edges to associations. According to the classification given in the beginning of this chapter, we partitioned the CMS metamodel into the components student administration, program portfolio administration as well as examination and lecture administration. Additionally, the metamodel contains a component facility management encapsulating information on rooms and their reservations. In the following, we describe the CMS metamodel from the bottom to the top, beginning with the facility management component.

The facility management component consists of the classes Room and Booking. A Room has an *attribute* cap of *domain* int representing its capacity (i. e., the number of available seats in the room). As for examinations, usually only a fraction of seats is used (to prevent cheating), the class Room has an additional attribute capEx, characterizing its examination capacity. To reserve a room, a Booking can be assigned to a Room via the bookings association. A Booking has a start and an end attribute, each defined by a value of domain long. The start and end times of

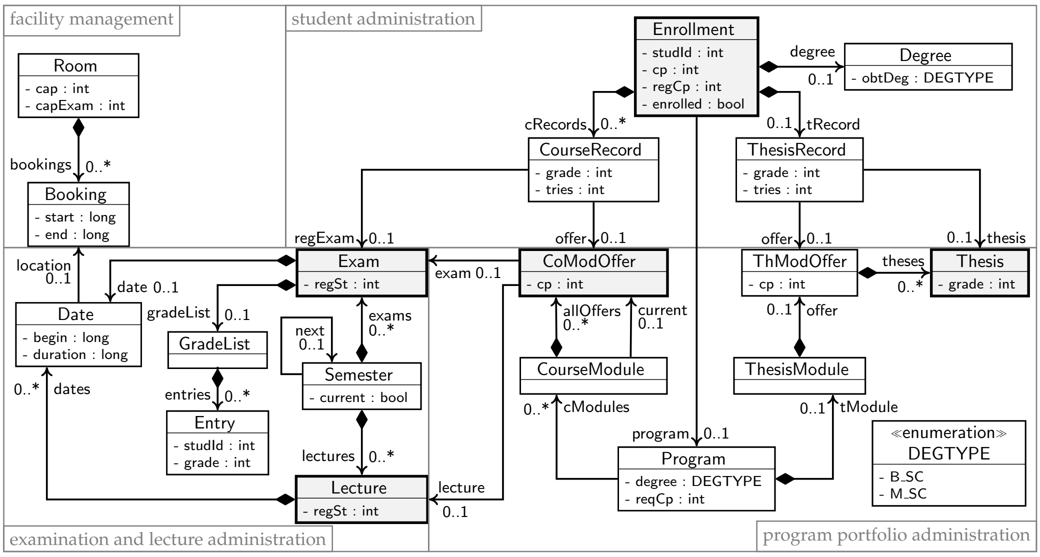


Figure 2.1: The metamodel for the campus management system

a reservation are given by means of the difference (measured in milliseconds) of the desired date with respect to midnight, January 1, 1970. A Booking belongs to at most one Room (indicated by containment symbol, i. e., the diamond at the source of the association). The label 0..* at the target side of association indicates that a Room may refer to at least no (indicated by 0) and at most an arbitrary number of Bookings (indicated by *).

The examination and lecture administration component contains the class Semester that serves as a container of all examinations (Exam) and Lectures offered in the corresponding semester. A Semester has an attribute current of domain bool to mark the current semester. The classes Exam and Lecture both have an attribute regSt of domain int to record the number of registered students. To store dates for examinations and lectures, the classes Exam and Lecture have an association to the class Date, respectively. The class Date provides long values to determine its begin and its duration. While examinations have at most one date, lectures may have several dates (during a semester). Additionally, the class Date has an association to a Booking that specifies the location for the date. To enter the results of an examination a GradeList can be uploaded. The grade list contains a collection of entries (Entry) to store for each student (i. e., studentId) the corresponding grade.

The program portfolio administration component contains the class Program. A Program refers to the modules (i. e., a collection of CourseModules and a ThesisModule) that may be absolved to collect the required credit points (reqCp : int) in order to obtain the degree defined by the degree attribute. In the current version of our CMS, only programs are supported that result in degree of type (DEGTYPE) Bachelor (B_SC) or Master of Science (M_SC). Note that the Program class does not own the course modules as two distinct programs may offer the same course module. A CourseModule contains a collection of course module offers (CourseModOffer). A course module offer has an attribute cp of domain int that determines the credit points that can be acquired by absolving the corresponding Lecture and passing the

Exam assigned to the course module offer. It is distinguished between modules and module offers to allow for changing of the credit points for a module, by simply adding a new course module offer with an adapted value for the credit points. In this way it is ensured that once a student has registered for a module (i. e., to a module offer) it will always acquire the same amount of credit points granted for the module offer at the time of registration. The class CourseModule has an association current that points to the most current course module offer. The structure of a ThesisModule is similar to that of a CourseModule, except that a Program has only one ThesisModule; instead of an Exam, a thesis module offer (ThesModOffer) has a Thesis.

The student administration component contains the class Enrollment to record the academic progress of a student registered for a specific Program. The student is assigned to an Enrollment by its unique student identifier (studId). Additionally, an Enrollment has an attribute enrolled of type bool that tracks the current enrollment status of a student. An Enrollment may own several CourseRecords and a ThesisRecord storing the actual achievements. The overall achievements are stored in the attributes regCp and cp. While, regCp stores the sum of all registered modules, the attribute cp stores the sum of credit points of all completed modules. A registration for a module is given by a course record that is assigned via an offer association to the corresponding module offer. A registration for an Exam (Thesis) is indicated by an exam (thesis) association from the record to the corresponding Exam (Thesis). The status of a course or thesis is stored by the attributes grade and tries, both of domain int. For simplicity we assume the following meaning of grade values: the best grade is 1 the worst is 5 indicating failed; a value equal to 6 is used as initial value.

Note that not every class represents an artifact; that is, some classes do not have their own life cycle; in fact, they rather belong to other artifacts. Thus, the classes that where identified to represent available artifacts are filled gray. For more information on the methodology to identify artifacts we refer to [NC03].

Concrete artifacts are represented by instance models. An *instance model* consists of objects (instances of classes), links (instances of associations), attribute slots (instances of attributes), and attribute values (concrete values of the domain given by the corresponding attribute domain).

Figure 2.2 shows an instance model of the CMS metamodel. Note that the instance model displays only a small section of a CMS system, which is assumed to comprise hundreds of rooms and exams and modules, and thousands of enrollments. The instance model contains an object en1 of *type* Enrollment. The Enrollment en1 is for a student with studId 1234567, and has a registration for the course module offer cmo1). The course module offer cmo1 is for the Exam algebra1 (exAlg1). The Exam exAlg1 has a Date daAlg1. To improve readability, we denote long values that represent dates in the DD.MM.YYYY;hh:mm format, where DD $\in \{1, \ldots, 31\}$ denotes the day, MM $\in \{1, \ldots, 12\}$ denotes the month, and YYYY denotes the year of a date; The time of a date is given in the 24 h clock format hh:mm; Accordingly, the Exam exAlg1 (will) take place at November 26 in the year 2042 at 1 pm and is expected to take 2 hours.

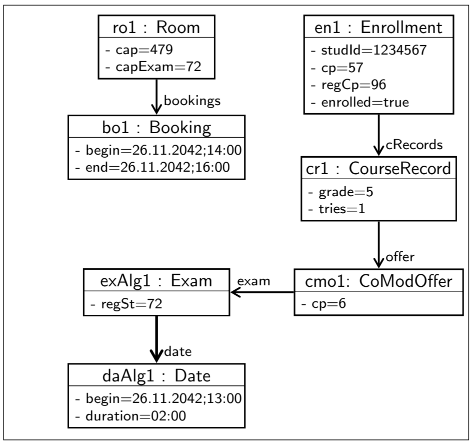


Figure 2.2: An instance model of the CMS metamodel

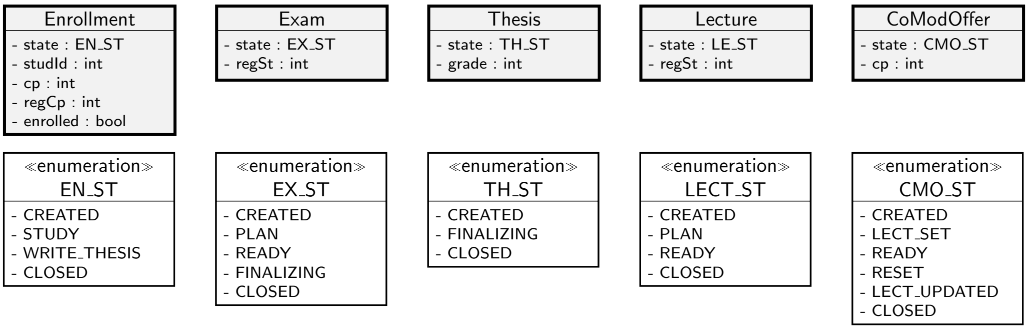


Figure 2.3: The artifact classes with corresponding state attributes

2.1.2 *The Life-Cycle of Artifacts*

To define the macro life cycle of artifacts, we assume that each class that represents an artifact has a state attribute that defines its current state. Figure 2.3 shows the artifact classes with their corresponding state attributes. For example, the class Exam is augmented by a state attribute of domain EX_ST which is an enumeration of the relevant stages in the life cycle of an examination artifact (i. e., an object of type Exam). In the following, we assume that CREATED and CLOSED denominate the initial and final states of any artifact, respectively.

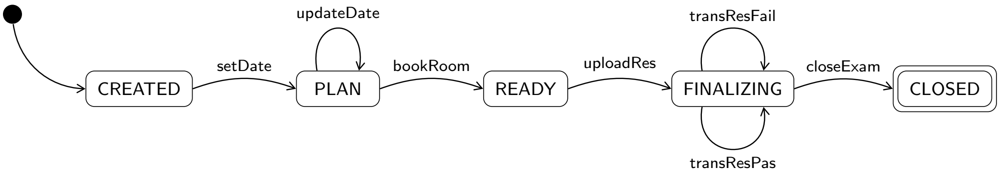


Figure 2.4: The life-cycle of an examination artifact

Figure 2.4 shows the intended live cycle of an examination artifact. The states denote the processing stages and the transitions are labeled with the tasks. More specifically, after the creation of an examination artifact (i. e., an examination artifact currently in the CREATED state) the setDate task can be performed in order to set the date of the corresponding examination. After setting the date once, it can be updated (updateDate), until a room is booked for the examination (bookRoom). After booking a room, the examination is READY to take place from an organizational perspective. Note that conducting the actual examination (in the real world) is not part of the examination artifact life cycle, as it does not immediately affect its data. However, after conducting the examination and correcting the exams, the resulting grades have to be uploaded (uploadRes). Subsequently, each grade is transferred to the records by the tasks transResPas (for passed examinations) and transResFail (for failed examinations). Finally, (i. e., after all grades are transferred) the examination artifact is closed (closeExam).

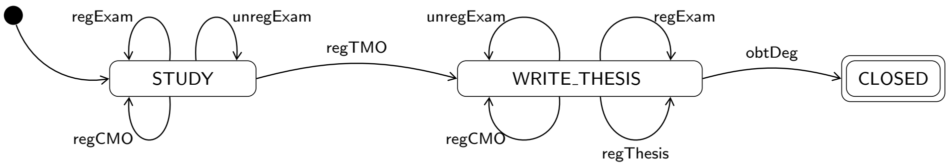


Figure 2.5: The life-cycle of an enrollment artifact

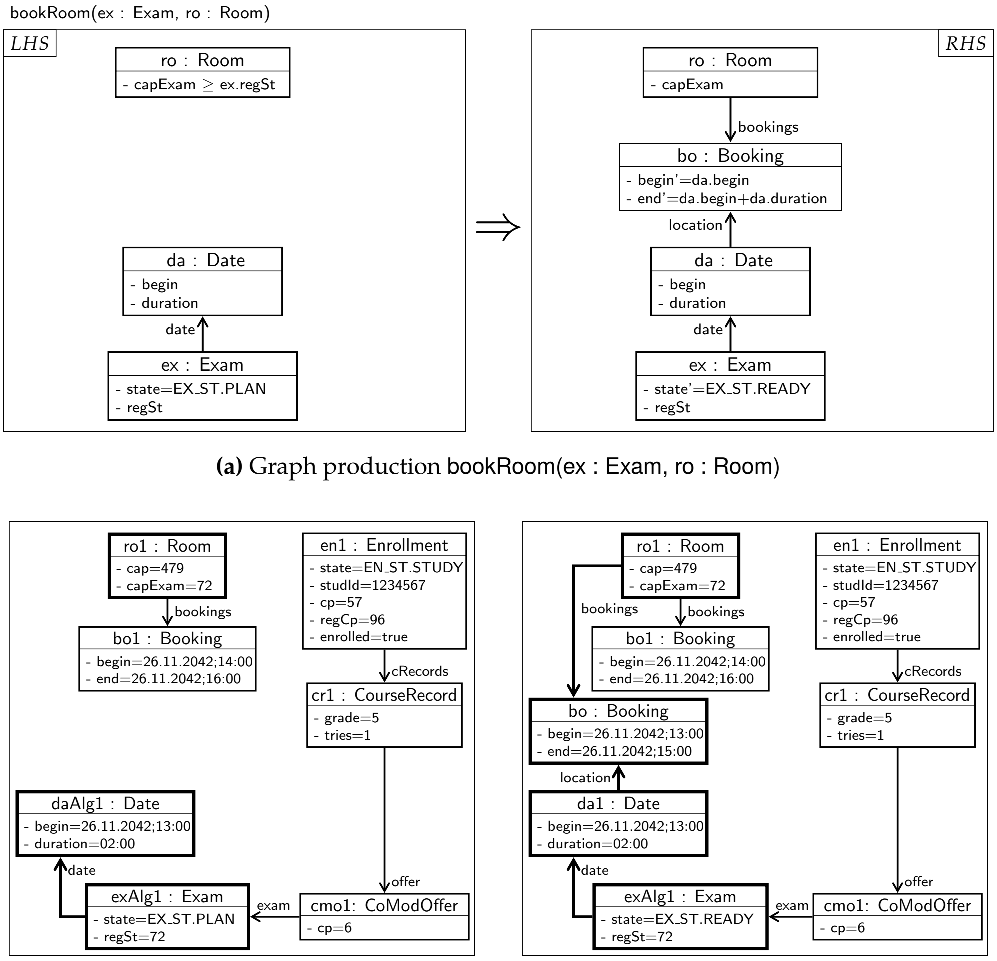
Analogously, the life cycle of an enrollment artifact is depicted in Figure 2.4. After the creation of an enrollment artifact the tasks regExam or regCMO can be invoked to register for an exam or course module offer. Task unregExam can be performed to unregister from an examination. After collecting a certain amount of credit points the task regTMO can be performed to register for the thesis module offer. Now, (i. e., in the WRITE_THESIS state) additionally, the regThesis task can be performed to register for a certain thesis. An enrollment finishes with obtaining a degree (obtDegree).

Note that the artifact life cycles shown in Figures 2.4 and 2.5 are only informal; that is, the notation is intended to give an overview of the desired life cycles. As tasks may invoke different artifacts at the same time, there may also be interactions between the life cycles of different artifacts. This kind of interaction is studied later by means of conflict analysis techniques in Section 2.2.1 and Chapter 8. However, to be able to study these interactions, we first have to assign a precise meaning to tasks.

2.1.3 *Modeling Tasks by Graph Transformations*

In the context of workflow (business process) modeling, tasks are usually specified abstractly by a contract that defines the precondition under which a task may be invoked and the postcondition, i. e., the effect of executing a task. In the following, we us graph transformation to define these contracts.

Graph transformation (GT) [EEPT06] provides a declarative formally founded language for specifying the manipulation of graph based models (such as instance models presented before). The manipulation of graph based models is specified by graph productions. A *graph production* consists of a left-hand side (LHS), which specifies its precondition, and a right-hand side (RHS), which specifies its postcon-



(a) Graph production bookRoom(ex : Exam, ro : Room)

(b) Instance model before applying production bookRoom

(c) Instance model after applying production bookRoom

Figure 2.6: The application of production bookRoom

dition. A graph production is applied to an instance model by (a) searching for such parts of the model that matches the LHS, and (b) updating the model by replacing the matched part by the RHS by first deleting those elements that are matched by the LHS but do not appear in the RHS; then creating those elements that are only in the RHS.

Figure 2.6 shows the application of the graph production that defines the task bookRoom (shown in Figure 2.6a) to an instance model (shown in Figure 2.6b). The production takes as input an Exam ex and a Room ro. The precondition (given by the LHS) is fulfilled if Exam ex is in the PLAN state, has a Date assigned, and the exam capacity of the given Room ro is larger or equal to the number of students registered for the Exam ex; This is denoted by the expression (capExam≥ex.regSt). The effect of the production is is given by the difference of the LHS and RHS; that is, the production is applied by creating a new Booking bo and assigning it to the Room ro. The begin value of the new Booking bo is set equal to the value of da.begin (i. e., the begin value of Date da). The end value of bo is set equal to the sum of da.begin and da.duration. This is defined by the expressions begin'=da.begin and end'=da.begin+da.duration, where primed variables refer to attribute values on RHS and nonprimed variables refer to attribute values on the LHS. For example, the expression i++ that increments i by one is denoted according to this convention as i'=i+1. Note that according to this scheme, attribute expressions can be defined unambiguously without the need to assigning them to a specific side (i. e., LHR or RHS) and object. As we will see later, attribute expression may be written as a conjunction below the production. However, to increase readability, we assign, whenever possible, attribute expression to the corresponding objects.

The production bookRoom is applied to the instance model shown in Figure 2.6b by first finding a match of the LHS. The current matched parts are drawn bold. As the Exam exAlg1 is in the PLANNING state and ro1.capExam is equal to exAlg.regSt, the production can be applied. As a result (shown in Figure 2.6c) a new Booking bo is created, whose begin is equal to da1.begin, and end is equal to the sum of daAlg1.begin and daAlg1.duration.

Figure 2.7a shows the graph production for the task regExam. The production takes as input an Enrollment en and an Exam ex. The precondition ensures that the production can only be applied if for the Enrollment en, there exists a registration for the corresponding course module offer; that is, there exists a CourseRecord cr assigned to the given Enrollment en and the course module offer cmo for the corresponding Exam ex. Moreover, it is only possible to register for an exam if the number of tries is smaller than three (i. e., at most three tries for an exam) the grade is larger than four (i. e., the exam was not passed before), and the student is enrolled (enrolled=true). Moreover, the attribute expression

$$((\text{en.state=EN\_STATE.STUDY}) \vee (\text{en.state=EN\_STATE.WRITE\_THESIS}))$$

ensures that the production can only be applied if the Enrollment en is either in the STUDY or the WRITE_THESIS state. In this case the attribute expression is denoted below the production.

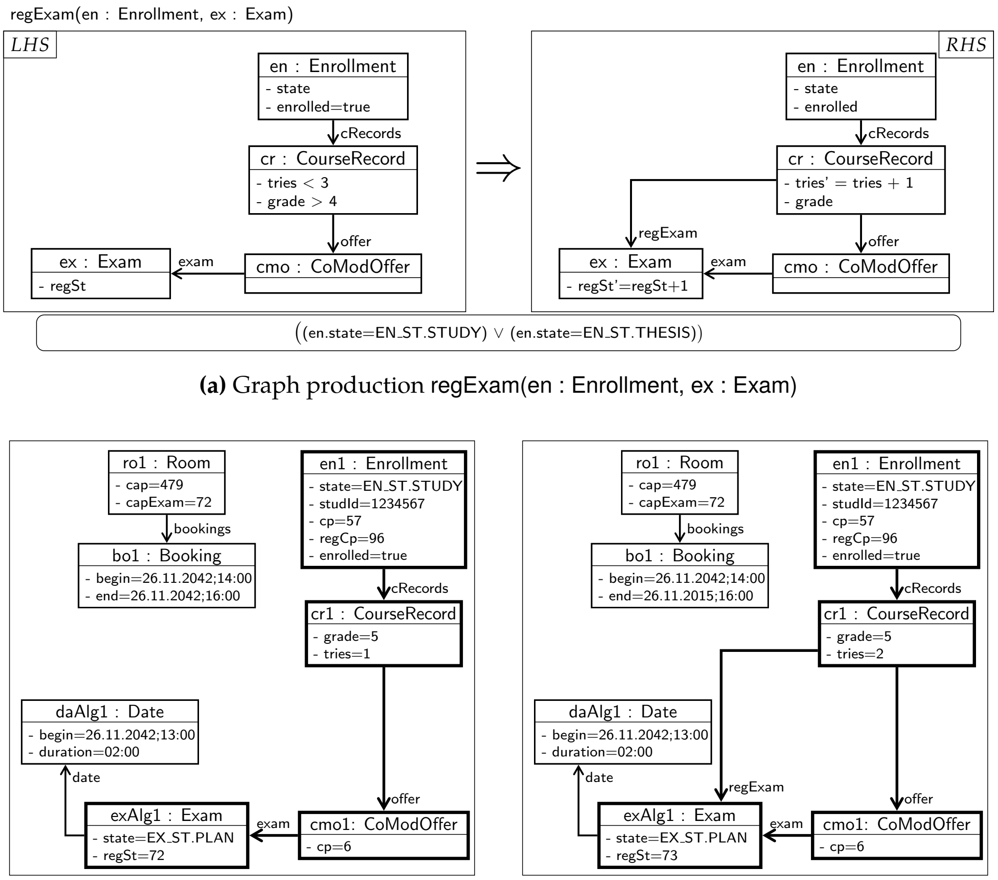
The production is applied by creating a new link of type regExam from the CourseRecord cr to the Exam ex. Additionally, the number of registered students for the Exam ex and the number of tries stored in the CourseRecord cr is incremented by one. The result of applying production regExam to an instance model (Figure 2.7b) is shown in Figure 2.7c.

In the following, we call the application of a production to an instance model at a specific match a *transformation*. Note that a transformation of an instance model is uniquely defined by a production together with a match.

2.2 Static Analysis and Verification by Graph Transformation

Static analysis and verification is concerned with determining, in advance, whether a process model exhibits certain desirable behaviours. Hence, careful analysis of process models at design time can greatly improve the reliability of such systems [WVvdA$^+$09].

The examples presented in Figure 2.6 and Figure 2.7 uncover basically two problems of the current CMS specification. The first problem addresses the integrity of the data stored in our CMS system. More specifically, as shown in Figure 2.6, after



(a) Graph production regExam(en : Enrollment, ex : Exam)

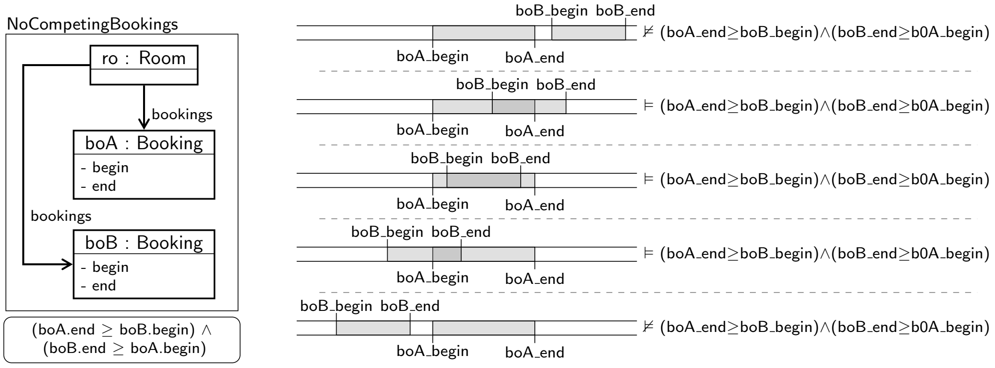
(b) Instance model before applying production regExam

(c) Instance model after applying production regExam

Figure 2.7: The application of production regForExam

applying production bookRoom there are two bookings for the same room with mutually overlapping time slots. As it is highly problematic to conduct, lets say, two exams at the same time in the same room, such a situation must be prevented in any case. In Section 2.2.1 we provide a first impression on how this problem can be solved by graph constraints. Later in Chapter 7 we provide the detailed constructions and proofs for these techniques in the context of attributed graph structures.

The second problem addresses the interaction of tasks. Up until now we only considered the effects of applying a production in isolation. However, in a real system several tasks may operate concurrently on the same set of artifacts. For example, regarding the CMS case study, we expect that dozens of tasks are invoked simultaneously on thousands of artifacts, which potentially leads to unintended effects. In Section 2.2.1 we provide a first impression on how conflict analysis techniques can help to detect potentially problematic interactions. Detailed constructions and proofs for conflict analysis for attributed graph structures are presented in Chapter 8.



(a) Negative constraint NoCompetingBookings

(b) Time line to illustrate NoOverlapInBookings

Figure 2.8: Negative graph constraint NoCompetingBookings, to forbid the existence of competing bookings

2.2.1 *Constraint Enforcement*

In the context of business process modeling constraints usually reflect certain business concerns. These concerns may range from external regulations such as legal requirements and standards up to internal directives on the procedures to guarantee seamless operation.

Graph constraints provide declarative means to place global constraints on the inner structure of a system. Moreover, by constraint enforcement techniques graph constraints can automatically be translated to preconditions over productions, which ensure that a production can only be applied if the resulting graph is consistent with respect to all graph constraints. In the following, we focus mainly on negative graph constraints and negative application conditions.

A negative graph constraint specifies forbidden system states. More specifically, a system state (i. e., an instance model) is consistent with respect to a negative graph constraint if there is no match of the constraint in the instance model. For example, Figure 2.8a shows the negative graph constraint NoCompetingBookings, which forbids the existence of a pair of bookings boA and boB for the same room with mutually overlapping time slots. The meaning of the formula is illustrated by the time lines depicted in Figure 2.8b, which lists all possible combinations of the begin and end dates of boA and boB, respectively. Notice that NoCompetingBookings is a negative constraint; that is, a pair of bookings is only consistent if it invalidates the formula, i. e., there are no bookings boA and boB whose time intervals overlap. Hence, the model shown in Figure 2.6c is inconsistent as the time intervals of bookings bo and bo1 overlap.

In Chapter 7 we show how attributed graph constraints can be enforced by automatically translating them to precondition application conditions. The result is a set of *extended productions* with application conditions that guarantee that each production may only be applied if the result is consistent with respect to the constraints. The result of translating the constraint NoCompetingBookings to an application condition for production bookRoom is the extended production shown

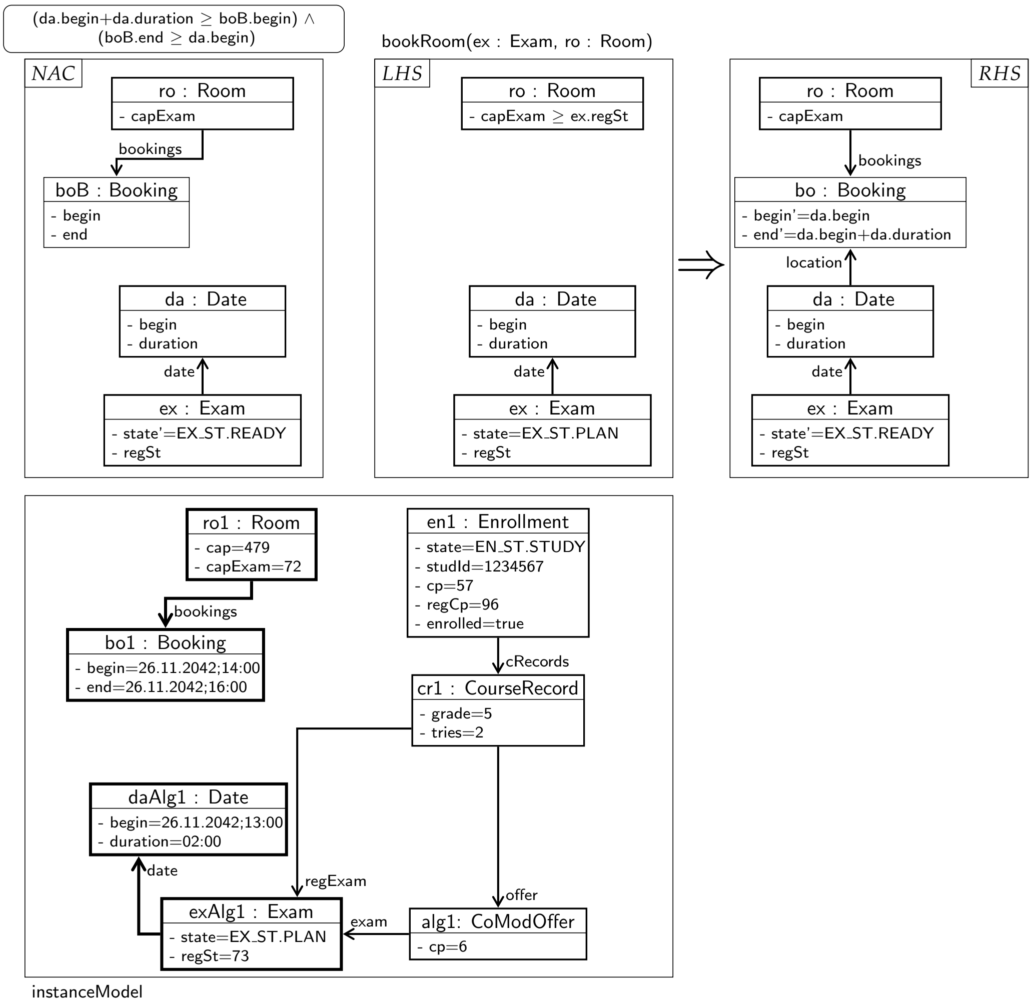


Figure 2.9: Extended production bookRoom

in Figure 2.9. The extended production can only be applied if the match of the LHS in a model cannot be extended to the negative application condition (NAC). More specifically, the production can only be applied if no Booking boB exists for the Room ro such that the time slots of boB overlap with the to be created Booking bo; that is, no Booking boB exists such that (da.begin + da.duration), which is the new value for bo.end', is smaller or equal to boB.begin, and the value of da.begin, which is the new value for bo.begin', is smaller or equal to boB.end.

2.2.2 *Conflict Analysis*

As mentioned before, for our campus management system we expect that dozens of tasks simultaneously operate on thousands of artifacts. In the following, we show how conflict analysis techniques can help to detect potentially problematic interactions of tasks given as graph productions. Basically, two transformations have a *parallel conflict* if one transformation modifies an element that is part of the match of the other. For example, consider the productions regExam and unregExam shown

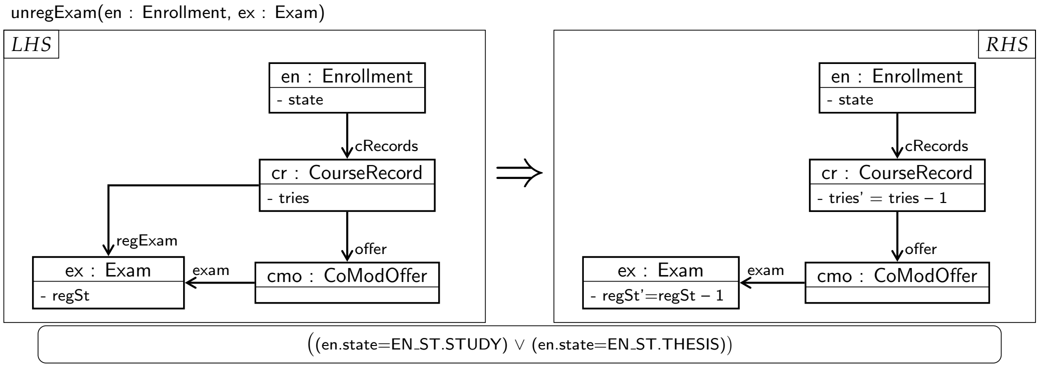


Figure 2.10: Graph production unregExam(en : Enrollment, ex : Exam)

in Figure 2.7a and Figure 2.10, respectively. Production regExam was presented in Section 2.1.3. The production unregExam is basically the inverse of regExam. More specifically, unregExam takes as input an Enrollment en and an Exam ex. By applying the production, the link regExam from cr : CourseRecord to ex : Ex is removed, as well as the number of tries (cr.tries) and the number of registrations (ex.regSt) are decremented by one. The productions have a parallel conflict, as both productions read and modify the regSt attribute of an Exam and the tries attribute of a CourseRecord.

However, in many scenarios one is interested whether the result of two conflicting transformations can be joined again, which leads us to the notion of conflict resolution. *A parallel conflict of two transformations can be resolved* if the output of the first productions can be transformed to the same result. For example, consider the parallel conflict shown on top of Figure 2.11 that arises if productions unregExam and regExam are applied to different enrollments but to the same exam. More specifically, production unregExam(en2, exAlg1) is applied to unregister Enrollment en2 from Exam exAlg1; production regExam(en1, exAlg1) is applied register Enrollment en2 for Exam exAlg1. The transformations have a parallel conflict as applying unregExam and regExam in parallel to the same Exam ex incorporate the parallel modification of attribute ex.regSt (i. e. unregExam decrements and production regExam increments the value of ex.regSt by one. The transformation can be joined by applying the production unregExam and regExam in opposite order to the result of the first productions. Note that, although these transformations are obtained by applying the same productions in opposite order to the results of the first pair of transformation, they constitutes different transformations as the matches were sightly changed in order to match the new attribute values. We shall study this detail at length in Chapter 8. Moreover, we are not forced to use the same productions to join such a pair of diverging transformations; that is, we may all productions that are part of the system specification in order to resolve a parallel conflict.

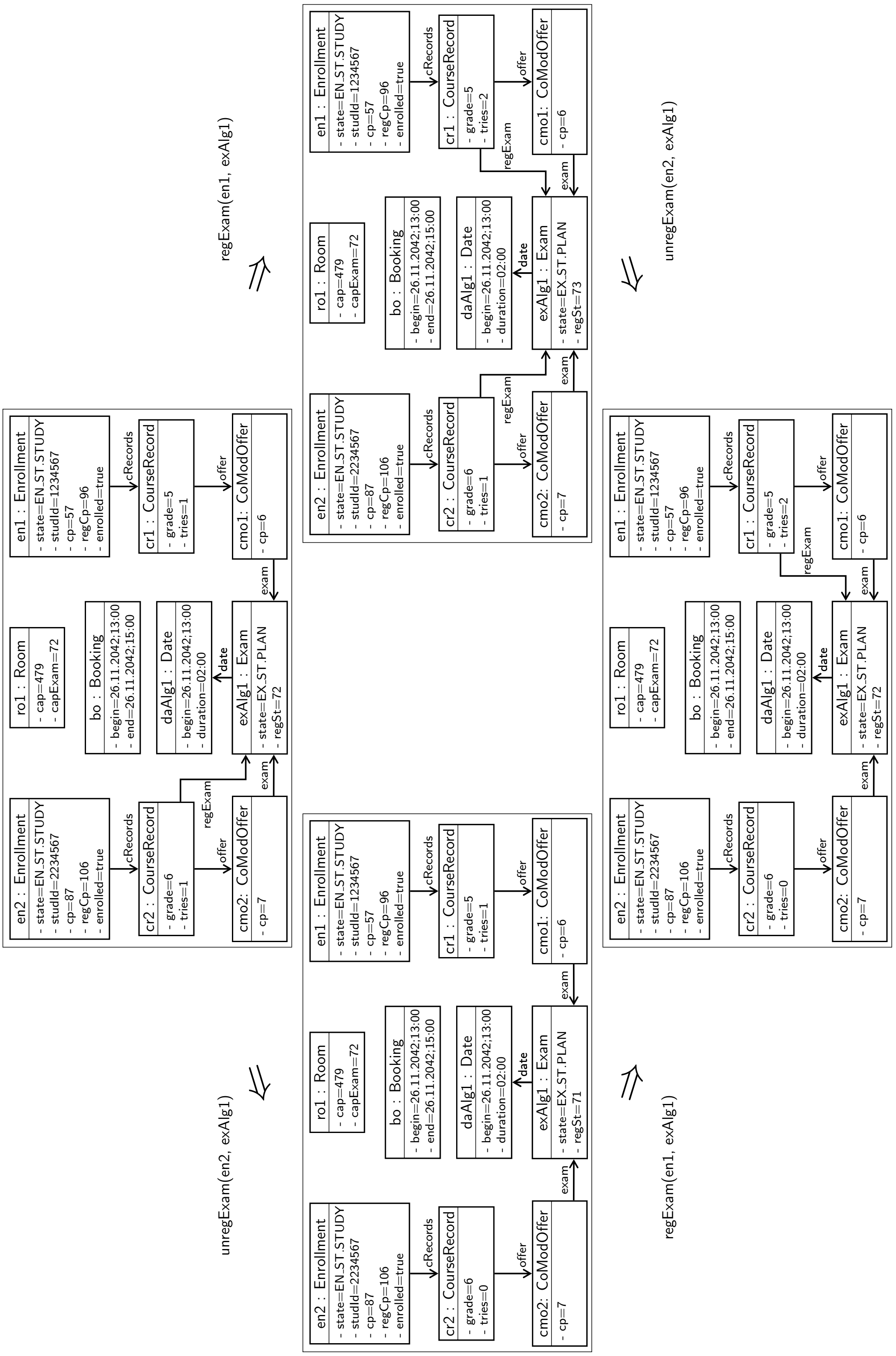


Figure 2.11: An example for a conflict of production unregExam and regExam that can be resolved.

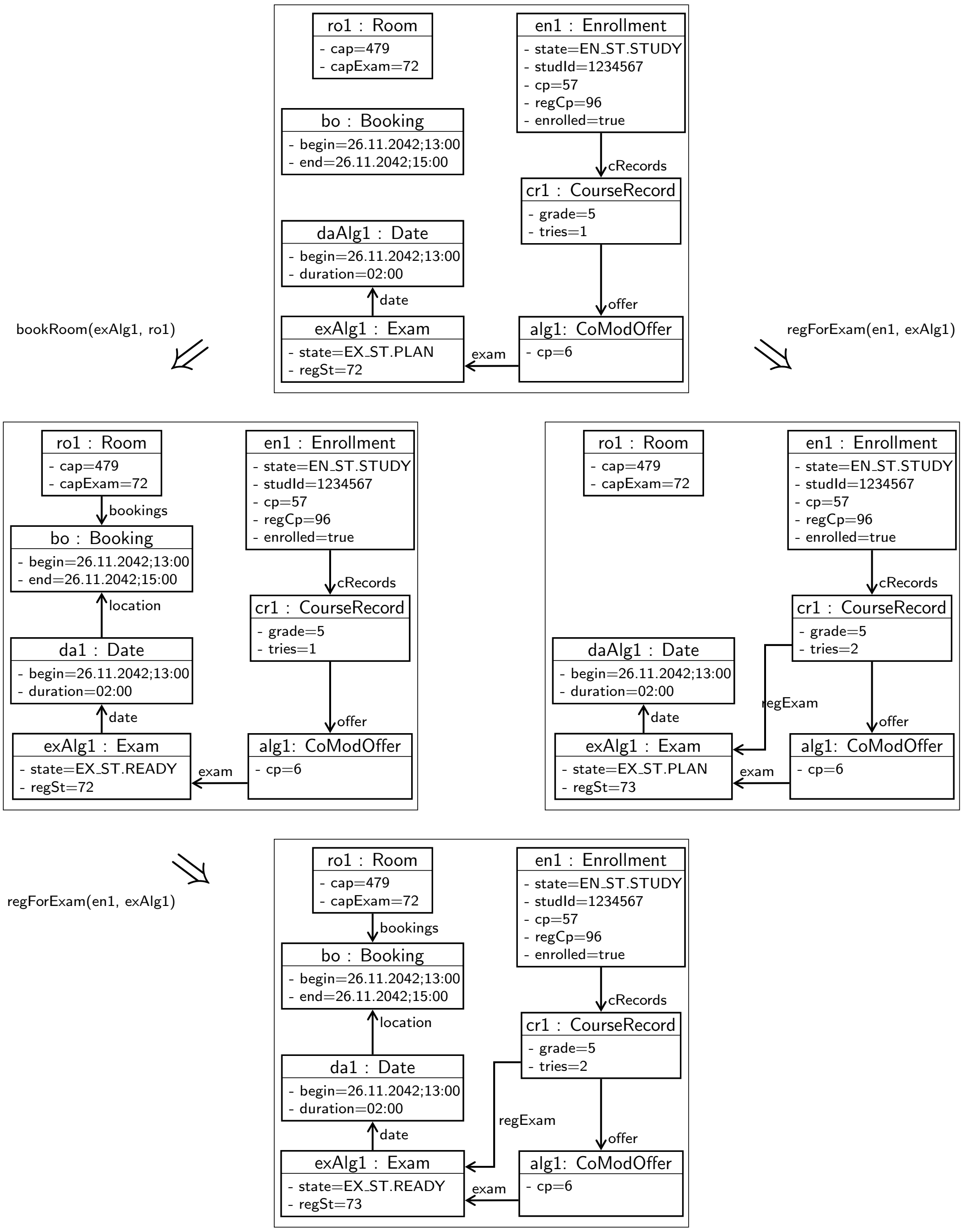


Figure 2.12: An example for a conflict of production bookRoom and regForExam that cannot be resolved.

An example for a parallel conflict that cannot be resolved is shown in Figure 2.12. More specifically, the productions bookRoom regExam have a conflict that cannot be resolved. The problem is that after registering for an exam the production bookRoom cannot be applied for the corresponding Room ro1 as the number of registered students exceeds the exam capacity of Room ro1. However, if we apply the production the other way around it is still possible to register Enrollment en1 for Exam ex, although this exceeds the exam capacity of the booked Room ro.

As we shall see later in Chapter 8 the concept of conflict resolution is closely related to the concept of local confluence known from term rewriting systems; that is, if any parallel conflict can be resolved the system is locally confluent. However, conflict detection and resolution techniques are not only interesting for confluent systems. In several applications conflicts may intended to model nondeterministic behavior of a system, for example, to model different options during a business process. As we shall see in Chapter 9, conflict detection and resolution techniques can also be used to statically analyse potential interactions. If these interaction lead to unintended effects, the system specification has to be adapted accordingly. These adaption can incorporate changing the productions, adding new consistency constraints to exclude conflicts from the consistent system behaviour, or adding new productions to the specification to resolve unintended parallel conflicts.

3

FUNDAMENTALS OF SYMBOLIC GRAPHS AND GRAPH TRANSFORMATIONS

This chapter provides the fundamental concepts for modeling and transforming attributed structures by symbolic graphs and symbolic graph transformations. By combining graphs with first-order formulas, symbolic graphs provide a powerful concept for defining transformations of attributed graph structures.

We start with a brief introduction to category theory in Section 3.1, to subsequently establish the concept of $(\mathcal{M}, \mathcal{N})$-adhesive transformation systems, which serves us as a framework for the definition of transformations. In Section 3.2, we define the syntax and semantics of first-order formulas that, combined with graphs, leads us to the concept of symbolic graphs, presented in Section 3.3. The application of symbolic graph transformations for model transformations is illustrated in Section 3.4 by means of the campus management case study. Based on this illustration, we discuss in Section 3.5 those aspects of symbolic graph transformations that actually prevent its application to achieve all objectives of this thesis. Based on this overview we motivate and detail the key contributions presented in the remainder of this thesis.

3.1 Introduction to Category Theory and Transformation Systems

This section provides a brief introduction to category theory and the categorical framework for high level transformation systems. This section is manly based on the notions provided in [EEPT06]. Hence, we only provide a reference if the content originates from an other source.

3.1.1 *Introduction to Category Theory*

A category can be taught as a system of functions among objects.

Definition 3.1 (Category)**.**
A category $\mathbb{C} = (Ob_{\mathbb{C}}, Mor_{\mathbb{C}}, \circ, id)$ is given by:

- a class $Ob_{\mathbb{C}}$ of objects
- a class $Mor_{\mathbb{C}}$ of morphisms given by the morphism sets $Mor_{\mathbb{C}}(A, B)$ for each pair of objects $A, B \in Ob_{\mathbb{C}}$

- for all objects $A, B, C \in Ob_{\mathbb{C}}$, a composition operation $\circ$, defined as

$$\circ : Mor_{\mathbb{C}}(B,C) \times Mor_{\mathbb{C}}(A,B) \to Mor_{\mathbb{C}}(A,C)$$

- an identity morphism $id_A \in Mor_{\mathbb{C}}(A,A)$ for each object $A \in Ob_{\mathbb{C}}$

such that the following conditions hold:

- *Composition is associative:* For all objects $A, B, C, D \in Ob_{\mathbb{C}}$ and morphisms $f : A \to B$, $g : B \to C$ and $h : C \to D$ we have $(h \circ g) \circ f = h \circ (g \circ f)$.
- *Identity morphisms act as identities w. r. t. composition:* For all objects $A, B \in Ob_{\mathbb{C}}$ and morphisms $f : A \to B$, we have $f \circ id_A = f$ and $id_B \circ f = f$.

Notice that $h : A \to B$ denotes the shorthand notation for $h \in Mor_{\mathbb{C}}(A,B)$; in the following we sometimes write $A \xrightarrow{h} B$ for $h : A \to B$.

For a morphism $h : A \to B$ the objects A and B are called the *domain* and *codomain* of h, respectively.

In general, a category is anything that satisfies the preceding definition; however, for this thesis it is sufficient to consider objects as structured sets, i. e., tuples of sets (called components), endowed with some structure. Morphisms are then structure preserving mappings.

Example 3.2 (The Categories of Sets $\mathbb{Set}$ and graphs $\mathbb{G}$)**.**
The basic example for a category is the category of sets $\mathbb{Set}$, with the class of all sets as objects and with all total functions as morphisms. The composite $(g \circ f)(x)$ of two morphisms $f : A \to B$ and $g : B \to C$ is defined as $g(f(x))$ for all $x \in A$. The identity $id_A : A \to A$ is given by $id_A(x) = x$ for all $x \in A$.

An example for a category of a structured set, is the category $\mathbb{G}$ of graphs. A graph $G = (V_G, E_G, s_G, t_G)$ consists of a set V_G of graph nodes, a set E_G of graph edges, and the source and target functions $s_G, t_G : E_G \to V_G$ mapping the edges to the source and target nodes. A graph morphism is a structure preserving mapping that preserves the source and target functions. More specifically, a graph morphism $f : G \to H$ that maps a graph $G = (V_G, E_G, s_G, t_G)$ to a graph $H = (V_H, E_H, s_H, t_H)$ is a tuple of total functions $f = (f_V, f_E)$, $f_V : V_G \to V_H$, $f_E : E_G \to E_H$ that commutes with source and target functions, i. e., $f_V \circ s_G = s_H \circ f_E$ and $f_V \circ t_G = t_H \circ f_E$. The identity morphism and composition operation can be defined componentwise in $\mathbb{Set}$; that is, the identity morphism $id_G = (id_{V_G}, id_{E_G})$ for a graph G is given for each component separately as the identity in $\mathbb{Set}$. The composition operation for graph morphisms $f : G \to D$ and $g : D \to H$, is defined componentwise by:

$$g \circ f = (g_V \circ f_V, g_E \circ f_E).$$

As f and g are graph morphisms we have

$$f_V \circ s_G = s_D \circ f_E,\ f_V \circ t_G = t_D \circ f_E,\ g_V \circ s_D = s_H \circ g_E,\ g_V \circ t_D = t_H \circ g_E.$$

Hence, we can conclude that $g \circ f$ is also a graph morphism as

$$\begin{aligned} g_V \circ f_V \circ s_G &= g_V \circ s_D \circ f_E = s_H \circ g_E \circ f_E \\ g_V \circ f_V \circ t_G &= g_V \circ t_D \circ f_E = t_H \circ g_E \circ f_E. \end{aligned}$$

Remark 3.3 (Componentwise construction).
In the previous example we have seen that the composition of graph morphisms can be constructed componentwise in the category of sets $\mathfrak{Set}$. This principle of *divide and conquer* is very common in category theory as many categorical constructions can be broken down to componentwise constructions in some *base* category. In the following, we shall see various examples for this principle.

Now we take a closer look on how to classify morphisms of a category according to certain properties. This leads us to the following concept of a morphism class.

Definition 3.4 (Morphism class).
Given a category $\mathbb{C}$ and a property P, a *class of morphisms* is formed from the sets

$$X(A, B) = \{x \in Mor_{\mathbb{C}}(A, B) \mid P(x)\}$$

for all pairs of objects $A, B \in Ob_{\mathbb{C}}$.

In category theory, a property P is usually given as an *abstract characterization*, which defines the role of a morphism in terms of its relations to adjacent objects and morphisms, rather than by internal properties of the morphism itself. In the following, we give an abstract characterization for the classes of monomorphisms, epimorphisms, and isomorphisms.

Definition 3.5 (Monomorphism, epimorphism, and isomorphism).
Given a category $\mathbb{C}$, the class of all *monomorphisms* consists of all morphisms $m : B \to C$, $m \in Mor_{\mathbb{C}}$ that satisfy the following property:
For all morphisms $f, g : A \to B$; $f, g \in Mor_{\mathbb{C}}$, it holds that $m \circ f = m \circ g$ implies $f = g$.

$$\begin{tikzcd} A \arrow[r, bend left, "f"] \arrow[r, bend right, "g"'] & B \arrow[r, "m"] & C \end{tikzcd}$$

The class of all *epimorphisms* consists of all morphisms $e : A \to B$, $e \in Mor_{\mathbb{C}}$ that satisfy the following property:
For all morphisms $f, g : B \to C$; $f, g \in Mor_{\mathbb{C}}$, it holds that $f \circ e = g \circ e$ implies $f = g$.

$$\begin{tikzcd} A \arrow[r, "e"] & B \arrow[r, bend left, "f"] \arrow[r, bend right, "g"'] & C \end{tikzcd}$$

The class of all *isomorphisms* consists of all morphisms $i : A \to B$, $i \in Mor_{\mathbb{C}}$ that satisfy the following property:

There is a morphism $i^{-1} : B \to A$, $i^{-1} \in Mor_{\mathbb{C}}$, such that $i \circ i^{-1} = id_B$ and $i^{-1} \circ i = id_A$.

$$\begin{tikzcd}
A \arrow[r, bend right, "i"'] & B \arrow[l, bend right, "i^{-1}"']
\end{tikzcd}$$

Two objects A and B in $\mathbb{C}$ are *isomorphic* (denoted as $A \simeq B$) iff there exists an isomorphism $i : A \to B$, $i \in Mor_{\mathbb{C}}$.

Example 3.6 (Mono- epi- and isomorphisms in categories $\mathfrak{Set}$ and $\mathbb{G}$)**.**
Monomorphisms, epimorphisms, and isomorphisms correspond to injective, surjective and bijective functions in the category $\mathfrak{Set}$, respectively.

In the category $\mathbb{G}$, a morphisms $f = (f_V, f_E)$ is a monomorphism, epimorphism, or isomorphism if and only if f is componentwise injective, surjective, or bijective, respectively.

We have seen how to use properties to abstractly characterize different morphism classes. In the following, we shall see how to characterize more complex categorical concepts in a similar way.

We begin with the definition of *pushouts*, which can be considered as the generalisation of the set theoretic union.

Definition 3.7 (Pushout (PO))**.**
Given morphisms $f : A \to B$ and $g : A \to C$ in a category $\mathbb{C}$, the triple (f', g', D) consisting of:

- a pushout object D
- morphisms $f' : C \to D$ and $g' : B \to D$ such that $f' \circ g = g' \circ f$

is a *pushout* over f and g in category $\mathbb{C}$ iff the following universal property holds: For all objects X and morphisms $h : B \to X$ and $k : C \to X$ with $k \circ g = h \circ f$, there is a unique morphism $x : D \to X$ such that $x \circ g' = h$ and $x \circ f' = k$.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] \arrow[rd, phantom, "="] & B \arrow[d, "g'"] \arrow[rdd, bend left, "h"] \arrow[rdd, phantom, "="] & \\
C \arrow[r, "f'"'] \arrow[rrd, bend right, "k"'] & D \arrow[rd, "x"] \arrow[d, phantom, "="] & \\
 & {} & X
\end{tikzcd}$$

Example 3.8 (Pushout in $\mathbb{S}\mathrm{et}$ and $\mathbb{G}$).
In $\mathbb{S}\mathrm{et}$ the pushout $(f' : C \to D, g' : B \to D, D)$ over the morphisms $f : A \to B$ and $g : A \to C$ can be constructed as follows:

The pushout object D is given as the quotient $B \dot{\cup} C|_{\equiv}$, where $\dot{\cup}$ denotes the disjoint union, and $\equiv$ the smallest equivalence relation generated from the relation $\sim$, where $\sim$ is given by $f(a) \sim g(a)$ for all $a \in A$. Here, *smallest equivalence relation generated from* $\sim$ means the reflexive, symmetric, and transitive closure of $\sim$. Let $[x] = \{y \in B \dot{\cup} C \mid x \equiv y\}$, then the morphisms f' and g' are given as $f'(c) = [c]$ and $g'(b) = [b]$ for all $c \in C$ and $b \in B$.

The pushout in the category of graphs $\mathbb{G}$ is defined componentwise by the pushouts on node and edge components in the category $\mathbb{S}\mathrm{et}$, respectively. The source and target functions are uniquely determined by the universal property of the pushout for the edge component. For example, the diagram below shows the construction of the source function s_D for the pushout object D. As shown the source function $s_D : E_D \to V_D$ is uniquely given by the universal property of the pushout for the edge component PO_E, with morphisms $g_V \circ s_B$ and $f_V \circ s_C$, where $s_B : E_B \to V_B$ and $s_C : E_C \to V_C$ are the source functions of graph B and C, respectively.

$$\begin{tikzcd}
E_A \arrow[r, "f_E"] \arrow[dd, "g'_E"'] \arrow[dr, phantom, "(PO_E)"] & E_B \arrow[dd, "g_E"] \arrow[ddr, bend left, "g_V \circ s_B"] & & \\
& & V_B \arrow[d, "g_V"] \arrow[dr, phantom, "(PO_V)"] & V_A \arrow[l, "f_V"'] \arrow[d, "g'_V"] \\
E_C \arrow[r, "f'_E"] \arrow[rr, bend right, "f_V \circ s_C"'] & E_D \arrow[r, "s_D"] & V_D & V_C \arrow[l, "f_V"]
\end{tikzcd}$$

The target function $t_D : E_D \to V_D$ can be constructed similarly.

In the categories $\mathbb{S}\mathrm{et}$ and $\mathbb{G}$, the construction of pushout objects can be simplified if one of the morphisms is a monomorphism.

Fact 3.9 (Pushouts in $\mathbb{S}\mathrm{et}$ and $\mathbb{G}$ along monomorphisms).
Let (f', g', D) be the pushout over morphisms $f : A \to B$ and $g : A \to C$ in category $\mathbb{S}\mathrm{et}$. If f is a monomorphism, then the following properties hold:

a) Morphism f' is a monomorphism, too.

b) The pushout object D is isomorphic to $D' = C \dot{\cup} (B \backslash f(A))$, where the disjoint union $\dot{\cup}$ is used to ensure that the elements of $(B \backslash f(A))$ are added as new elements to D'.

The properties (a) and (b) hold componentwise in $\mathbb{G}$.

The *pushout complement*, defined next, is a generalization of the set theoretic difference operation.

Definition 3.10 (Pushout complement)**.**
Let $f : A \to B$ and $g : B \to D$ be morphisms in a category $\mathbb{C}$, the triple (f', g', C), consisting of morphisms $f' : C \to D$, $g' : A \to C$ and object C, is a *pushout complement* in $\mathbb{C}$ if and only if (1) is a pushout in $\mathbb{C}$.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g'"] \arrow[dr, phantom, "(1)"] & B \arrow[d, "g"] \\
C \arrow[r, "f'"] & D
\end{tikzcd}$$

The *pullback* is the categorical dual of the pushout. Hence, it is a generalization of the set theoretic intersection.

Definition 3.11 (Pullback (PB) [EEHP06])**.**
Given morphisms $f : C \to D$ and $g : B \to D$ in a category $\mathbb{C}$, the triple (f', g', A) consisting of:

- a pullback object A
- morphisms $f' : A \to B$ and $g' : A \to C$ such that $g \circ f' = f \circ g'$

is a *pullback* over f and g in category $\mathbb{C}$, if and only if the following universal property holds: For all objects X and morphisms $h : X \to B$ and $k : X \to C$ with $f \circ k = g \circ h$, there is a unique morphism $x : X \to A$ such that $f' \circ x = h$ and $g' \circ x = k$.

$$\begin{tikzcd}
X \arrow[drr, bend left, "h"] \arrow[dr, "x"] \arrow[ddr, bend right, "k"'] & & \\
& A \arrow[r, "f'"] \arrow[d, "g'"] \arrow[dr, phantom, "="] & B \arrow[d, "g"] \\
& C \arrow[r, "f"] & D
\end{tikzcd}$$

Example 3.12 (Pullback in $\mathfrak{Set}$ and $\mathbb{G}$)**.**
In $\mathfrak{Set}$, the pullback (f', g', A) over morphisms $f : C \to D$ and $g : B \to D$ can be constructed as

$$A = \bigcup_{d \in D} f^{-1}(d) \times g^{-1}(d) = \{(c, b) | f(c) = g(b)\} \subseteq C \times B$$

with $f' : A \to B$ and $g' : A \to C$ given by $f'(c, b) = b$ and $g(c, b) = c$.

The pullback in the category of graphs $\mathbb{G}$ can be constructed componentwise for node and edge components in $\mathfrak{Set}$. The source and target functions are uniquely determined by the universal property of the pullback for the node components.

In the categories $\mathbb{S}et$ and $\mathbb{G}$ we have the following property for pullbacks along monomorphisms.

Fact 3.13 (Pullbacks in $\mathbb{S}et$ and $\mathbb{G}$ along monomorphisms)**.**
Let (f', g', A) be the pullback over morphisms $f : C \to D$ and $g : B \to D$ in category $\mathbb{S}et$ then the following properties hold:

a) If f is a monomorphism, then also f'.

b) If f and g are monomorphisms then A can be constructed as the intersection of B and C, i. e., $A = B \cap C$.

The properties (a) and (b) hold componentwise in $\mathbb{G}$.

The following properties are valid in any category that has pushouts and pullbacks.

Fact 3.14 (PO and PB properties)**.**
For any category $\mathbb{C}$ that has pushouts and pullbacks the following properties hold:

a) The pushout and pullback objects are unique up to isomorphism.

b) The composition and decomposition of pushouts (pullbacks) results again in a pushout (pullback):

composition:

- If (1) and (2) are pushouts, then (1) + (2) is a pushout.
- If (1) and (2) are pullbacks, then (1) + (2) is a pullback.

decomposition:

- If (1) and (1) + (2) are pushouts , then (2) is a pushout.
- If (2) and (1) + (2) are pullbacks, then (1) is a pullback.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & B \arrow[r, "e"] \arrow[d, "g'"] \arrow[dr, phantom, "(2)"] & E \arrow[d, "g''"] \\
C \arrow[r, "f'"'] & D \arrow[r, "e'"'] & F
\end{tikzcd}$$

c) For any morphism $f : A \to B$, the diagram (3) below is a pushout and a pullback; for any monomorphism $m : A \to B$ diagram (4) is a pullback.

$$\begin{tikzcd}
A \arrow[r, "f" description] \arrow[d, "id_A" description] \arrow[dr, phantom, "(3)"] & B \arrow[d, "id_B" description] \\
A \arrow[r, "f" description] & B
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "id_A" description] \arrow[d, "id_A" description] \arrow[dr, phantom, "(4)"] & A \arrow[d, "m" description] \\
A \arrow[r, "m" description] & B
\end{tikzcd}$$

The following concept of *jointly epimorphic pairs of morphisms* is the generalization of epimorphisms from single morphisms to pairs of morphism.

Definition 3.15 (Jointly epimorphic)**.**
A morphism pair (e_1, e_2) of morphisms $e_1 : A_1 \to B$ and $e_2 : A_2 \to B$ with the same codomain is jointly epimorphic if for all $g, h : B \to C$ such that $g \circ e_i = h \circ e_i$, for $i = 1, 2$, we have $g = h$.

$$\begin{tikzcd}
A_1 \arrow[dr, "e_1" description] & & \\
& B \arrow[r, bend left, "h" description] \arrow[r, bend right, "g" description] & C \\
A_2 \arrow[ur, "e_2" description] & &
\end{tikzcd}$$

In the following, we consider the slice construction for categories. The slice construction can be used to obtain categories for typed structures from an (untyped) base category. This is particularly interesting, as the slice construction preserves many properties of the base category (e. g., pushouts, pullbacks, and binary coproducts). Hence, in order to show that a category, obtained by slice construction from a base category, has a property of interest it is sufficient to show that the base category has this property.

Definition 3.16 (Slice category)**.**
Let $\mathbb{C}$ be a category and X any object of $\mathbb{C}$, the slice category $\mathbb{C}\backslash X$ is defined as follows:

- An object of $\mathbb{C}\backslash X$ is a morphism $(f : A \to X)$ from an object $A \in Ob_{\mathbb{C}}$ to X.
- A morphism $m : (f : A \to X) \to (g : B \to X) \in Mor_{\mathbb{C}\backslash X}$ from an object $(f : A \to X)$ to an object $(g : B \to X)$, is given by morphism $m : A \to B \in Mor_{\mathbb{C}}$ such that $f = g \circ m$.
- The composition of morphisms $m : (f : A \to X) \to (g : B \to X) \in Mor_{\mathbb{C}\backslash X}$ and $n : (g : B \to X) \to (h : C \to X) \in Mor_{\mathbb{C}\backslash X}$ is given by $n \circ m \in Mor_{\mathbb{C}}$.

The following fact lists properties that are preserved by slice construction.

Fact 3.17 (Slice category properties)**.**
The following properties hold for every slice category:

a) If the category $\mathbb{C}$ has pushouts, the pushouts in the slice category $\mathbb{C}\backslash X$ can be constructed over the pushouts in $\mathbb{C}$.

b) If the category $\mathbb{C}$ has pullbacks, the pullbacks in the slice category $\mathbb{C}\backslash X$ can be constructed over the pullbacks in $\mathbb{C}$.

c) If the category $\mathbb{C}$ has binary coproducts, the binary coproducts in the slice category $\mathbb{C}\backslash X$ can be constructed over the binary coproducts in $\mathbb{C}$.

3.1.2 *$(\mathcal{M},\mathcal{N})$-Adhesive Categories and Transformation Systems*

This section introduces the basics for transformations based on the double pushout approach. The double pushout approach was originally defined for directed labeled graphs [EPS73] and later generalized to other high-level structures, such as Petri nets or algebraic specifications, by introducing the categorical framework of high level replacement system (HLR systems) [EHKP90]. Basically, a HLR system is a category with a distinguished class $\mathcal{M}$ of morphisms that fulfils certain properties (called HLR properties). The HLR properties were originally provided as a list, consisting of all properties that were used to prove the fundamental results for the double pushout approach. In [LS04], it was shown that most of the HLR properties are consequences of a more general principle, leading to the notion of adhesive categories. An adhesive category is any category that provides a certain compatibility of pushouts and pullbacks, known as the *van Kampen property*. Over the years, it was shown that various weaker versions of the van Kampen property are still sufficient to provide the HLR properties, resulting in the notions of weak adhesive and $\mathcal{M}$-adhesive categories [EHPP04, EGH10]. A detailed overview on the various versions of the van Kampen property and their relations can be found in [EGH10]. In this thesis we use $(\mathcal{M},\mathcal{N})$-adhesive categories introduced by Habel and Plump in [HP12a].

Definition 3.18 ($(\mathcal{M},\mathcal{N})$-adhesive category [HP12a])**.**
A category $\mathbb{C}$ with morphism classes $\mathcal{M}$ and $\mathcal{N}$ is an *$(\mathcal{M},\mathcal{N})$-adhesive category* $(\mathbb{C},\mathcal{M},\mathcal{N})$ if the following properties hold:

a) $\mathcal{M}$ and $\mathcal{N}$ contain all isomorphisms:

- f being an isomorphism implies $f \in \mathcal{M}$ and $f \in \mathcal{N}$

$\mathcal{M}$ and $\mathcal{N}$ are closed under composition and decomposition: given morphisms $f : A \to B$ and $g : B \to C$ in $\mathbb{C}$ then

- $f, g \in \mathcal{X}$ implies $g \circ f \in \mathcal{X}$, for any $\mathcal{X} \in \{\mathcal{M},\mathcal{N}\}$
- $g \circ f \in \mathcal{X}$ and $g \in \mathcal{X}$ implies $f \in \mathcal{X}$, for any $\mathcal{X} \in \{\mathcal{M},\mathcal{N}\}$

$\mathcal{N}$ is closed under $\mathcal{M}$-decomposition: given morphisms $f : A \to B$ and $g : B \to C$ in $\mathbb{C}$ then

- $g \circ f \in \mathcal{N}$ and $g \in \mathcal{M}$ implies $f \in \mathcal{N}$

b) $\mathbb{C}$ has pushouts along $(\mathcal{M},\mathcal{N})$-morphisms and pullbacks along $\mathcal{M}$-morphisms:

- A pushout along $(\mathcal{M},\mathcal{N})$-morphisms, or $(\mathcal{M},\mathcal{N})$-pushout, is a pushout where one of the given morphisms is in $\mathcal{M}$ and the other morphism is in $\mathcal{N}$.
- A pullback along an $\mathcal{M}$-morphism, or $\mathcal{M}$-pullback, is a pullback where at least one of the given morphisms is in $\mathcal{M}$.

$\mathcal{M}$ and $\mathcal{N}$ are closed under pushouts and pullbacks:

- Given pushout (1), then $f \in \mathcal{X}$ implies $g \in \mathcal{X}$, for any $\mathcal{X} \in \{\mathcal{M}, \mathcal{N}\}$.
- Given pullback (1), then $g \in \mathcal{X}$ implies $f \in \mathcal{X}$, for any $\mathcal{X} \in \{\mathcal{M}, \mathcal{N}\}$.

c) Pushouts in $\mathbb{C}$ along $(\mathcal{M}, \mathcal{N})$-morphisms are $(\mathcal{M}, \mathcal{N})$-VK squares. A pushout (1) with $m \in \mathcal{M}$ and $f \in \mathcal{N}$ is a $(\mathcal{M}, \mathcal{N})$-VK square if for any commutative cube (2) with pushout (1) in the bottom, with pullbacks as back faces, and $b, c, d \in \mathcal{M}$, the following statement holds: the top face is a pushout if and only if the front faces are pullbacks.

$$\begin{tikzcd}
A \arrow[r, "m"] \arrow[d, "f"'] \arrow[dr, phantom, "(1)"] & B \arrow[d, "g"] \\
C \arrow[r, "n"] & D
\end{tikzcd}$$

$$\begin{tikzcd}
& A' \arrow[rr, "m'"] \arrow[dl, "f'"'] \arrow[dd, "a" near start] & & B' \arrow[dl, "g'"'] \arrow[dd, "b"] & \\
C' \arrow[rr, "n'"] \arrow[dd, "c"] & & D' \arrow[dd, "d" near start] & & \\
& A \arrow[rr, "m"] \arrow[dl, "f"'] & & B \arrow[dl, "g"] & (2) \\
C \arrow[rr, "n"] & & D & &
\end{tikzcd}$$

Remark 3.19 (Hierarchy of adhesive categories)**.**
As mentioned in the introduction of this section, there are various versions of adhesive categories. Hence, many results where obtained for adhesive categories other than $(\mathcal{M}, \mathcal{N})$-adhesive categories. In order to transfer these results to $(\mathcal{M}, \mathcal{N})$-adhesive categories, it is important to know their relations. To this end, we briefly review $\mathcal{M}$-*adhesive* and *adhesive HLR* categories and show how they relate to $(\mathcal{M}, \mathcal{N})$-adhesive categories.

In contrast to $(\mathcal{M}, \mathcal{N})$-adhesive categories, $\mathcal{M}$-adhesive categories presume only a single morphism class $\mathcal{M}$. An $\mathcal{M}$-adhesive category $(\mathbb{C}, \mathcal{M})$ can be defined in terms of an $(\mathcal{M}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{M}, \mathcal{N})$ by choosing $\mathcal{N} = Mor_{\mathbb{C}}$, i. e. $\mathcal{N}$ comprises all morphisms in $\mathbb{C}$. Adhesive HLR categories are similar to $\mathcal{M}$-adhesive categories with the difference that the VK square property must hold for $m \in \mathcal{M}$ only, instead of $(m, b, c, d \in \mathcal{M})$. Consequently any adhesive HLR category is also an $\mathcal{M}$-adhesive category.

To obtain an $(\mathcal{M}, \mathcal{N})$-adhesive category from an adhesive HLR category (or $\mathcal{M}$-adhesive category), for choices other than $\mathcal{N} = Mor_{\mathbb{C}}$, it must be shown that narrowing the class $\mathcal{N}$ does not destroy the closure properties. Hence, instead of verifying all properties stated in Definition 3.18, it is sufficient to show that:

a) $\mathcal{N}$ contains all isomorphisms.

b) $\mathcal{N}$ is closed under composition and decomposition.

c) $\mathcal{N}$ is closed under $\mathcal{M}$-decomposition.

d) $\mathcal{N}$ is closed under pushouts and pullbacks along $\mathcal{M}$-morphisms.

Moreover, given an $\mathcal{M}$-adhesive category $(\mathbb{C}, \mathcal{M})$ the previous properties are trivially fulfilled for category $(\mathbb{C}, \mathcal{M}, \mathcal{N})$ with $\mathcal{N} = \mathcal{M}$.

An $(\mathcal{M},\mathcal{N})$-adhesive category provides the following HLR properties.

Fact 3.20 (HLR properties of $(\mathcal{M},\mathcal{N})$-adhesive categories [HP12a])**.**
For any $(\mathcal{M},\mathcal{N})$-adhesive category $(\mathbb{C},\mathcal{M},\mathcal{N})$ the following properties hold:

a) *Pushouts along $(\mathcal{M},\mathcal{N})$-morphisms are pullbacks:* Given the $(\mathcal{M},\mathcal{N})$-pushout (1), then (1) is also a pullback.

b) *The $\mathcal{M}$–$\mathcal{M}$-pushout–pullback decomposition:* If (1)+(2) is an $(\mathcal{M},\mathcal{N})$-pushout with $l \in \mathcal{M}$ and $r \circ k \in \mathcal{N}$, and (2) a pullback with $w \in \mathcal{M}$, then (1) and (2) are pushouts as well as pullbacks.

c) *The cube $(\mathcal{M},\mathcal{N})$-pushout–pullback decomposition:* Given the commutative cube (3), where all morphisms in the top square and bottom square are in $\mathcal{M}$, all vertical morphisms are in $\mathcal{N}$, the top face is a pullback and the front faces are pushouts, then the following statement holds: the bottom face is a pullback if and only if the back faces are pushouts.

d) *Pushout complements along $(\mathcal{M},\mathcal{N})$-morphisms are unique:* Given morphisms $l : A \to B$ and $u : C \to D$, where $l \in \mathcal{M}$ and $u \in \mathcal{N}$, then there is at most one B (up to isomorphism) and morphisms $k : A \to B$ and $s : B \to D$, such that (1) is a pushout.

$$\begin{tikzcd}
A \arrow[r, "l"] \arrow[d, "k"] \arrow[dr, phantom, "(1)"] & C \arrow[d, "u"] \\
B \arrow[r, "s"] \arrow[d, "r"] \arrow[dr, phantom, "(2)"] & D \arrow[d, "w"] \\
E \arrow[r, "v"] & F
\end{tikzcd}$$

$$\begin{tikzcd}
 & & A' \arrow[dll, "f'"'] \arrow[dr, "m'"] \arrow[dd, "a"] & \\
C' \arrow[dr, "n'"] \arrow[dd, "c"] & & & B' \arrow[dll, "g'"'] \arrow[dd, "b"] \\
 & D' \arrow[dd, "d"] & A \arrow[dll, "f"'] \arrow[dr, "m"] & \\
C \arrow[dr, "n"] & & & B \arrow[dll, "g"'] \\
 & D & & (3)
\end{tikzcd}$$

Note that the $\mathcal{M}$–$\mathcal{M}$-pushout–pullback decomposition was originally refereed to as the the $\mathcal{M}$–$\mathcal{N}$-pushout–pullback decomposition property in [HP12a].

Another important result shows that $(\mathcal{M},\mathcal{N})$-adhesive categories are stable under slice constructions. As mentioned in the previous section the slice construction can be used to construct categories for typed structures from an untyped base category. Hence, to show that a category obtained by slice construction from a base category is $(\mathcal{M},\mathcal{N})$-adhesive, it is sufficient to show that the base category is $(\mathcal{M},\mathcal{N})$-adhesive.

Fact 3.21 (Slice construction of $(\mathcal{M},\mathcal{N})$-adhesive categories [PH15])**.**
If $(\mathbb{C},\mathcal{M},\mathcal{N})$ is an $(\mathcal{M},\mathcal{N})$-adhesive category, then for every object X in $\mathbb{C}$ the slice category $(\mathbb{C}\backslash X, \mathcal{M} \cap Mor_{\mathbb{C}\backslash X}, \mathcal{N} \cap Mor_{\mathbb{C}\backslash X})$ is also $(\mathcal{M},\mathcal{N})$-adhesive adhesive.

Now we define productions and transformations.

Definition 3.22 (Production).
Given an $(\mathcal{M}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{M}, \mathcal{N})$, a *production* $p = (L \leftarrow K \rightarrow R)$ consists of the objects L, K, and R (called the left-hand side, the interface, and the right-hand side, respectively) as well as *left production* morphism $l : K \rightarrow L$, $l \in \mathcal{M}$ and *right production* morphism $r : K \rightarrow R$, $r \in \mathcal{M}$.

Definition 3.23 (Transformation).
Given an $(\mathcal{M}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{M}, \mathcal{N})$, a *direct transformation* $G \overset{p@m}{\Longrightarrow} H$ via a production $p = (L \leftarrow K \rightarrow R)$ and a *match* $m : L \rightarrow G$, $m \in \mathcal{N}$ is given by the following double pushout diagram, where (1) and (2) are pushouts:

$$\begin{tikzcd}
L \arrow[d, "m"] \arrow[rd, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[rd, phantom, "(2)"] & R \arrow[d, "n"] \\
G & D \arrow[l, "g"] \arrow[r, "h"'] & H
\end{tikzcd}$$

The morphism $n : R \rightarrow H$ is called *comatch*.

In the following, we call the class $\mathcal{M}$ *production morphisms* and the class $\mathcal{N}$ *match morphisms*.

Finally we define $(\mathcal{M}, \mathcal{N})$-adhesive transformation systems.

Definition 3.24 ($(\mathcal{M}, \mathcal{N})$-adhesive transformation systems).
An *$(\mathcal{M}, \mathcal{N})$-adhesive transformation system* $((\mathbb{C}, \mathcal{M}, \mathcal{N}), \mathbf{P})$ is composed of an $(\mathcal{M}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{M}, \mathcal{N})$ and a set of productions $\mathbf{P}$.

Let $((\mathbb{C}, \mathcal{M}, \mathcal{N}), \mathbf{P})$ be an $(\mathcal{M}, \mathcal{N})$-adhesive transformation system. Given objects G and H in $\mathbb{C}$, such that there is a *direct transformation* $G \overset{p@m}{\Longrightarrow} H$ via $p \in \mathbf{P}$ we write $G \Longrightarrow H$. A *transformation* from G to H is a sequence of direct transformations $G \simeq G_0 \Longrightarrow \ldots \Longrightarrow G_n \simeq H$ for some $n \geq 0$, and is denoted as $G \overset{*}{\Longrightarrow} H$. For $n = 0$ we have the identical transformations $G \overset{id}{\Longrightarrow} H$ and $G \simeq H$. For $n \in \{0, 1\}$ we write $G \overset{0..1}{\Longrightarrow} H$.

Remark 3.25 (Applicability of productions and construction of transformations).
Let $((\mathbb{C}, \mathcal{M}, \mathcal{N}), \mathbf{P})$ be an $(\mathcal{M}, \mathcal{N})$-adhesive transformation system with a production $p = (L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$. Given a *match* $m : L \rightarrow G$, $m \in \mathcal{N}$, then *production* p *is applicable via match* m if pushout (1) can be constructed as pushout complement of m and l.

If p is applicable to G via match m, then p is applied to G at match m by first constructing D as the pushout complement of l and m and H by the pushout of k and r leading to the direct transformation $G \overset{p@m}{\Longrightarrow} H$ via p and m.

Notice that the context object D is unique (up to isomorphism) if it exists, as pushouts complements are unique in $(\mathcal{M}, \mathcal{N})$-adhesive categories.

In the following, we say that a *production* p *is applicable to an object* G if there exists a match such that p is applicable to G via match m.

Now, we show a criterion to decide whether a production is applicable or not, i. e., a criterion for the existence of pushout complements. To this end we need to define the concept of initial pushouts.

Definition 3.26 (($\mathcal{M},\mathcal{N}$)-initial pushout)**.**
Let $(\mathbb{C},\mathcal{M},\mathcal{N})$ be an $(\mathcal{M},\mathcal{N})$-adhesive category and given an $\mathcal{N}$-morphism $f : A \to F$. An $(\mathcal{M},\mathcal{N})$-pushout (1) with $b \in \mathcal{M}$ is an *$(\mathcal{M},\mathcal{N})$-initial pushout over f* if for every $(\mathcal{M},\mathcal{N})$-pushout (2) with $b' \in \mathcal{M}$ there exist unique morphisms $b^* : B \to D$ and $c^* : C \to E$ with $b^*, c^* \in \mathcal{M}$ such that $b' \circ b^* = b$, $c' \circ c^* = c$ and (3) is a pushout. Morphisms b and c are called the *boundary and context with respect to f*.

$$\begin{tikzcd}
B \arrow[rr, "b^*"] \arrow[dr, "b"] \arrow[dd] & & D \arrow[dl, "b'"'] \arrow[dd] \\
(1) & A \arrow[dd, "f"] \arrow[d, phantom, "(3)"] & (2) \\
C \arrow[rr, "c^*"] \arrow[dr, "c"'] & {} & E \arrow[dl, "c'"] \\
& F &
\end{tikzcd}$$

By means of initial pushouts we can define the following condition, which is sufficient and necessary for the existence of a pushout complement.

Fact 3.27 (Gluing condition)**.**
Let $((\mathbb{C},\mathcal{M},\mathcal{N}),\mathbf{P})$ be an $(\mathcal{M},\mathcal{N})$-adhesive transformation system and $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ a production in $\mathbf{P}$, given a *match* $m : L \to G$, $m \in \mathcal{N}$, then *production p is applicable to G via match m* (i. e., the pushout complement of m and l exists) if and only if there is a morphism $b^* : B \to K$ such that $l \circ b^* = b$ for $(\mathcal{M},\mathcal{N})$-initial pushout (1).

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[rr, bend left, "b^*"] \arrow[d] \arrow[dr, phantom, "(1)"] & L \arrow[d, "m"] & K \arrow[l, "l"'] \arrow[r, "r"] & R \\
C \arrow[r, "c"] & G & &
\end{tikzcd}$$

In addition to the gluing condition initial pushouts have further properties, which are required to prove the Local Confluence Theorem in Chapter 8.

Fact 3.28 (Closure property of ($\mathcal{M},\mathcal{N}$)-initial pushouts)**.**
Given an $(\mathcal{M},\mathcal{N})$-adhesive category $(\mathbb{C},\mathcal{M},\mathcal{N})$ with $(\mathcal{M},\mathcal{N})$-initial pushouts, then $(\mathcal{M},\mathcal{N})$-initial pushouts are closed under double pushouts along $\mathcal{M}$-morphisms; that is, given an $(\mathcal{M},\mathcal{N})$-initial pushout (1) over $f_0 \in \mathcal{N}$ and double pushout diagram (2) with pushout $(2a)$ and $(2b)$ and $b'_0, b'_1 \in \mathcal{M}$, the following holds:

a) The composition of (1) with $(2a)$ (i. e., pushout (3)) is again an initial pushout over d, where pushout (3) is derived from (1) and $(2a)$ using the initial pushout property of (1) (see Definition 3.26).

b) The composition of initial pushout (3) with pushout $(2b)$, leading to pushout (4), is an initial pushout over f_1.

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[d] \arrow[dr, phantom, "(1)"] & A_0 \arrow[d, "f_0"] & A_0 \arrow[d, "f_0"] \arrow[dr, phantom, "(2a)"] & D \arrow[l, "b'_0"'] \arrow[r, "b'_1"] \arrow[d, "d"] \arrow[dr, phantom, "(2b)"] & A_1 \arrow[d, "f_1"] & (2) \\
C \arrow[r, "c"] & F_0 & F_0 & E \arrow[l, "c'_0"] \arrow[r, "c'_1"] & F_1 &
\end{tikzcd}$$

$$\begin{tikzcd}
B \arrow[r, "b^*"] \arrow[d] \arrow[dr, phantom, "(3)"] & D \arrow[d, "d"] & B \arrow[r, "b'_1 \circ b^*"] \arrow[d] \arrow[dr, phantom, "(4)"] & A_1 \arrow[d, "f_1"] \\
C \arrow[r, "c^*"] & E & C \arrow[r, "c'_1 \circ c^*"] & F_1
\end{tikzcd}$$

3.1.3 *Negative Constraints and Negative Application Conditions*

In this section we extend the expressiveness of transformation systems by introducing negative constraints and negative application conditions. Constraints and application conditions were first introduced in [EH86] for graphs, and later lifted to adhesive transformation systems [EEHP06]. In this thesis we stick to negative constraints and negative application conditions. Moreover, for the rest of this section, we assume that we have fixed an $(\mathcal{M}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{M}, \mathcal{N})$ with distinguished classes $\mathcal{M}$ and $\mathcal{N}$ for production and match morphisms, respectively.

We begin with introducing the notion of constraints. Constraints define global conditions on the inner structure of objects. An object that satisfies these conditions is called consistent. A negative constraint defines a forbidden structure that must not appear in any consistent object.

Definition 3.29 (Negative constraint)**.**
A *simple negative constraint* $nc(N)$ is defined by an object N. An object G *is consistent with respect to a simple negative constraint* $nc(N)$, denoted as $G \Vdash nc(N)$, if there does not exist an $\mathcal{N}$-morphism $c : N \to G$.

A *negative constraint* NC is a set consisting of simple negative constraints. An object G *is consistent w. r. t. to a negative constraint* NC, denoted as $G \Vdash NC$, if G is consistent with respect to each simple negative constraint $nc(N)$ in NC.

Note that every object is consistent with respect to the empty negative constraint $NC = \emptyset$. In the following, we write $G \nVdash NC$ to express that $G \Vdash NC$ does not hold.

While negative constraints impose global restrictions on the inner structure of objects, negative application conditions place local restrictions on the context of a match.

Definition 3.30 (Negative application condition (NAC)).
A *simple negative application condition* $nac(L \xrightarrow{x} X)$ over an object L is defined by an $\mathcal{N}$-morphism $x : L \to X$.

An $\mathcal{N}$-morphism $m : L \to G$ *satisfies a simple negative application condition* $nac(L \xrightarrow{x} X)$, denoted as $m \Vdash nac(L \xrightarrow{x} X)$, if there does not exist an $\mathcal{N}$-morphism $p : X \to G$ such that $p \circ x = m$.

A *negative application condition* NAC_L *over* L is a set consisting of simple negative application conditions over L.

An $\mathcal{N}$-morphism $m : L \to G$ *satisfies* a negative application condition NAC_L, denoted as $m \Vdash NAC_L$, if morphism m satisfies all simple NACs $nac(L \xrightarrow{x} X)$ in NAC_L.

Similarly to the negative constraints, every $\mathcal{N}$-morphism satisfies the empty negative application condition $NAC_L = \emptyset$; we write $m \nVdash NAC_L$ for *not* $m \Vdash NAC_L$.

By adding negative application conditions to the left-hand sides (or right-hand sides) of productions, we gain additional means to control their applicability.

Definition 3.31 (Extended production).
Let $((\mathbb{C}, \mathcal{M}, \mathcal{N}), \mathbf{P})$ be an $(\mathcal{M}, \mathcal{N})$-adhesive transformation system with a production $p = (L \leftarrow K \rightarrow R) \in \mathbf{P}$, an *extended production*

$$\varrho = (p, NAC_L, NAC_R)$$

consists of a production p, a *precondition negative application condition* NAC_L over L, and a *postcondition negative application condition* NAC_R over R.

A *direct transformation* $G \xRightarrow{\varrho @ m} H$ *via extended production* ϱ is a direct transformation $G \xRightarrow{p @ m} H$ via (nonextended) production p such that the match m satisfies NAC_L and comatch n satisfies NAC_R.

Up until now, we considered constraints and application conditions separately; however, both concepts can be related via the notion of consistency preserving productions. Basically, a production is *consistency preserving with respect to a negative constraint* NC if for any object G that is consistent with respect to NC, any result of applying the production to G is also consistent with respect to NC.

Definition 3.32 (Consistency guaranteeing and preserving production).
Given negative constraint NC and extended production $\varrho = (p, NAC_L, NAC_R)$, the extended production ϱ is *consistency guaranteeing* with respect to a negative constraint NC if for all direct transformation $G \xRightarrow{\varrho @ m} H$, we have that

$$G \Vdash NC \text{ if and only if } H \Vdash NC.$$

The extended production ϱ is *consistency preserving* with respect to NC if for all direct transformation $G \xRightarrow{\varrho @ m} H$, we have that

$$G \Vdash NC \text{ implies } H \Vdash NC.$$

3.2 Introduction to First-Order Logic Languages

In this section, we give a brief introduction to many sorted first-order logic. More specifically, we provide the syntax and semantics of the first-order logic language, which is used in the next section to define symbolic graphs. This section is based on the books [Gal85, EFT94].

3.2.1 *Syntax of First-Order Logic*

Basically, the syntax of a first-order language is composed of two parts, the *logical part* consisting of logical connectives and variables, and the *nonlogical part* consisting of constant, function, and predicate symbols. While the logical part is fixed, the nonlogical part depends on the intended application of the language and is given by a signature.

We begin with the nonlogical part, but first we recall the concept of a *word*. Given an alphabet A (i. e., in the general case an infinite, but countable set of elements called symbols), *the set of all words over alphabet* A is denoted by A^*. *The set of all words of length* n *over alphabet* A is denoted as A^n, where $n \in \mathbb{N}^0$. The set $A^0 = \{\varepsilon\}$ contains the *empty word* ε only. The length of a word ω is denoted as $|\omega| \in \mathbb{N}^0$. *A word* ω *of length* n *is also written as* $u_1 \dots u_n$.

The syntax for the nonlogical part is provided by a signature, which consists of the symbols for naming available sorts, constants, functions, and predicates.

Definition 3.33 (Signature)**.**
A *signature* $\Sigma = (S, O)$ consists of a sort symbol alphabet S, an operation symbol alphabet O, and a function $a : O \to S^* \times S^1 \cup \{\varepsilon\}$ assigning an *arity* (ω, s) to each operation symbol $o \in O$. An operation symbol o with arity (ω, s) is

- a constant symbol if $\omega = \varepsilon$ and $s \neq \epsilon$
- a function symbol if $\omega \neq \varepsilon$ and $s \neq \epsilon$
- a predicate symbol if $\omega \neq \epsilon$ and $s = \varepsilon$

Instead of denoting the arity of a constant symbol $c \in O$ as (ε, s), we say c is of sort s. The arity of a function symbol $f \in O$ is denoted as $s_1 \dots s_n \to s$; for the arity of a predicate symbol $p \in O$ we write $s_1 \dots s_n$.

Given a signature and a set of variable symbols, terms are built by composing the variable symbols with the function and constant symbols in the following way:

Definition 3.34 (Variables and terms)**.**
Let $\Sigma = (S, O)$ be a signature and $X = (X_s)_{s \in S}$ an S-indexed family of sets, where each X_s contains the variables of sort s. We assume that the sets of variables X_s are pairwise disjoint, and disjoint with the set of operation symbols. The S-indexed family of Σ-terms $\mathcal{T}(X) = (\mathcal{T}_s(X))_{s \in S}$ is defined for each sort $s \in S$ by the smallest set $\mathcal{T}_s(X)$ such that

- $x \in \mathcal{T}_s(X)$, for all variables $x \in X_s$

- $c \in \mathcal{T}_s(X)$, for all constant symbols $c \in O$ of sort s
- $f(t_1, \ldots, t_n) \in \mathcal{T}_s(X)$, for all function symbols $f \in O$ with arity $s_1 \ldots s_n \rightarrow s$ and all terms $t_i \in \mathcal{T}_{s_i}(X)$ for $i \in \{1, \ldots, n\}$

The syntax of a first-order logic language over a given signature is given by the set $\mathcal{F}(X)$ consisting of all first-order Σ-formulas obtained from composing Σ-terms with predicate symbols and the logical symbols as defined next.

Definition 3.35 (First-order Σ-formula)**.**
Let Σ be a signature and $X = (X_s)_{s \in S}$ an S-indexed family of variable sets, the set of *first-order Σ-formulas* (or short Σ-formulas) over X is defined as the smallest set $\mathcal{F}(X)$ such that:

- $p(t_1, \ldots, t_n) \in \mathcal{F}(X)$, for each predicate symbol $p : s_1 \ldots s_n \in O$ and terms $t_i \in \mathcal{T}_{s_i}(X)$, with $i \in \{1, \ldots, n\}$
- $t_1 \overset{s}{=} t_2 \in \mathcal{F}(X)$, for terms $t_1, t_2 \in \mathcal{T}_s(X)$
- $\top \in \mathcal{F}(X)$ and $\bot \in \mathcal{F}(X)$
- $\neg\Phi \in \mathcal{F}(X)$, for each $\Phi \in \mathcal{F}(X)$
- $\Phi \wedge \Psi$, $\Phi \vee \Psi$, $\Phi \Rightarrow \Psi$, and $\Phi \Leftrightarrow \Psi$ are in $\mathcal{F}(X)$, for each Φ and Ψ in $\mathcal{F}(X)$
- $\exists x.\Phi \in \mathcal{F}(X)$ and $\forall x.\Phi \in \mathcal{F}(X)$, for each $\Phi \in \mathcal{F}(X)$ and $x \in X$

The logical connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ denote *negation*, *and*, *or*, *implication*, and *equivalence*, respectively; the logical symbols $\top$ and $\bot$ denote *true* and *false*, respectively. Although, not explicitly mentioned we also use parentheses with usual meaning in first-order Σ-formulas.

In first-order logic, variables may occur *free or bound by a quantifier* in a Σ-formula. The set consisting of all variables that occur free in a given Σ-formula is obtained as follows:

Definition 3.36 (Free variables)**.**
Let $\Sigma = (S, O)$ be a signature and $X = (X_s)_{s \in S}$ be an S-indexed family of variable sets, the S-indexed family of sets of variables $var(t) = (var_s(t))_{s \in S} \subseteq X$ of a term $t \in \mathcal{T}(X)$ is defined for each sort $s \in S$ by the set $var_s(t)$:

- $var_s(x) = \begin{cases} \{x\} & \text{if } x \in X_s, \\ \emptyset & \text{otherwise} \end{cases}$, for a variable x
- $var_s(c) = \emptyset$, for a constant c
- $var_s(f(t_1, \ldots, t_n)) = var_s(t_1) \cup \ldots \cup var_s(t_n)$, for a function symbol f of arity $s_1 \ldots s_n \rightarrow s$

The S-indexed family of sets $FV(\Phi) = (FV_s(\Phi))_{s\in S} \subseteq X$, containing the free variables of a Σ-formula $\Phi \in \mathcal{F}(X)$, is defined for each sort $s \in S$ by the set $FV_s(\Phi)$:

- $FV_s(p(t_1, \ldots, t_n) = var_s(t_1) \cup \ldots \cup var_s(t_n)$, for a predicate symbol p of arity $s_1 \ldots s_n$
- $FV_s(t_1 \overset{s}{=} t_2) = var_s(t_1) \cup var_s(t_2)$, for terms t_1 and t_2
- $FV_s(\neg\Phi) = FV_s(\Phi)$
- $FV_s(\Phi * \Psi) = FV_s(\Phi) \cup FV_s(\Psi)$, for $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
- $FV_s(\mathbf{Q}x.\Phi) = \begin{cases} FV_s(\Phi)\backslash\{x\} & \text{if } x \in X_s, \\ FV_s(\Phi) & \text{otherwise} \end{cases}$, for $\mathbf{Q} \in \{\exists, \forall\}$

Example 3.37 (Syntax of linear integer arithmetic).
The signature $\Sigma_{\mathbf{LIA}} = (\{int\}, \{+, -, 0, 1, <, \leq\})$ for linear integer arithmetic consists of a single sort symbol *int*, and the

function symbols:

- $+$ with arity $int\ int \rightarrow int$
- $-$ with arity $int\ int \rightarrow int$

constant symbols:

- 0 and 1 both of sort *int*

predicate symbol:

- $<$ with arity $int\ int$

Given a set of variables $X_{int} = \{x, z\}$ we can construct a $\Sigma_{\mathbf{LIA}}$-formula

$$\exists z.(z + z \overset{int}{=} x),$$

where x is the only variable that occurs free in the formula.

Note, as we have not defined any semantics for Σ-formulas, the formula shown in the previous example is up until now just a sequence of symbols.

3.2.2 *Semantics of First-Order Logic*

Basically, the semantics of a Σ-formula is obtained by assigning values to the free variables and by interpreting logical and nonlogical symbols. While the meaning of logical symbols is fixed, the meaning of the nonlogical symbols is given by a Σ-structure.

Definition 3.38 (Many-sorted Σ-structure)**.**
Given a signature Σ, *a many-sorted* Σ*-structure* $\mathcal{D} = (|\mathcal{D}|, \mathfrak{I})$ is a pair consisting of a S-indexed family of nonempty sets $|\mathcal{D}| = (|\mathcal{D}|_s)_{s\in S}$, each called the domain for sort s, and an interpretation function $\mathfrak{I}$ defined as follows:

- Each constant symbol $c \in O$ of sort s is interpreted as an element $\mathfrak{I}(c) = c^{\mathcal{D}}$, where $c^{\mathcal{D}} \in |\mathcal{D}|_s$.
- Each function symbol $f \in O$ of arity $s_1 \dots s_n \to s$ is interpreted as a function $\mathfrak{I}(f) = f^{\mathcal{D}}$, where $f^{\mathcal{D}} : |\mathcal{D}|_{s_1} \times \dots \times |\mathcal{D}|_{s_n} \to |\mathcal{D}|_s$.
- Each predicate symbol $p \in O$ of arity $s_1 \dots s_n \in O$ is interpreted as a relation $\mathfrak{I}(p) = p^{\mathcal{D}}$, where $p^{\mathcal{D}} \subseteq |\mathcal{D}|_{s_1} \times \dots \times |\mathcal{D}|_{s_n}$.

While a Σ-structure assigns meaning to constant, function, and predicate symbols, it does not assign any meaning to variables. Variables receive meaning by assigning values to them.

Definition 3.39 (Variable assignment)**.**
Given a Σ-structure $\mathcal{D}$, a *variable assignment* is any S-indexed family of functions $\zeta : X \to |\mathcal{D}| = (\zeta_s : X_s \to |\mathcal{D}|_s)_{s\in S}$ mapping the variables in X_s to elements of the domains $|\mathcal{D}|_s$, respectively.

In the following, we write $\zeta[x \mapsto a] : X \to |\mathcal{D}|$ for the new variable assignment that coincides with ζ except that it assigns to variable x of sort s the element a of the domain $|\mathcal{D}|_s$, i. e.,

$$\zeta[x \mapsto a] = \begin{cases} \zeta(y) & \text{if } x \neq y; \\ a & \text{otherwise.} \end{cases}$$

Given a variable assignment, a term is evaluated in a Σ-structure as follows:

Definition 3.40 (Term evaluation)**.**
The *evaluation of* Σ*-terms with respect to a variable assignment* ζ, is given by the S-indexed family of functions

$$[\![\cdot]\!]^{\mathcal{D}}_{\zeta} : \mathcal{T}(X) \to |\mathcal{D}| = ([\![\cdot]\!]^{\mathcal{D}}_{\zeta,s} : \mathcal{T}_s(X) \to |\mathcal{D}|_s)_{s\in S}$$

defined as follows:

- $[\![x]\!]^{\mathcal{D}}_{\zeta,s} = \zeta_s(x)$, for a variable x of sort s
- $[\![c]\!]^{\mathcal{D}}_{\zeta,s} = \mathfrak{I}(c)$, for a constant symbol c of sort s
- $[\![f(t_1, \dots, t_n)]\!]^{\mathcal{D}}_{\zeta,s} = \mathfrak{I}(f)([\![t_1]\!]^{\mathcal{D}}_{\zeta,s_1}, \dots, [\![t_n]\!]^{\mathcal{D}}_{\zeta,s_n})$, for a function symbol f with arity $s_1, \dots, s_n \to s$ and terms $t_i \in \mathcal{T}_{s_i}(X)$, $i \in \{1, \dots, n\}$

The semantics of Σ-formulas is given by the relation $\models$ defined next.

Definition 3.41 (Semantics of Σ-formulas)**.**
Let $\Phi, \Psi \in \mathcal{F}(X)$ be Σ-formulas over the variables X and $\zeta : X \to |\mathcal{D}|$ a variable assignment. The relation $\models$ is defined in the following where $\not\models$ is given by $(\mathcal{D}, \zeta) \not\models \Phi$ iff not $(\mathcal{D}, \zeta) \models \Phi$.

$$\begin{array}{lll}
(\mathcal{D}, \zeta) \models p(t_1, \ldots, t_n) & \text{iff} & (\llbracket t_1 \rrbracket^{\mathcal{D}}_{\zeta}, \ldots, \llbracket t_n \rrbracket^{\mathcal{D}}_{\zeta}) \in \mathfrak{I}(p) \\
(\mathcal{D}, \zeta) \models (t_1 \overset{s}{=} t_2) & \text{iff} & \llbracket t_1 \rrbracket^{\mathcal{D}}_{\zeta} = \llbracket t_2 \rrbracket^{\mathcal{D}}_{\zeta} \\
(\mathcal{D}, \zeta) \models \top & & \\
(\mathcal{D}, \zeta) \not\models \bot & & \\
(\mathcal{D}, \zeta) \models \neg\Phi & \text{iff} & (\mathcal{D}, \zeta) \not\models \Phi; \\
(\mathcal{D}, \zeta) \models \Phi \wedge \Psi & \text{iff} & (\mathcal{D}, \zeta) \models \Phi \text{ and } (\mathcal{D}, \zeta) \models \Psi \\
(\mathcal{D}, \zeta) \models \Phi \vee \Psi & \text{iff} & (\mathcal{D}, \zeta) \models \Phi \text{ or } (\mathcal{D}, \zeta) \models \Psi \\
(\mathcal{D}, \zeta) \models \Phi \Rightarrow \Psi & \text{iff} & (\mathcal{D}, \zeta) \not\models \Phi \text{ or } (\mathcal{D}, \zeta) \models \Psi \\
(\mathcal{D}, \zeta) \models \Phi \Leftrightarrow \Psi & \text{iff} & (\mathcal{D}, \zeta) \models \Phi \text{ iff } (\mathcal{D}, \zeta) \models \Psi \\
(\mathcal{D}, \zeta) \models \forall x.\Phi & \text{iff} & (\mathcal{D}, \zeta[x \mapsto a]) \models \Phi \text{ for all } a \in |\mathcal{D}|_s,\ s \in S,\ x \in X_s \\
(\mathcal{D}, \zeta) \models \exists x.\Phi & \text{iff} & (\mathcal{D}, \zeta[x \mapsto a]) \models \Phi \text{ for some } a \in |\mathcal{D}|_s,\ s \in S,\ x \in X_s
\end{array}$$

We now define the notion of satisfaction and validity.

Definition 3.42 (Satisfaction and validity)**.**
A Σ-formula Φ is *satisfiable* in $\mathcal{D}$, iff

$$(\mathcal{D}, \zeta) \models \Phi \text{ for some assignment } \zeta.$$

In this case, we say that ζ is a *solution* of Φ in $\mathcal{D}$.

A Σ-formula Φ is *valid* in $\mathcal{D}$ iff

$$(\mathcal{D}, \zeta) \models \Phi \text{ for every assignment } \zeta.$$

In this case we write $\mathcal{D} \models \Phi$.

The following fact states that the truth value of any Σ-formula Φ depends on the assignment of the free variables only.

Fact 3.43 (Free variables)**.**
Let $\mathcal{D}$ be a Σ-structure and Φ a Σ-formula with an S-indexed set of free variables $FV(\Phi) = (FV_s(\Phi))_{s \in S}$. For any two assignments $\zeta_1 = (\zeta_{1s})_{s \in S}$ and $\zeta_2 = (\zeta_{2s})_{s \in S}$, such that for all $s \in S$ and $x_{i(s)} \in FV_s(\Phi)$ it holds that $\zeta_{1s}(x_{i(s)}) = \zeta_{2s}(x_{i(s)})$, we have

$$(\mathcal{D}, \zeta_1) \models \Phi \text{ iff } (\mathcal{D}, \zeta_2) \models \Phi.$$

In the following, we assume that for every Σ-structure $\mathcal{D}$, the corresponding signature contains constant symbols for naming all elements in $|\mathcal{D}|$, so that these elements can be treated as constants in formulas.

Fact 3.44 (Solution)**.**
Let Φ a Σ-formula and ζ a solution of Φ in $\mathcal{D}$, the solution ζ can be represented as a conjunction of equality predicates such that

$$(\mathcal{D}, \zeta) \models \bigwedge_{s \in S} \left(\bigwedge_{x_{i(s)} \in FV_s(\Phi)} (x_{i(s)} \overset{s}{=} c_{i(s)}) \right),$$

where $c_{i(s)}$ are constants in $|\mathcal{D}|_s$.

Moreover, let $\underline{\Phi}$ be a conjunction of equality predicates as defined above, i. e., we have $(\mathcal{D}, \zeta) \models \underline{\Phi}$, then the statements

$$(\mathcal{D}, \zeta) \models \Phi \text{ and } \mathcal{D} \models \underline{\Phi} \Rightarrow \Phi.$$

are equivalent.

Proof. This fact is a direct consequence of Fact 3.43. □

Example 3.45 (Semantics of $\Sigma_{\mathbf{LIA}}$-formulas)**.**
Given the signature $\Sigma_{\mathbf{LIA}}$ provided in Example 3.37. Let $\mathcal{D}_{\mathbf{LIA}}$ be the $\Sigma_{\mathbf{LIA}}$-structure consisting of the domain of the integer numbers $\mathbb{Z}$. The function symbols $+^{\mathcal{D}}$ and $-^{\mathcal{D}}$ are interpreted as addition and subtraction on $\mathbb{Z}$. The constants symbols $0^{\mathcal{D}}$ and $1^{\mathcal{D}}$ are interpreted as the numbers zero and one, whereas $<^{\mathcal{D}}$ denotes the the usual ordering on $\mathbb{Z}$.

Now we can interpret the formula Φ given by $\exists z.(z + z = x)$ as *x is even.* Hence a possible solution is the assignment ζ_1 with $\zeta_1(x) = 4$ as we have $2 \in \mathbb{Z}$ such that $(\mathcal{D}_{\mathbf{LIA}}, \zeta_1[z \mapsto 2]) \models \exists z.(z + z = x)$.

According to Fact 3.44 we can represent the solution as the equality predicate $(x \overset{int}{=} 1 + 1 + 1 + 1)$ that can be rewritten to $(x \overset{int}{=} 4)$ if we assume that we have an constant symbol for every value in the domain $\mathbb{Z}$. Moreover we may obtain a valid formula $\mathcal{D}_{\mathbf{LIA}} \models (x \overset{int}{=} 4) \Rightarrow \exists z.(z + z = x)$.

Finally, we define substitution. To this end, we introduce substitution maps.

Definition 3.46 (Substitution map)**.**
Let Σ be a signature and $X = (X_s)_{s \in S}$ an S-indexed family of variable sets, then *a substitution map* for finite subsets $Y = (Y_s)_{s \in S}$ of $(X_s)_{s \in S}$ is a family of S-indexed functions $\hat{\sigma} : Y \to \mathcal{T}(X)_s = (\hat{\sigma} : Y_s \to \mathcal{T}(X)_s)_{s \in S}$.

In the following, we write

$$\hat{\sigma} : Y \to \mathcal{T}(X) : \frac{\hat{\sigma}(y_1) \ldots \hat{\sigma}(y_n)}{y_1 \ \ldots \ y_n} = \frac{t_1 \ldots t_n}{y_1 \ldots y_n},$$

to denote the substitution map $\hat{\sigma} : Y \to \mathcal{T}(X)$ that sends the variables $y_i \in Y$ to terms $t_i \in \mathcal{T}(X)$. A substitution map of the form

$$\hat{\sigma} : \frac{z_1 \dots z_n}{y_1 \dots y_n},$$

where $z_1, \dots, z_n$ are variables is called *variable map* in the following.

In general, substitution has to be performed carefully on Σ-formulas containing quantifiers, to not unintentionally alter their meaning. More specifically, given a Σ-formula Φ we have to ensure that a free variable x in Φ, is not replaced by a term containing a variable z, when z does not occur free at the position of x in Φ. For example consider the $\Sigma_{\mathbf{LIA}}$-formula $\exists z.(z + z = x)$ which expresses that x is even (as given in Example 3.45). If we now replace variable x by variable z, we obtain formula $\exists z.(z + z = z)$ that is always valid as $0 + 0 = 0$.

The standard solution for this problem is to replace the quantified variables by *fresh* variables, before performing a substitution in the scope of a quantifier.

Definition 3.47 (Substitution [EFT94]).
Let Σ be a signature with S-indexed family with variable set X and let Φ be a Σ-formula. Given a substitution map

$$\hat{\sigma} : Y \to \mathcal{T}(X) : \frac{t_1 \dots t_n}{y_1 \dots y_n},$$

the Σ-formula $\Phi[\hat{\sigma}]$, obtained from substituting Φ along $\hat{\sigma}$ is given as follows:

- $x[\hat{\sigma}] = \begin{cases} x & \text{if } x \notin dom(\hat{\sigma}), \\ \hat{\sigma}(x) & \text{otherwise,} \end{cases}$ for a variable x
- $c[\hat{\sigma}] = c$, for a constant c
- $f(t_1, \dots, t_n)[\hat{\sigma}] = f(t_1[\hat{\sigma}], \dots, t_n[\hat{\sigma}])$, for a function symbol f of arity $s_1 \dots s_n \to s$
- $\top[\hat{\sigma}] = \top$ and $\bot[\hat{\sigma}] = \bot$
- $p(t_1, \dots, t_n)[\hat{\sigma}] = p(t_1[\hat{\sigma}], \dots, t_n[\hat{\sigma}])$, for a predicate symbol p of arity $s_1 \dots s_n$
- $(t_1 \overset{s}{=} t_2)[\hat{\sigma}] = (t_1[\hat{\sigma}] \overset{s}{=} t_2[\hat{\sigma}])$, for terms t_1 and t_2
- $(\Phi * \Psi)[\hat{\sigma}] = (\Phi[\hat{\sigma}] * \Psi[\hat{\sigma}])$, for Σ-formulas Φ, Ψ, and $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
- $(\neg\Phi)[\hat{\sigma}] = \neg(\Phi[\hat{\sigma}])$, for a Σ-formula Φ
- Suppose $y_{i_1}, \dots, y_{i_r}$ $(i_1 < \dots < i_r)$ are exactly the variables $y_i \in Y_{s(i)}$ among $y_1, \dots, y_n$ such that

$$y_i \in FV_{s(i)}(\mathbf{Q}x.\Phi) \text{ and } y_i \neq t_i, \text{ where } \mathbf{Q} \in \{\exists, \forall\}.$$

Then set

$$(\mathbf{Q}x.\Phi)\left[\frac{t_1 \dots t_n}{y_1 \dots y_n}\right] = \mathbf{Q}u.\left(\Phi\left[\frac{t_{i_0} \dots t_{i_r}\ \ u}{y_{i_0} \dots y_{i_r}\ \ x}\right]\right)$$

where $u = x$ if x does not occur in $t_{i_0} \dots t_{i_r}$; otherwise choose u such that u is a variable in $X_{s(i)}$ that does not occur anywhere in Φ or $t_{i_0} \dots t_{i_r}$.

3.3 Symbolic Graphs ans Symbolic Graph Transformation

In this section we introduce symbolic graphs and symbolic graph transformation based on the DPO approach. Symbolic graphs were originally introduced by Orejas et al. in [OL10b], as an attribution concept for graphs. A symbolic graph consists of an E-graph and a Σ-formula, where an *E-graph* is a graph with an additional kind of *label nodes*, used as placeholder for attribute values. Instead of assigning concrete values directly to the label nodes, symbolic graphs provide a more expressive kind of attribution by considering the label nodes as variables and constraining their values by a Σ-formula.

3.3.1 *The Category of Symbolic Graphs*

Before we define symbolic graphs we have to establish the category of E-graphs.

Definition 3.48 (E-graph and E-graph morphism)**.**
An *E-graph* is a tuple $G = (V_G, X_G, E_G^V, E_G^X, s_G^V, s_G^X, t_G^V, t_G^X)$ consisting of:

- V_G and X_G the sets of graph and label nodes, respectively
- E_G^V and E_G^X the sets of graph and label edges, respectively

and the source and target functions:

- $s_G^V : E_G^V \to V_G$ and $t_G^V : E_G^V \to V_G$ relating two graph nodes
- $s_G^X : E_G^X \to V_G$ and $t_G^X : E_G^X \to X_G$ assigning a label node to a graph node

An *E-graph morphism* $h : G \to H$ from E-graph G to E-graph H is a tuple of total functions

$$h = (h_V : V_G \to V_H, h_X : X_G \to X_H, h_{EV} : E_G^V \to E_H^V, h_{EX} : E_G^X \to E_H^X),$$

such that h commutes with source and target functions, i.e., $h_V \circ s_G^V = s_H^V \circ h_{EV}$, $h_V \circ t_G^V = t_H^V \circ h_{EV}$, $h_V \circ s_G^X = s_H^X \circ h_{EX}$, $h_X \circ t_G^X = t_H^X \circ h_{EX}$.

Fact 3.49 (The category $\mathbb{EG}$)**.**
E-graphs together with their morphisms form the category $\mathbb{EG}$ of E-graphs.

Note that in contrast to the definition above, E-graphs are usually defined with an additional kind of edge label nodes and corresponding edge label edges. However, as attribution of edges is not supported of the most practical approaches we leave this out in order to improve readability.

Analogously to graphs, E-graphs can be extended by a type concept, leading to typed E-graphs and typed E-graph morphisms.

Definition 3.50 (Typed E-graph and E-graph morphism).
A *E-type graph* is a distinguished E-graph

$$TG = (V_{TG}, X_{TG}, E^V_{TG}, E^X_{TG}, s^V_{TG}, s^X_{TG}, t^V_{TG}, t^X_{TG}),$$

where V_{TG} , X_{TG}, E^V_{TG}, and E^X_{TG} define the graph node, label node, graph edge and label edge type alphabet, respectively.

A *typed E-graph* is a tuple $(G, type)$ consisting of an E-graph G and an E-graph morphism $type : G \to TG$.

A *typed E-graph morphism* $h : (G, type_1) \to (H, type_2)$ is an E-graph morphism $h : G \to H$ such that $type_2 \circ h = type_1$.

Typed E-graphs over an E-type graph TG together with typed E-graph morphisms constitute the category $\mathbb{TEG}_{TG}$.

Fact 3.51 (The category $\mathbb{TEG}_{TG}$).
Typed E-graphs over an E-type graph TG together with typed E-graph morphisms constitute the category $\mathbb{TEG}_{TG}$.

Note that given an E-type graph TG, the category $\mathbb{TEG}_{TG}$ of typed E-graphs over TG is isomorphic to the slice category $\mathbb{EG}\backslash TG$.

As we shall use various different kind of morphisms classes in this thesis, we introduce a self describing naming scheme for morphism classes, to improve the readability of this thesis. Since the mapping mode for label nodes often differs from the mapping mode for the other components, we apply the following naming scheme. A class of E-graph morphisms is denoted as $\mathcal{X}^y$, where $\mathcal{X} \in \{\mathcal{M}, \mathcal{I}, \mathcal{E}\}$ specifies the mode for mapping graph nodes, graph edges, and label edges; the superscript $y \in \{inj, bij, surj, *\}$ specifies mode for mapping the label nodes. More specifically, $\mathcal{M}$ means injective, $\mathcal{I}$ means bijective, and $\mathcal{E}$ means surjective for graph nodes an all kind of edges, respectively. The superscript *inj* denotes an injective, *bij* a bijective, *surj* a surjective, and $*$ an arbitrary mapping for label nodes. Typed E-graphs are denoted as $\mathcal{X}^y_{TG}$. According to this scheme the classes $\mathcal{I}^{bij}$ and $\mathcal{M}^{inj}$ correspond to isomorphisms and monomorphisms in category $\mathbb{EG}$; similarly, $\mathcal{I}^{bij}_{TG}$ and $\mathcal{M}^{inj}_{TG}$ correspond to isomorphisms and monomorphisms in category $\mathbb{TEG}_{TG}$.

Fact 3.52 (Properties of the category $\mathbb{TEG}_{TG}$ and $\mathbb{EG}$ [EEPT06]).
The following properties hold in the category $\mathbb{TEG}_{TG}$ over a type graph TG:

a) $\mathbb{TEG}_{TG}$ has pushouts and pullbacks along arbitrary morphisms in $\mathbb{TEG}_{TG}$

b) $\mathbb{TEG}_{TG}$ has binary coproducts

c) $\mathbb{TEG}_{TG}$ has an $\mathcal{E}$–$\mathcal{M}$ factorization for $\mathcal{E} = \mathcal{E}^{surj}_{TG}$ the class of all typed epimorphisms and $\mathcal{M} = \mathcal{M}^{inj}_{TG}$ the class of all typed monomorphisms.

Moreover, the category $\mathbb{TEG}_{TG}$ with the class $\mathcal{M}^{inj}_{TG}$ of typed E-graph monomorphism is $\mathcal{M}$-adhesive. Hence, the category $\mathbb{TEG}_{TG}$ with $\mathcal{M} = \mathcal{N} = \mathcal{M}^{inj}_{TG}$ is $(\mathcal{M}, \mathcal{N})$-adhesive (see Remark 3.19).

Note that the previous properties hold also for the category $\mathbb{EG}$ of (untyped) E-graphs with (untyped) mono and epimorphism $\mathcal{M}^{inj}$ and $\mathcal{E}^{surj}$.

Symbolic graphs can be defined by combining E-graphs with Σ-formulas as follows:

Definition 3.53 (Symbolic graphs [OL10b]).
Given a Σ-structure $\mathcal{D}$ and a set of variables $X = (X_s)_{s\in S}$, a *symbolic graph* $G^\Phi = \langle G, \Phi_G\rangle$ consists of an E-graph G and a Σ-formula $\Phi_G \in \mathcal{F}(X)$, such that $\dot{\bigcup}_{s\in S} FV(\Phi_G) \subseteq X_G$, where $\dot{\bigcup}$ denotes the disjoint union.

In the following, we sometimes write G^Φ as a shorthand notation for a symbolic graph $\langle G, \Phi_G\rangle$. As defined above, label nodes serve as variables for formulas. Usually, we use the term *label nodes* to refer to the corresponding nodes in the graph component, and the term *variables* to refer to the corresponding elements in the Σ-formula. However, we sometimes refer to the label nodes as variables, too.

Symbolic graph morphisms are defined as follows:

Definition 3.54 (Symbolic graph morphisms [OL10b]).
Given a Σ-structure $\mathcal{D}$, a *symbolic graph morphism* $h : G^\Phi \to H^\Phi$ from symbolic graph $G^\Phi = \langle G, \Phi_G\rangle$ to symbolic graph $H^\Phi = \langle H, \Phi_H\rangle$ is a pair $(h, \hat{h})$ consisting of an E-graph morphism $h : G \to H$ and a variable map

$$\hat{h} : FV(\Phi_G) \to FV(\Phi_H) = \left(\hat{h}_s : FV_s(\Phi_G) \to FV_s(\Phi_H)\right)_{s\in S}$$

such that (1) commutes and $\mathcal{D} \models \Phi_H \Rightarrow \Phi_G[\hat{h}]$, where the vertical morphisms in (1) are inclusions.

$$\begin{tikzcd}
FV_s(\Phi_G) \arrow[r, "\hat{h}_s"] \arrow[d, hook] \arrow[dr, phantom, "(1)"] & FV_s(\Phi_H) \arrow[d, hook] \\
X_G \arrow[r, "h_{G,X}"] & X_H
\end{tikzcd}$$

Fact 3.55 (The category $\mathbb{SG}_{\mathcal{D}}$).
Symbolic graphs over a Σ-structure $\mathcal{D}$ together with symbolic graph morphisms form the category $\mathbb{SG}_{\mathcal{D}}$.

Fact 3.56 ($\mathbb{SG}_{\mathcal{D}}$ has pushouts [OL10b]).
Given a Σ-structure $\mathcal{D}$, the category $\mathbb{SG}_{\mathcal{D}}$ has pushouts along arbitrary morphisms in $\mathbb{SG}_{\mathcal{D}}$.

Remark 3.57 (Construction of pushouts in $\mathbb{SG}_{\mathcal{D}}$).
Diagram (1) is a pushout in $\mathbb{SG}_{\mathcal{D}}$ iff diagram (2) is a pushout in $\mathbb{EG}$ and

$$\mathcal{D} \models \Phi_D \Leftrightarrow (\Phi_B[\hat{h}'] \wedge \Phi_C[\hat{g}']).$$

$$\begin{tikzcd}
\langle A, \Phi_A \rangle \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & \langle B, \Phi_B \rangle \arrow[d, "g'"] \\
\langle C, \Phi_C \rangle \arrow[r, "f'"] & \langle D, \Phi_D \rangle
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(2)"] & B \arrow[d, "g'"] \\
C \arrow[r, "f'"] & D
\end{tikzcd}$$

Note that, the Σ-formula component of the pushout object can be constructed purely syntactically.

Fact 3.58 ($\mathbb{SG}_{\mathcal{D}}$ has pullbacks [OL10b]).
Given a Σ-structure $\mathcal{D}$, the category $\mathbb{SG}_{\mathcal{D}}$ has pullbacks along arbitrary morphisms in $\mathbb{SG}_{\mathcal{D}}$.

Although, $\mathbb{SG}_{\mathcal{D}}$ has pullbacks along arbitrary morphisms in $\mathbb{SG}_{\mathcal{D}}$ [OL10b], for this thesis it is sufficient to consider the construction of pullbacks for the case that both morphisms are injective.

Remark 3.59 (Construction of pullbacks in $\mathbb{SG}_{\mathcal{D}}$). Diagram (1) with $f, g \in \mathcal{M}_{\Rightarrow}^{inj}$ is a pullback in $\mathbb{SG}_{\mathcal{D}}$ iff, (2) is a pullback in $\mathbb{EG}$ and

$$\mathcal{D} \models \Phi_A \Leftrightarrow (\exists b_1 \ldots \exists b_n . \Phi_B) \vee (\exists c_1 \ldots \exists c_n . \Phi_C),$$

where $\{b_1, \ldots, b_n\} = X_B \backslash f'_X(X_A)$ and $\{c_1, \ldots, c_n\} = X_C \backslash g'_X(X_A)$ are the label nodes of B and C that have no preimage in A, respectively.

$$\begin{tikzcd}
\langle A, \Phi_A \rangle \arrow[r, "f'"] \arrow[d, "g'"'] \arrow[dr, phantom, "(1)"] & \langle B, \Phi_B \rangle \arrow[d, "g"] \\
\langle C, \Phi_C \rangle \arrow[r, "f"] & \langle D, \Phi_D \rangle
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
A \arrow[r, "f'"] \arrow[d, "g'"'] \arrow[dr, phantom, "(2)"] & B \arrow[d, "g"] \\
C \arrow[r, "f"] & D
\end{tikzcd}$$

Analogously to E-graphs, symbolic graphs can be extended by a type concept. To this end we define a symbolic type graph $TG^{\Phi} = \langle TG, \bot \rangle$ as an E-type graph TG and formula $\bot$, which ensures that if there exists an E-graph morphism $type : G \rightarrow TG$ from an symbolic graph $\langle G, \Phi_G \rangle$ to the symbolic type graph $\langle TG, \bot \rangle$ then $type$ is also a symbolic morphism as $\mathcal{D} \vDash \bot \Rightarrow \Phi_G[\widehat{type}]$ in any Σ-structure $\mathcal{D}$.

Definition 3.60 (Typed symbolic graphs and morphisms [OL10b]).
Let $\mathcal{D}$ be any Σ-structure and S the sort alphabet of Σ, a symbolic type graph is a distinguished symbolic graph $TG^{\Phi} = \langle TG, \perp \rangle$, with E-type graph

$$TG = (V_{TG}, X_{TG}, E^{V}_{TG}, E^{X}_{TG}, s^{V}_{TG}, s^{X}_{TG}, t^{V}_{TG}, t^{X}_{TG}),$$

such that X_{TG} contains exactly one variable x_s for each sort symbol $s \in S$, i. e., $X_{TG} = \dot{\bigcup}_{s \in S} x_s$.

A *typed symbolic graph* is a tuple $(G^{\Phi}, \mathit{type})$ consisting of a symbolic graph G^{Φ} and a symbolic graph morphism $\mathit{type} : G^{\Phi} \to TG^{\Phi}$.

A *typed symbolic graph morphism* $h : (G^{\Phi}, \mathit{type}_1) \to (H^{\Phi}, \mathit{type}_2)$ is a symbolic graph morphism $h : G^{\Phi} \to H^{\Phi}$ such that $\mathit{type}_2 \circ h = \mathit{type}_1$.

Fact 3.61 (The category $\mathbb{TSG}_{\mathcal{D},TG}$).
Typed symbolic graphs over an symbolic type graph TG^{Φ} together with typed symbolic graph morphisms form the category $\mathbb{TSG}_{\mathcal{D},TG}$.

In order to denote classes of symbolic graph morphisms, the naming scheme for E-graph morphism classes is extended by adding to each E-graph morphism class symbol a subscript that specifies the operator for relating the formula components of the codomain an domain. Hence, a class of (typed) symbolic graph morphisms is denoted as $\mathcal{X}^{y}_{z}$ ($\mathcal{X}^{y}_{z,TG}$).

According tho this scheme we can define the classes $\mathcal{I}^{bij}_{\Leftrightarrow}$ and $\mathcal{I}^{bij}_{\Leftrightarrow,TG}$, which are indeed isomorphisms in $\mathbb{SG}_{\mathcal{D}}$ and $\mathbb{TSG}_{\mathcal{D},TG}$, respectively.

Fact 3.62 (Isomorphisms in $\mathbb{SG}_{\mathcal{D}}$ and $\mathbb{TSG}_{\mathcal{D},TG}$).
Let $\mathbb{SG}_{\mathcal{D}}$ be the category of symbolic graphs over a Σ-structure $\mathcal{D}$. Given a morphism $h : \langle G, \Phi_G \rangle \to \langle H, \Phi_H \rangle$ in $\mathbb{SG}_{\mathcal{D}}$, then $h \in \mathcal{I}^{bij}_{\Leftrightarrow}$ if h is bijective for all kinds of node and edges, and $\mathcal{D} \models \Phi_H \Leftrightarrow \Phi_G[\hat{h}]$. Moreover, if $h \in \mathcal{I}^{bij}_{\Leftrightarrow}$, then h is an isomorphism in $\mathbb{SG}_{\mathcal{D}}$.

The class of all typed symbolic graph isomorphisms $\mathcal{I}^{bij}_{\Leftrightarrow,TG}$ is given by

$$\mathcal{I}^{bij}_{\Leftrightarrow,TG} = \mathcal{I}^{bij}_{\Leftrightarrow} \cap Mor_{\mathbb{SG}_{\mathcal{D}} \backslash TG^{\Phi}}.$$

Proof. Given a morphism $h : \langle G, \Phi_G \rangle \to \langle H, \Phi_H \rangle$ in $\mathcal{I}^{bij}_{\Leftrightarrow}$, to show that h is an isomorphisms we have to verify that there exists a symbolic graph morphism $h^{-1} : \langle H, \Phi_H \rangle \to \langle G, \Phi_G \rangle$ such that $h \circ h^{-1} = id_H$ and $h^{-1} \circ h = id_G$. As h is an isomorphism in $\mathbb{EG}$ we know that there is an E-graph morphisms $h^{-1} : H \to G$ such that $h \circ h^{-1} = id_H$ and $h^{-1} \circ h = id_G$. Hence h^{-1} is obviously also a symbolic graph morphisms as $h \in \mathcal{I}^{bij}_{\Leftrightarrow}$ implies that $\mathcal{D} \models \Phi_H \Leftrightarrow \Phi_G[\hat{h}]$, so $\mathcal{D} \models \Phi_G \Rightarrow \Phi_H[\hat{h}^{-1}]$.

Given a symbolic graph isomorphism $h : \langle G, \Phi_G \rangle \to \langle H, \Phi_H \rangle$, there is an symbolic graph morphism $h^{-1} : \langle H, \Phi_H \rangle \to \langle G, \Phi_G \rangle$ such that $h \circ h^{-1} = id_H$ and $h^{-1} \circ h = id_G$.

From symbolic graph morphism h we obtain

$$\mathcal{D} \models \Phi_H \Rightarrow \Phi_G[\hat{h}]. \tag{3.1}$$

From symbolic graph morphism h^{-1} we obtain

$$\mathcal{D} \models \Phi_G \Rightarrow \Phi_H[\hat{h}^{-1}],$$

which can be rewritten as

$$\mathcal{D} \models \left(\Phi_G \Rightarrow \Phi_H[\hat{h}^{-1}]\right)[\hat{h}] \Leftrightarrow \left(\Phi_G[\hat{h}] \Rightarrow \Phi_H[\hat{h} \circ \hat{h}^{-1}]\right),$$

leading to

$$\mathcal{D} \models \Phi_G[\hat{h}] \Rightarrow \Phi_H, \tag{3.2}$$

as $h \circ h^{-1} = id_H$.
From expressions 3.1 and 3.2 follows

$$\mathcal{D} \models \Phi_H \Leftrightarrow \Phi_G[\hat{h}];$$

hence, $h \in \mathcal{I}^{bij}_{\Leftrightarrow}$.

Showing that $\mathcal{I}^{bij}_{\Leftrightarrow,TG}$ is the class of all typed symbolic graph isomorphisms is similar. □

3.3.2 *Typed Symbolic Graph Transformation Systems*

Based on the results originally presented in [OL10b], we define $(\mathcal{M}, \mathcal{N})$-adhesive transformation systems for the category of typed symbolic graphs.

Hence, we have to fix the classes $\mathcal{M}$ and $\mathcal{N}$ such that $(\mathbb{TSG}_{\mathcal{D},TG}, \mathcal{M}, \mathcal{N})$ is an $(\mathcal{M}, \mathcal{N})$-adhesive category. One may wonder whether typed symbolic graphs with the class $\mathcal{M}$ consisting of all typed symbolic graph monomorphisms is an $(\mathcal{M}, \mathcal{N})$-adhesive category; unfortunately, as shown in [OL10b], the answer is no.

However, the category $\mathbb{TSG}_{\mathcal{D},TG}$ becomes an $(\mathcal{M}, \mathcal{N})$-adhesive category by choosing $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow,TG}$. The classes $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{M}^{*}_{\Rightarrow,TG}$ are given as follows:

Definition 3.63 ($\mathcal{M}^{bij}_{\Leftrightarrow,TG}$, and $\mathcal{M}^{*}_{\Rightarrow,TG}$-morphisms)**.**
Let $\mathbb{SG}_{\mathcal{D}}$ be the category of typed symbolic graphs over a Σ-structure $\mathcal{D}$, given a morphisms $h : \langle G, \Phi_G \rangle \rightarrow \langle H, \Phi_H \rangle$ in $\mathbb{SG}_{\mathcal{D}}$ then:

- $h \in \mathcal{M}^{bij}_{\Leftrightarrow}$ if h is injective for graph nodes and all kinds of edges, bijective for label nodes, and $\mathcal{D} \models \Phi_H \Leftrightarrow \Phi_G[\hat{h}]$.
- $h \in \mathcal{M}^{*}_{\Rightarrow}$ if h is injective for graph nodes and all kinds of edges, arbitrary for label nodes, and $\mathcal{D} \models \Phi_H \Rightarrow \Phi_G[\hat{h}]$.

Let $\mathbb{TSG}_{\mathcal{D},TG}$ be the category of typed symbolic graphs over a Σ-structure $\mathcal{D}$ and a symbolic type graph TG^{Φ}, the classes $\mathcal{M}^{*}_{\Rightarrow,TG}$ and $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$ are given as

$$\mathcal{M}^{*}_{\Rightarrow,TG} = \mathcal{M}^{*}_{\Rightarrow} \cap Mor_{\mathbb{SG}_{\mathcal{D}} \backslash TG^{\Phi}} \text{ and } \mathcal{M}^{bij}_{\Leftrightarrow,TG} = \mathcal{M}^{bij}_{\Leftrightarrow} \cap Mor_{\mathbb{SG}_{\mathcal{D}} \backslash TG^{\Phi}},$$

respectively.

Note that in [OL10b] it was shown that the category of symbolic graphs with $\mathcal{M}^{bij}_{\Leftrightarrow}$-morphisms is an $\mathcal{M}$-adhesive HLR category (therefore, also $\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$-morphisms). Hence, to define transformations we may choose the class for match morphisms as the class of all symbolic graph morphisms. However, more advanced concepts, e. g., the construction of precondition application conditions from constraints, are not applicable to arbitrary match morphisms. In [DV14] we have shown that the construction of precondition application conditions from constraints is possible for the choices $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and for $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow,TG}$.

Theorem 3.64 ($\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow,TG}$ is $(\mathcal{M},\mathcal{N})$-adhesive)**.**

Let $\mathbb{TSG}_{\mathcal{D},TG}$ be the category of typed symbolic graphs over a given Σ-structure $\mathcal{D}$ and symbolic type graph TG^{Φ}, then the category $\mathbb{TSG}_{\mathcal{D},TG}$ with morphism classes $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow,TG}$ is $(\mathcal{M},\mathcal{N})$-adhesive.

Proof. In [OL10b], it was shown that $\mathbb{SG}_{\mathcal{D}}$ with $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow}$ is $\mathcal{M}$-adhesive. According to Remark 3.19, to show that $\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow,TG}$ is $(\mathcal{M},\mathcal{N})$-adhesive, we have to ensure that (a) $\mathcal{M}^{*}_{\Rightarrow}$ contains all isomorphisms, (b) $\mathcal{M}^{*}_{\Rightarrow}$ is closed under composition and decomposition, (c) $\mathcal{M}^{*}_{\Rightarrow}$ is closed under $\mathcal{M}^{bij}_{\Leftrightarrow}$ decomposition, and (d) $\mathcal{M}^{*}_{\Rightarrow}$ is closed under pushouts and pullbacks along $\mathcal{M}^{bij}_{\Leftrightarrow}$-morphisms:

a) It is obvious that $\mathcal{I}^{bij}_{\Leftrightarrow}$ is a subclass of $\mathcal{M}^{*}_{\Rightarrow}$.

b) $\mathcal{M}^{*}_{\Rightarrow}$ is closed under composition and decomposition, as injectivity is preserved componentwise under composition and decomposition, and any symbolic graph morphism that is injective for graph nodes and all kind of edges is in $\mathcal{M}^{*}_{\Rightarrow}$.

c) As $\mathcal{M}^{bij}_{\Leftrightarrow}$ is a subclass of $\mathcal{M}^{*}_{\Rightarrow}$, it follows from (ii) that $\mathcal{M}^{*}_{\Rightarrow}$ is closed under $\mathcal{M}^{bij}_{\Leftrightarrow}$-decomposition.

d) $\mathcal{M}^{*}_{\Rightarrow}$ is closed under pushouts and pullbacks along $\mathcal{M}^{bij}_{\Leftrightarrow}$-morphisms, as injectivity is preserved componentwise by pushouts and pullbacks along arbitrary morphisms (Fact 3.9 and Fact 3.13), and any symbolic graph morphism that is injective for graph nodes and all kind of edges is in $\mathcal{M}^{*}_{\Rightarrow}$.

As a direct consequence of Fact 3.21, we have that $\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow,TG}$ is $(\mathcal{M},\mathcal{N})$-adhesive, as it is isomorphic to the slice category $\mathbb{SG}_{\mathcal{D}}\backslash TG^{\Phi}$ with $\mathcal{M} = (\mathcal{M}^{bij}_{\Leftrightarrow} \cap Mor_{\mathbb{SG}_{\mathcal{D}}\backslash TG})$ and $\mathcal{N} = (\mathcal{M}^{*}_{\Rightarrow} \cap Mor_{\mathbb{SG}_{\mathcal{D}}\backslash TG})$. □

Based on the category $\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$-morphisms we can define typed symbolic productions and transformation systems.

Definition 3.65 (Typed symbolic productions and typed symbolic transformation systems [OL10b])**.**
A typed symbolic production $p = (\langle L, \Phi_L\rangle \xleftarrow{l} \langle K, \Phi_K\rangle \xrightarrow{r} \langle R, \Phi_R\rangle)$ consists of a

left production morphism $l : \langle L, \Phi_L \rangle \rightarrow \langle K, \Phi_K \rangle$ and *a right production morphism* $r : \langle R, \Phi_R \rangle \rightarrow \langle K, \Phi_K \rangle$, which are both of class $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$.

A *typed symbolic graph transformation system* **TSGTS**, is given by a finite set of typed symbolic productions **P** and a the class of $\mathcal{M}^{*}_{\Rightarrow,TG}$ of match morphisms.

Note, that $l, r \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ implies that Φ_L, Φ_K, and Φ_R are equivalent, often they are the same formula.

Corollary 3.66 (**TSGTS** are $(\mathcal{M}, \mathcal{N})$-adhesive)**.**
Any typed symbolic graph transformation system **TSGTS** constitutes an $(\mathcal{M}, \mathcal{N})$-adhesive transformation system in the sense of Definition 3.24.

Proof. This is a direct consequence of the fact that the category $\mathbb{TSG}_{\mathcal{D},TG}$ with morphism classes $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow,TG}$ is $(\mathcal{M}, \mathcal{N})$-adhesive. □

3.4 Model Transformation by Symbolic Graph Transformation

Up until now, we introduced the concepts of symbolic graphs and symbolic graph transformation on a very abstract level. In the following, we present how the concepts introduced for the campus management system (introduced in Chapter 2) are represented by symbolic graphs and symbolic graph transformations.

Similar to the case of nonattributed graphs, metamodels correspond to a type graph, except that a symbolic type graph contains edges and nodes for representing attributes and attribute types. This is shown in the following example.

Example 3.67 (Symbolic type graph for the CMS metamodel)**.**
Figure 3.1 shows the symbolic type graph for the facility management component of CMS metamodel originally presented in Figure 2.1. It contains a graph node (solid box) for the classes Room and Booking, a graph edge (solid line) for the association bookings, a label node (rounded box) for the attribute types int and long, as well as a label edge (dashed line) for each attribute; that is, a label edge for the attributes cap, capExam, start, and end, respectively.

An instance model is represented as a symbolic graph, where objects correspond to graph nodes, links to graph edges, attribute slots to label edges, and attribute

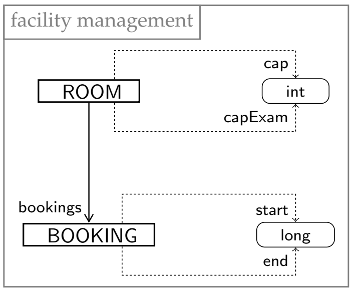


Figure 3.1: The type graph for the CMS metamodel

values to label nodes. However not every symbolic graph is an instance model. Recall, that an instance model represents a snapshot of a system at a specific time (see Section 2.1.1). Hence, every attribute slot of an object has to point to a concrete value, which leads us to the notion of grounded symbolic graphs.

Basically, a *grounded* symbolic graph over a Σ-structure with domain $|\mathcal{D}|$ may be considered as a symbolic graph containing a variable x_v for representing every value $v \in |\mathcal{D}|$.

Definition 3.68 (Typed grounded symbolic graphs [OL10b])**.**
Given the category $\mathbb{TSG}_{\mathcal{D},TG}$ over a Σ-structure $\mathcal{D}$ and symbolic type graph TG^{Φ}, a typed symbolic graph $\underline{G}^{\Phi} = \langle \underline{G}, \underline{\Phi} \rangle$ is *grounded* iff

- X_G includes a variable x_v for each value $v \in |\mathcal{D}|$
- for every assignment ζ such that $(\mathcal{D}, \zeta) \models \underline{\Phi}$, we have $\zeta(x_v) = v$, for each variable $x_v \in X_G$

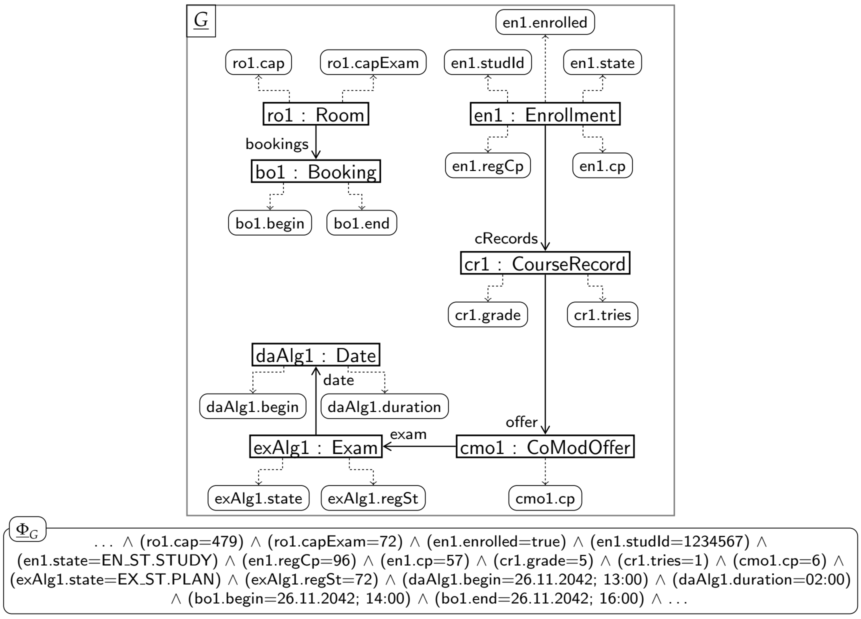

$\underline{G}$
en1.enrolled
ro1.cap
ro1.capExam
en1.studId
en1.state
ro1 : Room
en1 : Enrollment
bookings
bo1 : Booking
en1.regCp
en1.cp
bo1.begin
bo1.end
cRecords
cr1 : CourseRecord
cr1.grade
cr1.tries
daAlg1 : Date
date
daAlg1.begin
daAlg1.duration
offer
exam
exAlg1 : Exam
cmo1 : CoModOffer
exAlg1.state
exAlg1.regSt
cmo1.cp
$\underline{\Phi}_G$
. . . ∧ (ro1.cap=479) ∧ (ro1.capExam=72) ∧ (en1.enrolled=true) ∧ (en1.studId=1234567) ∧
(en1.state=EN_ST.STUDY) ∧ (en1.regCp=96) ∧ (en1.cp=57) ∧ (cr1.grade=5) ∧ (cr1.tries=1) ∧ (cmo1.cp=6) ∧
(exAlg1.state=EX_ST.PLAN) ∧ (exAlg1.regSt=72) ∧ (daAlg1.begin=26.11.2042; 13:00) ∧ (daAlg1.duration=02:00)
∧ (bo1.begin=26.11.2042; 14:00) ∧ (bo1.end=26.11.2042; 16:00) ∧ . . .

Figure 3.2: A typed grounded symbolic graph representing a model of the CMS

Next, we present an example of a typed grounded symbolic graph.

Example 3.69 (Grounded symbolic graphs as a models)**.**
Figure 3.2 depicts a grounded symbolic graph, typed by the CMS type graph. Note that Figure 3.2 displays only a section of the CMS system, which is assumed to comprise hundreds of rooms and exams, and thousands of enrollments. The graph nodes are depicted by solid boxes labeled with an node identifier followed by a colon and the type identifier (i. e., the identifier of the corresponding element in the CMS type graph). The graph edges are denoted by solid arrows. Graph edges are tagged with the identifier of the corresponding type identifier in the CMS type graph. Label edges are denoted by dashed

arrows. The rounded boxes represent label nodes. To improve readability, label nodes are tagged by the identifier of the corresponding object and the attribute identifier (i. e., the type identifier of the label edge) separated by a period. However, note that in general, these labels can be chosen arbitrarily. We do not explicitly denote the type of the label nodes as it is uniquely determined by the type of the attribute. The corresponding Σ-formula is shown on the bottom of Figure 3.2. As stated in Definition 3.68, a grounded symbolic graph has a label node for each value in the domain. As it is impossible to display all of them, we depict only those label nodes and only the formula parts which are of interest. Similar to the previous chapter we denote dates in the DD.MM.YYYY; hh:mm and times the 24 h clock format hh:mm;

One may assume that is sufficient to include label nodes for only a subset of $|\mathcal{D}|$. However, this is not the case. The problem is, that by definition, *symbolic direct transformations cannot add label nodes;* thus, all required values have to be present in a symbolic graph before applying a symbolic production.

Definition 3.70 (Construction of typed symbolic direct transformations)**.**
Given a typed symbolic production $p = (\langle L, \Phi_L \rangle \xleftarrow{l} \langle K, \Phi_K \rangle \xrightarrow{r} \langle R, \Phi_R \rangle)$ and a match $m : \langle L, \Phi_L \rangle \rightarrow \langle G, \Phi_G \rangle$. Production p is applied to $\langle G, \Phi_G \rangle$ at match m as follows:

- *Construct D as the pushout complement of l and m:* Delete those nodes and edges in G that are in the match (i. e., $m(L)$) but keep the image of K (i. e., $m(l(K))$); that is, $D = (G \backslash m(L) \cup m(l(K)))$. The pushout complement exists if D does not contain dangling edges, i. e., edges whose source or target node was deleted.

- *Construct H as the pushout of r and k:* Add to D those nodes and edges that are in R but not in L; that is, $H = D \;\dot{\cup}\; (R \backslash r(K))$.

$$\begin{tikzcd}
\langle L, \Phi_K \rangle \arrow[d, "m"] \arrow[dr, phantom, "(1)"] & \langle K, \Phi_L \rangle \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dr, phantom, "(2)"] & \langle R, \Phi_R \rangle \arrow[d, "n"] \\
\langle G, \Phi_G \rangle & \langle D, \Phi_D \rangle \arrow[l, "g"] \arrow[r, "h"'] & \langle H, \Phi_H \rangle
\end{tikzcd}$$

As l and r are $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$-morphisms and $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$-morphisms are closed under pushouts we know that also $g, h \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$. Hence, $\mathcal{D} \models \Phi_G[\hat{g}] \Leftrightarrow \Phi_D \Leftrightarrow \Phi_H[\hat{h}]$. Moreover, as g and h are bijective for label nodes and we may assume that $E^X_D = E^X_G = E^X_H$ as well as $\Phi_D = \Phi_G$ and $\Phi_H = \Phi_G$ (and consequently $\Phi_D = \Phi_H$). As a direct consequence, if Φ_G is grounded then also Φ_D and Φ_C.

Example 3.71 (Direct transformation via a symbolic production)**.**
Figure 3.3 shows an example for a direct transformation. The symbolic version symbBookRoom of production bookRoom originally presented in Chapter 2

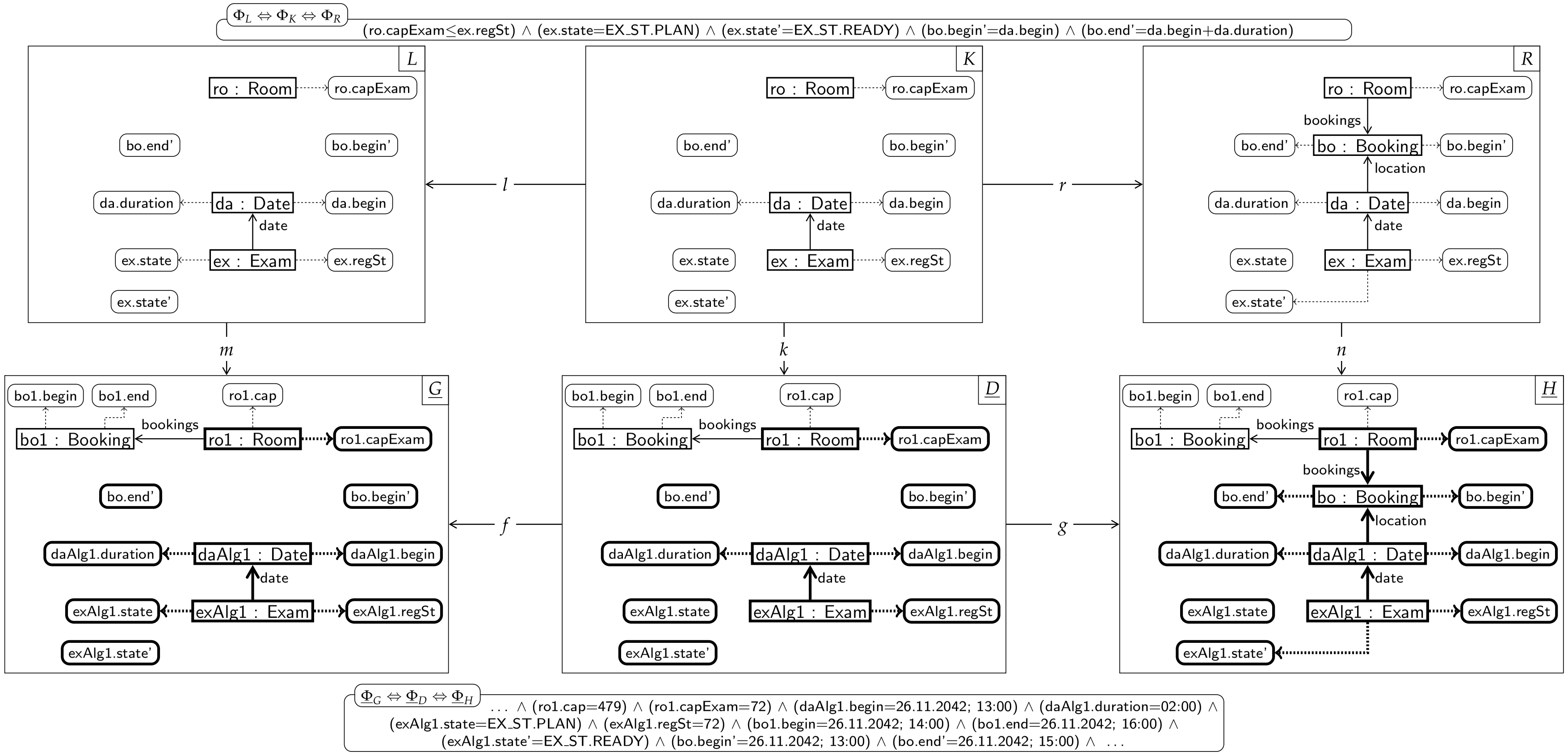

$\Phi_L \Leftrightarrow \Phi_K \Leftrightarrow \Phi_R$
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧ (ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧ (bo.end'=da.begin+da.duration)
L
K
R
l
r
m
k
n
f
g
G
D
H
ro : Room
ro.capExam
bo.end'
bo.begin'
bookings
bo : Booking
location
da.duration
da : Date
da.begin
date
ex.state
ex : Exam
ex.regSt
ex.state'
bo1.begin
bo1.end
ro1.cap
bo1 : Booking
ro1 : Room
ro1.capExam
daAlg1.duration
daAlg1 : Date
daAlg1.begin
exAlg1.state
exAlg1 : Exam
exAlg1.regSt
exAlg1.state'
$\underline{\Phi_G} \Leftrightarrow \underline{\Phi_D} \Leftrightarrow \underline{\Phi_H}$
... ∧ (ro1.cap=479) ∧ (ro1.capExam=72) ∧ (daAlg1.begin=26.11.2042; 13:00) ∧ (daAlg1.duration=02:00) ∧
(exAlg1.state=EX_ST.PLAN) ∧ (exAlg1.regSt=72) ∧ (bo1.begin=26.11.2042; 14:00) ∧ (bo1.end=26.11.2042; 16:00) ∧
(exAlg1.state'=EX_ST.READY) ∧ (bo.begin'=26.11.2042; 13:00) ∧ (bo.end'=26.11.2042; 15:00) ∧ ...

Figure 3.3: The application of symbolic production symbBookRoom to an cutout of the grounded symbolic graph shown in Example 3.69

is shown on top of Figure 3.3. To improve readability, we use the same labeling scheme for variables (label nodes) as introduced in Chapter 2; that is, nonprimed variables denote the attribute values before and primed variables represent the newattribute values after applying the production, although they appear on the left and right-hand side in case of symbolic productions. The production is applied to an excerpt of the grounded symbolic graph $\langle \underline{G}, \underline{\Phi}_G \rangle$ (shown Figure 3.2). To improve readability we have left out the parts irrelevant for applying production symbBookRoom The production is applied by first finding a match m for the left hand side $\langle L, \Phi_L \rangle$ in $\langle \underline{G}, \underline{\Phi}_G \rangle$. The actual match for the E-graph component L in G is drawn bold; that is, m maps graph nodes ro to ro1, da to daAlg1, and ex to exAlg1. The mapping of the label nodes is given by the variable map $\hat{m}$:

$$\frac{\begin{array}{ccccc}\text{ro1.capExam} & \text{bo.begin'} & \text{bo.end'} & \text{daAlg1.begin} & \text{daAlg1.duration}\end{array}}{\begin{array}{ccccc}\text{ro.capExam} & \text{bo.begin'} & \text{bo.end'} & \text{da.begin} & \text{da.duration}\end{array}} \dots$$

$$\dots \frac{\begin{array}{ccc}\text{exAlg1.state} & \text{exAlg1.regSt} & \text{exAlg1.state'}\end{array}}{\begin{array}{ccc}\text{ex.state} & \text{ex.regSt} & \text{ex.state'}\end{array}}$$

Note, as $\langle \underline{G}, \underline{\Phi}_G \rangle$ is a grounded symbolic graph, we can assume that $\langle \underline{G}, \underline{\Phi}_G \rangle$ contains label nodes bo.begin' and bo.end', and exAlg1.state' with the corresponding equality predicates (as shown in Figure 3.3).

To verify that m is a symbolic graph morphism we have to show that

$$\mathcal{D} \models \underline{\Phi}_G \Rightarrow \Phi_L[\hat{m}].$$

To this end, we rewrite $\underline{\Phi}_G$ and $\Phi_L[\hat{m}]$ as shown in the following table:

$\underline{\Phi}_G$		$\Phi_L[\hat{m}]$
. . . ∧ (ro1.capExam=72) ∧ (daAlg1.begin=26.11.2015; 13:00) ∧ (daAlg1.duration=02:00) ∧ (exAlg1.state=EX_ST.PLAN) ∧ (exAlg1.state'=EX_ST.READY) ∧ (exAlg1.regSt=72) ∧ (bo.begin'=26.11.2015; 13:00) ∧ (bo.end'=26.11.2015;15:00) ∧ . . .	$\Rightarrow$	(ro1.capExam≤ exAlg1.regSt) (exAlg1.state=EX_ST.PLAN) (exAlg1.state'=EX_ST.READY) ∧ (bo.begin'=daAlg1.begin) ∧ (bo.end'=daAlg1.begin+daAlg1.duration) ∧

As:

$$\begin{array}{c}\text{(ro1.capExam=72)} \land \\ \text{(exAlg1.regSt=72)}\end{array} \Rightarrow \text{(ro1.capExam}\leq\text{exAlg1.regSt),}$$

$$\begin{array}{c}\text{(exAlg1.state=EX\_ST.PLAN)} \land \\ \text{(exAlg1.state'=EX\_ST.READY)}\end{array} \Rightarrow \begin{array}{c}\text{(exAlg1.state=EX\_ST.PLAN)} \land \\ \text{(exAlg1.state'=EX\_ST.READY),}\end{array}$$

$$\begin{array}{c}(\text{bo.begin'=26.11.2015; 13:00}) \wedge \\ (\text{daAlg1.begin=26.11.2015; 13:00})\end{array} \Rightarrow (\text{bo.begin'=daAlg1.begin}), \text{ and}$$

$$\begin{array}{c}(\text{bo.end'=26.11.2015;15:00}) \wedge \\ (\text{daAlg1.begin=26.11.2015; 13:00}) \wedge \\ (\text{daAlg1.duration=02:00})\end{array} \Rightarrow (\text{bo.end'=daAlg1.begin+daAlg1.duration}),$$

the implication $\mathcal{D} \models \underline{\Phi}_G \Rightarrow \Phi_L[\hat{m}]$ is valid. Recall that although dates and times are denoted in the DD.MM.YYYY;hh:mm and hh:mm format, they are internally represented as values of type long. Hence, arithmetic on dates and times is, in fact, arithmetic on long values.

The production is applied by first constructing $\langle \underline{D}, \underline{\Phi}_D \rangle$ as the pushout complement of m and l, and afterwards constructing $\langle \underline{H}, \underline{\Phi}_H \rangle$ as the pushout of r and k. Note, as l and r are bijections on label, also f and g are bijection on label nodes (Definition 3.20). Moreover, we have $\mathcal{D} \models \Phi_G \Leftrightarrow \Phi_D \Leftrightarrow \Phi_H$. Consequently, the fact that $\langle \underline{G}, \underline{\Phi}_G \rangle$ is a grounded symbolic graph, implies that also $\langle \underline{D}, \underline{\Phi}_D \rangle$ and $\langle \underline{H}, \underline{\Phi}_H \rangle$ are grounded symbolic graphs.

The resulting grounded symbolic graph $\langle \underline{H}, \underline{\Phi}_H \rangle$ contains a new booking bo for room ro1, for the date daAlg1 of exam exAlg1.

3.5 Open Issues of Symbolic Graph Transformations

Based on the previous overview, we evaluate symbolic graph transformations with respect to their applicability to fulfill the objectives presented in the introduction of this thesis. Accordingly, we discuss the application of symbolic graph transformations regarding (i) their feasibility to finitely represent attributed graphs and transformations, as well as (ii) their ability to be used for consistency constraint verification and conflict analysis. Moreover, based on this overview we further motivate and detail the key contributions presented in the remainder of this thesis.

We begin with discussing the notion of grounded symbolic graphs for representing models. Recall that a grounded symbolic graph contains a label node for each data value in the domain. Hence, in case of unbounded domains this leads to infinite graphs, although the corresponding instance model might be finite. Another aspect is the coding of attribute values as conjunctions of equality predicates that leads to infinite formulas in case of unbounded domains, which are not permitted by the definition of Σ-formulas. Although, the problem might be solved by coding attribute values as sets of Σ-formulas, the conceptual difference between grounded symbolic graphs and the notion of instance models (see Section 2.1.1) still remains.

Now we consider the application of symbolic productions to nongrounded symbolic graphs. As symbolic graph transformation systems are $(\mathcal{M}, \mathcal{N})$-adhesive, it seems reasonable to expect that symbolic productions may also be applied to arbitrary (i. e., nongrounded) symbolic graphs. While from a theoretical point of view this is possible, in practice, the application of symbolic production to nongrounded symbolic graphs does not always behave as expected, which is demonstrated by the following example.

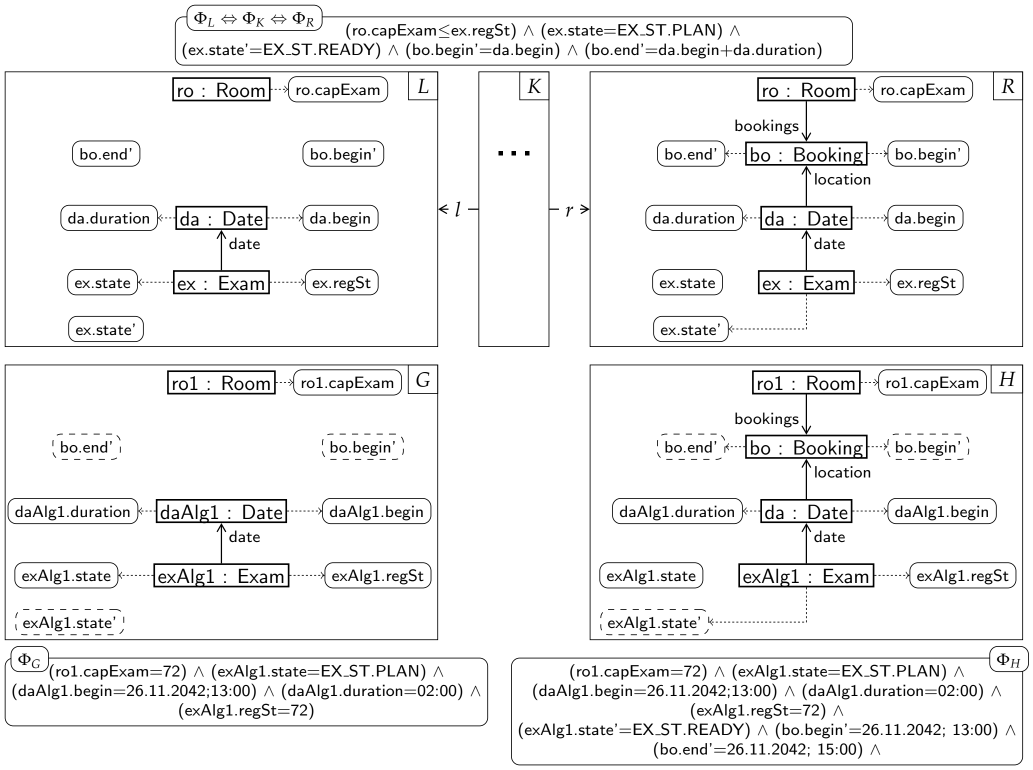

ΦL ⇔ ΦK ⇔ ΦR
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧ (bo.end'=da.begin+da.duration)
L
K
R
l
r
ro : Room
ro.capExam
bo.end'
bo.begin'
da.duration
da : Date
da.begin
date
ex.state
ex : Exam
ex.regSt
ex.state'
bookings
bo : Booking
location
G
H
ro1 : Room
ro1.capExam
daAlg1.duration
daAlg1 : Date
daAlg1.begin
exAlg1.state
exAlg1 : Exam
exAlg1.regSt
exAlg1.state'
ΦG
(ro1.capExam=72) ∧ (exAlg1.state=EX_ST.PLAN) ∧
(daAlg1.begin=26.11.2042;13:00) ∧ (daAlg1.duration=02:00) ∧
(exAlg1.regSt=72)
ΦH
(ro1.capExam=72) ∧ (exAlg1.state=EX_ST.PLAN) ∧
(daAlg1.begin=26.11.2042;13:00) ∧ (daAlg1.duration=02:00) ∧
(exAlg1.regSt=72) ∧
(exAlg1.state'=EX_ST.READY) ∧ (bo.begin'=26.11.2042; 13:00) ∧
(bo.end'=26.11.2042; 15:00) ∧

Figure 3.4: Limitations of symbolic productions

Example 3.72 (Limitations of symbolic productions)**.**
We might expect that it should be possible to apply production symbBookRoom to typed symbolic graph $\langle G, \Phi_G \rangle$ (depicted on the bottom left in Figure 3.4) resulting in the typed symbolic graph $\langle H, \Phi_H \rangle$ (depicted below right in Figure 3.4); However, this is not possible. Lets assume first that the dashed label nodes bo.begin', bo.end', and exAlg1.state' do not exists in $\langle G, \Phi_G \rangle$. Hence, we cannot find a match for the left-hand side graph L in graph G. Notice that, although, the match need not to be injective for label nodes, we may not map label nodes bo.begin' and bo.end' (from L) to any other label node in G as $\Phi_G \Rightarrow \Phi_L[\hat{m}]$ would not be valid.

Now, assume that G is supplied with an unlimited number of label nodes; hence, we may assume that G includes label nodes bo.begin', bo.end', and exAlg1.state'. However, we still cannot apply the production symbbookRoom to $\langle G, \Phi_G \rangle$, because $\Phi_G \Rightarrow \Phi_L[\hat{m}]$ is not valid.

To solve this problem we have to assume that $\langle G, \Phi_G \rangle$ contains a label node for each value in $\mathcal{D}$ with corresponding equality predicate in Φ_G, which means that $\langle G, \Phi_G \rangle$ is a grounded symbolic graph.

As shown in the previous example, to apply a symbolic production, the match has to include also a mapping of the primed label nodes (which represent the new attribute values). Grounded symbolic graphs guarantee the existence of such an appropriate mapping by including a certain label node for each value of a domain. However, in general (i. e., for arbitrary symbolic graphs) this is not the case. Hence, the application of symbolic productions to arbitrary symbolic graphs does not lead to the expected results, i. e., they are just not applicable in most cases. Nevertheless, especially to apply conflict analysis techniques to symbolic graphs (see Chapter 8), it is necessary to apply productions also to nongrounded symbolic graphs.

Orejas et al. approached this issue by introducing lazy graph transformations in [OL10a]. The basic idea of lazy graph transformations is to permit productions to create label nodes and constraints. Consequently, the required label nodes need not to be provisioned beforehand. To this end, lazy graph transformations are based on productions with different classes for left and right production morphisms. Similar to symbolic productions, the left production morphisms are restricted to class $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$. The right production morphisms has to be only a symbolic graph monomorphism (instead of in class $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$ as for symbolic productions). However, by permitting symbolic graph monomorphisms for the right production morphisms, a lazy transformation may affect the values of label nodes that are not part of its match. This problem is illustrated in the following example.

Example 3.73 (Nonlocal effects of lazy graph transformations)**.**
The lazy production shown on top of Figure 3.5 further restricts the value of existing label node x by adding constraint $(\mathsf{x}=5)$. The direct transformation derived by applying this production to a symbolic graph $\langle G, \Phi_G \rangle$ is shown on the bottom of Figure 3.5. In addition to a label node x the graph G contains an other label node y, whose value is given by $(\mathsf{x}=\mathsf{y})$. Applying the production to symbolic graph $\langle G, \Phi_G \rangle$ results in symbolic graph $\langle H, \Phi_H \rangle$ with $\Phi_H \Leftrightarrow (\mathsf{x} = \mathsf{y}) \wedge (\mathsf{x} = 5)$. Hence, in addition to setting te value of x equal to 5, the application implicitly sets also the value of y equal to 5, although label node y was not in the match. In this way, lazy transformations violate the locality property.

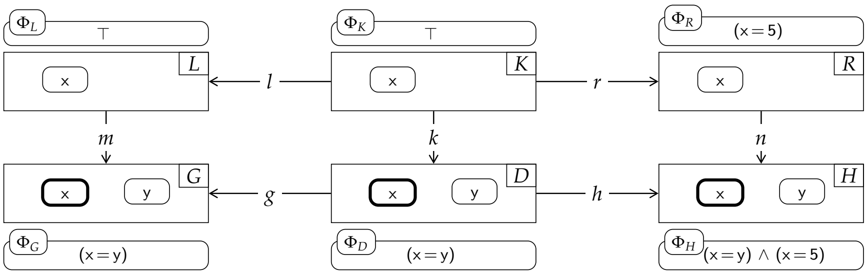


Figure 3.5: Example of a transformation via a lazy production which has nonlocal effects.

The locality property allows for inferring global properties of productions (i. e., they are valid for all graphs) by only considering a finite subset of graphs. In this way, the locality property forms the basis of almost all analysis techniques for transformation systems, including the results for constraint verification and conflict analysis techniques. As a consequence, constraint verification and conflict analysis techniques cannot be applied to lazy transformations.

To sum up, in order to use symbolic graphs as a basis to achieve our objectives, we need a transformation approach that (i) shows the desired behaviour when applied to nongrounded graphs and (ii) provides the properties required for the constraint verification and conflict analysis techniques at the same time. To this end we propose *projective graph transformations* in the next chapter. Basically, projective graph transformations can be considered as a compromise between the nice formal properties of symbolic graph transformations and the expressive power of lazy transformations. Projective graph transformations follow the idea of lazy graph transformations; that is, projective productions have different classes for left and right production morphism. However, to ensure that projective transformations have only local effects, we require the right production morphisms to be in the class of projection morphisms. The class of projection morphisms and the concept of projective graph transformations is introduced in Chapter 4. Moreover, we show that projective graph transformations provide the desired behaviour when applied to nongrounded graphs. In order to show that projective graph transformations provide the required formal properties, we first introduce $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems in Chapter 5. Basically, $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems are a generalization of $(\mathcal{M}, \mathcal{N})$-adhesive transformation systems in order to cope with productions that require different classes for left and right production morphisms. In Chapter 6, we show that projective graph transformation systems fit into the framework of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems. Subsequently, we show in Chapter 7 and Chapter 8 that constraint verification and conflict analysis techniques lead to the expected results when applied to projective graph transformation systems.

4

PROJECTIVE GRAPH TRANSFORMATIONS

In this chapter we present projective graph transformation systems as an extension of symbolic graph transformations. In contrast to symbolic productions, projective productions can create label nodes and constraints. Hence, projective productions are appropriate to transform also nongrounded symbolic graphs, as the required label nodes can be created on the fly.

We begin with introducing the class of projection morphisms in Section 4.1. Based on this morphism class we define in Section 4.2 the notion of projective graph transformation systems. In Section 4.3 we discuss the application of projective graph transformations in the context of model transformations and show that projective graph transformations solves the shortcomings of symbolic graph transformations in this context.

In the following, we present all constructions directly for typed symbolic graphs. To this end, we assume for the rest of this chapter that category $\mathbb{TSG}_{\mathcal{D},TG}$ is given by a symbolic type graph TG^{Φ} and a Σ-structure $\mathcal{D}$.

4.1 Projection Morphisms

In this section we introduce the new class $\mathcal{M}^{inj}_{Proj,TG}$, of typed projection morphisms. A projection morphism $a : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$ is a symbolic graph monomorphism such that Φ_A is a projection of Φ_B. Intuitively, Φ_A is a projection of Φ_B, if Φ_A is equivalent to $\exists x_1 \ldots \exists x_n . \Phi_B$, where $\{x_1, \ldots, x_n\} = X_B \backslash a_X(X_A)$ is the set consisting of all variables that are in X_B but not in X_A. In such a way any solution for Φ_A can be extended to a solution of Φ_B.

In the following, we give a more general definition of projection morphisms, whereas $\mathcal{M}^{inj}_{\Rightarrow,TG}$ is the class of typed symbolic monomorphisms.

Definition 4.1 (Typed projection morphism)**.**
An $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $a : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$ is *a typed projection morphism*, i. e., $a \in \mathcal{M}^{inj}_{Proj,TG}$, if and only if for any typed symbolic graph $\langle Z, \Phi_Z \rangle$ with typed E-graph morphisms $z : Z \to B$ and $z' : Z \to A$ such that $z = a \circ z'$, the following statement is true:

$$\mathcal{D} \models \Phi_B \Rightarrow \Phi_Z[\hat{z}] \text{ iff } \mathcal{D} \models \Phi_A \Rightarrow \Phi_Z[\hat{z}'].$$

$$\begin{tikzcd}
& \langle Z, \Phi_Z \rangle \arrow[dl, "z'"'] \arrow[dr, "z"] & \\
\langle A, \Phi_A \rangle \arrow[rr, "a"] & & \langle B, \Phi_B \rangle
\end{tikzcd}$$

In other words, z is a typed symbolic graph morphism if and only if z' is a typed symbolic graph morphism.

We refer to this property in the following as the *projection property*. Moreover, we shall see in the next sections that projection property is very useful to derive further facts for projective graph transformations.

Up until now, we provided only an intuition for the construction of projection morphisms by existential quantification. In the following, we show that the projection of a symbolic graph $\langle B, \Phi_B \rangle$ to a given subgraph A of B can be constructed as a pullback.

Definition 4.2 (Construction of projections).
Given a typed symbolic graph $\langle B, \Phi_B \rangle$ and a typed E-graph A, both typed over the same type graph TG. For an E-graph monomorphism $a' : A \to B$, the projection morphism $a : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$ is constructed as the *projection $Proj(\langle B, \Phi_B \rangle, a')$* of $\langle B, \Phi_B \rangle$ to A via a', which is defined as follows: First construct symbolic graph $\langle B, \perp \rangle$ with typed symbolic graph morphism $i : \langle B, \Phi_B \rangle \to \langle B, \perp \rangle$ given by $i = id_B$, where id_B is the identity morphisms of B in the category $\mathbb{TEG}_{TG}$. In a second step, symbolic graph $\langle A, \perp \rangle$ is obtained from typed E-graph A and $\perp$. Note that morphism $i : \langle B, \Phi_B \rangle \to \langle B, \perp \rangle$ and $a' : \langle A, \perp \rangle \to \langle B, \perp \rangle$ are symbolic, as $\mathcal{D} \models \perp \Rightarrow \Phi_B$ and $\mathcal{D} \models \perp \Rightarrow \perp[\hat{a}']$ hold in any Σ-structure $\mathcal{D}$.

$$\begin{tikzcd}
\langle A, \perp \rangle \arrow[r, "a'"] & \langle B, \perp \rangle \\
\langle A, \Phi_A \rangle \arrow[u, "i'"] \arrow[r, "a"] \arrow[ur, phantom, "(1)"] & \langle B, \Phi_B \rangle \arrow[u, "i"']
\end{tikzcd}$$

Finally, the projection morphism $a : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$ is obtained by the pullback (1) of a' and i.

Remark 4.3 (Properties of projection construction via pullbacks).
Note that pullback (1) in the previous definition is also a pushout in $\mathbb{TEG}_{TG}$, as morphisms $i = id_B$ and $i' = id_A$ are given by the identities in $\mathbb{TEG}_{TG}$, and every commuting square along identities is a pushout (Fact 3.14.c). Moreover, pullback (1) is also a pushout in $\mathbb{TSG}_{\mathcal{D},TG}$, as $\mathcal{D} \models \perp \Leftrightarrow (\perp \wedge \Phi_B)$ is valid in any Σ-structure $\mathcal{D}$; hence, the Σ-formula $\perp$ of the pushout object $\langle B, \perp \rangle$ is always equivalent to the conjunction of the Σ-formulas of $\langle A, \perp \rangle$ and $\langle B, \Phi_B \rangle$.

Note, as (1) is a pullback and a' as well as i are monomorphisms, Φ_A is equivalent to $\exists b_1 \ldots \exists b_n . \Phi_B \vee \perp$, which simplifies to $\exists b_1 \ldots \exists b_n . \Phi_B$, where $\{b_1, \ldots, b_n\} = X_B \backslash a_X(X_A)$.

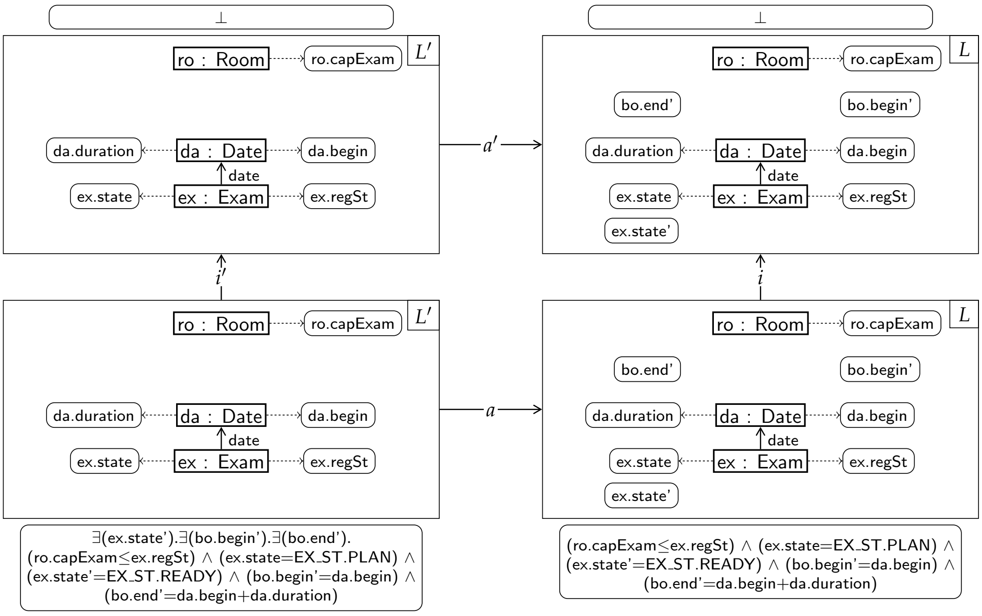

⊥
⊥
ro : Room
ro.capExam
L′
da.duration
da : Date
da.begin
date
ex.state
ex : Exam
ex.regSt
a′
L
bo.end'
bo.begin'
ex.state'
i′
i
a
∃(ex.state').∃(bo.begin').∃(bo.end').
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧
(bo.end'=da.begin+da.duration)
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧
(bo.end'=da.begin+da.duration)

Figure 4.1: Construction of Projections

Example 4.4 (Construction of projections)**.**
Figure 4.1 shows the construction of a projection morphism according to Definition 4.2, where $\langle L, \Phi_L \rangle$ corresponds to the left-hand side of production symbBookRoom presented in Example 3.71. The idea is to use projection to remove all label nodes that are not assigned to a graph node. To this end, we derive L' from L by removing all label nodes not assigned to a graph node. Hence, there is an E-graph monomorphism $a' : L' \to L$ which is given by the inclusion of L' in L. Then we construct the graphs $\langle L, \perp \rangle$ and $\langle L', \perp \rangle$ with symbolic graph morphism i. Finally, we obtain projection morphism $a : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$ by the pullback of a' and i. Note, as both a' and i are injective, we can construct L' according to Remark 3.59. Hence, Φ'_L is equivalent to $\exists(\mathsf{ex.state'}).\exists(\mathsf{bo.begin'}).\exists(\mathsf{bo.end'}).\Phi_L$.

In the following lemma we show that any morphism obtained according to Definition 4.2 is a projection morphism, and every projection morphism can be obtained according to Definition 4.2.

Lemma 4.5 (Construction of projection morphisms)**.**
Let $\mathbb{TSG}_{\mathcal{D},TG}$ be the category of typed symbolic graphs, every morphism $a : \langle A', \Phi'_A \rangle \to \langle A, \Phi_A \rangle$ is in $\mathcal{M}^{inj}_{Proj,TG}$ if and only if it is constructed according to Definition 4.2.

Proof.
If. Given projection morphism $a : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$, we have to show that the commuting diagram (1) (shown below) is a pullback in $\mathbb{TSG}_{\mathcal{D},TG}$. This is shown by verifying the universal pullback property of (1); that is, for any typed symbolic graph $\langle Z, \Phi_Z \rangle$ and typed symbolic graph morphisms $z : \langle Z, \Phi_Z \rangle \to \langle B, \Phi_B \rangle$ and

$z'' : \langle Z, \Phi_Z \rangle \to \langle A, \bot \rangle$ with $i \circ z = a' \circ z''$, there is a unique typed symbolic graph morphism $z' : \langle Z, \Phi_Z \rangle \to \langle A, \Phi_A \rangle$ with $z'' = i' \circ z'$ and $z = a \circ z'$.

As i' is given uniquely by $i' = id_A$ in $\mathbb{TEG}_{TG}$, we may obtain a unique typed E-graph morphism $z' : Z \to A$ as $z' = z''$. Thus we have $z'' = i' \circ z'$. As the E-graph components of a' and a are the same, we also have $z = a \circ z'$. By assumption, z is a typed symbolic graph morphism and a a typed projection morphism; hence, we can conclude from the projection property of a that z' is also a typed symbolic graph morphism. As z' is unique in $\mathbb{TEG}_{TG}$ it is also unique in $\mathbb{TSG}_{\mathcal{D},TG}$ (up to isomorphism).

$$\begin{tikzcd}
 & \langle A, \bot \rangle \arrow[r, "a'"] & \langle B, \bot \rangle \\
 & \langle A, \Phi_A \rangle \arrow[u, "i'"] \arrow[r, "a"] \arrow[ur, phantom, "(1)"] & \langle B, \Phi_B \rangle \arrow[u, "i"'] \\
\langle Z, \Phi_Z \rangle \arrow[uur, "z''"] \arrow[ur, "z'"] \arrow[urr, "z"'] & &
\end{tikzcd}$$

Only if. Given pullback (1) constructed according to Definition 4.2, we have to verify the projection property of $a : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$; that is, for any typed E-graph morphisms $z : Z \to B$ and $z' : Z \to A$ with $z = a \circ z'$, we have to show that (i) if z' is a typed symbolic graph morphism, then also z; and (ii) if z is a typed symbolic graph morphism, then also z'.

(i). As z' and a are typed symbolic graph morphisms, then also its composition $z = a \circ z'$.

(ii). First we construct typed E-graph morphism $z'' : Z \to A$ as $z'' = i' \circ z'$. Morphism z'' is also a typed symbolic graph morphisms as $\mathcal{D} \models \bot \Rightarrow \Phi_Z[\hat{z}'']$ holds in any Σ-structure $\mathcal{D}$. From the commutativity of (1) we know $a' \circ i' = i \circ a$. Hence, from $z'' = i' \circ z'$ and $z = a \circ z'$ we obtain

$$a' \circ z'' = a' \circ i' \circ z' = i \circ a \circ z' = i \circ z.$$

Consequently, we may use the universal property of pullback (1) with typed symbolic graph morphisms z'' and z to construct unique $x : \langle Z, \Phi_Z \rangle \to \langle A, \Phi_A \rangle$. To show that z' is a typed symbolic graph morphism, we have to show that $x = z'$. From the universal property of pullback (1) we know that $z = a \circ x$ and $z'' = i' \circ x$. By construction we know that $z = a \circ z'$ and $z'' = i' \circ z'$. As x is unique it must hold that $x = z'$. Consequently, z' is a typed symbolic graph morphism. □

4.2 Projective Graph Transformation Systems

Based on the class of projection morphism we define projective productions and projective graph transformation systems in the following.

One may wonder whether $\mathbb{TSG}_{\mathcal{D},TG}$ with morphism classes $\mathcal{M} = \mathcal{M}^{inj}_{Proj,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$ is $(\mathcal{M}, \mathcal{N})$-adhesive, which would imply that it is sufficient to choose $\mathcal{M}^{inj}_{Proj,TG}$ for the left and right production morphisms. Unfortunately, this is not the case as shown by the following example.

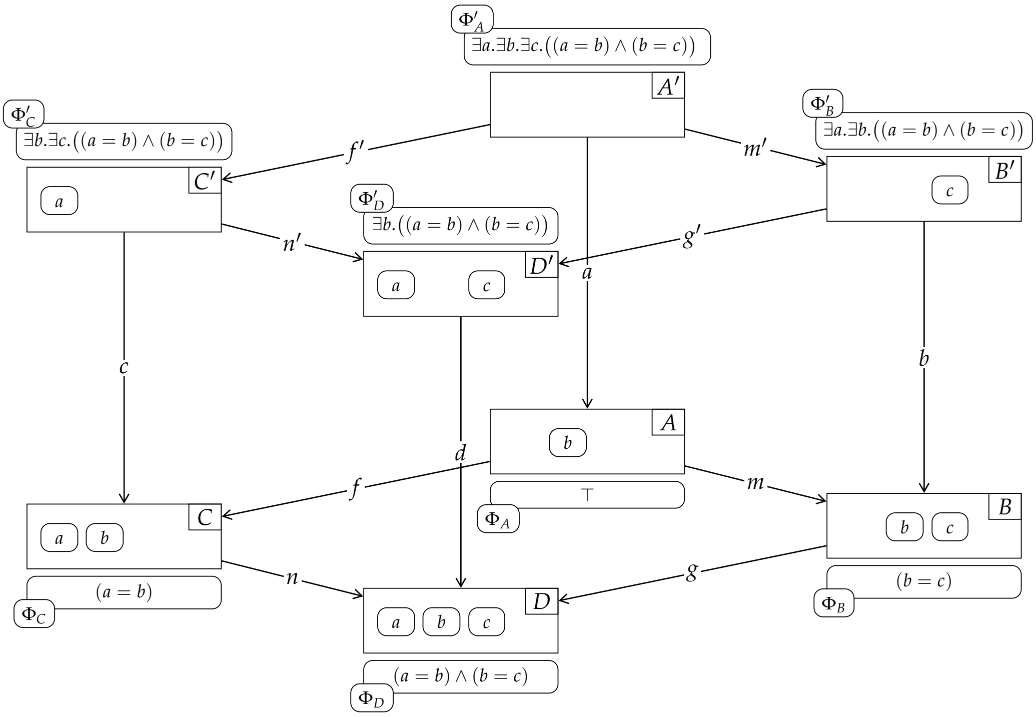


Figure 4.2: Category $\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{M} = \mathcal{M}^{inj}_{Proj,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$ is not $(\mathcal{M}, \mathcal{N})$-adhesive

Example 4.6 (The category $\mathbb{TSG}_{\mathcal{D},TG}$ with morphism classes $\mathcal{M} = \mathcal{M}^{inj}_{Proj,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$ is not $(\mathcal{M}, \mathcal{N})$-adhesive)**.**
Consider the commutative cube shown in Figure 4.2. It can be checked that the bottom face is a pushout with $m \in \mathcal{M}$, the front and back faces are pullbacks in $\mathbb{TSG}_{\mathcal{D},TG}$, and $c, d, b \in \mathcal{M}^{inj}_{Proj,TG}$, $f \in \mathcal{M}^{inj}_{\Rightarrow,TG}$. Unfortunately, the commutative cube is not a VK-square, as the top face is not a pushout. The problem is that the formula

$$\exists a.\exists b.\Phi_B \land \exists b.\exists c.\Phi_C,$$

obtained by first projecting $\langle B, \Phi_B \rangle$ to B' as well as $\langle C, \Phi_C \rangle$ to C' and then constructing the pushout of $\langle B', \Phi'_B \rangle$ and $\langle C', \Phi'_C \rangle$, is not equivalent to the formula

$$\exists b.(\Phi_B \land \Phi_C),$$

obtained by first constructing $\langle D, \Phi_D \rangle$ as the pushout of $\langle B, \Phi_B \rangle$ and $\langle C, \Phi_C \rangle$ and then projecting $\langle D, \Phi_D \rangle$ to D'.

The problem shown in the previous example arises from the fact that D' possibly does not contain the label nodes of D to which B and C are glued together. The problem can be avoided by requiring that morphism a (in the previous example) is a bijection on label nodes (i. e., $a \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$), which enforces that B' and C' are glued along the same label nodes as B and C. *This observation gives rise to the definition of typed projective productions and transformation systems with a certain class for left and right production morphisms, respectively. Moreover, we shall see in the next chapter that* *transformation systems with a certain class for left, right, and match morphisms can be generalized to the concept of* $(\mathcal{L}, \mathcal{R}, \mathcal{N})$*-adhesive transformation systems.*

Definition 4.7 (Typed projective productions and typed projective graph transformation systems (**TPGTS**))**.**
A typed projective production $p = (\langle L, \Phi_L\rangle \xleftarrow{l} \langle K, \Phi_K\rangle \xrightarrow{r} \langle R, \Phi_R\rangle)$ consists of a *left production morphism* $l : \langle L, \Phi_L\rangle \to \langle K, \Phi_K\rangle$ of class $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and *a right production morphism* $r : \langle R, \Phi_R\rangle \to \langle K, \Phi_K\rangle$ of class $\mathcal{M}^{inj}_{Proj,TG}$.

A *typed projective graph transformation system* **TPGTS**, is given by a finite set of typed projective productions **P** and a the class $\mathcal{M}^{inj}_{\Rightarrow,TG}$ for match morphisms.

Note that the choice $r \in \mathcal{M}^{inj}_{Proj,TG}$ is still sufficient to create label nodes on the fly. Hence, it is possible to apply projective productions also to nongrounded graphs. Moreover, the choice $r \in \mathcal{M}^{inj}_{Proj,TG}$ guarantees that a projective production may only put additional constraints on the values of created label nodes. For this reason, projective graph transformations may not change the values of label nodes that are not part of the match. Consequently, projective graph transformations have only local effects, which is one of the fundamental requirements to prove the results required for constraint verification and conflict analysis techniques.

Before we actually show that these fundamental properties indeed apply for projective graph transformations, we first show that projective graph transformations address the main deficiencies of symbolic graph transformation systems.

4.3 Model Transformation by Projective Graph Transformation

Similar as done for symbolic graph transformations we show how the concepts of instance models and transformations can be represented in the context of projective graph transformations. Moreover, we show that the shortcomings of symbolic graph transformations (explained in Section 3.5) can be avoided by projective graph transformations.

We begin with an alternative notion for instance models. As mentioned before, by projective graph transformations it is possible to create label nodes and constraints on the fly. Hence, it is possible to define useful productions for the transformation of symbolic graphs that *do not* include label nodes for all domain values. This leads us to the notion of definite symbolic graphs to represent instance models. The idea of definite symbolic graphs is similar to that of grounded symbolic graph (that is, we assign to each variable a definite value); however, in contrast to grounded symbolic graphs, we do not have to include a label node for each domain value.

Definition 4.8 (Typed definite symbolic graphs)**.**
Given the category $\mathbb{TSG}_{\mathcal{D},TG}$ over a Σ-structure $\mathcal{D}$ and symbolic type graph TG^{Φ}, a typed symbolic graph $\underline{G}^{\Phi} = \langle \underline{G}, \underline{\Phi}\rangle$ is a *definite symbolic graph* iff:

- For all assignments ζ_1 and ζ_2 such that $(\mathcal{D}, \zeta_1) \models \underline{\Phi}$ and $(\mathcal{D}, \zeta_2) \models \underline{\Phi}$, we have $\zeta_1(x) = \zeta_2(x)$ for each label node $x \in X_G$.

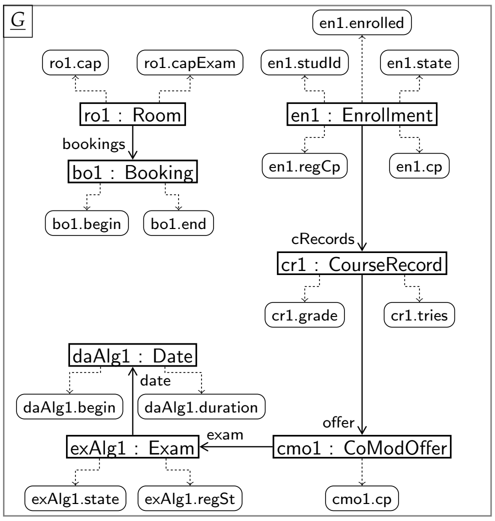
- $\underline{G}$ is *linear*; that is, for each label node $x \in X_G$, there is at most one label edge e such that $x = t^X_G(e)$.

Notice that we require $\underline{G}$ to be linear, in order to compensate that match morphisms have to be injective for label nodes for projective transformations. However, this is not really a restriction as every symbolic graph $\langle G, \Phi_G \rangle$ can be transformed to a linear symbolic graph. This is achieved by repeatedly creating for each pair of label edges e_1 and e_2 with same the target node x (i. e., $t^X_G(e_1) = x = t^X_G(e_2)$) a new label node y as well as conjuncting Φ_G with $(x = y)$, and redirecting e_2 to y (i. e., set $t^X_G(e_2) = y$).

The idea of definite symbolic graphs is similar to that of a solution for Σ-formulas. According to Fact 3.44, any solution ζ of a Σ-formula Φ in a Σ-structure $\mathcal{D}$ (i. e., $(\mathcal{D}, \zeta) \models \Phi$) can be represented as a (finite) conjunction of equality predicates $\underline{\Phi}$ such that $(\mathcal{D}, \zeta) \models \underline{\Phi}$. Hence, given a symbolic graph $\langle G, \Phi_G \rangle$ with a finite set of label nodes and a solution ζ of Φ_G, we can construct a definite symbolic graph, just by replacing Φ_G with $\underline{\Phi}_G$. Moreover, there exists a symbolic graph morphism $s : \langle G, \Phi_G \rangle \to \langle G, \underline{\Phi}_G \rangle$ (given as id_G in E-graphs), as $\mathcal{D} \models \underline{\Phi}_G \Rightarrow \Phi_G$ (according to Fact 3.44). Consequently, any finite instance model can be represented by a finite definite symbolic graph also for unbounded domains. This representation of instance model works perfectly for projective transformations as, in contrast to symbolic transformations, projective transformations can create the required label nodes on the fly, including the corresponding constraints for representing new attribute values.

Example 4.9 (Definite symbolic graphs as models)**.**
Figure 4.3 depicts the definite symbolic graph for the instance model model originally presented in Figure 2.2. At a first glance, the definite symbolic graph looks similar to the grounded symbolic graph shown in Figure 3.2. The difference is that the definite symbolic graph does not contain any label node and equality predicate in addition to those depicted in Figure 4.3, in contrast to the grounded symbolic graph of Figure 3.2, which was assumed to contain an infinite number of label nodes.

In order to compare projective graph transformations with symbolic graph transformations, recall that every $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$-morphisms is also an $\mathcal{M}^{inj}_{Proj,TG}$-morphism. Hence, every symbolic production is also a projective production. This means, on the one hand, that projective graph transformation systems can be considered as a generalization of symbolic graph transformation systems. However, on the other hand, we might run into the same problems as we have with symbolic graph transformations. As discussed in Section 3.5 one of the limitations of symbolic graph transformation is caused by the fact that matches for the left-hand sides have to be defined also for those label nodes that are not assigned to a graph node. Hence, from a practical point of view it seems reasonable to consider only projective productions whose left-hand sides contain no unassigned label nodes, which are called *auxiliary variables* in the following.



$\underline{\Phi_G}$

(ro1.cap=479) ∧ (ro1.capExam=72) ∧ (en1.enrolled=true) ∧ (en1.studId=1234567) ∧
(en1.state=EN_ST.STUDY) ∧ (en1.regCp=96) ∧ (en1.cp=57) ∧ (cr1.grade=5) ∧ (cr1.tries=1) ∧ (cmo1.cp=6) ∧
(exAlg1.state=EX_ST.PLAN) ∧ (exAlg1.regSt=72) ∧ (daAlg1.begin=26.11.2042; 13:00) ∧ (daAlg1.duration=02:00)
∧ (bo1.begin=26.11.2042; 14:00) ∧ (bo1.end=26.11.2042; 16:00)

Figure 4.3: The typed definite symbolic graph for the instance model originally shown in Figure 2.2

Definition 4.10 (Auxiliary variables and normal form).
Given a (typed) symbolic graph $\langle L, \Phi_L \rangle$, the set of auxiliary variables is defined as the set

$$aux(L) = \{x \in X_L \,|\, \text{not exists } e \in E_L^X \text{ s. t. } x = t_L^X(e)\}$$

A symbolic graph $\langle L, \Phi_L \rangle$ is in *normal form* if $aux(L) = \emptyset$

Now we show a construction to transform any (typed) symbolic graph $\langle G, \Phi_G \rangle$ to normal form by projecting it to (typed) E-graph G' that is obtained by removing all auxiliary variables from G. Later,in Section 8.3 we show that this construction indeed leads to symbolic graphs in normal form.

Definition 4.11 (Construction of symbolic graphs in normal form).
For any (typed) symbolic graph $\langle G, \Phi_G \rangle$ with $aux(G) \neq \emptyset$ we can construct a (typed) symbolic graph in normal form $nor(\langle G, \Phi_G \rangle) = \langle G', \Phi'_G \rangle$, given by the projection $Proj(\langle G, \Phi_G \rangle, a')$ of $\langle G, \Phi_G \rangle$ to G' via $a' : G' \to G$, where G' is obtained by removing all auxiliary variables from G and a' is the inclusion of G' in G. Moreover, this construction induces the (typed) projection morphisms $a : \langle L', \Phi'_L \rangle \to \langle L, \Phi_L \rangle$.

Example 4.12 (Direct transformation via a projective production).
Figure 4.4 shows the application of projective production projBookRoom to an cutout of the definite symbolic graph presented in Example 4.9. On top of Figure 4.4 the projective production is depicted. The production projBookRoom is linear and in normal form. In the following we show how projective pro-

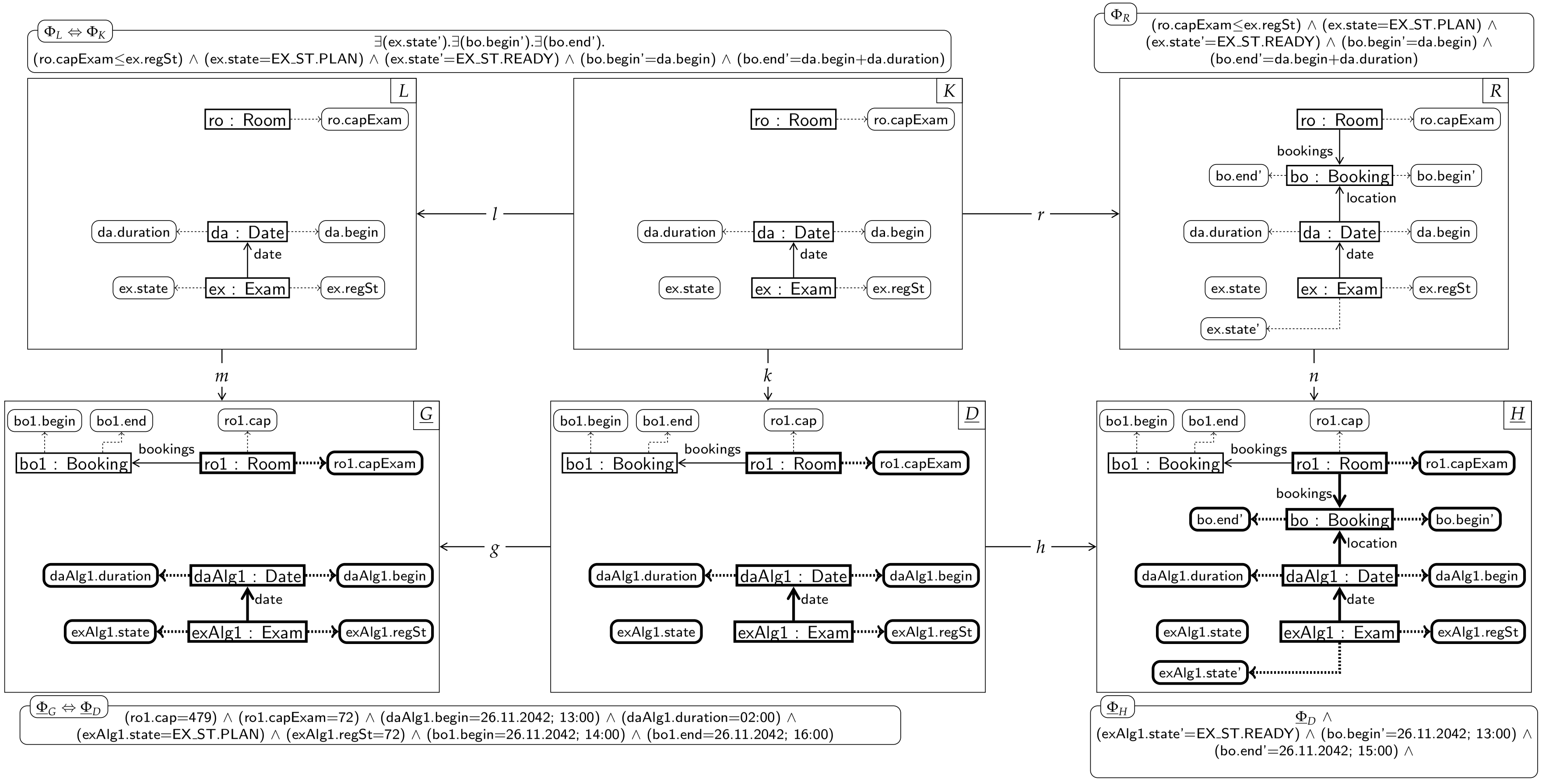

$\Phi_L \Leftrightarrow \Phi_K$
∃(ex.state').∃(bo.begin').∃(bo.end').
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧ (ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧ (bo.end'=da.begin+da.duration)
Φ_R
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧
(bo.end'=da.begin+da.duration)
L
K
R
ro : Room
ro.capExam
bo : Booking
bookings
location
bo.end'
bo.begin'
da : Date
da.duration
da.begin
date
ex : Exam
ex.state
ex.regSt
ex.state'
l
r
m
k
n
g
h
G
D
H
bo1.begin
bo1.end
ro1.cap
bo1 : Booking
ro1 : Room
ro1.capExam
daAlg1 : Date
daAlg1.duration
daAlg1.begin
exAlg1 : Exam
exAlg1.state
exAlg1.regSt
exAlg1.state'
$\underline{\Phi_G} \Leftrightarrow \underline{\Phi_D}$
(ro1.cap=479) ∧ (ro1.capExam=72) ∧ (daAlg1.begin=26.11.2042; 13:00) ∧ (daAlg1.duration=02:00) ∧
(exAlg1.state=EX_ST.PLAN) ∧ (exAlg1.regSt=72) ∧ (bo1.begin=26.11.2042; 14:00) ∧ (bo1.end=26.11.2042; 16:00)
$\underline{\Phi_H}$
$\underline{\Phi_D}$ ∧
(exAlg1.state'=EX_ST.READY) ∧ (bo.begin'=26.11.2042; 13:00) ∧
(bo.end'=26.11.2042; 15:00) ∧

Figure 4.4: The application of projective production projBookRoom to an cutout of the definite symbolic graph presented in Example 4.9

duction projBookRoom can be obtained from symbolic production symbBookRoom shown in Figure 3.3. To this end, assume that production projBookRoom is given by $p' = (\langle L', \Phi'_L \rangle \xleftarrow{l'} \langle K', \Phi'_K \rangle \xrightarrow{r'} \langle R', \Phi'_R \rangle)$ and production symbBookRoom is given by $p = (\langle L, \Phi_L \rangle \xleftarrow{l} \langle K, \Phi_K \rangle \xrightarrow{r} \langle R, \Phi_R \rangle)$. Basically, the E-graph components L', K', are constructed by removing all primed label nodes from the graph part and existentially quantify the corresponding variables in the formula. More specifically:

- L' is obtained from L by removing all auxiliary variables from L, i. e., L' is identical to L except that $X'_L = X_L \backslash aux(L)$
- K' is obtained from K by removing all variables from K whose image under l is an auxiliary variable in L, i. e.,

$$X'_K = X_K \backslash \{x_K \in X_K \mid l_X(x_K) \in aux(L)\}.$$

The symbolic graphs $\langle L', \Phi'_L \rangle$ and $\langle K', \Phi'_K \rangle$ are obtained as $Proj(\langle L, \Phi_L \rangle, a'_L)$ and $Proj(\langle K, \Phi_K \rangle, a'_K)$, where morphisms $a'_L : L' \to L$ and $a'_K : K' \to K$ are the inclusions of L' in L and K' in K, respectively. The symbolic graph $\langle R', \Phi'_R \rangle$ is identical to $\langle R, \Phi_R \rangle$. Note that $\mathcal{D} \models \Phi_K \Leftrightarrow \Phi_L$.

The production is applied to definite symbolic graph $\langle \underline{G'}, \underline{\Phi'_G} \rangle$ via match m', by first deriving $\langle \underline{D'}, \underline{\Phi'_D} \rangle$ as the pushout complement of m' and l', and afterwards constructing $\langle \underline{H'}, \underline{\Phi'_H} \rangle$ as the pushout of r' and k'.

The resulting symbolic graph $\langle \underline{H'}, \underline{\Phi'_H} \rangle$ contains a new booking bo for room ro1 and the created label nodes exAlg1.state', bo.begin', and bo.end'; as well as the created predicates

(bo.begin'=daAlg1.begin), (bo.end'=daAlg1.begin+daAlg1.duration),

and

(exAlg1.state'=EX_ST.READY).

Note that $\langle \underline{H'}, \underline{\Phi'_H} \rangle$ is definite.

One may wonder whether any projective direct transformation of a definite symbolic graph results again in a definite symbolic graph. The answer is no, as the application of a projective production may create label nodes without assigning definite values to them. However, a pushout along an $\mathcal{M}^{inj}_{Proj,TG}$-morphism $f : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$ preserves definiteness if the values of the created variables (i. e., the variables $b^* \in X_B \backslash f_X(X_A)$) are functionally determined by means of the variables $b \in f_X(X_A)$, which leads us to the notion of *functional projective symbolic* graph morphisms.

Definition 4.13 (The class $\mathcal{M}^{inj}_{Func,TG}$ of typed functional projective morphisms)**.**

Given the category $\mathbb{TSG}_{\mathcal{D},TG}$ over a Σ-structure $\mathcal{D}$ and symbolic type graph TG^{Φ}. A typed symbolic graph morphism $f : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$ is in $\mathcal{M}^{inj}_{Func,TG}$ if f is in $\mathcal{M}^{inj}_{Proj,TG}$, and additionally for any variable $b^*_i \in \{b^*_1, \ldots, b^*_n\} = X_B \backslash a_X(X_A)$

of sort $s(i)$ (i. e., $b_i^* \in X_{s(i)}$), there is a term $t_i \in \mathcal{T}_{s(i)}$ with $var(t_i) \subseteq f_X(X_A)$ such that

$$\mathcal{D} \models \Phi_B \Leftrightarrow (\Phi_A[\hat{f}] \wedge (b_1^* \overset{s(1)}{=} t_1) \wedge \ldots \wedge (b_n^* \overset{s(n)}{=} t_n)).$$

Basically, *typed functional projective productions* can be defined by replacing the class $\mathcal{M}^{inj}_{Proj,TG}$ in Definition 4.7 by $\mathcal{M}^{inj}_{Func,TG}$ leading to the following definition.

Definition 4.14 (Typed functional projective productions).
A typed functional projective production $p = (\langle L, \Phi_L \rangle \xleftarrow{l} \langle K, \Phi_K \rangle \xrightarrow{r} \langle R, \Phi_R \rangle)$ consists of a *left production morphism* $l : \langle L, \Phi_L \rangle \to \langle K, \Phi_K \rangle$ of class $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and *a right production morphism* $r : \langle R, \Phi_R \rangle \to \langle K, \Phi_K \rangle$ of class $\mathcal{M}^{inj}_{Func,TG}$.

Note that $\mathcal{M}^{inj}_{Func,TG}$ is a subclass of $\mathcal{M}^{inj}_{Proj,TG}$. Consequently, every typed functional projective production is also a typed projective production; thus, functional projective productions enjoy the properties of projective productions. Additionally, direct transformations along functional projective productions preserve definiteness; that is, the result of applying a functional projective production to a definite symbolic graph is again a definite symbolic graph. In order to proof this, we first show that pushouts along functional projective morphisms preserve definiteness.

Lemma 4.15 (Pushouts along $\mathcal{M}^{inj}_{Func,TG}$-morphisms preserve definiteness).
Let $\mathbb{TSG}_{\mathcal{D},TG}$ be the category of typed symbolic graphs over a Σ-structure $\mathcal{D}$ and symbolic type graph TG^{Φ}, then for any pushout (1) with $f \in \mathcal{M}^{inj}_{Func,TG}$ and $g \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ we have that if $\langle C, \Phi_C \rangle$ is definite, then $\langle D, \Phi_D \rangle$ is definite.

$$\begin{tikzcd}
\langle A, \Phi_A \rangle \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & \langle B, \Phi_B \rangle \arrow[d, "g'"] \\
\langle C, \Phi_C \rangle \arrow[r, "f'"] & \langle D, \Phi_D \rangle
\end{tikzcd}$$

Proof. We have to show that if $\langle C, \Phi_C \rangle$ is definite , then $\langle D, \Phi_D \rangle$ is definite. As f and g are E-graph monomorphisms and (1) is a pushout in $\mathbb{TEG}_{TG}$, we may assume without loss of generality (see Fact 3.9) that $X_B = X_A \cup X_B^*$ and $X_C = X_A \cup X_C^*$ with $f_X(a) = a$ and $g_X(a) = a$ for all $a \in X_A$; as well as $X_D = X_A \cup X_C^* \cup X_B^*$ with $f'_X(c) = c$ for all $c \in X_C$ and $g'_X(b) = b$ for all $b \in X_B$. Moreover, we may assume that X_A, X_B^*, and X_C^* are pairwise disjoint. Hence, showing that $\langle D, \Phi_D \rangle$ is definite if $\langle C, \Phi_C \rangle$ is definite becomes equivalent to show that for any two assignments ζ_1 and ζ_2 such that $\zeta_1(c) = \zeta_2(c)$ for each label node $c \in X_C \subseteq X_D$, we have that $(\mathcal{D}, \zeta_1) \models \Phi_D$ and $(\mathcal{D}, \zeta_2) \models \Phi_D$ implies $\zeta_1(d) = \zeta_2(d)$ for each $d \in X_D$.
As (1) is a pushout in $\mathbb{TSG}_{\mathcal{D},TG}$ we know that

$$\mathcal{D} \models \Phi_D \Leftrightarrow (\Phi_C \wedge \Phi_B). \tag{4.1}$$

From $f \in \mathcal{M}^{inj}_{Func,TG}$ we obtain

$$\mathcal{D} \models \Phi_B \Leftrightarrow (\Phi_A \wedge (b_1^* \overset{s(1)}{=} t_1) \wedge \ldots \wedge (b_n^* \overset{s(n)}{=} t_n)), \tag{4.2}$$

with $\{b_1^*, \ldots, b_n^*\} = X_B^*$. By combining Statements 4.1 and 4.2 we obtain

$$\mathcal{D} \models \Phi_D \Leftrightarrow (\Phi_C \wedge \Phi_A \wedge (b_1^* \overset{s(1)}{=} t_1) \wedge \ldots \wedge (b_n^* \overset{s(n)}{=} t_n)).$$

As by assumption $\langle C, \Phi_C \rangle$ is definite, we have for all assignments ζ_1 and ζ_2 such that $(\mathcal{D}, \zeta_1) \models \Phi_D$ and $(\mathcal{D}, \zeta_2) \models \Phi_D$, that $\zeta_1(c) = \zeta_2(c)$ for each label node $c \in X_C \subseteq X_D$. From $(\mathcal{D}, \zeta_1) \models (b_i^* \overset{s(i)}{=} t_i)$ we obtain $[\![b_i^*]\!]^{\mathcal{D}}_{\zeta_1} = \zeta_1(b_1^*) = [\![t_i]\!]^{\mathcal{D}}_{\zeta_1}$; from $(\mathcal{D}, \zeta_2) \models (b_i^* \overset{s(i)}{=} t_i)$ we obtain $[\![b_i^*]\!]^{\mathcal{D}}_{\zeta_2} = \zeta_2(b_i^*) = [\![t_i]\!]^{\mathcal{D}}_{\zeta_2}$ for all $i \in \{1, \ldots, n\}$. As $var(t_i) \subseteq X_A \subseteq X_C$ and $\zeta_1(c) = \zeta_2(c)$ for each label node $c \in X_C$, so $[\![t_i]\!]^{\mathcal{D}}_{\zeta_1} = [\![t_i]\!]^{\mathcal{D}}_{\zeta_2}$ for all $i \in \{1, \ldots, n\}$; hence, $[\![b_i^*]\!]^{\mathcal{D}}_{\zeta_1} = [\![b_i^*]\!]^{\mathcal{D}}_{\zeta_2}$ for all $i \in \{1, \ldots, n\}$. □

Now, we can show that direct transformations along functional projective productions preserve definiteness .

Lemma 4.16 (Functional direct transformations preserve definiteness)**.**
Given direct transformation $\langle G, \Phi_G \rangle \xRightarrow{p@m} \langle H, \Phi_G \rangle$ via an functional projective production $p = (\langle L, \Phi_L \rangle \xleftarrow{l} \langle K, \Phi_K \rangle \xrightarrow{r} \langle R, \Phi_R \rangle)$ and *match* $m : L \rightarrow G$, $m \in \mathcal{M}^{inj}_{\Rightarrow,TG}$, then $\langle H, \Phi_H \rangle$ is definite if $\langle G, \Phi_G \rangle$ is definite.

Proof. First we show that if $\langle G, \Phi_G \rangle$ is definite then also $\langle D, \Phi_D \rangle$. As $l \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $m \in \mathcal{M}^{inj}_{\Rightarrow,TG}$, pushout (1) implies $g \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ (Lemma 6.1). Accordingly, $\mathcal{D} \models \Phi_G \Leftrightarrow \Phi_D[\hat{g}]$. Hence, if $\langle G, \Phi_G \rangle$ is definite then also $\langle D, \Phi_D \rangle$. It remains to show that if $\langle D, \Phi_D \rangle$ is definite then also $\langle H, \Phi_H \rangle$, which is a direct consequence of Lemma 4.15, as (2) is a pushout along $\mathcal{M}^{inj}_{Func,TG}$-morphism r.

$$\begin{tikzcd}
\langle L, \Phi_L \rangle \arrow[d, "m"] \arrow[dr, phantom, "(1)"] & \langle K, \Phi_L \rangle \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(2)"] & \langle R, \Phi_R \rangle \arrow[d, "n"] \\
\langle G, \Phi_G \rangle & \langle D, \Phi_D \rangle \arrow[l, "g"] \arrow[r, "h"'] & \langle H, \Phi_H \rangle
\end{tikzcd}$$

□

Note that all productions used in this thesis (also those presented later) are functional projective productions, as it is quite natural to define the values of the created variables in terms of the preserved variables.

5

$(\mathcal{L}, \mathcal{R}, \mathcal{N})$-ADHESIVE CATEGORIES AND TRANSFORMATION SYSTEMS

In this chapter we generalize the concept of $(\mathcal{M}, \mathcal{N})$-adhesive transformation systems to $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems. The concept of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems provides the categorical foundations for projective graph transformation systems, i. e., for transformation systems that distinguish between left and right production morphisms. Intuitively, an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation system can be considered as an $(\mathcal{M}, \mathcal{N})$-adhesive transformation system where the class $\mathcal{M}$ of production morphism is split up into the classes $\mathcal{L}$ and $\mathcal{R}$ of left and right production morphisms.

The main contribution of this chapter is to show that the fundamental results of the double pushout approach remain valid for $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems.

To this end, we define the concept of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories and transformation systems in Section 5.1, including an adapted notion and proofs for the HLR properties. In order to prove the results for consistency constraints verification as well as for conflict detection, we require in addition to the HLR properties additional properties that cannot be derived from the axioms of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories. These additional properties are usually referred to as HLR^{+}properties [HP12a, GBEG14] and are introduced in Section 5.2. Subsequently we show that properties of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories together with the HLR and HLR^{+}properties are sufficient to prove the basic results required for constraint verification and conflict detection. More specifically, in Section 5.3 we verify the results for constructing application conditions from consistency constraints. In Section 5.4, we provide proofs for the Local Church–Rosser, Embedding, and Extension Theorems as well as for Critical Pair Lemma in the context of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems.

5.1 $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-Adhesive Categories and Transformation Systems

In this section we introduce the notion of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories and transformation systems, and show that HLR-properties [HP12a] (see Section 3.1.2) of $(\mathcal{M}, \mathcal{N})$-adhesive categories can be lifted to $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories.

Recall that for $(\mathcal{M}, \mathcal{N})$-adhesive categories we distinguished between the class $\mathcal{M}$ of production morphisms and the class $\mathcal{N}$ of match morphisms. For $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories we further split the class of production morphisms into the classes $\mathcal{L}$ and $\mathcal{R}$ to distinguish between *left-hand and right-hand production morphisms*, respectively. This leads us to following notion of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories.

Definition 5.1 ($(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category)**.**
A category $\mathbb{C}$ with morphism classes $\mathcal{L}$, $\mathcal{R}$, and $\mathcal{N}$ is an *$(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category* $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ if the following requirements are satisfied:

1) *Closure Properties:*

   a) *$\mathcal{L}$, $\mathcal{R}$ and $\mathcal{N}$ contain all isomorphisms.* f being an isomorphism implies $f \in \mathcal{L}$, $f \in \mathcal{R}$ and $f \in \mathcal{N}$.

   b) *$\mathcal{L}$, $\mathcal{R}$ and $\mathcal{N}$ are closed under composition.* Let $f : A \to B$ and $g : B \to C$ be morphisms in $\mathbb{C}$, then $f, g \in \mathcal{X}$ implies $(g \circ f) \in \mathcal{X}$, where $\mathcal{X} \in \{\mathcal{L},\mathcal{R},\mathcal{N}\}$.

   c) *$\mathcal{L}$, $\mathcal{R}$ and $\mathcal{N}$ are closed under decomposition.* Let $f : A \to B$ and $g : B \to C$ be morphisms in $\mathbb{C}$, then $(g \circ f) \in \mathcal{X}$ and $g \in \mathcal{X}$ implies $f \in \mathcal{X}$ where $\mathcal{X} \in \{\mathcal{L},\mathcal{R},\mathcal{N}\}$.

   d) *$\mathcal{L}$ is a subclass of $\mathcal{R}$.*

   e) *$\mathcal{N}$ is closed under $\mathcal{R}$-composition.* Let $f : A \to B$ and $g : B \to C$ be morphisms in $\mathbb{C}$, then $f \in \mathcal{N}$ and $g \in \mathcal{R}$ implies $(g \circ f) \in \mathcal{N}$.

   f) *$\mathcal{N}$ is closed under $\mathcal{R}$-decomposition.* Let $f : A \to B$ and $g : B \to C$ be morphisms in $\mathbb{C}$, then $(g \circ f) \in \mathcal{N}$ and $g \in \mathcal{R}$ implies $f \in \mathcal{N}$.

2) *Pushouts and Pullbacks:*

   a) *$\mathbb{C}$ has $(\mathcal{R},\mathcal{N})$-pushouts.*

   b) *$\mathbb{C}$ has $\mathcal{R}$-pullbacks.*

   c) *$\mathcal{L}$, $\mathcal{R}$, and $\mathcal{N}$ are closed under pushouts.* Given pushout (1), then $f \in \mathcal{X}$ implies $g \in \mathcal{X}$, for any $\mathcal{X} \in \{\mathcal{L},\mathcal{R},\mathcal{N}\}$.

   d) *$\mathcal{L}$, $\mathcal{R}$, and $\mathcal{N}$ are closed under pullbacks.* Given pullback (1), then $g \in \mathcal{X}$ implies $f \in \mathcal{X}$, for any $\mathcal{X} \in \{\mathcal{L},\mathcal{R},\mathcal{N}\}$.

3) *Pushouts along $(\mathcal{R},\mathcal{N})$-morphisms are $(\mathcal{L},\mathcal{R},\mathcal{N})$-VK squares.* A pushout (1) with $m \in \mathcal{R}$ and $f \in \mathcal{N}$ is an $(\mathcal{L},\mathcal{R},\mathcal{N})$-VK square if for any commutative cube (2) with (1) in the bottom, where the back faces are pullbacks and $b, c, d \in \mathcal{R}$, $a \in \mathcal{L}$,the following statement holds: the top face is a pushout if and only if the front faces are pullbacks.

4) *Pushouts along $(\mathcal{L},\mathcal{N})$-morphisms are $(\mathcal{L},\mathcal{N})$-VK squares.* A pushout (1) with $m \in \mathcal{L}$ and $f \in \mathcal{N}$ is an $(\mathcal{L},\mathcal{N})$-VK square if for any commutative cube (2) with (1) in the bottom, where the back faces are pullbacks and $b, c, d \in \mathcal{R}$, the following statement holds: the top face is a pushout if and only if the front faces are pullbacks.

$$\begin{tikzcd}
A \arrow[r, "m"] \arrow[d, "f"'] \arrow[dr, phantom, "(1)"] & B \arrow[d, "g"] \\
C \arrow[r, "n"] & D
\end{tikzcd}
\qquad
\begin{tikzcd}
& A' \arrow[dl, "f'"'] \arrow[rr, "m'"] \arrow[dd, "a"] & & B' \arrow[dl, "g'"'] \arrow[dd, "b"] \\
C' \arrow[rr, "n'"] \arrow[dd, "c"] & & D' \arrow[dd, "d"] & \\
& A \arrow[dl, "f"'] \arrow[rr, "m"] & & B \arrow[dl, "g"] & (2) \\
C \arrow[rr, "n"] & & D &
\end{tikzcd}$$

An *$(\mathcal{R},\mathcal{N})$-pushout* is a pushout where one of the given morphisms is in $\mathcal{R}$ and the other morphism is in $\mathcal{N}$; an *$\mathcal{R}$-pullback* is a pullback where at least one of the given morphisms is in $\mathcal{R}$.

An $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category provides the following HLR properties.

Theorem 5.2 (HLR properties of $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive categories)**.**
For any $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$, the following properties hold:

1) *Pushouts along $(\mathcal{R},\mathcal{N})$-morphisms are pullbacks.* Given $(\mathcal{R},\mathcal{N})$-pushout (1), then (1) is also a pullback.

2) *The $\mathcal{R}$–$\mathcal{R}$-pushout–pullback decomposition.* If (1)+(2) is an $(\mathcal{R},\mathcal{N})$-pushout with $l \in \mathcal{R}$ and $r \circ k \in \mathcal{N}$, and (2) a pullback with $w \in \mathcal{R}$, then (1) and (2) are pushouts as well as pullbacks.

3) *The $(\mathcal{L},\mathcal{R},\mathcal{N})$-cube pushout–pullback decomposition.* Given the commutative cube (3) where all morphisms in the top square and bottom square are in $\mathcal{R}$ and additionally $n' \in \mathcal{L}$, all vertical morphisms are in $\mathcal{N}$, the top face is a pullback and the front faces are pushouts, then the following statement holds: the bottom face is a pullback if and only if the back faces are pushouts.

4) *Pushout complements along $(\mathcal{R},\mathcal{N})$-morphisms are unique.* Given morphisms $l : A \to C$ and $u : C \to D$, where $l \in \mathcal{R}$ and $u \in \mathcal{N}$, there is at most one B (up to isomorphism) and morphisms $k : A \to B$ and $s : B \to D$, such that (1) is a pushout.

$$\begin{tikzcd}
A \arrow[r, "l"] \arrow[d, "k"'] \arrow[dr, phantom, "(1)"] & C \arrow[d, "u"] \\
B \arrow[r, "s"] \arrow[d, "r"'] \arrow[dr, phantom, "(2)"] & D \arrow[d, "w"] \\
E \arrow[r, "v"] & F
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=small]
& & A' \arrow[dll, "f'"'] \arrow[drr, "m'"] \arrow[ddd, "a"] & & \\
C' \arrow[drr, "n'"'] \arrow[ddd, "c"'] & & & & B' \arrow[dll, "g'"] \arrow[ddd, "b"] \\
& & D' \arrow[ddd, "d"] & & \\
& & A \arrow[dll, "f"'] \arrow[drr, "m"] & & \\
C \arrow[drr, "n"'] & & & & B \arrow[dll, "g"] \\
& & D & & (3)
\end{tikzcd}$$

Proof. The following proofs are extended versions of the corresponding proofs given in [EEPT06].

1) Consider the following commutative cube (4) with $(\mathcal{R},\mathcal{N})$-pushout (1) in both the bottom and the front left face. We have to show that (1) is also a pullback.

$$\begin{tikzcd}
A \arrow[r, "l"] \arrow[d, "k"'] \arrow[dr, phantom, "(1)"] & C \arrow[d, "u"] \\
B \arrow[r, "s"] & D
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=small]
& & A \arrow[dll, "id_A"'] \arrow[drr, "l"] \arrow[ddd, "id_A"] & & \\
A \arrow[drr, "l"'] \arrow[ddd, "k"'] & & & & C \arrow[dll, "id_C"] \arrow[ddd, "id_C"] \\
& & C \arrow[ddd, "u"] & & \\
& & A \arrow[dll, "k"'] \arrow[drr, "l"] & & \\
B \arrow[drr, "s"'] & & & & C \arrow[dll, "u"] \\
& & D & & (4)
\end{tikzcd}$$

The bottom face is the $(\mathcal{R},\mathcal{N})$-pushout (1) along $l \in \mathcal{R}$ and $k \in \mathcal{N}$; thus, an $(\mathcal{L},\mathcal{R},\mathcal{N})$-VK square. Moreover, we have:

- $id_A \in \mathcal{L}$ and $id_C \in \mathcal{N}$, as identity morphisms are isomorphisms and $\mathcal{L}$ as well as $\mathcal{N}$ contain all isomorphisms
- $u \in \mathcal{N}$, as (1) is a pushout, $k \in \mathcal{N}$ and $\mathcal{N}$ is closed under pushouts

From Fact 3.14.c we know that:

- the back left face is a pullback
- the back right face is a pullback
- the top is a pushout

From the VK square property, we conclude that the front faces are pullbacks. Hence, (1) is a pullback.

2) Consider the following commutative cube (5) with pushout (1)+(2) in the bottom.

$$\begin{tikzcd}
 & & & & A \arrow[dll, "k"'] \arrow[dr, "l"] \arrow[ddd, "id_A"] & \\
 & & B \arrow[dll, "id_B"'] \arrow[dr, "s"] \arrow[ddd, "id_B"] & & & C \arrow[dll, "u"'] \arrow[ddd, "id_C"] \\
B \arrow[dr, "s"] \arrow[ddd, "r"] & & & D \arrow[dll, "id_D"'] \arrow[ddd, "id_D"] & & \\
 & D \arrow[ddd, "w"] & & & A \arrow[dll, "k"'] \arrow[dr, "l"] & \\
 & & B \arrow[dll, "r"'] \arrow[dr, "s"] & & & C \arrow[dll, "u"'] \\
E \arrow[dr, "v"] & & & D \arrow[dll, "w"'] & & (5) \\
 & F & & & &
\end{tikzcd}$$

We have the following properties:

- The bottom (1)+(2) is a pushout along the $\mathcal{R}$-morphism l and $\mathcal{N}$-morphism $r \circ k$; thus, an $(\mathcal{L},\mathcal{R},\mathcal{N})$-VK square.
- The back right face is a pullback (Fact 3.14.c).
- The back left face is a pullback, as it is composed of two pullbacks (Fact 3.14.b and 3.14.c).
- The front right face is a pullback, as it is composed of two pullbacks (Fact 3.14.b and 3.14.c).
- The front left face is a pullback by assumption.

Hence, we have $r \in \mathcal{R}$, as $w \in \mathcal{R}$ and $\mathcal{R}$ is closed under pullbacks. Moreover, we have $id_C \in \mathcal{R}$ and $id_A \in \mathcal{L}$, as identities are isomorphism and $\mathcal{R}$ as well as $\mathcal{L}$ contain all isomorphisms. From the VK square property we can conclude that the top face, which corresponds to square (1), is a pushout. As the left back face is a pullback along $\mathcal{N}$-morphisms $r \circ k$, we have $id_B \circ k \in \mathcal{N}$. As $id_B \in \mathcal{N}$ and $\mathcal{N}$-morphisms are closed under decomposition, we have $k \in \mathcal{N}$. Consequently, (1) is an $(\mathcal{R},\mathcal{N})$-pushout, and therefore also a pullback. By pushout decomposition we can conclude that also (2) is a pushout.

3) **If.** Assume, the bottom of the original cube (3) is a pullback, we have to show that the back faces are pushouts. Consider the turned cube (6).

$$\begin{tikzcd}
& A' \arrow[dl, "f'"'] \arrow[dr, "m'"] \arrow[ddd, "a"] & \\
C' \arrow[dr, "n'"'] \arrow[ddd, "c"'] & & B' \arrow[dl, "g'"'] \arrow[ddd, "b"] \\
& D' \arrow[ddd, "d"] & \\
& A \arrow[dl, "f"'] \arrow[dr, "m"] & \\
C \arrow[dr, "n"'] & & B \arrow[dl, "g"] \\
& D & (3)
\end{tikzcd}
\qquad
\begin{tikzcd}
& A' \arrow[dl, "a"'] \arrow[dr, "f'"] \arrow[ddd, "m'"] & \\
A \arrow[dr, "f"'] \arrow[ddd, "m"'] & & C' \arrow[dl, "c"'] \arrow[ddd, "n'"] \\
& C \arrow[ddd, "n"] & \\
& B' \arrow[dl, "b"'] \arrow[dr, "g'"] & \\
B \arrow[dr, "g"'] & & D' \arrow[dl, "d"] \\
& D & (6)
\end{tikzcd}$$

The following properties then apply to (6):

- The bottom face is a pushout along the $\mathcal{R}$-morphism g' and $\mathcal{N}$-morphism b; thus, an $(\mathcal{L},\mathcal{R},\mathcal{N})$-VK square.
- The front left face is a pullback (by assumption).
- The front right face is a pushout along $n' \in \mathcal{R}$ (as $\mathcal{L}$ is a subclass of $\mathcal{R}$) and $c \in \mathcal{N}$, and therefore also a pullback.
- The back right face is a pullback (by definition).
- The back left face is a pullback (by pullback composition and decomposition).
- We have $m, n, n', g' \in \mathcal{R}$ and $b \in \mathcal{N}$ (by definition).
- The morphism m' is in $\mathcal{L}$, as $n' \in \mathcal{L}$, the back right face is a pullback, and $\mathcal{L}$ is closed under pullbacks.

From the $(\mathcal{L},\mathcal{R},\mathcal{N})$-VK property follows that the top face is a pushout; hence, the back left face in the original cube (3) is a pushout.

By turning the cube once more we obtain cube (7).

$$\begin{tikzcd}
& A' \arrow[dl, "f'"'] \arrow[dr, "m'"] \arrow[ddd, "a"] & \\
C' \arrow[dr, "n'"'] \arrow[ddd, "c"'] & & B' \arrow[dl, "g'"'] \arrow[ddd, "b"] \\
& D' \arrow[ddd, "d"] & \\
& A \arrow[dl, "f"'] \arrow[dr, "m"] & \\
C \arrow[dr, "n"'] & & B \arrow[dl, "g"] \\
& D & (3)
\end{tikzcd}
\qquad
\begin{tikzcd}
& A' \arrow[dl, "a"'] \arrow[dr, "m'"] \arrow[ddd, "f'"] & \\
A \arrow[dr, "m"'] \arrow[ddd, "f"'] & & B' \arrow[dl, "b"'] \arrow[ddd, "g'"] \\
& B \arrow[ddd, "g"] & \\
& C' \arrow[dl, "c"'] \arrow[dr, "n'"] & \\
C \arrow[dr, "n"'] & & D' \arrow[dl, "d"] \\
& D & (7)
\end{tikzcd}$$

The following properties then apply to (7):

- The bottom face is a pushout along the $\mathcal{L}$-morphism n' and $\mathcal{N}$-morphism c; thus, an $(\mathcal{L},\mathcal{N})$-VK square.
- The front left face is a pullback (by assumption).
- The front right face is a pushout along $g' \in \mathcal{R}$ and $b \in \mathcal{N}$; thus, also a pullback (by Theorem 5.2.1).
- The back right face is a pullback (by definition).
- The back left face is a pushout along $f' \in R$ and $a \in \mathcal{N}$; thus, also a pullback (by Theorem 5.2.1).

- We have $f, g, g' \in \mathcal{R}$, $c \in \mathcal{N}$, and $n' \in \mathcal{L}$.

From the $(\mathcal{L},\mathcal{N})$-VK property follows that the top face is a pushout, which corresponds to (3) in the original cube.

Only if. Assume, the back faces of the original cube (3) are pushouts, we have to show that the bottom face is a pullback. Considering again the turned cube (7), we have the following properties:

- The bottom face is a pushout along the $\mathcal{L}$-morphism n' and $\mathcal{N}$-morphism c; thus, a $(\mathcal{L},\mathcal{N})$-VK square.
- The top face is a pushout (by assumption).
- The back left face is a is a pushout along $f' \in \mathcal{R}$ and $a \in \mathcal{N}$, and therefore a pullback (by Theorem 5.2.1).
- The back right face is a pullback (by definition).
- We have $f, g, g' \in \mathcal{R}$, $c \in \mathcal{N}$, and $n' \in \mathcal{L}$ (by definition).

From the $(\mathcal{L},\mathcal{N})$-VK property follows that the front faces are pullbacks. Hence, the bottom face of the original cube is a pullback.

4) Suppose that commutative squares (8) and (9) below are pushouts with $k \in \mathcal{R}$ and $l, l' \in \mathcal{N}$. To show that pushout complements are unique we have to show that C and C' are isomorphic. As $\mathcal{R}$ is closed under pushouts we have $u, u' \in \mathcal{R}$. Consider the following cube (10), where (8) is in the bottom and (9) is the front right face.

$$\begin{tikzcd}
A \arrow[r, "k"] \arrow[d, "l"'] \arrow[dr, phantom, "(8)"] & B \arrow[d, "s"] \\
C \arrow[r, "u"] & D
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "k"] \arrow[d, "l'"'] \arrow[dr, phantom, "(9)"] & B \arrow[d, "s"] \\
C' \arrow[r, "u'"] & D
\end{tikzcd}
\qquad
\begin{tikzcd}
 & A \arrow[dl, "h"'] \arrow[drr, "id_A"] \arrow[ddd, "id_A"] & & \\
U \arrow[drr, "x"] \arrow[ddd, "y"'] & & & A \arrow[dl, "l'"'] \arrow[ddd, "k"] \\
 & & C' \arrow[ddd, "u'"] & \\
 & A \arrow[dl, "l"'] \arrow[drr, "k"] & & \\
C \arrow[drr, "u"'] & & & B \arrow[dl, "s"] \\
 & & D &
\end{tikzcd}
\quad (10)$$

The front left face with $C \xleftarrow{y} U \xrightarrow{x} C'$ is constructed as the pullback over $C \xrightarrow{u} D \xleftarrow{u'} C'$. Morphisms x, y are in $\mathcal{R}$, as $u, u' \in \mathcal{R}$ and $\mathcal{R}$ is closed under pullbacks. The morphism $h : A \to U$ is obtained from the universal property of pullback $C \xleftarrow{y} U \xrightarrow{x} C'$ with morphism l and l'. Hence, it holds that $x \circ h = l'$ and $y \circ h = l$. Moreover, the following properties apply to cube (10):

- The bottom face (8) is a pushout along the $\mathcal{R}$-morphism k and $\mathcal{N}$-morphism l; thus, a $(\mathcal{L},\mathcal{R},\mathcal{N})$-VK square.
- The back right face is a pullback (Fact 3.14.c).
- The front right face is a pushout along $k \in \mathcal{R}$ and $l' \in \mathcal{N}$, and therefore also a pullback (by Theorem 5.2.1).
- The front left face is a pullback by construction.

- The back left face is a pullback (by pullback composition and decomposition).
- We have $k, u', y \in \mathcal{R}$, $l \in \mathcal{N}$, and $id_A \in \mathcal{L}$.

From the $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-VK property follows that the top face is a pushout. As id_A is an isomorphism and pushouts preserve isomorphisms, x is also an isomorphism. By exchanging C and C', we can conclude that y is also an isomorphism. Consequently C and C' are isomorphic. □

Corollary 5.3 (Induced HLR properties of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories).
Note that according to Definition 5.1 $\mathcal{L}$ is a subclass of $\mathcal{R}$. Consequently, any $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category has also $(\mathcal{L}, \mathcal{N})$-pushouts, $\mathcal{L}$-pullbacks, as well as $\mathcal{N}$ is closed under $\mathcal{L}$-composition and $\mathcal{L}$-decomposition. Moreover, in any $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category $(\mathcal{L}, \mathcal{N})$-pushouts are pullbacks and it has also the $\mathcal{L}$–$\mathcal{L}$-pushout–pullback decomposition property.

Remark 5.4 (Correspondence of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$ and $(\mathcal{M}, \mathcal{N})$-adhesive categories).
Note that according to Definition 5.1 $\mathcal{L}$ is a subclass of $\mathcal{R}$. This implies that all HLR properties given in Theorem 5.2 are also valid if we set $\mathcal{R} = \mathcal{L}$. Consequently, every $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N})$ with $\mathcal{R} = \mathcal{L}$ is also an $(\mathcal{M}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{M}, \mathcal{N})$. One might wonder whether $(\mathcal{M}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{M}, \mathcal{N})$ is equivalent to $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N})$ with $\mathcal{L} = \mathcal{R}$. However, this is not the case as there is no correspondence for the closure of $\mathcal{N}$ under $\mathcal{R}$-composition in $(\mathcal{M}, \mathcal{N})$-adhesive categories.

Similar to $(\mathcal{M}, \mathcal{N})$-adhesive categories, also $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories are stable under slice construction, which means: if we can show that a base category is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive, then any category that is derived by slice construction from this base category is also $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive. Recall that the slice construction can be used to extend a base category by a typing concept.

Theorem 5.5 (Slice construction of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories).
Given $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N})$, then for every object X in $\mathbb{C}$ the slice category $(\mathbb{C}\backslash X, \mathcal{L}', \mathcal{R}', \mathcal{N}')$ is also $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive, where $\mathcal{L}' = \mathcal{L} \cap Mor_{\mathbb{C}\backslash X}$, $R' = \mathcal{R} \cap Mor_{\mathbb{C}\backslash X}$, and $\mathcal{N}' = \mathcal{N} \cap Mor_{\mathbb{C}\backslash X}$.

Proof. Morphisms, pullbacks and pushouts can be constructed componentwise for slice categories. This componentwise construction ensures that also $\mathcal{L}'$, $\mathcal{R}'$, and $\mathcal{N}'$ are closed under composition, decomposition, pushouts and pullbacks. □

Finally, we define $(\mathcal{L}, \mathcal{R})$-productions and $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-transformations, which together lead to $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems.

Definition 5.6 ($(\mathcal{L}, \mathcal{R})$-production).
Given an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N})$, an *$(\mathcal{L}, \mathcal{R})$-production* $p = (L \leftarrow K \rightarrow R)$ consists of the objects L, K, and R as well as left morphism $l : K \rightarrow L$, $l \in \mathcal{L}$ and right morphism $r : K \rightarrow R$, $r \in \mathcal{R}$.

Definition 5.7 ($(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems)**.**
An $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation system $((\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N}), \mathbf{P})$ is given by an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N})$ together with a finite set **P** consisting of $(\mathcal{L}, \mathcal{R})$-productions.

5.2 HLR$^+$Properties for $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-Adhesive Categories

In order to prove the results for consistency constraints verification as well as for conflict detection, we require in addition to the HLR properties additional properties, which cannot be concluded from the axioms of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories. These additional properties are usually referred to as HLR$^+$properties [HP12a, GBEG14]. In contrast to classical HLR systems, we have to deal with productions where the left and right morphisms might belong to different classes. Accordingly, we identified the following HLR$^+$properties for $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems:

- Binary coproducts
- $\mathcal{E}$–$\mathcal{N}$ factorization
- The $\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition property
- The $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property
- $(\mathcal{L}, \mathcal{N})$-initial pushouts

Moreover, we introduce the notion of quasi $(\mathcal{L}, \mathcal{N})$-initial pushout, in order to deal with $(\mathcal{L}, \mathcal{R})$-productions.

We start with the well-known concept of binary coproducts. Binary coproducts can be considered as a generalization of the concept of disjoint union.

Definition 5.8 (Binary Coproduct)**.**
Let A and B objects of a category $\mathbb{C}$, the triple $(A{+}B, i_1, i_2)$ consisting of:

- a coproduct object $A{+}B$
- a pair of morphisms $i_1 : A \to A{+}B$ and $i_2 : B \to A{+}B$ called *coproduct injections*

is a *binary coproduct* in $\mathbb{C}$, if and only if the following universal property holds: For all objects C with morphisms $f_1 : A \to C$ and $f_2 : A \to C$ there is a morphism $c : A{+}B \to C$ such that the following diagram commutes, i. e., $c \circ i_1 = f_1$ and $c \circ i_2 = f_2$.

$$\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "f_1"'] & A+B \arrow[d, dashed, "c"] \arrow[dl, phantom, "="] \arrow[dr, phantom, "="] & B \arrow[l, "i_2"'] \arrow[dl, "f_2"] \\
 & C &
\end{tikzcd}$$

The following concept of an $\mathcal{E}$–$\mathcal{N}$ *factorization* is a generalization of the fact, known from set theory, which states that every function can be uniquely decomposed in a surjective and an injective component. In the general setting, we just require a category with an additional morphism class $\mathcal{E}$, such that every morphism can be decomposed in an $\mathcal{E}$-morphism and an $\mathcal{N}$-morphism.

Definition 5.9 ($\mathcal{E}$–$\mathcal{N}$ factorization)**.**
An $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ has an $\mathcal{E}$–$\mathcal{N}$ factorization for a given morphism class $\mathcal{E}$ if for each morphism $f : A \to B$ in $\mathbb{C}$, there exists a unique (up to isomorphism) decomposition $e : A \to C$, $m : C \to B$ with $m \circ e = f$ such that $e \in \mathcal{E}$ and $m \in \mathcal{N}$.

$$\begin{tikzcd}
A \arrow[rr, "f"] \arrow[dr, "e"'] & {} \arrow[d, phantom, "="] & B \\
& C \arrow[ur, "m"'] &
\end{tikzcd}$$

The concept of $\mathcal{E}'$–$\mathcal{N}$ pair factorizations, defined next, is the generalisation of $\mathcal{E}$–$\mathcal{N}$ factorizations to pairs of morphisms with the same codomain.

Definition 5.10 ($\mathcal{E}'$–$\mathcal{N}$ pair factorization)**.**
Given a class $\mathcal{E}'$ of morphism pairs with same codomain, an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ has an $\mathcal{E}' - \mathcal{N}$ pair factorization if for each pair of morphisms $f_1 : A_1 \to C$ and $f_2 : A_2 \to C$, there exists an object K with a morphism pair $e_1 : A_1 \to K$, $e_2 : A_2 \to K$, $(e_1, e_2) \in \mathcal{E}'$ and an $\mathcal{N}$-morphism $m : K \to C$ such that the following diagram commutes, i. e., $m \circ e_1 = f_1$ and $m \circ e_2 = f_2$.

$$\begin{tikzcd}
\langle A_1, \Phi_1 \rangle \arrow[dr, "e_1"] \arrow[drr, bend left, "f_1"] & & \\
& \langle K, \Phi_K \rangle \arrow[r, "m"] & \langle C, \Phi_C \rangle \\
\langle A_2, \Phi_2 \rangle \arrow[ur, "e_2"'] \arrow[urr, bend right, "f_2"'] & &
\end{tikzcd}$$

The following lemma shows that any category with binary coproducts and $\mathcal{E}$–$\mathcal{N}$ factorization has also an $\mathcal{E}'$–$\mathcal{N}$ pair factorization. Moreover, we shall see that the class $\mathcal{E}'$ consists of jointly epimorphic morphism pairs.

Lemma 5.11 ($\mathcal{E}'$–$\mathcal{N}$ pair factorization)**.**
For any $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ with binary coproducts and $\mathcal{E}$–$\mathcal{N}$ factorization, there exists a class $\mathcal{E}'$ of morphism pairs with an $\mathcal{E}'$–$\mathcal{N}$ pair factorization. Moreover, for any $\mathcal{E}'$–$\mathcal{N}$ pair factorization of morphisms $f_1 : A_1 \to C$ and $f_2 : A_2 \to C$, the following properties hold:

i) If $f_1, f_2 \in \mathcal{N}$, then $e_1, e_2 \in \mathcal{N}$.

ii) Any pair $(e_1, e_2) \in \mathcal{E}'$ is jointly epimorphic.

Proof. Given morphisms $f_1 : A_1 \to C$ and $f_2 : A_2 \to C$. First, we construct the binary coproduct $(A_1 + A_2, i_1, i_2)$ of A_1 and A_2, as shown in the following diagram.

The coproduct together with morphisms $f_1 : A_1 \to C$ and $f_2 : A_2 \to C$ induces the unique morphism $c : A_1 + A_2 \to C$. Now we take the $\mathcal{E}$–$\mathcal{N}$ factorization $c = m \circ e$ of the morphism c and define $e_1 = e \circ i_1$ and $e_2 = e \circ i_2$, whereas morphisms $i_1 : A_1 \to A_1 + A_2$ and $i_1 : A_1 \to A_1 + A_2$ are the coproduct injections.

$$\begin{tikzcd}
A_1 \arrow[rr, "i_1"] \arrow[drrr, "e_1"] \arrow[ddrr, "f_1"'] & & A_1 + A_2 \arrow[dd, "c"] \arrow[dr, "e"] & & A_2 \arrow[ll, "i_2"'] \arrow[dl, "e_2"'] \arrow[ddll, "f_2"] \\
& & & K \arrow[dl, "m"] & \\
& & C & &
\end{tikzcd}$$

As $m \circ e$ is the $\mathcal{E}$–$\mathcal{N}$ factorization of c, we may conclude that $m \in \mathcal{N}$. Moreover, $m \circ e_1 = f_1$ and $m \circ e_2 = f_2$. From the construction above, we know that $e_1 = e \circ i_1$ and $e_2 = e \circ i_2$. The coproduct gives us $f_1 = c \circ i_1$ and $f_2 = c \circ i_2$. Hence, $m \circ e_1 = m \circ e \circ i_1 = c \circ i_1 = f_1$ and $m \circ e_2 = m \circ e \circ i_2 = c \circ i_2 = f_2$. Consequently, the pair (e_1, e_2) together with morphism m is an $\mathcal{E}'$–$\mathcal{N}$ pair factorization of f_1 and f_2.
It it remains to show that:

(i) If $f_1, f_2 \in \mathcal{N}$, then also $e_1, e_2 \in \mathcal{N}$. This is a direct consequence of the closure of $\mathcal{N}$ under decomposition and $m, f_1, f_2 \in \mathcal{N}$.

(ii) Any pair $(e_1, e_2) \in \mathcal{E}'$ is jointly epimorphic. Consider the diagram below with morphisms as constructed above. We have to show for any morphism pair $g, h : K \to D$, that if $g \circ e_i = h \circ e_i$, with $i = 1, 2$ then $g = h$.

$$\begin{tikzcd}
A_1 \arrow[r, "i_1"] \arrow[dr, "f_1"'] & A_1 + A_2 \arrow[d, "e"] & A_2 \arrow[l, "i_2"'] \arrow[dl, "f_2"] \\
& K \arrow[d, bend right, "g"'] \arrow[d, bend left, "h"] & \\
& D &
\end{tikzcd}$$

As we know from the coproduct that $e_1 = e \circ i_1$ and $e_2 = e \circ i_2$, this is equivalent to show that $g \circ e \circ i_i = h \circ e \circ i_i$, with $i = 1, 2$ implies $g = h$, which is a direct consequence of the fact that e is an epimorphism. □

The $\mathcal{R}$–$\mathcal{N}$-pushout–pullback and the $\mathcal{L}$–$\mathcal{N}$-pushout–pullback decomposition property defined in the following are generalizations of the $\mathcal{R}$–$\mathcal{R}$-pushout–pullback decomposition property stated in Theorem 5.2.

Definition 5.12 (The $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property).
Given the following commutative diagram with $l \in \mathcal{R}$, $(r \circ k) \in \mathcal{N}$, and $w \in \mathcal{N}$, an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ has *$\mathcal{R}$–$\mathcal{N}$-pushout–pullback decomposition property* if the following property holds: if (1)+(2) is a pushout and (2) a pullback, then (1) and (2) are pushouts as well as pullbacks.

$$\begin{tikzcd}
A \arrow[r, "k"] \arrow[d, "l"'] \arrow[dr, phantom, "(1)"] & B \arrow[r, "r"] \arrow[d, "s"] \arrow[dr, phantom, "(2)"] & E \arrow[d, "v"] \\
C \arrow[r, "u"'] & D \arrow[r, "w"'] & F
\end{tikzcd}$$

Definition 5.13 (The $\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition property)**.**
Given the following commutative diagram with $l \in \mathcal{L}$, $(r \circ k) \in \mathcal{N}$, and $w \in \mathcal{N}$, an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ has *$\mathcal{R}$–$\mathcal{N}$-pushout–pullback decomposition property* if the following property holds: if (1)+(2) is a pushout and (2) a pullback, then (1) and (2) are pushouts as well as pullbacks.

$$\begin{tikzcd}
A \arrow[r, "k"] \arrow[d, "l"'] \arrow[dr, phantom, "(1)"] & B \arrow[r, "r"] \arrow[d, "s"] \arrow[dr, phantom, "(2)"] & E \arrow[d, "v"] \\
C \arrow[r, "u"'] & D \arrow[r, "w"'] & F
\end{tikzcd}$$

Corollary 5.14 (The $\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition property)**.**
Any $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ which provides the $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property has also the $\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition property.

Proof. This is a direct consequence of the fact that by definition of $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive categories $\mathcal{L}$ is a subclass of $\mathcal{R}$ (Definition 5.1). □

Remark 5.15 (The $\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition property)**.**
One might wonder why the $\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition property is also part of the HLR⁺properties, although the $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property implies the $\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition property. However, the opposite is not true. As we shall see later, for some results the $\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition property suffices (e. g., for proof the Completeness of Critical Pairs Lemma in Section 5.4).

The following definition of $(\mathcal{L},\mathcal{N})$-initial pushouts is the adaption of $(\mathcal{M},\mathcal{N})$-initial pushouts stated in Definition 3.26.

Definition 5.16 ($(\mathcal{L},\mathcal{N})$-initial pushout)**.**
Let $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ be an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category, given an $\mathcal{N}$-morphism $f : A \to F$, then $(\mathcal{L},\mathcal{N})$-pushout (1) with $b \in \mathcal{L}$ is an *$(\mathcal{L},\mathcal{N})$-initial pushout over f* if for every $(\mathcal{L},\mathcal{N})$-pushout (2) with $b' \in \mathcal{L}$, there are unique $\mathcal{L}$-morphisms $b^* : B \to D$ and $c^* : C \to E$ such that $b' \circ b^* = b$, $c' \circ c^* = c$ and (3) is a pushout.

$$\begin{tikzcd}
B \arrow[rrrr, "b^*"] \arrow[rrd, "b"] \arrow[dd] & & & & D \arrow[lld, "b'"'] \arrow[dd] \\
& (1) & A \arrow[dd, "f"] \arrow[r, phantom, "(3)"] & (2) & \\
C \arrow[rrrr, "c^*" near end] \arrow[rrd, "c"'] & & & & E \arrow[lld, "c'"] \\
& & F & &
\end{tikzcd}$$

The morphisms b and c are called the *boundary and context with respect to f*.

Note that $(\mathcal{L},\mathcal{N})$-initial pushouts are initial pushouts in the sense of Definition 3.26. As for $(\mathcal{L},\mathcal{R})$-productions the left-hand and right-hand morphisms are not in the same morphism class, we have to give an alternative definition of initial pushouts in order to retain the closure properties. This leads us to the notion of quasi $(\mathcal{L},\mathcal{N})$-initial pushouts. Basically, a quasi $(\mathcal{L},\mathcal{N})$-initial pushout is an $(\mathcal{R},\mathcal{N})$-pushout that is initial for $(\mathcal{L},\mathcal{N})$-pushouts in the following sense:

Definition 5.17 (Quasi $(\mathcal{L},\mathcal{N})$-initial pushout)**.**
Let $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ be an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category, given an $\mathcal{N}$-morphism $m : L \to G$, then $(\mathcal{R},\mathcal{N})$-pushout (1) with $b' \in \mathcal{R}$ is a *quasi $(\mathcal{L},\mathcal{N})$-initial pushout over m* if for every $(\mathcal{L},\mathcal{N})$-pushout (2) with $l \in \mathcal{L}$, there are unique $\mathcal{R}$-morphisms $b^* : B' \to K$ and $c^* : C' \to D$ such that $l \circ b^* = b'$, $g \circ c^* = c'$ and (3) is a pushout.

$$\begin{tikzcd}
B' \arrow[rr, "b^*"] \arrow[dr, "b'"] \arrow[dd, "e"] & & K \arrow[dl, "l"'] \arrow[dd, "k"] \\
\arrow[r, phantom, "(1)"] & L \arrow[dd, "m"'] \arrow[dd, phantom, "(3)" near start, shift left=3ex] \arrow[r, phantom, "(2)"] & {} \\
C' \arrow[rr, "c^*"] \arrow[dr, "c'"] & & D \arrow[dl, "g"] \\
& G &
\end{tikzcd}$$

Note that $b^* \in \mathcal{R}$ and $c^* \in \mathcal{R}$ is a direct consequence of $b' \in \mathcal{R}$ and $l \in \mathcal{L}$ as well as $c' \in \mathcal{R}$ and $g \in \mathcal{L}$ and the closure of $\mathcal{R}$ under decomposition (note that any $\mathcal{L}$-morphism is also an $\mathcal{R}$-morphism). We call this concept quasi $(\mathcal{L},\mathcal{N})$-initial pushout, as although morphisms $b', c' \in \mathcal{R}$ as well as $b^*, c^* \in \mathcal{R}$ we require for any pushout (3) and $\mathcal{L}$-morphism l that (2) is an $(\mathcal{L},\mathcal{N})$-pushout. Moreover, every $(\mathcal{L},\mathcal{N})$-initial pushout is also a quasi $(\mathcal{L},\mathcal{N})$-initial pushout as by definition $\mathcal{L}$ is a subclass of $\mathcal{R}$.

The following lemma states that any $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category with $(\mathcal{L},\mathcal{N})$-initial pushouts has also quasi $(\mathcal{L},\mathcal{N})$-initial pushouts.

Lemma 5.18 (Quasi $(\mathcal{L},\mathcal{N})$-initial pushouts in $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive categories)**.**
Let $(\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N})$ be an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive category and (1) an $(\mathcal{L},\mathcal{N})$-initial pushout over $\mathcal{N}$-morphism $m : L \to G$, then $(1^\diamond)$+(1) is a quasi $(\mathcal{L},\mathcal{N})$-initial pushout over m if and only if $(1^\diamond)$ is an $(\mathcal{R},\mathcal{N})$-pushout with $b^\diamond \in \mathcal{R}$.

$$\begin{tikzcd}
B' \arrow[r, "b^\diamond"] \arrow[rr, bend left, "b'"] \arrow[d] \arrow[dr, phantom, "(1^\diamond)"] & B \arrow[r, "b"] \arrow[d] \arrow[dr, phantom, "(1)"] & L \arrow[d, "m"] \\
C' \arrow[r, "c^\diamond"] \arrow[rr, bend right, "c'"'] & C \arrow[r, "c"] & G
\end{tikzcd}$$

Proof.
If. We have to show that if $(1^\diamond)$ is an $(\mathcal{R},\mathcal{N})$-pushout with $b^\diamond \in \mathcal{R}$, then $(1^\diamond)$+(1) is a quasi $(\mathcal{L},\mathcal{N})$-initial pushout over m. Assume given $(\mathcal{L},\mathcal{N})$-initial pushout (1) over $m \in \mathcal{N}$ and $(\mathcal{R},\mathcal{N})$-pushout $(1^\diamond)$, then for any $(\mathcal{L},\mathcal{N})$-pushout (2) there exist unique morphisms $b^*, c^* \in \mathcal{L}$ such that (3) is a pushout. Hence, by composing pushouts $(1^\diamond)$ and (3), we obtain pushout $(1^\diamond)$+(3) and unique morphisms $b^* \circ b^\diamond$ and $c^* \circ c^\diamond$ such that $l \circ b^* \circ b^\diamond = b \circ b^\diamond$ and $g \circ c^* \circ c^\diamond = c \circ c^\diamond$. Consequently, $(1^\diamond)$+(1) is a quasi $(\mathcal{L},\mathcal{N})$-initial pushout.

$$\begin{tikzcd}
B' \arrow[r, "b^\diamond"] \arrow[rrr, bend left, "b^* \circ b^\diamond"] \arrow[dd, "e'"'] & B \arrow[rr, "b^*"] \arrow[dr, "b"] \arrow[dd, "e"'] & & K \arrow[dl, "l"'] \arrow[dd, "k"] \\
\arrow[r, phantom, "(1^\diamond)"] & \arrow[r, phantom, "(1)"] & L \arrow[dd, "m", "(3)"'] \arrow[r, phantom, "(2)"] & \\
C' \arrow[r, "c^\diamond"] \arrow[rrr, bend right, "c^* \circ c^\diamond"'] & C \arrow[rr, "c^*"] \arrow[dr, "c"] & & D \arrow[dl, "g"'] \\
& & G &
\end{tikzcd}$$

Only if. We have to show that if (1′) is a quasi $(\mathcal{L},\mathcal{N})$-initial pushout over m, then there exists an $(\mathcal{R},\mathcal{N})$-pushout $(1^\diamond)$. Assume given $(\mathcal{L},\mathcal{N})$-initial pushout (1) and quasi $(\mathcal{L},\mathcal{N})$-initial pushout (1′), both over $\mathcal{N}$-morphism m. From pushout (1), $b, c \in \mathcal{L}$, and the initiality of (1′) follows that there are unique $\mathcal{R}$-morphisms $b^\diamond$ and $c^\diamond$ such that $(1^\diamond)$ is an $(\mathcal{R},\mathcal{N})$-pushout and $(1')=(1^\diamond)+(1)$.

$$\begin{tikzcd}
& B \arrow[dr, "b"] \arrow[dd, "e"] & \\
B' \arrow[ur, dashed, "b^\diamond"] \arrow[rr, "b'"] \arrow[dd, "e'"'] \arrow[r, phantom, "(1^\diamond)"'] & \arrow[r, phantom, "(1)"] & L \arrow[dd, "m"] \\
& C \arrow[dr, "c"] \arrow[l, phantom, "(1')"] & \\
C' \arrow[ur, dashed, "c^\diamond"] \arrow[rr, "c'"] & & G
\end{tikzcd}$$ □

It remains to show that quasi $(\mathcal{L},\mathcal{N})$-initial pushouts are closed under transformations along $(\mathcal{L},\mathcal{R})$-productions.

Lemma 5.19 (Closure property of quasi $(\mathcal{L},\mathcal{N})$-initial pushouts).
Quasi $(\mathcal{L},\mathcal{N})$-initial pushouts are closed under transformations along $(\mathcal{L},\mathcal{R})$-productions; that is, given a quasi $(\mathcal{L},\mathcal{N})$-initial pushout (1′) over $m \in \mathcal{N}$ and the double pushout diagram (2) with pushouts $(2a)$ and $(2b)$ and $l \in \mathcal{L}$, $r \in \mathcal{R}$, then the following holds:

a) The composition of (1′) with $(2a)$, leading to pushout (3), is again a quasi $(\mathcal{L},\mathcal{N})$-initial pushout over k, where pushout (3) is derived from (1) and $(2a)$ using the initiality property of (1) (see Definition 5.17).

b) The composition of quasi $(\mathcal{L},\mathcal{N})$-initial pushout (3) with $(\mathcal{R},\mathcal{N})$-pushout $(2b)$, leading to pushout (4), is a quasi $(\mathcal{L},\mathcal{N})$-initial pushout over n.

$$\begin{tikzcd}
B' \arrow[r, "b'"] \arrow[d] \arrow[dr, phantom, "(1')"] & L \arrow[d, "m"] \\
C' \arrow[r, "c'"] & G
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[d, "m"] \arrow[dr, phantom, "(2a)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dr, phantom, "(2b)"] & R \arrow[d, "n"] & (2) \\
G & D \arrow[l, "g"] \arrow[r, "h"] & H &
\end{tikzcd}$$

$$\begin{tikzcd}
B' \arrow[r, "b^*"] \arrow[d] \arrow[dr, phantom, "(3)"] & K \arrow[d, "k"] \\
C' \arrow[r, "c^*"] & D
\end{tikzcd}
\qquad
\begin{tikzcd}
B' \arrow[r, "r \circ b^*"] \arrow[d] \arrow[dr, phantom, "(4)"] & R \arrow[d, "n"] \\
C' \arrow[r, "h \circ c^*"] & H
\end{tikzcd}$$

Proof. **Item a).** Quasi $(\mathcal{L},\mathcal{N})$-initial pushouts are closed under $(\mathcal{L},\mathcal{N})$-pushouts in the opposite direction; that is, given quasi $(\mathcal{L},\mathcal{N})$-initial pushout (1′) over morphism $m \in \mathcal{N}$ and pushout (2a) with $l \in \mathcal{L}$, then there is a quasi $(\mathcal{L},\mathcal{N})$-initial pushout (3) over $k \in \mathcal{N}$ with $l \circ b^* = b'$ and $g \circ c^* = c'$.

$$\begin{tikzcd}
B' \arrow[d, "e'"'] \arrow[rd, "b'"] \arrow[rrrd, "b^*"] & & & \\
C' \arrow[rd, "c'"'] \arrow[rrrd, "c^*"] \arrow[r, phantom, "(1')"] & L \arrow[d, "m"] \arrow[r, phantom, "(3)"] & (2a) & K \arrow[ll, "l"'] \arrow[d, "k"] \\
& G & & D \arrow[ll, "g"]
\end{tikzcd}
\qquad
\begin{tikzcd}
B' \arrow[d, "e'"'] \arrow[rd, "b'"] \arrow[rrrr, dashed, "b^\diamond"] & & (6) & & B \arrow[ld, "b"'] \arrow[dd] \\
C' \arrow[rd, "c'"'] \arrow[rrrrd, dashed, "c^\diamond"] \arrow[r, phantom, "(1')"] & L \arrow[d, "m"] & (2a) & K \arrow[ll, "l"'] \arrow[dd, "k"] & (5) \\
& G & & & C \arrow[ld, "c"] \\
& & & D \arrow[llu, "g"] &
\end{tikzcd}$$

Assume (5) is the $(\mathcal{L},\mathcal{N})$-initial pushout over $k \in \mathcal{N}$. As (2a) and (5) are $(\mathcal{L},\mathcal{N})$-pushouts, so their composition (2a)+(5). From the initiality of (1), we obtain unique morphisms $b^\diamond : B' \to B$ and $c^\diamond : C' \to C$ such that (6) is an $(\mathcal{R},\mathcal{N})$-pushout. Hence, according to Lemma 5.18, the composition of (6)+(5), leading to (3), is a quasi $(\mathcal{L},\mathcal{N})$-initial pushout over k.

Item b). Quasi $(\mathcal{L},\mathcal{N})$-initial pushouts are closed under $(\mathcal{R},\mathcal{N})$-pushouts in the same direction; that is, given quasi $(\mathcal{L},\mathcal{N})$-initial pushout (3) over morphism $k \in \mathcal{N}$ and pushout (2b) with $r \in \mathcal{R}$ then the composition of pushouts (3) and (2b), leading to pushout (4), is a quasi $(\mathcal{L},\mathcal{N})$-initial pushout over $n \in \mathcal{N}$.

$$\begin{tikzcd}
B' \arrow[d, "e'"'] \arrow[rd, "b^*"] \arrow[rrd, "r \circ b^*"] & & \\
C' \arrow[rd, "c^*"'] \arrow[rrd, "h \circ c^*"] \arrow[r, phantom, "(3)"] & K \arrow[r, "r"] \arrow[d, "k"] & R \arrow[d, "n"] \\
& D \arrow[r, "h"] & H
\end{tikzcd}$$

Assume (7) (in the diagram next) is the $(\mathcal{L},\mathcal{N})$-initial pushout over morphism $n \in \mathcal{N}$. First we construct the pullback along the morphisms $r : K \to R$ and $b : B \to R$ as well as the pullback along morphisms $h : D \to H$ and $c : C \to H$, leading to the top and bottom squares of the cube shown next. As $\mathcal{L}$ as well as $\mathcal{R}$-morphisms are closed under pullbacks, $h, r \in \mathcal{R}$ and $c, b \in \mathcal{L}$ imply $c^\diamond_b, b^\diamond_b \in \mathcal{R}$ and $v, w \in \mathcal{L}$. Morphism $x : V \to W$ is obtained from the universal property of the bottom pullback and morphisms $k \circ v$ and $e \circ b^\diamond$. From the closure of $\mathcal{N}$ under $\mathcal{R}$-composition, we obtain from $e \in \mathcal{N}$ and $b^\diamond \in \mathcal{R}$ that $(e \circ b^\diamond) \in \mathcal{N}$. From the closure of $\mathcal{N}$ under $\mathcal{R}$-decomposition we obtain from $(e \circ b^\diamond) \in \mathcal{N}$ and $c^\diamond_b \in \mathcal{R}$ that $x \in \mathcal{N}$. Hence, all vertical morphisms are in $\mathcal{N}$ and all horizontal morphisms are in $\mathcal{R}$ (note that any $\mathcal{L}$-morphism is also an $\mathcal{R}$-morphisms). Moreover, the bottom and top faces are pullbacks, the front faces (i. e., (2b) and (7)) are pushouts. As b is in $\mathcal{L}$ we may apply the $(\mathcal{L},\mathcal{R},\mathcal{N})$-cube pushout–pullback decomposition property. Accordingly, (8) and (9) are pushouts. As (8) is an $(\mathcal{L},\mathcal{N})$-pushout along $\mathcal{L}$-morphism v, we obtain from the quasi initiality of (3) the unique morphisms $b_a^\diamond : B' \to V$ and $c_a^\diamond : C' \to W$ such that (10) is a pushout. Consequently, the composition of $(\mathcal{R},\mathcal{N})$-pushouts (10) and (9) is again an $(\mathcal{R},\mathcal{N})$-pushout. As (7) is the $(\mathcal{L},\mathcal{N})$-initial pushout over n, it follows from Lemma 5.18 that the composition (10)+(9)+(7), which corresponds to (4) in the previous diagram, is a quasi $(\mathcal{L},\mathcal{N})$-initial pushout over n.

$$\begin{tikzcd}
B' \arrow[dd] \arrow[drrr, dashed, "b_a^\diamond"] \arrow[ddr, "b^*"] & & & & \\
& (10) & & V \arrow[dll, dashed, "v"'] \arrow[dr, dashed, "b_b^\diamond"] \arrow[ddd, dashed, "x"] & \\
C' \arrow[ddrrr, dashed, "c_a^\diamond"] \arrow[dddr, "c^*"'] \arrow[r, phantom, "(3)"] & K \arrow[dr, "r"] \arrow[ddd, "k"] \arrow[ddrr, phantom, "(8)"] & & & B \arrow[dll, "b"'] \arrow[ddd, "e"] \arrow[ddl, phantom, "(9)"] \\
& & R \arrow[ddd, "n"] \arrow[ddrr, phantom, "(7)"] & & \\
& & & W \arrow[dll, dashed, "w"] \arrow[dr, dashed, "c_b^\diamond"] & \\
& D \arrow[dr, "h"'] \arrow[uur, phantom, "(2b)"] & & & C \arrow[dll, "c"] \\
& & H & &
\end{tikzcd}$$

□

5.3 Constraints and Application Conditions

The construction of application conditions from constraints was first introduced in [HW95] for plain graphs. Accordingly, a constraint is first transformed into an equivalent right application condition. Subsequently, the right application condition is transformed into an equivalent left application condition. In this section we show that the HLR$^+$properties for $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive categories are sufficient to prove these techniques for $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation systems. In the following we focus on negative constraints and negative application condition. However, the corresponding constructions and proofs can be easily extended to nested constraints and application conditions (see for example [HP12a, EEPT06, EGH$^+$12]). Table 5.1 summarizes which of the HLR$^+$properties are required for the translation of consistency constraints to application conditions and for the translation of right NACs to left NACs, respectively.

Note that in this section the corresponding techniques are shown in the abstract setting of $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation systems. For detailed examples, we refer to Chapter 7, where these techniques are instantiated for functional projective graph transformation systems.

5.3.1 *Construction of Equivalent Negative Application Conditions*

Basically, for a simple negative constraint $nc(N)$, an equivalent NAC over R is constructed from all gluings of N and R. A gluing Y of two objects N and R is defined as the morphism pair $(R \xrightarrow{y} Y, N \xrightarrow{c} Y)$ such that $(y, c) \in \mathcal{E}'$, i. e., the pair (y, c) is jointly epimorphic (see Definition 3.15 and Lemma 5.11). By constructing

Table 5.1: Overview for the required HLR$^+$properties.

		Negative constraints to NACs	Right NACs to left NACs
HLR$^+$Properties	$\mathbb{C}$ has binary coproducts	x	
	$\mathbb{C}$ has $\mathcal{E}$–$\mathcal{N}$ factorization	x	
	$\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition		x
	$\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition		x
	$(\mathcal{L}, \mathcal{N})$-initial pushouts		

all gluings, we can capture all potential interactions of N and R. Thus, adding all these gluings as simple negative application conditions to NAC_R ensures that for any object H that is inconsistent with respect to negative constraint $nc(N)$, there either does not exist a match of R in H, or all matches do not satisfy NAC_R, as H must contain one of the gluings of N and R.

Definition 5.20 (Construction of negative application conditions from negative constraints for $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories)**.**
Given an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N})$. The construction of a negative application condition over an object R from a simple negative constraint $nc(N)$ is defined as

$$Acc_R(nc(N)) = \bigcup_{i \in \mathbf{I}} \{nac_R(R \xrightarrow{y_i} Y_i)\},$$

where $\mathbf{I}$ ranges over all triples (Y_i, y_i, c_i) with morphisms $y_i : R \rightarrow Y_i$ and $c_i : N \rightarrow Y_i$ such that the pair (y_i, c_i) is in $\mathcal{E}'$.

$$\begin{tikzcd}
R \arrow[d, "y_i"] & \\
Y_i & N \arrow[l, "c_i"]
\end{tikzcd}$$

For a negative constraint NC, the construction is given by

$$Acc_R(NC) = \bigcup Acc_R(nc(N)) \text{ for all } nc(N) \in NC.$$

It remains to show that the construction given in Definition 5.20 indeed leads to equivalent negative application conditions in the following sense:

Theorem 5.21 (Construction of equivalent NACs from negative constrains)**.**
Consider an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N})$ with binary coproducts

and $\mathcal{E}$-$\mathcal{N}$ factorizations, then for any negative constraint NC and every object R in $\mathbb{C}$ with $\mathcal{N}$-morphism $n : R \to H$, we have

$$n \Vdash Acc_R(NC) \text{ iff } H \Vdash NC.$$

Proof. Instead of proving

$$n \Vdash Acc_R(NC) \text{ iff } H \Vdash NC,$$

we equivalently show that

$$n \nVdash Acc_R(NC) \text{ iff } H \nVdash NC.$$

If. Let $n \nVdash Acc_R(NC)$, we have to show that $H \nVdash NC$; that is, for all triples (Y_i, y_i, c_i) derived according to Definition 5.20, and every object H in $\mathbb{C}$ and $\mathcal{N}$-morphisms $n : R \to H$, $n' : Y_i \to H$ such that $n'_i \circ y_i = n$, there has to be an $\mathcal{N}$-morphism $c' : N \to H$.

$$\begin{tikzcd}
R \arrow[d, "y_i"] \arrow[dd, bend right=60, "n"'] & \\
Y_i \arrow[d, "n'"] & N \arrow[l, "c_i"'] \arrow[dl, "c'"] \\
H &
\end{tikzcd}$$

For any triple (Y_i, y_i, c_i) and morphism n' in $\mathcal{N}$ as above, we define $c' = n' \circ c_i$. Then $c' \in \mathcal{N}$ as $\mathcal{N}$ is closed under composition.

Only if. Let $H \nVdash NC$, we have to show that $n \nVdash Acc_R(NC)$; that is, for any simple negative constraint $nc(N) \in NC$ and $\mathcal{N}$-morphisms $c' : N \to H$ and $n : R \to H$, there is a triple (Y_i, y_i, c_i) with morphisms $y_i : R \to Y_i$ and $c_i : N \to Y_i$ such that the pair (y_i, c_i) is in $\mathcal{E}'$; and there is an $\mathcal{N}$-morphism $n' : Y_i \to H$ such that $n' \circ y_i = n$.

As any $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category with binary coproducts and $\mathcal{E}$-$\mathcal{N}$ factorizations has $\mathcal{E}'$–$\mathcal{N}$ pair factorizations (see Lemma 5.11), we may construct the triple (Y_i, y_i, c_i) and morphisms $n' : Y_i \to H$ as the $\mathcal{E}'$–$\mathcal{N}$ pair factorization of n and c'. Consequently, the pair (y_i, c_i) is in $\mathcal{E}'$. As $n, c' \in \mathcal{N}$, so $n' \in \mathcal{N}$ and $n' \circ y_i = n$. □

5.3.2 *Construction of Equivalent Left NACs from Right NACs*

In the following, we present the construction of equivalent left NACs from right NACs for $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems and prove that this construction indeed leads to equivalent application conditions.

Definition 5.22 (Construction of left from right NACs for $(\mathcal{L}, \mathcal{R})$-productions).

Let ϱ be an extended production over $(\mathcal{L}, \mathcal{R})$-production $p = (L \leftarrow K \to R)$

and right negative application condition NAC_R. For a simple right negative application condition $nac_R(R \xrightarrow{y} Y) \in NAC_R$, let

$$\mathit{shift}_\varrho(nac_R((R \xrightarrow{y} Y)) = \{nac_L(L \xrightarrow{x} X)\}$$

be the singleton set constructed from $nac_R(R \xrightarrow{y} Y)$ as follows:

$$\begin{tikzcd}
L \arrow[d, "x"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "z"] \arrow[l, phantom, "(2)", shift right=6ex] & R \arrow[d, "y"] \\
X & Z \arrow[l, "l'"] \arrow[r, "r'"'] & Y
\end{tikzcd}$$

$$\text{(2)} \qquad \text{(1)}$$

If the pair $r : K \to R$ and $y : R \to Y$ has a pushout complement, choose $\mathit{shift}_\varrho(nac_R(R \xrightarrow{y} Y)) = \{nac_L(L \xrightarrow{x} X)\}$, where x is defined by the pushouts (1) and (2); otherwise $\mathit{shift}_\varrho(nac_R(R \xrightarrow{y} Y)) = \emptyset$.
A left NAC is obtained as follows:

$$\mathit{shift}_\varrho(NAC_R) = \bigcup \mathit{shift}_\varrho(nac_R(R \xrightarrow{y_i} Y_i)) \text{ for all } nac_R(R \xrightarrow{y_i} Y_i) \in NAC_R.$$

Theorem 5.23 (Construction of equivalent left NACs from right NACs for $(\mathcal{L}, \mathcal{R})$-productions)**.**
Consider an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive category $(\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N})$ with the $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property. Given extended production $\varrho = (p, \mathit{shift}_\varrho(NAC_R), NAC_R)$ over $(\mathcal{L}, \mathcal{R})$-production p with left NAC $\mathit{shift}_\varrho(NAC_R)$ derived according to Definition 5.22, then for any direct transformation $G \overset{\varrho@m}{\Longrightarrow} H$ via ϱ with match m and comatch n in $\mathcal{N}$ we have

$$m \Vdash \mathit{shift}_p(NAC_R) \text{ iff } n \Vdash NAC_R.$$

Proof. Instead of proving

$$m \Vdash \mathit{shift}_p(NAC_R) \text{ iff } n \Vdash NAC_R,$$

we equivalently show that

$$m \nVdash \mathit{shift}_p(NAC_R) \text{ iff } n \nVdash NAC_R.$$

We start with simple negative application conditions; that is, given an $(\mathcal{L}, \mathcal{R})$-production $p = (L \leftarrow K \to R)$ and a simple negative application condition $nac_R(R \xrightarrow{y} Y)$ over R, we have to show that

$$m \nVdash \mathit{shift}_p(nac_R(R \xrightarrow{y} Y)) \text{ iff } n \nVdash nac_R(R \xrightarrow{y} Y).$$

If. We have tho show that if

$$n \nVdash nac_R(R \xrightarrow{y} Y), \text{ then also } m \nVdash \mathit{shift}_p(nac_R(R \xrightarrow{y} Y)).$$

Given a direct transformation $G \xRightarrow{p@m} H$. As $n \not\models nac_R(R \xrightarrow{y} Y)$, there must be a morphism $n' : Y \to H$ such that $n' \circ y = n$ (as shown in the diagram next). Since $h : D \to H$ is an $\mathcal{R}$-morphism, we can construct (3) as the $\mathcal{R}$-pullback of h and n'. The closure of $\mathcal{N}$ and $\mathcal{R}$ under pullbacks leads to $k' \in \mathcal{N}$ and $r' \in \mathcal{R}$. From the universal pullback property of (3), we obtain unique morphism $z : K \to Z$. As $r \in \mathcal{R}$, $k \in \mathcal{N}$, and $n' \in \mathcal{N}$, we can apply the $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property to pushout (1)+(3) and pullback (3). Accordingly (1) and (3) are pushouts as well as pullbacks. Since $\mathcal{N}$ is closed under decomposition $k, k' \in \mathcal{N}$ implies $z \in \mathcal{N}$. Now we construct object X as the $(\mathcal{L}, \mathcal{N})$-pushout of $L \xleftarrow{l} K \xrightarrow{z} Z$, leading to pushout (2) with $\mathcal{N}$-morphism $x : L \to X$ and $\mathcal{L}$-morphism $l' : Z \to X$. By the universal property of pushout (2), we obtain unique morphism $m' : X \to G$. The decomposition of pushout (2)+(4) with pushout (2), implies that (4) is a pushout. From the closure of $\mathcal{N}$ under pushouts, we know from $k' \in \mathcal{N}$ that also $m' \in \mathcal{N}$. Moreover, as diagrams (2) and (4) commute, we may assume that $m' \circ x = m$; hence, $m \not\models nac_L(L \xrightarrow{x} X)$. As pushouts complements are unique in $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories, the pushouts (1) and (2) are identical to the shift construction. Thus, the simple NAC $nac_L(L \xrightarrow{x} X)$ is identical to the simple NAC $\mathit{shift}_\varrho(nac_R(R \xrightarrow{y} Y))$ (obtained according to Definition 5.22).

$$\begin{tikzcd}
L \arrow[d, dotted, "x"] \arrow[dd, bend right, "m"'] \arrow[dr, phantom, "(2)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, dotted, "z"] \arrow[dd, bend right, "k"'] \arrow[dr, phantom, "(1)"] & R \arrow[d, "y"] \arrow[dd, bend left, "n"] \\
X \arrow[d, dotted, "m'"] \arrow[dr, phantom, "(4)"] & Z \arrow[l, dotted, "l'"'] \arrow[r, dotted, "r'"] \arrow[d, dotted, "k'"] \arrow[dr, phantom, "(3)"] & Y \arrow[d, "n'"] \\
G & D \arrow[l, "g"'] \arrow[r, "h"] & H
\end{tikzcd}$$

Only if. The proof of the "only if" direction can be obtained similarly by, starting with constructing diagram (4).

Consequently, the statement

$$m \not\models \mathit{shift}_p(nac_R(R \xrightarrow{y} Y)) \text{ iff } n \not\models nac_R(R \xrightarrow{y} Y).$$

holds for any simple NAC $nac_R(R \xrightarrow{y} Y)$.

As $\mathit{shift}_\varrho(NAC_R)$ is derived as the union of $\mathit{shift}_\varrho(nac_R(R \xrightarrow{y_i} Y_i))$ for each simple NAC $nac_R(R \xrightarrow{y_i} Y_i) \in NAC_R$, we may conclude that statement

$$m \not\models \mathit{shift}_p(NAC_R) \text{ iff } n \not\models NAC_R$$

is also valid. □

5.4 Local Church–Rosser, Embedding, and Critical Pairs

In this section we show that the classic results of HLR systems can be lifted to $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems. More specifically, we focus on those results that are mandatory for conflict detection and resolution, namely the parallel part of the Local Church–Rosser Theorem, the Embedding and Extension

Theorems, as well as the Completeness of Critical Pairs Lemma. This does not mean that the other results such as the sequential part of the Local Church–Rosser Theorem, the Parallelism Theorem and the Concurrency Theorem do not hold for $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation systems. In fact, after inspecting the corresponding proofs, we are convinced that these results can also be lifted to $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation systems. However, as these results are not required for conflict detection and resolution, we leave them for future work.

Table 5.1 summarizes which of the HLR^{+}properties are required in the proofs of the corresponding results.

Table 5.2: Overview for the required HLR^{+}properties.

		Parallel Local Church–Rosser Theorem	Embedding Theorem	Extension Theorem	Completeness of Critical Pairs Lemma
HLR^{+}Properties	$\mathbb{C}$ has binary coproducts				x
	$\mathbb{C}$ has $\mathcal{E}$–$\mathcal{N}$ factorization				x
	$\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition				
	$\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition				x
	$(\mathcal{L},\mathcal{N})$-initial pushouts		x	x	

Note that in this section these techniques are shown in the abstract setting of $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation systems. For more detailed explanations and motivating examples, we refer to Chapter 8, which discusses the application of these techniques for projective graph transformation systems.

5.4.1 *Parallel Independence and Local Church–Rosser*

We begin with introducing the concept of parallel independence for $(\mathcal{L},\mathcal{R})$-productions, leading to the Parallel Local Church–Rosser Theorem for $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation systems. Intuitively, two direct transformations of the same object are parallel independent if none of the involved transformations deletes an element that is in the match of the other.

Definition 5.24 (Parallel independence).
Let $((\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N}),\mathbf{P})$ be an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation system, then two direct transformations

$$H_1 \overset{p_1@m_1}{\Longleftarrow} G \overset{p_2@m_2}{\Longrightarrow} H_2 \text{ with } p_1, p_2 \in \mathbf{P}$$

are parallel independent if there exist $\mathcal{N}$-morphisms

$$i : L_1 \to D_2 \text{ and } j : L_2 \to D_1$$

such that $g_2 \circ i = m_1$ and $g_1 \circ j = m_2$.

$$\begin{tikzcd}
R_1 \arrow[d, "n_1"] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "k_1"] & L_1 \arrow[dr, "m_1"] \arrow[drrr, "i"] & & L_2 \arrow[dl, "m_2"'] \arrow[dlll, "j"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "k_2"] & R_2 \arrow[d, "n_2"] \\
H_1 & D_1 \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & G & & D_2 \arrow[ll, "g_2"'] \arrow[r, "h_2"] & H_2
\end{tikzcd}$$

Given two parallel independent transformations of the same object, then the result of executing them in parallel is the same as executing them sequentially. This leads us to the Parallel Local Church–Rosser Theorem.

Theorem 5.25 (Parallel Local Church–Rosser Theorem for $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems)**.**
Let $((\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N}), \mathbf{P})$ be an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation system and let

$$H_1 \xLeftarrow{p_1@m_1} G \xRightarrow{p_2@m_2} H_2 \text{ with } p_1, p_2 \in \mathbf{P}$$

be two parallel independent direct transformations, then there is an object H_3 and direct transformations

$$H_1 \xRightarrow{p_2@m_2'} H_3 \xLeftarrow{p_1@m_1'} H_2.$$

Proof. The proof is an adapted version of the corresponding proof shown in [HP12b]. Let $H_1 \xLeftarrow{p_1@m_1} G \xRightarrow{p_2@m_2} H_2$ be parallel independent. Then there are $\mathcal{N}$-morphisms $i : L_1 \to D_2$ and $j : L_2 \to D_1$ such that $g_2 \circ i = m_1$ and $g_1 \circ j = m_2$ (see the figure below).

$$\begin{tikzcd}
R_1 \arrow[d, "n_1"] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "k_1"] & L_1 \arrow[dr, "m_1"] \arrow[drrr, "i"] & & L_2 \arrow[dl, "m_2"'] \arrow[dlll, "j"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "k_2"] & R_2 \arrow[d, "n_2"] \\
H_1 & D_1 \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & G & & D_2 \arrow[ll, "g_2"'] \arrow[r, "h_2"] & H_2
\end{tikzcd}$$

Since $\mathbb{C}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive, it has $\mathcal{L}$-pullbacks (see Corollary 5.3). Hence, D_0 can be constructed as pullback object of $D_1 \xrightarrow{g_1} G \xleftarrow{g_2} D_2$ (as shown in the figure next). Since $\mathcal{L}$ is closed under pullbacks the morphisms $D_0 \to D_i$ (for $i = 1, 2$) are in $\mathcal{L}$. By the universal pullback property, there are unique morphisms $K_i \to D_0$ (for $i = 1, 2$) such that (11) and (31) and the corresponding triangles commute, respectively. By the $\mathcal{R}$–$\mathcal{R}$-PO–PB decomposition property (note that we actually use the $\mathcal{L}$–$\mathcal{L}$-PO–PB decomposition property, see Corollary 5.3), diagrams (11), (12), (31), and (32) are pushouts as well as pullbacks (note that (12) and (32) are identical). Since $\mathcal{N}$ is closed under pullbacks, the morphisms $K_i \to D_0$ $(i = 1, 2)$ are in $\mathcal{N}$. Since $\mathbb{C}$ has $(\mathcal{R}, \mathcal{N})$-pushouts, D_i' $(i = 1, 2)$ can be constructed as the pushouts over $K_i \to R_i$ in $\mathcal{R}$ and $K_i \to D_0$ in $\mathcal{N}$. From the closure of $\mathcal{R}$ and $\mathcal{N}$ under pushouts, we obtain that morphisms $R_1 \to D_2'$ and $R_2 \to D_1'$ are in $\mathcal{N}$ as well as that morphisms $D_0 \to D_2'$ and $D_0 \to D_1'$ are in $\mathcal{R}$. By the universal pushout property, we can construct morphisms $g_2' : D_2' \to H_1$ and $g_1' : D_1' \to H_2$ such that (22) and (42) commute. By pushout decomposition, we can conclude that (22) and (42) are pushouts. The closure of $\mathcal{L}$ under pushouts implies that morphisms g_1' and g_2' are in $\mathcal{L}$. Now all squares in the following figure are pushouts.

$$\begin{tikzcd}
R_1 \arrow[d, dotted] \arrow[dd, bend right, "n_1"'] \arrow[dr, phantom, "(21)"] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, dotted] \arrow[dd, bend right, "k_1"'] \arrow[dr, phantom, "(11)"] & L_1 \arrow[d, "i"] \arrow[ddr, "m_1"] & & L_2 \arrow[d, "j"] \arrow[ddl, "m_2"'] \arrow[dr, phantom, "(31)"] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, dotted] \arrow[dd, bend left, "k_2"] \arrow[dr, phantom, "(41)"] & R_2 \arrow[d, dotted] \arrow[dd, bend left, "n_2"] \\
D_2' \arrow[d, dotted, "g_2'"] \arrow[dr, phantom, "(22)"] & D_0 \arrow[l, dotted] \arrow[r, dotted] \arrow[d, dotted] \arrow[dr, phantom, "(12)"] & D_2 \arrow[dr, "g_2"] & & D_1 \arrow[dl, "g_1"'] \arrow[dr, phantom, "(32)"] & D_0 \arrow[l, dotted] \arrow[r, dotted] \arrow[d, dotted] \arrow[dr, phantom, "(42)"] & D_1' \arrow[d, dotted, "g_1'"] \\
H_1 & D_1 \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & G & & D_2 \arrow[ll, "g_2"'] \arrow[r, "h_2"] & H_2
\end{tikzcd}$$

The pushouts can be rearranged as shown in the figures below. By definition $j, i \in \mathcal{N}$ and $h_1, h_2 \in \mathcal{R}$. As $\mathcal{N}$ is closed under $\mathcal{R}$ composition the matches $m_2' = h_1 \circ j$ and $m_1' = h_2 \circ i$ are in $\mathcal{N}$. As (31), (22), (11), and (42) are pushouts, so (31)+(22) and (11)+(42). The closure of $\mathcal{N}$ under pushouts implies $k_1', k_2' \in \mathcal{N}$. Finally, we obtain H_3 as the pushout object of $D_1' \leftarrow D_0 \rightarrow D_2'$. As (21), (41) and (5) are pushouts, so (21)+(5) and (41)+(5). The closure of $\mathcal{N}$ under pushouts together with $k_1', k_2' \in \mathcal{N}$ implies n_1', n_2' in $\mathcal{N}$. Hence there is a object H_3 and direct transformations $H_1 \xRightarrow{p_2 @ m_2'} H_3 \xLeftarrow{p_1 @ m_1'} H_2$.

$$\begin{tikzcd}
L_1 \arrow[d, "i"] \arrow[dd, bend right, "m_1"'] \arrow[dr, phantom, "(11)"] & K_1 \arrow[l, "l_1"'] \arrow[r, "r_1"] \arrow[d] \arrow[dd, bend right, "k_1"'] \arrow[dr, phantom, "(21)"] & R_1 \arrow[d] \arrow[ddr, "n_1"] & & L_2 \arrow[d, "j"] \arrow[ddl, dotted, "m_2'"'] \arrow[dr, phantom, "(31)"] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d] \arrow[dd, dotted, bend left, "k_2'"] \arrow[dr, phantom, "(41)"] & R_2 \arrow[d] \arrow[dd, dotted, bend left, "n_2'"] \\
D_2 \arrow[d, "g_2"] \arrow[dr, phantom, "(12)"] & D_0 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(22)"] & D_2' \arrow[dr, "g_2'"] & & D_1 \arrow[dl, "h_1"'] \arrow[dr, phantom, "(22)"] & D_0 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(5)"] & D_1' \arrow[d, dotted] \\
G & D_1 \arrow[l, "g_1"'] \arrow[rr, "h_1"] & & H_1 & & D_2' \arrow[ll, "g_2'"'] \arrow[r, dotted, "h_2'"] & H_3
\end{tikzcd}$$

$$\begin{tikzcd}
L_2 \arrow[d, "j"] \arrow[dd, bend right, "m_2"'] \arrow[dr, phantom, "(31)"] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d] \arrow[dd, bend right, "k_2"'] \arrow[dr, phantom, "(41)"] & R_2 \arrow[d] \arrow[ddr, "n_2"] & & L_1 \arrow[d, "i"] \arrow[ddl, dotted, "m_1'"'] \arrow[dr, phantom, "(11)"] & K_1 \arrow[l, "l_1"'] \arrow[r, "r_1"] \arrow[d] \arrow[dd, dotted, bend left, "k_1'"] \arrow[dr, phantom, "(21)"] & R_1 \arrow[d] \arrow[dd, dotted, bend left, "n_1'"] \\
D_1 \arrow[d, "g_1"] \arrow[dr, phantom, "(12)"] & D_0 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(42)"] & D_1' \arrow[dr, "g_1'"] & & D_2 \arrow[dl, "h_2"'] \arrow[dr, phantom, "(42)"] & D_0 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(5)"] & D_2' \arrow[d, dotted] \\
G & D_2 \arrow[l, "g_2"'] \arrow[rr, "h_2"] & & H_2 & & D_1' \arrow[ll, "g_1'"'] \arrow[r, dotted, "h_1'"] & H_3
\end{tikzcd}$$

□

5.4.2 *Embedding and Extension*

The Embedding and Extension Theorems states under what conditions a transformation sequence can be extended to a larger context. The extension of a transformation $t : G_0 \overset{*}{\Longrightarrow} G_n$ to a transformation $t' : G_0' \overset{*}{\Longrightarrow} G_n'$ via an extension morphism $k_0 : G_0 \rightarrow G_0'$, $k_0 \in \mathcal{N}$ is given by an extension diagram.

Definition 5.26 (Extension diagram)**.**
Let $((\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N}), \mathbf{P})$ be an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation system, then diagram (1) is an *extension diagram* over transformation $t : G_0 \overset{*}{\Longrightarrow} G_n$ and *extension morphism* $k_0 : G_0 \to G_0'$, $k_0 \in \mathcal{N}$, where t and t' are transformations via the same sequence of productions $p_0, \ldots, p_n \in \mathbf{P}$ with matches $(m_0, \ldots, m_{n-1})$ and $(k_0 \circ m_0, \ldots, k_{n-1} \circ m_{n-1})$, respectively, given by the double pushout diagrams on the right.

$$\begin{tikzcd}
G_0 \arrow[r, Rightarrow, "t" description, "*"{pos=1}] \arrow[d, "k_0"'] \arrow[dr, phantom, "(1)"] & G_n \arrow[d, "k_n"] \\
G_0' \arrow[r, Rightarrow, "t'" description, "*"{pos=1}] & G_n'
\end{tikzcd}$$

$$\begin{tikzcd}
L_i \arrow[d, "m_i"'] \arrow[dr, phantom, "(PO)"] & K_i \arrow[l, "l_i"'] \arrow[r, "r_i"] \arrow[d, "s_i"] \arrow[dr, phantom, "(PO)"] & R_i \arrow[d, "n_i"] \\
G_i \arrow[d, "k_i"'] \arrow[dr, phantom, "(PO)"] & D_i \arrow[l, "g_i"'] \arrow[r, "h_i"] \arrow[d, "d_i"] \arrow[dr, phantom, "(PO)"] & G_{i+1} \arrow[d, "k_{i+1}"] \\
G_i' & D_i' \arrow[l, "g_i'"'] \arrow[r, "h_i'"] & G_{i+1}'
\end{tikzcd}$$

The following definition of a derived span describes how to combine the changes of a transformation $t : G_0 \overset{*}{\Longrightarrow} G_n$ (i. e., a sequence of direct transformations) into a direct transformation $t : G_0 \Longrightarrow G_n$. In this way, any transformation sequence can be treated like a single transformation step.

Definition 5.27 (Derived span)**.**
Let $((\mathbb{C}, \mathcal{L}, \mathcal{R}, \mathcal{N}), \mathbf{P})$ be an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation system, the derived span of a direct transformation $G \overset{p@m}{\Longrightarrow} H$ with $p \in \mathbf{P}$, is given by

$$der(G \overset{p@m}{\Longrightarrow} H) = G \leftarrow D \rightarrow H.$$

Given a transformation sequence

$$t : G_0 \overset{*}{\Longrightarrow} G_{n-1} \Longrightarrow G_n,$$

via productions in $\mathbf{P}$, and with derived spans

$$s_1 = der(G_0 \overset{*}{\Longrightarrow} G_{n-1}) = (G_0 \leftarrow D' \rightarrow G_{n-1})$$
$$s_2 = der(G_{n-1} \Longrightarrow G_n) = (G_{n-1} \leftarrow D'' \rightarrow G_n)$$

as shown in the following diagram:

$$\begin{tikzcd}
 & & D \arrow[dll, "d_0"'] \arrow[dl, "v"'] \arrow[d, phantom, "(1)"] \arrow[dr, "w"] \arrow[drr, "d_n"] & & \\
G_0 & D' \arrow[l, "g_0"'] \arrow[r, "g_{n-1}"] & G_{n-1} & D'' \arrow[l, "f_{n-1}"'] \arrow[r, "f_n"] & G_n
\end{tikzcd}$$

The derived span

$$der(t) = G_0 \xleftarrow{d_0} D \xrightarrow{d_n} G_n,$$

of transformation sequence t is given by the composition of derived spans s_1 and s_2 via pullback (1), where $d_0 = g_0 \circ v$ and $d_n = f_n \circ w$.

Remark 5.28 (Derived Span)**.**
The derived span is unique up to isomorphism and does not depend on the order of the pullback constructions [EEPT06]. Moreover, given the derived span $der(t) = G_0 \xleftarrow{d_0} D \xrightarrow{d_n} G_n$, then morphism d_0 is in $\mathcal{L}$ and morphism d_n is in $\mathcal{R}$. This can be easily shown by induction, as we may obtain in each step from $f_{n-1} \in \mathcal{L}$ and $g_{n-1} \in \mathcal{R}$, that $v \in \mathcal{L}$ and $w \in \mathcal{R}$ (closure of $\mathcal{L}$ and $\mathcal{R}$ under pullbacks); from $g_0 \in \mathcal{L}$ and $f_n \in \mathcal{R}$ we obtain $d_0 \in \mathcal{L}$ and $d_n \in \mathcal{R}$ (closure under composition).

Based on the notion of initial pushouts we now define consistency and show that consistency is sufficient (Theorem 5.30) and necessary (Theorem 5.31) to guarantee the existence of an extension diagram.

Definition 5.29 (Consistency)**.**
Let $((\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N}),\mathbf{P})$ be an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation system and let $t : G_0 \overset{*}{\Longrightarrow} G_n$ be a transformation sequence via productions in $\mathbf{P}$ with derived span $der(t) = (G_0 \leftarrow D \rightarrow G_n)$.

$$\begin{tikzcd}
B \arrow[r, "b_0"] \arrow[rr, bend left, "b"] \arrow[d] \arrow[dr, phantom, "(1)"] & G_0 \arrow[d, "k_0"] & D \arrow[l, "d_0"'] \arrow[r, "d_n"] & G_n \\
C \arrow[r] & G_0' & &
\end{tikzcd}$$

A morphism $k_0 : G_0 \rightarrow G_0'$, $k_0 \in \mathcal{N}$ is consistent with respect to transformation t if there exists a quasi $(\mathcal{L},\mathcal{N})$-initial pushout (1) over k_0 and a morphism $b \in \mathcal{R}$ with $d_0 \circ b = b_0$.

Theorem 5.30 (Embedding Theorem)**.**
Let $((\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N}),\mathbf{P})$ be an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation system, where $\mathbb{C}$ has $(\mathcal{L},\mathcal{N})$-initial pushouts. Given transformation $t : G_0 \overset{*}{\Longrightarrow} G_n$ via productions in $\mathbf{P}$ and an $\mathcal{N}$-morphisms $k_0 : G_0 \rightarrow G_0'$ such that k_0 is consistent with respect to transformation t, then there is an extension diagram over t and k_0.

Proof. If $\mathbb{C}$ has $(\mathcal{L},\mathcal{N})$-initial pushouts, it has also quasi $(\mathcal{L},\mathcal{N})$-initial pushouts (see Lemma 5.18). The proof is similar to the proof of Theorem 6.14 in [EEPT06]. More specifically, the proof can be obtained from the proof in [EEPT06] by using quasi $(\mathcal{L},\mathcal{N})$-initial pushouts instead of initial pushouts. □

Theorem 5.31 (Extension Theorem)**.**
Let $((\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N}),\mathbf{P})$ be an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation system with

$(\mathcal{L}, \mathcal{N})$-initial pushouts, given a transformation $t : G_0 \overset{*}{\Longrightarrow} G_n$ via productions in **P** with derived span

$$der(t) = (G_0 \xleftarrow{d_0} D \xrightarrow{d_n} G_n)$$

and extension diagram (1)

$$\begin{tikzcd}
B \arrow[r, "b_0"] \arrow[d] \arrow[rd, phantom, "(2)"] & G_0 \arrow[r, Rightarrow, "t", "*"'] \arrow[d, "k_0"] \arrow[rd, phantom, "(1)"] & G_n \arrow[d, "k_n"] \\
C \arrow[r] & G_0' \arrow[r, Rightarrow, "t'", "*"'] & G_n'
\end{tikzcd}$$

with $(\mathcal{L}, \mathcal{N})$-initial pushout (2), then we have the following:

a) Morphism k_0 is consistent with respect to transformation t.

b) There is a direct transformation $G_0' \overset{der(t)@k_0}{\Longrightarrow} G_n'$ given by the following double pushout diagram.

$$\begin{tikzcd}
G_0 \arrow[d, "k_0"] \arrow[rd, phantom, "(3)"] & D \arrow[l, "d_0"'] \arrow[r, "d_n"] \arrow[d, "h"] \arrow[rd, phantom, "(4)"] & G_n \arrow[d, "k_n"] \\
G_0' & D' \arrow[l] \arrow[r] & G_n'
\end{tikzcd}$$

c) There are quasi $(\mathcal{L}, \mathcal{N})$-initial pushouts (5) and (6).

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[d] \arrow[rd, phantom, "(5)"] & D \arrow[d, "h"] \\
C \arrow[r] & D'
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[r, "d_n \circ b"] \arrow[d] \arrow[rd, phantom, "(6)"] & G_n \arrow[d, "k_n"] \\
C \arrow[r] & G_n'
\end{tikzcd}$$

Proof. If $\mathbf{C}$ has $(\mathcal{L}, \mathcal{N})$-initial pushouts, it has also quasi $(\mathcal{L}, \mathcal{N})$-initial pushouts (see Lemma 5.18). The proof is similar to the proof of Theorem 6.16 in [EEPT06]. More specifically, the proof can be obtained from the proof in [EEPT06] by using quasi $(\mathcal{L}, \mathcal{N})$-initial pushouts instead of initial pushouts. □

5.4.3 *Critical pairs*

Now we define symbolic critical pairs and show that symbolic critical pairs are complete; that is, for any pair of parallel dependent transformations, there exists a critical pair that can be extended to the corresponding pair of parallel dependent transformations.

Definition 5.32 (Critical pair)**.**
Let $((\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N}),\mathbf{P})$ be an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation system, a *critical pair* is a pair of parallel depended direct transformations

$$P_1 \overset{p_1@o_1}{\Longleftarrow} K \overset{p_2@o_2}{\Longrightarrow} P_2$$

with $p_1, p_2 \in \mathbf{P}$ such that the morphism pair (o_1, o_2) is in $\mathcal{E}'$.

Lemma 5.33 (Completeness of critical pairs)**.**
Let $((\mathbb{C},\mathcal{L},\mathcal{R},\mathcal{N}),\mathbf{P})$ be an $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive transformation system with an $\mathcal{E}$–$\mathcal{N}$ factorizations, binary coproducts, and $\mathcal{L}$–$\mathcal{N}$-pushout–pullback decomposition. The critical pairs are then complete. This means that for each pair of parallel dependent direct transformations

$$H_1 \overset{p_1@m_1}{\Longleftarrow} G \overset{p_2@m_2}{\Longrightarrow} H_2,$$

with $p_1, p_2 \in \mathbf{P}$, there exists a symbolic critical pair

$$P_1 \overset{p_1@o_1}{\Longleftarrow} K \overset{p_2@o_2}{\Longrightarrow} P_2$$

with the following extension diagrams (1) and (2) over extension morphism m.

$$\begin{tikzcd}
P_1 \arrow[d] & K \arrow[l, Rightarrow] \arrow[r, Rightarrow] \arrow[d, "m"] \arrow[dl, phantom, "(1)"] \arrow[dr, phantom, "(2)"] & P_2 \arrow[d] \\
H_1 & G \arrow[l, Rightarrow] \arrow[r, Rightarrow] & H_2
\end{tikzcd}$$

Proof. The following proof is an extended version of Lemma 6.22 in [EEPT06].

As $\mathbb{C}$ has $\mathcal{E}$–$\mathcal{N}$ factorization and binary coproducts, it has also an $\mathcal{E}'$–$\mathcal{N}$ pair factorization (see Lemma 5.11). From the $\mathcal{E}'$–$\mathcal{N}$ pair factorization for m_1 and m_2, we obtain object K and morphisms $o_1 : L_1 \to K$, $o_2 : L_2 \to K$ with $(o_1, o_2) \in \mathcal{E}'$, and $m : K \to G$ such that $m_1 = m \circ o_1$ and $m_2 = m \circ o_2$. As $m_1, m_2 \in \mathcal{N}$, so $m, o_1, o_2 \in \mathcal{N}$ (see Lemma 5.11).

$$\begin{tikzcd}
R_1 \arrow[dd, bend right, "n_1"'] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[dd, bend left, "k_1"] & L_1 \arrow[dr, "o_1"] \arrow[ddr, bend right, "m_1"'] & & L_2 \arrow[dl, "o_2"'] \arrow[ddl, bend left, "m_2"] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[dd, bend right, "k_2"'] & R_2 \arrow[dd, bend left, "n_2"] \\
& & & K \arrow[d, "m"] & & & \\
H_1 & D_1 \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & G & & D_2 \arrow[ll, "g_2"'] \arrow[r, "h_2"] & H_2
\end{tikzcd}$$

To construct the required extension diagram we first construct the pullback (3) over g_1 and m and derive the induced morphism t_1. By applying the $\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition property, we find that both squares (3) and (4) are pushouts, because

$l_1 \in \mathcal{L}$ and $m \in \mathcal{N}$. The closure of $\mathcal{N}$ under pushouts implies that t_1 and s_1 are in $\mathcal{N}$, because $o_1, m \in \mathcal{N}$.

$$\begin{tikzcd}
R_1 \arrow[dd, bend right, "n_1"'] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "t_1"] \arrow[dd, bend left, "k_1"] \arrow[dr, phantom, "(4)"] & L_1 \arrow[dr, "o_1"] \arrow[ddr, "m_1"'] & & L_2 \arrow[dl, "o_2"'] \arrow[ddl, "m_2"] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "t_2"] \arrow[dd, bend right, "k_2"'] & R_2 \arrow[dd, bend left, "n_2"] \\
& N_1 \arrow[rr, "v_1"] \arrow[d, "s_1"] \arrow[dr, phantom, "(3)"] & & K \arrow[d, "m"] & & N_2 \arrow[ll, "v_2"'] \arrow[d, "s_2"] & \\
H_1 & D_1 \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & G & & D_2 \arrow[ll, "g_2"'] \arrow[r, "h_2"] & H_2
\end{tikzcd}$$

Now we construct pushout (5) as the $(\mathcal{R}, \mathcal{N})$-pushout over $r_1 \in \mathcal{R}$ and $t_1 \in \mathcal{N}$ and derive the induced morphism z_1. By pushout decomposition, the square (6) is a pushout. We apply the same construction to the second transformations. This results in the following extension diagrams, where the lower part corresponds to the required extension diagrams (1) and (2) with $m \in \mathcal{N}$ (given in the definition of Lemma 5.33).

$$\begin{tikzcd}
R_1 \arrow[d, "u_1"] \arrow[dd, bend right, "n_1"'] \arrow[dr, phantom, "(5)"] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "t_1"] \arrow[dd, bend left, "k_1"] & L_1 \arrow[dr, "o_1"] \arrow[ddr, "m_1"'] & & L_2 \arrow[dl, "o_2"'] \arrow[ddl, "m_2"] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "t_2"] \arrow[dd, bend right, "k_2"'] & R_2 \arrow[d, "u_2"] \arrow[dd, bend left, "n_2"] \\
P_1 \arrow[d, "z_1"] \arrow[dr, phantom, "(6)"] & N_1 \arrow[l, "w_1"'] \arrow[rr, "v_1"] \arrow[d, "s_1"] & & K \arrow[d, "m"] & & N_2 \arrow[ll, "v_2"'] \arrow[r, "w_2"] \arrow[d, "s_2"] & P_2 \arrow[d, "z_2"] \\
H_1 & D_1 \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & G & & D_2 \arrow[ll, "g_2"'] \arrow[r, "h_2"] & H_2
\end{tikzcd}$$

Finally, we show that $P_1 \Leftarrow K \Rightarrow P_2$ is a critical pair. We know from construction that $(o_1, o_2) \in \mathcal{E}'$. It remains to show that the pair $P_1 \xLeftarrow{p_1@o_1} K \xRightarrow{p_2@o_2} P_2$ is parallel dependent. Assume there are morphisms $i : L_1 \to N_2$ and $j : L_2 \to N_1$ with $v_2 \circ i = o_1$ and $v_1 \circ j = o_2$. Then $g_2 \circ s_2 \circ i = m \circ v_2 \circ i = m \circ o_1 = m_1$ and $g_1 \circ s_1 \circ j = m \circ v_1 \circ j = m \circ o_2 = m_2$, which means that $H_1 \xLeftarrow{p_1@m_1} G \xRightarrow{p_2@m_2} H_2$ are parallel independent, which is a contradiction. Consequently, $P_1 \Leftarrow K \Rightarrow P_2$ is a critical pair. □

6

PROJECTIVE GRAPH TRANSFORMATION SYSTEMS ARE $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-ADHESIVE

In this chapter we prove that the theoretical results shown for $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems apply also for typed projective graph transformation systems. To this end, we prove in Section 6.1 that typed projective graph transformation systems are $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive. Subsequently, we show in Section 6.2 that typed projective graph transformation systems provide the defined HLR$^+$properties.

6.1 HLR Properties for Projective Graph Transformation Systems

We begin with proving the required properties for untyped projective graph transformation systems. Subsequently, we use the closure of $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive categories (see Theorem 5.5) to lift the results to typed projective graph transformation systems. In order to show that projective graph transformation systems are $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive we need to show that the category of symbolic graphs with morphism classes $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj}$, and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive, where:

- $\mathcal{M}^{bij}_{\Leftrightarrow}$, is the class of all symbolic graph morphisms $l : \langle G, \Phi_G \rangle \to \langle H, \Phi_H \rangle$ that are injective for graph nodes and all kind of edges, bijective for label nodes, and $\mathcal{D} \models \Phi_H \Leftrightarrow \Phi_G[\hat{l}]$.
- $\mathcal{M}^{inj}_{Proj}$, is the class of projection morphisms (see Definition 4.1).
- $\mathcal{M}^{inj}_{\Rightarrow}$ is the class of all symbolic graph morphisms $m : \langle G, \Phi_G \rangle \to \langle H, \Phi_H \rangle$ that are injective for all kind of nodes and edges, such that $\mathcal{D} \models \Phi_H \Rightarrow \Phi_G[\hat{m}]$.

To actually show that category $\mathbb{SG}_{\mathcal{D}}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive, we have to verify that the properties given in Definition 5.1 are valid for this choice of morphism classes.

For technical reasons, we begin with proving that $\mathbb{SG}_{\mathcal{D}}$ has $(\mathcal{R}, \mathcal{N})$-pushouts and $\mathcal{R}$-pullbacks as well as that morphism classes $\mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{M}^{inj}_{Proj}$, $\mathcal{M}^{inj}_{\Rightarrow}$ are closed under pushouts and pullbacks. According to Facts 3.56 and 3.58, the category $\mathbb{SG}_{\mathcal{D}}$ has pushouts and pullbacks along arbitrary symbolic graph monomorphisms. Consequently, it remains to verify the closure properties for pushouts and pullbacks.

Lemma 6.1 (Closure properties for pushouts)**.**
The morphism classes $\mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{M}^{inj}_{Proj}$, and $\mathcal{M}^{inj}_{\Rightarrow}$ are closed under pushouts.

Proof. As the category $\mathbb{SG}_{\mathcal{D}}$ with $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow}$ and $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow}$ is $(\mathcal{M},\mathcal{N})$-adhesive (Fact 3.64), we know that $\mathcal{M}^{bij}_{\Leftrightarrow}$ is closed under pushouts (Definition 3.18). The closure of $\mathcal{M}^{inj}_{\Rightarrow}$ under pushouts is a direct consequence of the closure of E-graph monomorphisms under pushouts and the fact that any symbolic graph monomorphism is in $\mathcal{M}^{inj}_{\Rightarrow}$, as class $\mathcal{M}^{inj}_{\Rightarrow}$ do not claim stronger restrictions on the involved Σ-formulas than symbolic graph morphisms. Hence, it remains to show that $\mathcal{M}^{inj}_{Proj}$ is closed under pushouts.

Given pushout (1) with $f \in \mathcal{M}^{inj}_{Proj}$ and $g \in \mathcal{M}^{inj}_{\Rightarrow}$, we show that $f' \in \mathcal{M}^{inj}_{Proj}$, by verifying that $\mathcal{D} \models \Phi_C \Leftrightarrow \exists d_1 \dots \exists d_n.\Phi_D$ where $\{d_1 \dots d_n\} = X_D \backslash f'_X(X_C)$ (see Remark 4.3).

$$\begin{tikzcd}
\langle A, \Phi_A \rangle \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & \langle B, \Phi_B \rangle \arrow[d, "g'"] \\
\langle C, \Phi_C \rangle \arrow[r, "f'"] & \langle D, \Phi_D \rangle
\end{tikzcd}$$

As f and g are E-graph monomorphisms, the mappings for the label nodes $f_X : X_A \to X_B$ and $g_X : X_A \to X_C$ are injective. Hence, we may assume without loss of generality that $X_A \subseteq X_B$ and $X_A \subseteq X_C$; so, $f_X(a) = a$ and $g_X(a) = a$ for all $a \in X_A$. Moreover, we may assume that $X_B = X_A \cup X^*_B$ and $X_C = X_A \cup X^*_C$, where $X^*_B = X_B \backslash f_X(X_A) = X_B \backslash X_A$ and $X^*_C = X_C \backslash g_X(X_A) = X_C \backslash X_A$, such that $X^*_B \cap X^*_C = \emptyset$.

As (1) is a pushout in $\mathbb{EG}$, and pushouts in $\mathbb{EG}$ are defined componentwise in $\mathbb{Set}$, we may assume by the injectivity of f_X that $X_D = X_C \,\dot{\cup}\, (X_B \backslash f_X(X_A))$ (Fact 3.9.b), which is equivalent to $X_D = X_A \cup X^*_C \cup X^*_B$ (as $X^*_B \cap X^*_C = \emptyset$).

According to Fact 3.9.a, monomorphisms are closed under pushouts in $\mathbb{EG}$; thus, also f'_X and g'_X are injective and we may assume without loss of generality that $f'_X(c) = c$ for all $c \in X_C$ as well as $g'_X(b) = b$ for all $b \in X_B$. Hence, we have

$$X_D \backslash f'_X(X_C) = (X_A \cup X^*_C \cup X^*_B) \backslash (X_A \cup X^*_C) = X^*_B.$$

Thus, proving

$$\mathcal{D} \models \Phi_C \Leftrightarrow \exists d_1 \dots \exists d_n.\Phi_D \text{ where } \{d_1 \dots d_n\} = X_D \backslash f'_X(X_C)$$

becomes equivalent to show

$$\mathcal{D} \models \Phi_C \Leftrightarrow \exists b^*_1 \dots \exists b^*_n.\Phi_D, \text{ where } \{b^*_1 \dots b^*_n\} = X^*_B. \tag{6.1}$$

From pushout (1) we obtain

$$\mathcal{D} \models \Phi_D \Leftrightarrow (\Phi_C[\hat{f}'] \wedge \Phi_B[\hat{g}']),$$

which is equivalent to

$$\mathcal{D} \models \Phi_D \Leftrightarrow (\Phi_C \wedge \Phi_B), \tag{6.2}$$

because $f'_X(c) = c$ for all $c \in X_C$ as well as $g'_X(b) = b$ for all $b \in X_B$.
By combining Statements (6.1) and (6.2) we obtain

$$\mathcal{D} \models \Phi_C \Leftrightarrow \exists b^*_1 \dots \exists b^*_n.(\Phi_C \wedge \Phi_B), \text{ where } \{b^*_1 \dots b^*_n\} = X^*_B.$$

As Φ_C does not have free variables in X^*_B, this is equivalent to

$$\mathcal{D} \models \Phi_C \Leftrightarrow \Phi_C \wedge \exists b^*_1 \ldots \exists b^*_n.\Phi_B, \text{ where } \{b^*_1 \ldots b^*_n\} = X^*_B. \tag{6.3}$$

The "($\Leftarrow$)" direction of Statement (6.3) trivially holds; the "($\Rightarrow$)" direction of Statement (6.3) is equivalent to show that

$$\mathcal{D} \models \Phi_C \Rightarrow \exists b^*_1 \ldots \exists b^*_n.\Phi_B, \text{ where } \{b^*_1 \ldots b^*_n\} = X^*_B. \tag{6.4}$$

Due to the fact that g is a symbolic graph morphism and f a projection morphism, we know that

$$\mathcal{D} \models \Phi_C \Rightarrow \Phi_A \tag{6.5}$$

and

$$\mathcal{D} \models \Phi_A \Leftrightarrow \exists b^*_1 \ldots \exists b^*_n.\Phi_B, \text{ where } \{b^*_1 \ldots b^*_n\} = X^*_B. \tag{6.6}$$

By inserting (6.6) in (6.5) we obtain

$$\mathcal{D} \models \Phi_C \Rightarrow \exists b^*_1 \ldots \exists b^*_n.\Phi_B, \text{ where } \{b^*_1 \ldots b^*_n\} = X^*_B,$$

which is equivalent to Statement (6.4); thus, f' is in $\mathcal{M}^{inj}_{Proj}$. □

Now we prove that $\mathcal{M}^{inj}_{Proj}$ is closed under pullbacks.

Lemma 6.2 (Closure properties for pullbacks)**.**
$\mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{M}^{inj}_{Proj}$, and $\mathcal{M}^{inj}_{\Rightarrow}$ are closed under pullbacks.

Proof. As the category $\mathbb{SG}_{\mathcal{D}}$ with $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow}$ and $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow}$ is $(\mathcal{M},\mathcal{N})$-adhesive (Fact 3.64), we know that $\mathcal{M}^{bij}_{\Leftrightarrow}$ is closed under pullbacks. The closure of $\mathcal{M}^{inj}_{\Rightarrow}$ under pullbacks is a direct consequence of the closure of E-graph monomorphisms under pullbacks, as $\mathcal{M}^{inj}_{\Rightarrow}$-morphisms do not claim stronger restrictions on the involved Σ-formulas than symbolic graph morphisms. Hence, it remains to show that $\mathcal{M}^{inj}_{Proj}$ is closed under pullbacks.

Let (1) be a pullback in $\mathbb{SG}_{\mathcal{D}}$ with $f \in \mathcal{M}^{inj}_{Proj}$ (given in the diagram shown next), we show that $f' \in \mathcal{M}^{inj}_{Proj}$ by verifying the projection property of f'; that is, for any symbolic graph $\langle Z, \Phi_Z\rangle$ with E-graph morphisms $z : Z \rightarrow B$ and $z' : Z \rightarrow A$ such that $z = f' \circ z'$, we have to show that z is a symbolic graph morphism if and only if z' is a symbolic graph morphism.

$$\begin{tikzcd}
\langle Z,\Phi_Z\rangle \arrow[dr, "z'"] \arrow[drr, "z"] \arrow[ddr, "v'"'] \arrow[ddrr, bend left=60, "v"] & & \\
& \langle A,\Phi_A\rangle \arrow[r, "f'"] \arrow[d, "g'"'] \arrow[dr, phantom, "(1)"] & \langle B,\Phi_B\rangle \arrow[d, "g"] \\
& \langle C,\Phi_C\rangle \arrow[r, "f"] & \langle D,\Phi_D\rangle
\end{tikzcd}$$

If. We have to show for any symbolic graph $\langle Z, \Phi_Z \rangle$ with E-graph morphisms z and z' such that $z = f' \circ z'$, that if z' is a symbolic graph morphism, then also z. This, trivially holds as $z = f' \circ z'$ and f' and z' are symbolic graph morphisms.

Only if. Given a symbolic graph $\langle Z, \Phi_Z \rangle$ with symbolic graph morphism z and E-graph morphism z' such that $z = f' \circ z'$ (in $\mathbb{EG}$), we have to show that z' is a symbolic graph morphism. First, we derive E-graph morphism v' as $v' = g' \circ z'$ and symbolic graph morphism v as $v = g \circ z$. As (1) is a pullback, we have $f \circ v' = v$. Hence, we can use the projection property of f with symbolic graph morphism v to conclude that E-graph morphism v' is also a symbolic graph morphism. By the universal pullback property of (1) we obtain from symbolic graph morphism v' and z the unique symbolic graph morphism $x : \langle Z, \Phi_Z \rangle \to \langle A, \Phi_A \rangle$ such that $x = f' \circ z'$ and $x = g' \circ z'$. As also $z = f' \circ z'$ and $z = g' \circ v'$ we obtain from the uniqueness of x that $z' = x$. Hence z' is a symbolic graph morphism. □

Now we show the closure properties for morphism classes $\mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{M}^{inj}_{Proj}$, and $\mathcal{M}^{inj}_{\Rightarrow}$ (i. e., Properties 1a–1f of Definition 5.1).

Lemma 6.3 (Closure under composition and decomposition)**.**
Given the category $\mathbb{SG}_{\mathfrak{D}}$ with morphism classes $\mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{M}^{inj}_{Proj}$, and $\mathcal{M}^{inj}_{\Rightarrow}$, the following properties hold:

1a) $\mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{M}^{inj}_{Proj}$ and $\mathcal{M}^{inj}_{\Rightarrow}$ contain all isomorphisms.

1b) $\mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{M}^{inj}_{Proj}$ and $\mathcal{M}^{inj}_{\Rightarrow}$ are closed under composition.

1c) $\mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{M}^{inj}_{Proj}$ and $\mathcal{M}^{inj}_{\Rightarrow}$ are closed under decomposition.

1d) $\mathcal{M}^{bij}_{\Leftrightarrow}$ is a subclass of $\mathcal{M}^{inj}_{Proj}$.

1e) $\mathcal{M}^{inj}_{\Rightarrow}$ is closed under $\mathcal{M}^{inj}_{Proj}$-composition.

1f) $\mathcal{M}^{inj}_{\Rightarrow}$ is closed under $\mathcal{M}^{inj}_{Proj}$-decomposition.

Proof. Properties 1a and 1d are straightforward to prove. Properties 1b and 1c for $\mathcal{M}^{bij}_{\Leftrightarrow}$-morphisms are direct consequences of the $(\mathcal{M}, \mathcal{N})$-adhesivity of $\mathbb{SG}_{\mathfrak{D}}$ with $\mathcal{M} = \mathcal{M}^{bij}_{\Leftrightarrow}$ and $\mathcal{N} = \mathcal{M}^{*}_{\Rightarrow}$) (see Fact 3.64); Properties 1b and 1c for $\mathcal{M}^{inj}_{\Rightarrow}$-morphisms follow from the closure of E-graph monomorphisms under composition and decomposition and the fact that any symbolic graph monomorphism is in $\mathcal{M}^{inj}_{\Rightarrow}$. Properties 1e and 1f follow directly from the closure of $\mathcal{M}^{inj}_{\Rightarrow}$ under composition and decomposition as any $\mathcal{M}^{inj}_{Proj}$-morphism is also an $\mathcal{M}^{inj}_{\Rightarrow}$-morphism. Hence, also their composition.

It remains to verify Propertiy 1b and 1c for $\mathcal{M}^{inj}_{Proj}$-morphisms; that is, for any pair of symbolic graph morphisms

$$f : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle,\ g : \langle B, \Phi_B \rangle \to \langle C, \Phi_C \rangle$$

we have to show:

Closed under composition (Property 1b): Given morphisms $f, g \in \mathcal{M}^{inj}_{Proj}$, we show that also $(g \circ f) \in \mathcal{M}^{inj}_{Proj}$, by verifying the projection property of $(g \circ f)$; that is, given symbolic graph $\langle Z, \Phi_Z\rangle$ with E-graph morphisms $z : Z \to C$ and $z' : Z \to A$ such that $z = g \circ f \circ z'$, we have to show that z is a symbolic graph morphism if and only if z' is a symbolic graph morphism.

If. We may define z as $z = g \circ f \circ z'$. Hence z is a symbolic graph morphism as z and $g \circ f$ are symbolic graph morphisms.

$$\begin{tikzcd}
\langle Z,\Phi_Z\rangle \arrow[dr, dashed, "z'"] \arrow[drrr, "z"] \arrow[ddrr, bend right, "z''"'] & & & \\
& \langle A,\Phi_A\rangle \arrow[rr, "g\circ f"] \arrow[dr, "f"] & & \langle C,\Phi_C\rangle \\
& & \langle B,\Phi_B\rangle \arrow[ur, "g"'] &
\end{tikzcd}$$

Only if. Given symbolic graph morphism z and E-graph morphism z' such that $z = g \circ f \circ z'$ in $\mathbb{EG}$. First we derive E-graph morphism z'' as $z'' = f \circ z'$. As z is a symbolic graph morphism and $g \circ z'' = z$, we know by the projection property of g that z'' is a symbolic graph morphism. Analogously, from the projection property of f we can conclude that z' is a symbolic graph morphism, as z'' is a symbolic graph morphism.

Closed under decomposition (Property 1c): Given symbolic graph morphisms g and f such that $(g \circ f) \in \mathcal{M}^{inj}_{Proj}$ and $g \in \mathcal{M}^{inj}_{Proj}$, we have to show that $f \in \mathcal{M}^{inj}_{Proj}$. In fact, this is a consequence of the closure of $\mathcal{M}^{inj}_{Proj}$ under pullbacks. Given commuting diagrams (1) and (2) below, according to Fact 3.14.c the diagrams (1) and (2) are pullbacks in $\mathbb{SG}_{\mathfrak{D}}$, respectively. By pullback composition (1)+(2) is a pullback. As $(g \circ f) \in \mathcal{M}^{inj}_{Proj}$, and $\mathcal{M}^{inj}_{Proj}$ is closed under pullbacks we have $f = (id_B \circ f) \in \mathcal{M}^{inj}_{Proj}$.

$$\begin{tikzcd}
\langle A,\Phi_A\rangle \arrow[rr, "g\circ f"] \arrow[dr, "f"'] & & \langle B,\Phi_C\rangle \\
& \langle B,\Phi_B\rangle \arrow[ur, "g"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
\langle A,\Phi_A\rangle \arrow[r, "f"] \arrow[d, "id_A"'] \arrow[dr, phantom, "(1)"] & \langle B,\Phi_B\rangle \arrow[r, "id_B"] \arrow[d, "id_B"'] \arrow[dr, phantom, "(2)"] & \langle B,\Phi_B\rangle \arrow[d, "g"] \\
\langle A,\Phi_A\rangle \arrow[r, "f"] & \langle B,\Phi_B\rangle \arrow[r, "g"] & \langle C,\Phi_C\rangle
\end{tikzcd}$$

□

To verify Property 3 of Definition 5.1, we have to show that $(\mathcal{R}, \mathcal{N})$-pushouts with $\mathcal{R} = \mathcal{M}^{inj}_{Proj}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow}$ are $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-VK squares.

Lemma 6.4 ($(\mathcal{R}, \mathcal{N})$-pushouts are $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-VK squares)**.**
In $\mathbb{SG}_{\mathfrak{D}}$ with $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow}$, $(\mathcal{R}, \mathcal{N})$-pushouts are $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-VK squares.

Proof. **If.** Consider the commutative cube (5) shown next, where the back and front faces are pullbacks in $\mathbb{SG}_{\mathcal{D}}$, the bottom face is a pushout in $\mathbb{SG}_{\mathcal{D}}$ along $\mathcal{M}^{inj}_{Proj}$-morphism m and $\mathcal{M}^{inj}_{\Rightarrow}$-morphism f, and we have $c, d, b \in \mathcal{M}^{inj}_{Proj}$, $f \in \mathcal{M}^{inj}_{\Rightarrow}$, as well as $a \in \mathcal{M}^{bij}_{\Leftrightarrow}$. We have to show that the top face is a pushout in $\mathbb{SG}_{\mathcal{D}}$.

$$\begin{tikzcd}
& \langle A', \Phi'_A \rangle \arrow[dl, "f'"'] \arrow[dr, "m'"] \arrow[ddd, "a"] & \\
\langle C', \Phi'_C \rangle \arrow[dr, "n'"'] \arrow[ddd, "c"'] & & \langle B', \Phi'_B \rangle \arrow[dl, "g'"] \arrow[ddd, "b"] \\
& \langle D', \Phi'_D \rangle \arrow[ddd, "d"] & \\
& \langle A, \Phi_A \rangle \arrow[dl, "f"'] \arrow[dr, "m"] & \\
\langle C, \Phi_C \rangle \arrow[dr, "n"'] & & \langle B, \Phi_B \rangle \arrow[dl, "g"] \\
& \langle D, \Phi_D \rangle &
\end{tikzcd}$$

(5)

As $c, d, b \in \mathcal{M}^{inj}_{Proj}$ and $f \in \mathcal{M}^{inj}_{\Rightarrow}$, they can be considered as monomorphisms in $\mathbb{EG}$. Consequently, we may assume that the top face is a pushout in $\mathbb{EG}$. Hence, according to Fact 3.56 it is sufficient to verify

$$\mathcal{D} \models \Phi'_D \Leftrightarrow (\Phi'_C[\hat{n}'] \wedge \Phi'_B[\hat{g}']), \tag{6.7}$$

in order to show that the top face is also a pushout in $\mathbb{SG}_{\mathcal{D}}$.

We have the following properties for the cube (6):

- $g, f', g' \in \mathcal{M}^{inj}_{\Rightarrow}$, as $f \in \mathcal{M}^{inj}_{\Rightarrow}$ and $\mathcal{M}^{inj}_{\Rightarrow}$ is closed under pushouts and pullbacks.
- $m', n, n' \in \mathcal{M}^{inj}_{Proj}$, as $m \in \mathcal{M}^{inj}_{Proj}$ and $\mathcal{M}^{inj}_{Proj}$ is closed under pushouts and pullbacks.

Note that all morphisms are injective for the label node components; a_X is in addition bijective for label nodes. Hence, without loss of generality, we may define the label node sets and mappings as follows:

- $X_{A'} = X_A$, $a_X(a') = a'$ for all $a' \in X_{A'} = X_A$
- $X_{B'} = X_A \cup X^{\diamond}_{B'}$ where $m'_X(a') = a'$ for all $a' \in X_{A'}$ and $X^{\diamond}_{B'} = X_{B'} \backslash m'(X_A)$
- $X_{C'} = X_A \cup X^{\diamond}_{C'}$ where $f'_X(a') = a'$ for all $a' \in X_{A'}$ and $X^{\diamond}_{C} = X_{C'} \backslash f'(X_A)$
- $X_{D'} = X_A \cup X^{\diamond}_{B'} \cup X^{\diamond}_{C'}$ where $n'_X(c') = c'$ for all $c' \in X_{C'}$, $g'_X(b') = b'$ for all $b' \in X_{B'}$
- $X_B = X_{B'} \cup X^*_B = X_A \cup X^{\diamond}_{B'} \cup X^*_B$ where $m_X(a) = a$ for all $a \in X_A$, $b_X(b') = b'$ for all $b' \in X_{B'}$, and $X^*_B = X_B \backslash X_{B'}$
- $X_C = X_{C'} \cup X^*_C = X_A \cup X^{\diamond}_{C'} \cup X^*_C$ where $f_X(a) = a$ for all $a \in X_A$, $c_X(c') = c'$ for all $c' \in X_{C'}$, and $X^*_C = X_C \backslash X_{C'}$
- $X_D = X_A \cup X^{\diamond}_{B'} \cup X^*_B \cup X^{\diamond}_{C'} \cup X^*_C$ where $n_X(c) = c$ for all $c \in X_C$, $g_X(b) = b$ for all $b' \in X_{B'}$, $d_X(d') = d'$ for all $d' \in X_{D'}$

Hence the Statement (6.7) becomes equivalent to

$$\mathcal{D} \vDash \Phi'_D \Leftrightarrow (\Phi'_C \wedge \Phi'_B). \tag{6.8}$$

From $d \in \mathcal{M}^{inj}_{Proj}$ we obtain

$$\mathcal{D} \vDash \Phi'_D \Leftrightarrow \exists c^*_1 \ldots \exists c^*_n . \exists b^*_1 \ldots \exists b^*_n . \Phi_D, \tag{6.9}$$

where $\{c^*_1, \ldots, c^*_n\} \in X^*_C$ and $\{b^*_1, \ldots, b^*_n\} \in X^*_B$.
As the bottom face is a pushout in $\mathbb{SG}_{\mathcal{D}}$ we obtain

$$\mathcal{D} \vDash \Phi_D \Leftrightarrow (\Phi_C \wedge \Phi_B). \tag{6.10}$$

By inserting Statement (6.10) in (6.9), and the result in (6.8) we obtain

$$\mathcal{D} \vDash \exists c^*_1 \ldots \exists c^*_n . \exists b^*_1 \ldots \exists b^*_n . (\Phi_C \wedge \Phi_B) \Leftrightarrow (\Phi'_C \wedge \Phi'_B), \tag{6.11}$$

where $\{c^*_1, \ldots, c^*_n\} \in X^*_C$ and $\{b^*_1, \ldots, b^*_n\} \in X^*_B$.
As $c, b \in \mathcal{M}^{inj}_{Proj}$ we have

$$\mathcal{D} \vDash \Phi'_C \Leftrightarrow \exists c^*_1 \ldots \exists c^*_n . \Phi_C, \text{ where} \{c^*_1, \ldots, c^*_n\} \in X^*_C \tag{6.12}$$

and

$$\mathcal{D} \vDash \Phi'_B \Leftrightarrow \exists b^*_1 \ldots \exists b^*_n . \Phi_B, \text{ where} \{b^*_1, \ldots, b^*_n\} \in X^*_B. \tag{6.13}$$

By combining Statement (6.11) with (6.12) and (6.13) we obtain

$$\mathcal{D} \vDash \exists c^*_1 \ldots \exists c^*_n . \exists b^*_1 \ldots \exists b^*_n . (\Phi_C \wedge \Phi_B) \Leftrightarrow (\exists c^*_1 \ldots \exists c^*_n . \Phi'_C \wedge \exists b^*_1 \ldots \exists b^*_n . \Phi_B),$$

with $\{c^*_1, \ldots, c^*_n\} \in X^*_C$ and $\{b_1*, \ldots, b_n*\} \in X^*_B$, which is valid as X^*_B and X^*_C are disjoint.

Only if. Given the following commutative cube (6) (i. e., the backmost cube). Assume the top face is a pushout in $\mathbb{SG}_{\mathcal{D}}$, we have to show that the front faces (3) and (3′) are pullbacks in $\mathbb{SG}_{\mathcal{D}}$.

By assumption, the morphisms $b, c, d, m \in \mathcal{M}^{inj}_{Proj}$ and morphism $f \in \mathcal{M}^{inj}_{\Rightarrow}$; hence, they are monomorphisms in $\mathbb{EG}$. According to Fact 3.52, the category $\mathbb{EG}$ with $\mathcal{M} = \mathcal{N} = \mathcal{M}^{inj}$ is $(\mathcal{M}, \mathcal{N})$-adhesive. By using the $(\mathcal{M}, \mathcal{N})$-VK property we can conclude that the front faces (3) and (3′) are pullbacks in $\mathbb{EG}$.

It remains to show that (3) and (3′) are also pullbacks in category $\mathbb{SG}_{\mathcal{D}}$. From $c, d \in \mathcal{M}^{inj}_{Proj}$ and Lemma 4.5 we obtain pullbacks (1) and (4) in $\mathbb{SG}_{\mathcal{D}}$. As (3) is a pullback in $\mathbb{EG}$, we know that also (2) is a pullback in $\mathbb{EG}$. Moreover, it can be easily checked that (2) is also a pullback in $\mathbb{SG}_{\mathcal{D}}$. By pullback composition we have that (1)+(2) (i. e., the diagonal square) is a pullback. Finally, by pullback decomposition we know that (3) is a pullback in $\mathbb{SG}_{\mathcal{D}}$ as (3)+(4)=(1)+(2) and (4) are pullbacks in $\mathbb{SG}_{\mathcal{D}}$.

The proof for the right front face (i. e., commuting square (3′)) can be obtained analogously, as also $b, d \in \mathcal{M}^{inj}_{Proj}$.

$$\begin{tikzcd}
 & & & & \langle A', \Phi'_A \rangle \arrow[dll, "f'"'] \arrow[drr, "m'"] \arrow[ddd, "a"] & (6) & & \\
 & & \langle C', \Phi'_C \rangle \arrow[dll, "id_{C'}"'] \arrow[dr, "n'"] \arrow[ddl] \arrow[ddd, "c"] & & & & \langle B', \Phi'_B \rangle \arrow[dlll, "g'"'] \arrow[dr] \arrow[ddl] \arrow[ddd, "b"] & \\
\langle C', \perp \rangle \arrow[dr, "n'"] \arrow[ddd] & & & \langle D', \Phi'_D \rangle \arrow[dll, "id_{D'}"'] \arrow[drr] \arrow[ddd, "d"] & & & & \langle B', \perp \rangle \arrow[dll] \arrow[ddd] \\
(1) & \langle D', \perp \rangle \arrow[ddd] & & (3) & \langle A, \Phi_A \rangle \arrow[dll, "f"'] \arrow[drr, "m"] & \langle D', \perp \rangle \arrow[ddd] & (3') & (1') \\
(2) & & \langle C, \Phi_C \rangle \arrow[dll, "id_C"'] \arrow[dr, "n"] \arrow[ddl] & (4) & (4') & & \langle B, \Phi_B \rangle \arrow[dlll, "g"'] \arrow[dr] \arrow[ddl] & (2') \\
\langle C, \perp \rangle \arrow[dr, "n"] & & & \langle D, \Phi_D \rangle \arrow[dll, "id_D"'] \arrow[drr] & & & & \langle B, \perp \rangle \arrow[dll] \\
 & \langle D, \perp \rangle & & & & \langle D, \perp \rangle & &
\end{tikzcd}$$

□

To verify Property 4 of Definition 5.1, we have to show that $(\mathcal{L}, \mathcal{N})$-pushouts are $(\mathcal{L}, \mathcal{N})$-VK squares for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow}$.

Lemma 6.5 ($(\mathcal{L}, \mathcal{N})$-pushouts are $(\mathcal{L}, \mathcal{N})$-VK squares)**.**
In $\mathbb{SG}_{\mathfrak{D}}$ with $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow}$, $(\mathcal{L}, \mathcal{N})$-pushouts are $(\mathcal{L}, \mathcal{N})$-VK squares.

Proof. **If.** Consider the commutative cube (1). Assume, the back and front faces are pullbacks in $\mathbb{SG}_{\mathfrak{D}}$, the bottom face is a pushout in $\mathbb{SG}_{\mathfrak{D}}$ along $m \in \mathcal{M}^{bij}_{\Leftrightarrow}$ and $f \in \mathcal{M}^{inj}_{\Rightarrow}$; the morphisms c, d, b are in $\mathcal{M}^{inj}_{Proj}$. We have to show that the top face is a pushout in $\mathbb{SG}_{\mathfrak{D}}$.

$$\begin{tikzcd}
 & & \langle A', \Phi'_A \rangle \arrow[dll, "f'"'] \arrow[drr, "m'"] \arrow[ddd, "a"] & & \\
\langle C', \Phi'_C \rangle \arrow[dr, "n'"] \arrow[ddd, "c"] & & & & \langle B', \Phi'_B \rangle \arrow[dlll, "g'"'] \arrow[ddd, "b"] \\
 & \langle D', \Phi'_D \rangle \arrow[ddd, "d"] & & & \\
 & & \langle A, \Phi_A \rangle \arrow[dll, "f"'] \arrow[drr, "m"] & & \\
\langle C, \Phi_C \rangle \arrow[dr, "n"] & & & & \langle B, \Phi_B \rangle \arrow[dlll, "g"'] \\
 & \langle D, \Phi_D \rangle & & (1) &
\end{tikzcd}$$

Similar to the proof of Lemma 6.4, we may assume that the cube (1) is a VK-square in $\mathbb{EG}$. Consequently, the top face is a pushout in $\mathbb{EG}$. From $m \in \mathcal{M}^{bij}_{\Leftrightarrow}$ follows that also m' and n are in $\mathcal{M}^{bij}_{\Leftrightarrow}$, as the bottom face is a pushout and the back right face an pullback, and $\mathcal{M}^{bij}_{\Leftrightarrow}$ is closed under pushouts and pullbacks. As $m' \in \mathcal{M}^{bij}_{\Leftrightarrow}$, the top face has to be a pushout along $m' \in \mathcal{M}^{bij}_{\Leftrightarrow}$. Since, $\mathcal{M}^{bij}_{\Leftrightarrow}$ is closed under pushouts, it is sufficient to verify that $n' \in \mathcal{M}^{bij}_{\Leftrightarrow}$. As the left front face is a pullback and $n \in \mathcal{M}^{bij}_{\Leftrightarrow}$, we can conclude that $n' \in \mathcal{M}^{bij}_{\Leftrightarrow}$, as $\mathcal{M}^{bij}_{\Leftrightarrow}$ is closed under pullbacks.

Only if. Given the commutative cube (1). Assume, the top face is a pushout in $\mathbb{SG}_{\mathfrak{D}}$, we have to show that the front faces are pullbacks in $\mathbb{SG}_{\mathfrak{D}}$. As $b, c, d \in \mathcal{M}^{inj}_{Proj}$, $f \in \mathcal{M}^{inj}_{\Rightarrow}$, and $m \in \mathcal{M}^{bij}_{\Leftrightarrow}$, we have that b, c, d, f, m are monomorphisms in the category $\mathbb{EG}$. Hence, the proof is the same as for the **Only If** direction of Lemma 6.4. □

Note that the Lemma 6.5 can also be derived as a consequence of the fact that the category of symbolic graphs with $\mathcal{M}^{bij}_{\Leftrightarrow}$-morphisms is an adhesive HLR category [OL10b].

Now we can show that the category of symbolic graphs $\mathbb{SG}_{\mathcal{D}}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive.

Theorem 6.6 ($\mathbb{SG}_{\mathcal{D}}$ with $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow}, \mathcal{R} = \mathcal{M}^{inj}_{Proj}, \mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive)**.**

The category $\mathbb{SG}_{\mathcal{D}}$ with morphism classes $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj}$, and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive.

Proof. This is a consequence of Definition 5.1 and Lemmas 6.1–6.5. □

Up until now, we have shown that the category of symbolic graphs with our corresponding choice of morphisms classes is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive. It remains to show that also the category of typed symbolic graphs is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive.

Corollary 6.7 ($\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$, $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive)**.**
The category $\mathbb{TSG}_{\mathcal{D},TG}$ of typed symbolic graphs over type graph TG^{Φ} with morphism classes $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$, and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive, where:

- $\mathbb{TSG}_{\mathcal{D},TG} = \mathbb{SG}_{\mathcal{D}} \backslash TG^{\Phi}$
- $\mathcal{M}^{bij}_{\Leftrightarrow,TG} = \mathcal{M}^{bij}_{\Leftrightarrow} \cap Mor_{\mathbb{SG}_{\mathcal{D}} \backslash TG^{\Phi}}$
- $\mathcal{M}^{inj}_{Proj,TG} = \mathcal{M}^{inj}_{Proj} \cap Mor_{\mathbb{SG}_{\mathcal{D}} \backslash TG^{\Phi}}$
- $\mathcal{M}^{inj}_{\Rightarrow,TG} = \mathcal{M}^{inj}_{\Rightarrow} \cap Mor_{\mathbb{SG}_{\mathcal{D}} \backslash TG^{\Phi}}$

Proof. This is a direct consequence of Theorem 5.5 and Theorem 6.6. □

It remains to show that typed projective graph transformation systems, as defined in Section 4.2, are $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive.

Corollary 6.8 (Typed projective graph transformation systems are $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation systems)**.**
Any typed projective graph transformation system **TPGTS** in the sense of Definition 4.7, is an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation system.

Proof. According to Corollary 6.7 the category $\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$, and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive. Hence any typed projective graph transformation system in the sense of Definition 4.7 is an $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive transformation system. □

6.2 HLR⁺Properties

In this section we prove the HLR⁺properties for typed projective graph transformation systems. Unfortunately, it turned out that the category $\mathbb{TSG}_{\mathcal{D},TG}$ with morphism classes $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj}$, and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow}$ does not provide the $\mathcal{R}$–$\mathcal{N}$-pushout–pullback decomposition property, which is required to transform right application condition to left application conditions. Nevertheless, by choosing typed functional projective morphisms for right production morphisms (i. e., $\mathcal{R} = \mathcal{M}^{inj}_{Func,TG}$), we can show the category $\mathbb{TSG}_{\mathcal{D},TG}$ of typed symbolic graphs provides this property. Note that $\mathcal{M}^{inj}_{Func,TG}$ is a subclass of $\mathcal{M}^{inj}_{Proj,TG}$. Consequently, every typed functional projective production is also a typed projective production; thus, typed functional projective productions enjoy the same properties as typed projective productions. Table 6.1 lists the HLR⁺properties with respect to the actual choice for $\mathcal{R}$.

Table 6.1: Overview of the HLR⁺properties

$\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$, $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$	Binary Coproducts	$\mathcal{E}$–$\mathcal{N}$ Factorization	$\mathcal{R}$–$\mathcal{N}$-PO–PB Decomp.	$\mathcal{L}$–$\mathcal{N}$-PO–PB Decomp.	$(\mathcal{L},\mathcal{N})$-Initial PO
$\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$	✓	✓		✓	✓
$\mathcal{R} = \mathcal{M}^{inj}_{Func,TG}$	✓	✓	✓	✓	✓

We start with showing that $\mathbb{TSG}_{\mathcal{D},TG}$ has binary coproducts for arbitrary morphisms in $\mathbb{TSG}_{\mathcal{D},TG}$.

Proposition 6.9 ($\mathbb{TSG}_{\mathcal{D},TG}$ has binary coproducts)**.**
For any two graphs $G^{\Phi}_1 = \langle G_1, \Phi_1 \rangle$ and $G^{\Phi}_2 = \langle G_2, \Phi_2 \rangle$ in $\mathbb{TSG}_{\mathcal{D},TG}$, there exists a binary coproduct $(\langle G_{1+2}, \Phi_{1+2} \rangle, i_1, i_2)$.

Construction. The triple $(\langle G_{1+2}, \Phi_{1+2} \rangle, i_1, i_2)$ is a binary coproduct in $\mathbb{TSG}_{\mathcal{D},TG}$ if (G_{1+2}, i_1, i_2) is a binary coproduct in $\mathbb{TEG}_{TG}$ and

$$\mathcal{D} \models \Phi_{1+2} \Leftrightarrow \left(\Phi_1[\hat{i}_1] \land \Phi_2[\hat{i}_2]\right).$$

Proof. According to Fact 3.52.b, the category $\mathbb{TEG}_{TG}$ has binary coproducts. Consequently given typed symbolic graphs $\langle G_1, \Phi_1 \rangle$ and $\langle G_2, \Phi_2 \rangle$ we can construct typed E-graph G_{1+2} with typed E-graph morphisms $i_1 : G_1 \to G_{1+2}$ and $i_2 : G_2 \to G_{1+2}$ such that (G_{1+2}, i_1, i_2) is a binary coproduct in $\mathbb{TEG}_{TG}$. Moreover, given typed symbolic graph morphisms

$$f_1 : \langle G_1, \Phi_1 \rangle \to \langle G_0, \Phi_0 \rangle \text{ and } f_2 : \langle G_2, \Phi_2 \rangle \to \langle G_0, \Phi_0 \rangle,$$

we can obtain unique morphism $c : G_{1+2} \to G_0$ in $\mathbb{TEG}_{TG}$ such that diagram (2) commutes, i.e., $c \circ i_1 = f_1$ and $c \circ i_2 = f_2$.

$$\begin{tikzcd}
\langle G_1, \Phi_1\rangle \arrow[r, "i_1"] \arrow[dr, "f_1"'] \arrow[d, phantom, "(1)"] & \langle G_{1+2}, \Phi_{1+2}\rangle \arrow[d, "c"] & \langle G_2, \Phi_2\rangle \arrow[l, "i_2"'] \arrow[dl, "f_2"] \\
 & \langle G_0, \Phi_0\rangle &
\end{tikzcd}
\qquad
\begin{tikzcd}
G_1 \arrow[r, "i_1"] \arrow[dr, "f_1"'] \arrow[d, phantom, "(2)"] & G_{1+2} \arrow[d, "c"] & G_2 \arrow[l, "i_2"'] \arrow[dl, "f_2"] \\
 & G_0 &
\end{tikzcd}$$

To show that diagram (1) is a coproduct in $\mathbb{TSG}_{\mathcal{D},TG}$, we have to verify that morphisms i_1, i_2, and c are typed symbolic graph morphisms.

The morphisms i_1 and i_2 are typed symbolic graph morphisms since

$$\mathcal{D} \models (\Phi_1[\hat{i}_1] \wedge \Phi_2[\hat{i}_2]) \Rightarrow \Phi_1[\hat{i}_1] \text{ and } \mathcal{D} \models (\Phi_1[\hat{i}_1] \wedge \Phi_2[\hat{i}_2]) \Rightarrow \Phi_2[\hat{i}_2].$$

To show that $c : \langle G_{1+2}, \Phi_{1+2}\rangle \to \langle G_0, \Phi_0\rangle$ is a typed symbolic graph morphism, we have to verify that

$$\mathcal{D} \models \Phi_0 \Rightarrow (\Phi_1[\hat{i}_1] \wedge \Phi_2[\hat{i}_2])[\hat{c}].$$

According to the definition of substitution (Definition 3.47), this is equivalent to

$$\mathcal{D} \models \Phi_0 \Rightarrow (\Phi_1[\hat{i}_1][\hat{c}] \wedge \Phi_2[\hat{i}_2][\hat{c}]).$$

From $f_1 = c \circ i_1$ and $f_2 = c \circ i_2$, we obtain

$$\Phi_0 \Rightarrow (\Phi_1[\hat{f}_1] \wedge \Phi_2[\hat{f}_2]),$$

which is valid because

$$\mathcal{D} \models \Phi_0 \Rightarrow \Phi_1[\hat{f}_1] \text{ and } \mathcal{D} \models \Phi_0 \Rightarrow \Phi_2[\hat{f}_2],$$

as morphisms f_1 and f_2 are typed symbolic graph morphisms (by definition). □

In order to define $\mathcal{E}$–$\mathcal{N}$ factorization we have to assign a concrete morphism class to $\mathcal{E}$. It turned out that the choice $\mathcal{E} = \mathcal{E}^{surj}_{\Leftrightarrow,TG}$ satisfies the requirements, whereas $\mathcal{E}^{surj}_{\Leftrightarrow,TG}$ is defined as follows:

Definition 6.10 (The class $\mathcal{E}^{surj}_{\Leftrightarrow,TG}$).
The class $\mathcal{E}^{surj}_{\Leftrightarrow,TG}$ consists of all morphisms $e : \langle G, \Phi_G\rangle \to \langle H, \Phi_H\rangle$ in $\mathbb{TSG}_{\mathcal{D},TG}$ that are surjective for all kinds of nodes and edges and $\mathcal{D} \models \Phi_H \Leftrightarrow \Phi_G[\hat{e}]$.

Lemma 6.11 ($\mathbb{TSG}_{\mathcal{D},TG}$ has an $\mathcal{E}$–$\mathcal{N}$ factorization for $\mathcal{E} = \mathcal{E}^{surj}_{\Leftrightarrow,TG}$, $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$).
Given any morphism $f : \langle A, \Phi_A\rangle \to \langle B, \Phi_B\rangle$ in $\mathbb{TSG}_{\mathcal{D},TG}$, then there exists a unique decomposition into morphisms $e : \langle A, \Phi_A\rangle \to \langle C, \Phi_C\rangle$ and $m : \langle C, \Phi_C\rangle \to \langle D, \Phi_D\rangle$, with $e \in \mathcal{E}^{surj}_{\Leftrightarrow,TG}$ and $m \in \mathcal{M}^{inj}_{\Rightarrow,TG}$, such that $f = m \circ e$.

Construction. Given a morphism $f : \langle A, \Phi_A\rangle \to \langle B, \Phi_B\rangle$ in $\mathbb{TSG}_{\mathcal{D},TG}$, then morphisms e, m, and E-graph C are constructed as the $\mathcal{E}$–$\mathcal{M}$ factorization in $\mathbb{TEG}_{TG}$, where $\mathcal{E} = \mathcal{E}^{surj}_{TG}$ and $\mathcal{M} = \mathcal{M}^{inj}_{TG}$ are the classes of typed E-graph epimorphisms and monomorphisms, respectively. The Σ-formula Φ_C is chosen so that $\mathcal{D} \models \Phi_C \Leftrightarrow \Phi_A[\hat{e}]$.

Proof. According to Fact 3.52.c the category $\mathbb{TEG}_{TG}$ has such a factorization for the classes of typed E-graph epimorphisms and monomorphisms. Hence, we can assume that given morphism $f : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$ in $\mathbb{TSG}_{\mathcal{D},TG}$, there exists a unique decomposition into E-graph morphisms $e : A \to C$, $e \in \mathcal{E}^{surj}_{TG}$ and $m : C \to D$, $m \in \mathcal{M}^{inj}_{TG}$ such that $f = m \circ e$.

$$\begin{tikzcd}
\langle A,\Phi_A\rangle \arrow[rr, "f"] \arrow[dr, "e"'] & & \langle B,\Phi_B\rangle \\
& \langle C,\Phi_C\rangle \arrow[ur, "m"'] \arrow[u, phantom, "="] &
\end{tikzcd}$$

Let $\mathcal{D} \models \Phi_C \Leftrightarrow \Phi_A[\hat{e}]$, it remains to show that $e \in \mathcal{E}^{surj}_{\Leftrightarrow,TG}$ and $m \in \mathcal{M}^{inj}_{\Rightarrow,TG}$. Morphism e is in $\mathcal{E}^{surj}_{\Leftrightarrow,TG}$ as $\mathcal{D} \models \Phi_C \Leftrightarrow \Phi_A[\hat{e}]$ (see Definition 6.10). Morphism m is in $\mathcal{M}^{inj}_{\Rightarrow,TG}$, as $\mathcal{D} \models \Phi_B \Rightarrow \Phi_A[\hat{f}]$ and $f = m \circ e$ implies $\mathcal{D} \models \Phi_B \Rightarrow \Phi_A[\hat{e}][\hat{m}]$, which is equivalent to $\mathcal{D} \models \Phi_B \Rightarrow \Phi_C[\hat{m}]$ as $\mathcal{D} \models \Phi_C \Leftrightarrow \Phi_A[\hat{e}]$. □

Before we show the $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property, we first show an other nice property of $\mathcal{M}^{inj}_{Func,TG}$-morphisms, which basically states that a pushout complement along an $\mathcal{M}^{inj}_{Func,TG}$-morphism exists if the pushout complement for the E-graph component exists.

Lemma 6.12 (Pushout complements along $\mathcal{M}^{inj}_{Func,TG}$ and $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphisms)**.** Given symbolic graph morphisms $f : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$, $f \in \mathcal{M}^{inj}_{Func,TG}$ and $g' : \langle B, \Phi_B \rangle \to \langle D, \Phi_D \rangle$, $g' \in \mathcal{M}^{inj}_{\Rightarrow,TG}$, then the following holds: If there exists a (unique) E-graph C such that (2) is a pushout in $\mathbb{TEG}_{TG}$ then there exists a unique Φ_C such that (1) is a pushout in $\mathbb{TSG}_{\mathcal{D},TG}$.

$$\begin{tikzcd}
\langle A,\Phi_A\rangle \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & \langle B,\Phi_B\rangle \arrow[d, "g'"] \\
\langle C,\Phi_C\rangle \arrow[r, "f'"'] & \langle D,\Phi_D\rangle
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(2)"] & B \arrow[d, "g'"] \\
C \arrow[r, "f'"'] & D
\end{tikzcd}$$

Construction. As f and g' are E-graph monomorphisms and (2) is a pushout in $\mathbb{TEG}_{TG}$, we may assume without loss of generality (similar to the proof of Lemma 6.1) that $X_B = X_A \cup X^*_B$ and $X_C = X_A \cup X^*_C$ with $f_X(a) = a$ and $g_X(a) = a$ for all $a \in X_A$; as well as $X_D = X_A \cup X^*_C \cup X^*_B$ with $f'_X(c) = c$ for all $c \in X_C$ and $g'_X(b) = b$ for all $b \in X_B$. Moreover we may assume that X_A,

X^*_B, and X^*_C are pairwise disjoint. As $f \in \mathcal{M}^{inj}_{Func,TG}$ we know that there is a decomposition of Φ_B such that

$$\mathcal{D} \models \Phi_B \Leftrightarrow (\Phi_A \wedge (b^*_1 \stackrel{s(1)}{=} t_1) \wedge \ldots \wedge (b^*_n \stackrel{s(n)}{=} t_n)),$$

where $b^*_i \in \{b^*_1, \ldots, b^*_n\} = X^*_B$ and terms $t_i \in \mathcal{T}_{s(i)}$ of corresponding sorts with $var(t_i) \subseteq X_A$. Now we define Φ_C such that

$$\mathcal{D} \models \Phi_C \Leftrightarrow \Phi_D \left[\frac{t_1 \ \ldots \ t_n}{b^*_1 \ \ldots \ b^*_n}\right].$$

Proof. Given diagram (1) above with $f \in \mathcal{M}^{inj}_{Func,TG}$ and $g' \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ such that (2) is a pushout in $\mathbb{TEG}_{TG}$ and assume Φ_C is constructed as defined above.
First we show that $f' : \langle C, \Phi_C \rangle \to \langle D, \Phi_D \rangle$ is a symbolic graph morphism, which is equivalent to show

$$\mathcal{D} \models \Phi_D \Rightarrow \Phi_D \left[\frac{t_1 \ \ldots \ t_n}{b^*_1 \ \ldots \ b^*_n}\right]. \tag{6.14}$$

As $g' : \langle B, \Phi_B \rangle \to \langle D, \Phi_D \rangle$ is a symbolic graph morphism, we have

$$\mathcal{D} \models \Phi_D \Rightarrow \Phi_A \wedge (b^*_1 \stackrel{s(1)}{=} t_1) \wedge \ldots \wedge (b^*_n \stackrel{s(n)}{=} t_n). \tag{6.15}$$

Hence, for any assignment ζ such that $(\mathcal{D}, \zeta) \models \Phi_D$, we know that

$$(\mathcal{D}, \zeta) \models (b^*_1 \stackrel{s(1)}{=} t_1) \wedge \ldots \wedge (b^*_n \stackrel{s(n)}{=} t_n)$$

is valid; thus, also $\zeta(b^*_i) = [\![t_i]\!]^{\mathcal{D}}_{\zeta}$ for all $i \in \{1, \ldots, n\}$, as $var(t_i) \subseteq X_A$. Thus, Statement (6.14) is valid, as $(\mathcal{D}, \zeta) \models \Phi_D$ implies $(\mathcal{D}, \zeta) \models \Phi_C$ for any assignment ζ.

According to Remark 3.57, to show that (1) is a pushout in $\mathbb{TSG}_{\mathcal{D},TG}$ it is sufficient to verify that

$$\mathcal{D} \models \Phi_D \Leftrightarrow (\Phi_B \wedge \Phi_C). \tag{6.16}$$

The "($\Rightarrow$)" direction is a direct consequence of the fact that f' and g' are symbolic graph morphisms, so $\mathcal{D} \models \Phi_D \Rightarrow \Phi_C$ and $\mathcal{D} \models \Phi_D \Rightarrow \Phi_B$.

The "($\Leftarrow$)" direction is equivalent to show

$$\mathcal{D} \models \Big(\Phi_A \wedge (b^*_1 \stackrel{s(1)}{=} t_1) \wedge \ldots \wedge (b^*_n \stackrel{s(n)}{=} t_n) \wedge \Phi_D \left[\frac{t_1 \ \ldots \ t_n}{b^*_1 \ \ldots \ b^*_n}\right]\Big) \Rightarrow \Phi_D, \tag{6.17}$$

which is valid if

$$\mathcal{D} \models \Big((b^*_1 \stackrel{s(1)}{=} t_1) \wedge \ldots \wedge (b^*_n \stackrel{s(n)}{=} t_n) \wedge \Phi_D \left[\frac{t_1 \ \ldots \ t_n}{b^*_1 \ \ldots \ b^*_n}\right]\Big) \Rightarrow \Phi_D. \tag{6.18}$$

Now assume any assignment ζ such that

$$(\mathcal{D}, \zeta) \models (b^*_1 \stackrel{s(1)}{=} t_1) \wedge \ldots \wedge (b^*_n \stackrel{s(n)}{=} t_n) \wedge \Phi_D \left[\frac{t_1 \ \ldots \ t_n}{b^*_1 \ \ldots \ b^*_n}\right]. \tag{6.19}$$

According to Definition 3.41, assignment ζ is as solution of Statement (6.19) iff

$$(\mathcal{D}, \zeta) \models (b_i^* \overset{s(i)}{=} t_i) \text{ for all } i \in \{1, \ldots, n\} \tag{6.20}$$

and

$$(\mathcal{D}, \zeta) \models \Phi_D \left[\frac{t_1 \, \ldots \, t_n}{b_1^* \, \ldots \, b_n^*} \right]. \tag{6.21}$$

From Statement (6.20) we obtain that

$$[\![b_i^*]\!]_\zeta^{\mathcal{D}} = \zeta(b_i^*) = [\![t_i]\!]_\zeta^{\mathcal{D}} \text{ for all } i \in \{1, \ldots, n\} \tag{6.22}$$

Hence, b_i^* and t_i evaluate to the same values in $\mathcal{D}$ under any assignment ζ. Consequently, if

$$(\mathcal{D}, \zeta) \models (b_1^* \overset{s(1)}{=} t_1) \wedge \ldots \wedge (b_n^* \overset{s(n)}{=} t_n) \wedge \Phi_D \left[\frac{t_1 \, \ldots \, t_n}{b_1^* \, \ldots \, b_n^*} \right], \text{then } (\mathcal{D}, \zeta) \models \Phi_D.$$

then also

$$(\mathcal{D}, \zeta) \models \Phi_D \tag{6.23}$$

for any assignment ζ. Thus, Statement (6.18) and, consequently, Statement (6.17) are valid.

Finally, we have to show that $g : \langle A, \Phi_A \rangle \to \langle C, \Phi_c \rangle$ is a symbolic graph morphism. From symbolic graph morphism $g' : \langle B, \Phi_B \rangle \to \langle D, \Phi_D \rangle$ we obtain

$$\mathcal{D} \models \Phi_D \Rightarrow (\Phi_A \wedge (b_1^* \overset{s(1)}{=} t_1) \wedge \ldots \wedge (b_n^* \overset{s(n)}{=} t_n)). \tag{6.24}$$

Substituting t_i for b_i^* for all $i \in \{1, \ldots, n\}$ in Statement (6.24) results in

$$\mathcal{D} \models \Big(\Phi_D \Rightarrow (\Phi_A \wedge (b_1^* \overset{s(1)}{=} t_1) \wedge \ldots \wedge (b_n^* \overset{s(n)}{=} t_n)) \Big) \left[\frac{t_1 \, \ldots \, t_n}{b_1^* \, \ldots \, b_n^*} \right], \tag{6.25}$$

which is equivalent to

$$\mathcal{D} \models \Phi_D \left[\frac{t_1 \, \ldots \, t_n}{b_1^* \, \ldots \, b_n^*} \right] \Rightarrow \Phi_A, \tag{6.26}$$

because

$$\mathcal{D} \models \Phi_A \left[\frac{t_1 \, \ldots \, t_n}{b_1^* \, \ldots \, b_n^*} \right] \Leftrightarrow \Phi_A$$

as Φ_A has no free variables in X_B^*, and

$$((b_1^* \overset{s(1)}{=} t_1) \wedge \ldots \wedge (b_n^* \overset{s(n)}{=} t_n)) \left[\frac{t_1 \, \ldots \, t_n}{b_1^* \, \ldots \, b_n^*} \right] \Leftrightarrow \top$$

is trivially valid. Hence, from Statement (6.26) follows that $g : \langle A, \Phi_A \rangle \to \langle C, \Phi_c \rangle$ is a symbolic graph morphism, as by definition

$$\mathcal{D} \models \Phi_C \Leftrightarrow \Phi_D \left[\frac{t_1 \, \ldots \, t_n}{b_1^* \, \ldots \, b_n^*} \right].$$

The uniqueness of Φ_C is a direct consequence of the uniqueness of pushout complements along $\mathcal{M}_{Proj,TG}^{inj}$-morphisms and the fact that $\mathcal{M}_{Func,TG}^{inj}$ is a subclass of $\mathcal{M}_{Proj,TG}^{inj}$. □

Remark 6.13 (Pushout complements along $\mathcal{M}^{inj}_{Func,TG}$ and $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphisms)**.**
As shown within Lemma 6.12, it is possible to construct the formula component of symbolic graph $\langle C, \Phi_C\rangle$ by syntactical means (note that substitution is an operation at the syntactical level of Σ-formulas). Moreover, as a direct consequence of Lemma 6.12, we can formulate a syntactical criterion to decide whether a pushout complement along an $\mathcal{M}^{inj}_{Func,TG}$-morphisms exists. More specifically, to decide whether the pushout complement for morphisms $f : \langle A, \Phi_A\rangle \to \langle B, \Phi_B\rangle$, $f \in \mathcal{M}^{inj}_{Func,TG}$ and $g' : \langle B, \Phi_B\rangle \to \langle D, \Phi_D\rangle$, $g' \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ exists in $\mathbb{TSG}_{\mathcal{D},TG}$, it is sufficient (and neccesary) to check the existence of the pushout complement for morphisms $f : A \to B$ and $g' : B \to D$ in $\mathbb{TEG}_{TG}$. Thus, no semantic reasoning over the Σ-formulas of the involved symbolic graphs is required. This property is important to guarantee the soundness of our implementation for transforming right to left application conditions, as shown later in Chapter 9.

Now, we can show that $\mathbb{TSG}_{\mathcal{D},TG}$ has the $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property for $\mathcal{R} = \mathcal{M}^{inj}_{Func,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$.

Lemma 6.14 ($\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition for $\mathcal{R} = \mathcal{M}^{inj}_{Func,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$)**.**
The category $\mathbb{TSG}_{\mathcal{D},TG}$ has $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition for $\mathcal{R} = \mathcal{M}^{inj}_{Func,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$; that is, given the following commuting diagram:

$$\begin{tikzcd}
\langle A, \Phi_A\rangle \arrow[r, "k"] \arrow[d, "l"'] \arrow[dr, phantom, "(1)"] & \langle B, \Phi_B\rangle \arrow[r, "r"] \arrow[d, "s"] \arrow[dr, phantom, "(2)"] & \langle E, \Phi_E\rangle \arrow[d, "v"] \\
\langle C, \Phi_C\rangle \arrow[r, "u"'] & \langle D, \Phi_D\rangle \arrow[r, "w"'] & \langle F, \Phi_F\rangle
\end{tikzcd}$$

if (1)+(2) is a pushout with $l \in \mathcal{M}^{inj}_{Func,TG}$ and $(r \circ k) \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ and (2) a pullback with $w \in \mathcal{M}^{inj}_{\Rightarrow,TG}$, then (1) and (2) are pushouts and pullbacks in $\mathbb{TSG}_{\mathcal{D},TG}$.

Proof. Recall that category $\mathbb{TEG}_{TG}$ with the class $\mathcal{L} = \mathcal{N} = \mathcal{M}^{inj}_{TG}$ of typed E--graph monomorphisms is $(\mathcal{M}, \mathcal{N})$-adhesive (see Fact 3.52). Hence, typed E-graphs have the $\mathcal{M}$–$\mathcal{N}$-pushout–pullback decomposition property (see Fact 3.52). As $l \in \mathcal{M}^{inj}_{Func,TG}$ and $w \in \mathcal{M}^{inj}_{\Rightarrow,TG}$, l and w are typed E-graph monomorphisms, we may assume that (1) and (2) are pushouts as well as pullbacks in $\mathbb{TEG}_{TG}$.

Now consider the following diagram, where (1′) is the pushout constructed according to Lemma 6.12 from pushout (1) leading to a unique Φ'_B; (2) is the pullback of w and v in $\mathbb{TSG}_{\mathcal{D},TG}$.

$$\begin{tikzcd}
\langle A,\Phi_A\rangle \arrow[r, dashed, "k"] \arrow[d, "l"] \arrow[rrrr, bend left, "k\circ r"] \arrow[dr, phantom, "(1')"] & \langle B,\Phi'_B\rangle \arrow[rr, dashed, bend right, "b"'] \arrow[dr, "s'"] & & \langle B,\Phi_B\rangle \arrow[r, dashed, "r"] \arrow[dl, "s"'] \arrow[ll, dashed, bend right, "b^{-1}"'] & \langle E,\Phi_E\rangle \arrow[d, "v"] \arrow[dl, phantom, "(2)"] \\
\langle C,\Phi_C\rangle \arrow[rr, "u"] \arrow[rrrr, bend right, "u\circ w"'] & & \langle D,\Phi_D\rangle \arrow[rr, "w"] & & \langle F,\Phi_F\rangle
\end{tikzcd}$$

As $\langle B,\Phi'_B\rangle$ and $\langle B,\Phi_B\rangle$ have isomorphic E-graph components, there is a typed E-graph isomorphism $b : B' \to B$ with inverse $b^{-1} : B \to B'$. It remains to show that b is an isomorphism in $\mathbb{TSG}_{\mathcal{D},TG}$, which is equivalent to verify that b and b^{-1} are typed symbolic graph morphisms. We obtain from $l \in \mathcal{M}^{inj}_{Func,TG}$ that $l \in \mathcal{M}^{inj}_{Proj,TG}$, as $\mathcal{M}^{inj}_{Func,TG}$ is a subclass of $\mathcal{M}^{inj}_{Proj,TG}$. Recall that morphisms in $\mathcal{M}^{inj}_{Proj,TG}$ are closed under pushouts and pullbacks. Thus, $l \in \mathcal{M}^{inj}_{Proj,TG}$ and pushout (1)+(2) implies $v \in \mathcal{M}^{inj}_{Proj,TG}$. From pushout (1′) and $l \in \mathcal{M}^{inj}_{Proj,TG}$ we obtain $s' \in \mathcal{M}^{inj}_{Proj,TG}$; pullback (2′) and $v \in \mathcal{M}^{inj}_{Proj,TG}$ implies $s \in \mathcal{M}^{inj}_{Proj,TG}$. Using the projection properties of s and s' we can conclude that b and b^{-1} are symbolic graph morphisms, respectively. Hence, $b : B' \to B$ is also an isomorphism in $\mathbb{TSG}_{\mathcal{D},TG}$. Consequently, pushout (1′) is isomorphic to (1).

From pushout decomposition of pushouts (1) and (1)+(2) in $\mathbb{TSG}_{\mathcal{D},TG}$, follows that also (2) is a pushout in $\mathbb{TSG}_{\mathcal{D},TG}$. The closure of $\mathcal{M}^{inj}_{\Rightarrow,TG}$ under pullbacks and $w \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ implies $r \in \mathcal{M}^{inj}_{\Rightarrow,TG}$. The closure of $\mathcal{M}^{inj}_{\Rightarrow,TG}$ under decomposition and $r, (k \circ r) \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ implies $k \in \mathcal{M}^{inj}_{\Rightarrow,TG}$. Hence (1) is a pushout and a pullback. □

Remark 6.15 ($\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$)**.** The category $\mathbb{TSG}_{\mathcal{D},TG}$ has also the *$\mathcal{L}$–$\mathcal{N}$-PO–PB decomposition for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$*. This is a direct consequence of Corollary 5.14 and Lemma 6.14. Note that this property is independent from the actual choice of $\mathcal{R}$, although we deduced it from Lemma 6.14 with $\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$.

It remains to show that $\mathbb{TSG}_{\mathcal{D},TG}$ has $(\mathcal{L},\mathcal{N})$-initial pushouts.

Definition 6.16 (Construction of $(\mathcal{L},\mathcal{N})$-initial pushouts for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$)**.**
Given a typed symbolic graph morphisms $m : \langle L,\Phi_L\rangle \to \langle G,\Phi_G\rangle$,

$$\begin{tikzcd}
\langle B,\Phi_B\rangle \arrow[r, "b"] \arrow[d, "e"] \arrow[dr, phantom, "(1)"] & \langle L,\Phi_L\rangle \arrow[d, "m"] \\
\langle C,\Phi_C\rangle \arrow[r, "c"] & \langle G,\Phi_G\rangle
\end{tikzcd}$$

the symbolic boundary graph $\langle B, \Phi_B \rangle$ with morphism $b : \langle B, \Phi_B \rangle \to \langle L, \Phi_L \rangle$, $b \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ is constructed as follows:

- The set of graph nodes V_B of B consists of all graph nodes $n \in V_L$ such that $m_V(n)$ is adjacent to an graph or label edge in $G \backslash m(L)$.

- The set of label nodes X_B is given by $X_B = X_L$.

Morphism $b : \langle B, \Phi_B \rangle \to \langle L, \Phi_L \rangle$ is the inclusion of B in L. The formula Φ_B is set such that $\mathcal{D} \models \Phi_B[\hat{b}] \Leftrightarrow \Phi_L$. Hence, $b \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$.

The context graph $\langle C, \Phi_C \rangle$ with morphism $c : \langle C, \Phi_C \rangle \to \langle G, \Phi_G \rangle$, $c \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ is given as $C = (G \backslash m(L)) \cup m(b(B))$. Morphisms $c : \langle C, \Phi_C \rangle \to \langle G, \Phi_G \rangle$ and $e : \langle B, \Phi_B \rangle \to \langle C, \Phi_C \rangle$ are given by the inclusion of C in G and B in C, respectively. The formula Φ_C is set such that $\mathcal{D} \models \Phi_C[\hat{c}] \Leftrightarrow \Phi_C$. As $X_B = X_L$ we have $X_C = (X_G \backslash m_X(X_L)) \cup m_X(b_X(X_B)) = (X_G \backslash m_X(X_L)) \cup m_X(X_L) = X_G$. Hence, morphism c is isomorphic for label nodes, so $c \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$.

Basically the boundary object contains all nodes of L that are adjacent to an edge in $G \backslash m(L)$, which are exactly those nodes that have to preserved by a production when applied to to G via match m in order to prevent dangling edges.

Lemma 6.17 (($\mathcal{L}, \mathcal{N}$)-initial pushouts for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$).
Every pushout constructed according to Definition 6.16 is an ($\mathcal{L}, \mathcal{N}$)-initial pushout in $\mathbb{TSG}_{\mathcal{D},TG}$ in the sense of Definition 3.26.

Proof. Consider the diagram shown next. Given initial pushout (1) in $\mathbb{TSG}_{\mathcal{D},TG}$ constructed according to Definition 6.16, we show that for every pushout (2) with $m, g \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $m, k \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ there exist unique $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$-morphisms $b^* : \langle B, \Phi_B \rangle \to \langle K, \Phi_K \rangle$ and $c^* : \langle C, \Phi_C \rangle \to \langle D, \Phi_D \rangle$, such that $l \circ b^* = b$, $g \circ c^* = c$, and (3) is a pushout $\mathbb{TSG}_{\mathcal{D},TG}$.

$$\begin{tikzcd}
\langle B, \Phi_B \rangle \arrow[rr, "b^*"] \arrow[dr, "b"] \arrow[dd, "e"'] & & \langle K, \Phi_K \rangle \arrow[dl, "l"'] \arrow[dd, "k"] \\
& \langle L, \Phi_L \rangle \arrow[dd, "m"] & \\
\langle C, \Phi_C \rangle \arrow[rr, "c^*"] \arrow[dr, "c"'] & & \langle D, \Phi_D \rangle \arrow[dl, "g"] \\
& \langle G, \Phi_G \rangle &
\arrow[from=1-1, to=3-1, phantom, shift left=8ex, "(1)"]
\arrow[from=2-2, to=3-2, phantom, shift left=3ex, "(3)"]
\arrow[from=1-3, to=3-3, phantom, shift right=8ex, "(2)"]
\end{tikzcd}$$

As the construction, given in Definition 6.16, is similar as the construction of initial pushouts for typed graphs in [EEPT06] and we do not delete label nodes, we may assume that there exists pushout (3) in $\mathbb{TEG}_{TG}$ such that $l \circ b^* = b$ and $g \circ c^* = c$. As morphisms $b, c, l, g, b^*, c^* \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ we may assume that $X_B = X_L = X_K$ and $X_C = X_G = X_D$ as well as $b_X(b) = b^*_X(b) = l(b) = c$ for all $b \in X_B = X_L = X_K$ and $c_X(c) = c^*_X(c) = l(c) = c$ for all $c \in X_C = X_G = X_D$. Thus, from the commutativity of (1), (2) and (3) we obtain that $e(b) = m(b) = g(b) = b$ for all $b \in X_B = X_L = X_K$. From pushout (1) we obtain

$$\mathcal{D} \models \Phi_G \Leftrightarrow \Phi_L[\hat{m}] \wedge \Phi_C[\hat{c}],$$

which is equivalent to

$$\mathcal{D} \models \Phi_G \Leftrightarrow \Phi_L[\hat{m}] \wedge \Phi_C,$$

as $c_X(c) = c$. From $b, c, l, g, b^*, c^* \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ we obtain

$$\mathcal{D} \models \Phi_B \Leftrightarrow \Phi_L \Leftrightarrow \Phi_K$$

and

$$\mathcal{D} \models \Phi_C \Leftrightarrow \Phi_G \Leftrightarrow \Phi_D$$

thus,

$$\mathcal{D} \models \Phi_D \Leftrightarrow \Phi_K[\hat{g}] \wedge \Phi_C,$$

which means that (3) is a pushout in $\mathbb{TSG}_{\mathcal{D},TG}$. □

Note that according to Lemma 5.18 category $\mathbb{TSG}_{\mathcal{D},TG}$ has quasi $(\mathcal{L}, \mathcal{N})$-pushouts.

The following remark states that gluing condition (see Fact 3.27) for a typed projective production can be reduced to the gluing condition in typed E-graphs.

Remark 6.18 (Pushout complements along $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$ and $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism pairs)**.** To ensure the existence of a pushout complement (see Fact 3.27) for a projective production $p = (\langle L, \Phi_L \rangle \xleftarrow{l} \langle K, \Phi_K \rangle \xrightarrow{r} \langle R, \Phi_R \rangle)$ with match $m : \langle L, \Phi_L \rangle \to \langle G, \Phi_G \rangle$, $m \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ as shown below (assuming $(\mathcal{D} \models \Phi_L \Leftrightarrow \Phi_K[\hat{l}])$),

$$\begin{tikzcd}
\langle B, \Phi_B \rangle \arrow[r, "b"] \arrow[rr, bend left, "b^*"] \arrow[d] \arrow[dr, phantom, "(1)"] & \langle L, \Phi_L \rangle \arrow[d, "m"] & \langle K, \Phi_K \rangle \arrow[l, "l"'] \arrow[r, "r"] & \langle R, \Phi_R \rangle \\
\langle C, \Phi_C \rangle \arrow[r, "c"] & \langle G, \Phi_G \rangle & &
\end{tikzcd}$$

it is sufficient to ensure the existence of E-graph $\mathcal{M}^{bij}_{TG}$-morphism $b^* : B \to K$ as $\mathcal{D} \models \Phi_K \Rightarrow \Phi_B[\hat{b}^*]$ is trivially valid if $b, l \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$. Hence, to decide whether pushout complement (1) exists, no reasoning on the formula component is needed. Accordingly, the construction of direct transformation $\langle G, \Phi_G \rangle \overset{p@m}{\Longrightarrow} \langle H, \Phi_H \rangle$ can be performed purely syntactically. This property is important to guarantee the soundness of our implementation, as shown later in Chapter 9.

7

VERIFICATION OF SYMBOLIC CONSISTENCY CONSTRAINTS

The construction of application conditions from constraints was first introduced in [HW95] for plain graphs. Accordingly, a constraint is first transformed into a set of equivalent right application conditions. Subsequently, the right application conditions are transformed into equivalent left application conditions. In [EEHP06] it was shown that this approach can be generalized to $\mathcal{M}$-adhesive categories if the underlying category has some extra properties, which are referred to as HLR$^+$properties. Although the category of symbolic graphs is an $\mathcal{M}$-adhesive category, it was unclear whether it provides these extra properties. In [DV14] we have shown that the category of symbolic graphs with $\mathcal{M}_{\Leftrightarrow}^{bij}$-morphisms indeed provides the required HLR$^+$properties. *The main contribution of this chapter is to extend these results to typed projective graph transformations.* Unfortunately, it turned out that the construction of left application conditions from right application conditions is not possible for arbitrary projective productions. Nevertheless, we show that this construction is valid for functional projective productions.

In the following, we present all constructions directly for typed symbolic graphs. To this end, we assume for the rest of this chapter that category $\mathbb{TSG}_{\mathcal{D},TG}$ is given by a symbolic type graph TG^{Φ} and a Σ-structure $\mathcal{D}$.

7.1 Construction Equivalent NACs From Negative Constraints

Towards the construction of consistency preserving typed functional projective productions, we shall see in this section that for every typed negative symbolic constraint NC and typed symbolic graph R^{Φ}, we can construct an equivalent application condition; that is, a typed negative application condition NAC_R such that any match of R^{Φ} in a typed symbolic graph H^{Φ} satisfies NAC_R if and only if H^{Φ} is consistent with respect to NC. If we consider R^{Φ} as the right-hand side of a typed functional projective production p, we can construct the extended production $\varrho = (p, \emptyset, NAC_R)$ with equivalent right negative application condition NAC_R, so that comatch n satisfies NAC_R if and only if $H^{\Phi} \Vdash NC$.

For a simple negative constraint $nc(N^{\Phi})$ in $\mathbb{TSG}_{\mathcal{D},TG}$, an equivalent NAC over R^{Φ} is constructed from all gluings of N^{Φ} and R^{Φ} along any common subgraph (including the empty graph). These gluings represent all possible combinations of N^{Φ} and R^{Φ} that may occur in a graph. Adding these gluings as simple negative application conditions to NAC_R ensures that for any typed symbolic graph H^{Φ} that is inconsistent with respect to $nc(N^{\Phi})$, there either does not exists a match of R^{Φ} in H^{Φ}, or all matches do not satisfy NAC_R, as H^{Φ} must contain one of the gluings.

To precisely define this construction, we first have to formalize the notion of *a gluing of two typed symbolic graphs*. Basically, a gluing Y^Φ of two typed symbolic graphs N^Φ and R^Φ can be defined as a pair $(R^\Phi \xrightarrow{y} Y^\Phi, N^\Phi \xrightarrow{c} Y^\Phi)$ of jointly epimorphic morphisms (see Definition 3.15). Consequently, whenever we can find a match $n' : Y^\Phi \to H^\Phi$ of Y^Φ to a typed symbolic graph H^Φ then we have also matches n and c' given by $n = n' \circ y$ and $c' = n' \circ c$, respectively.

$$\begin{tikzcd}
\langle R, \Phi_R \rangle \arrow[d, "y"] \arrow[dd, bend right=60, "n"'] & \\
\langle Y, \Phi_Y \rangle \arrow[d, "n'"] & \langle N, \Phi_N \rangle \arrow[l, "c"'] \arrow[dl, "c'"] \\
\langle H, \Phi_H \rangle &
\end{tikzcd}$$

The concept of jointly epimorphic morphism pairs is captured by the class $\mathcal{E}'^{surj}_{\Leftrightarrow,TG}$ given as follows:

Definition 7.1 (The class $\mathcal{E}'^{surj}_{\Leftrightarrow,TG}$)**.**
The class $\mathcal{E}'^{surj}_{\Leftrightarrow,TG}$ is given by all pairs (e_1, e_2) of typed symbolic graph morphisms $e_1 : \langle A_1, \Phi_1 \rangle \to \langle K, \Phi_K \rangle$ and $e_1 : \langle A_2, \Phi_2 \rangle \to \langle K, \Phi_K \rangle$ with the same codomain, such that there exists an $\mathcal{E}^{surj}_{\Leftrightarrow,TG}$-morphism $e : \langle A_{1+2}, \Phi_{1+2} \rangle \to \langle K, \Phi_K \rangle$ induced by the coproduct $(\langle A_{1+2}, \Phi_{1+2} \rangle, i_1, i_2)$.

$$\begin{tikzcd}
\langle A_1, \Phi_1 \rangle \arrow[r, "i_1"] \arrow[dr, "e_1"'] & \langle A_{1+2}, \Phi_{1+2} \rangle \arrow[d, "e"] & \langle A_2, \Phi_2 \rangle \arrow[l, "i_2"'] \arrow[dl, "e_2"] \\
& \langle K, \Phi_K \rangle &
\end{tikzcd}$$

Remark 7.2 (Construction of jointly epimorphic gluings)**.**
As e is in $\mathcal{E}^{surj}_{\Leftrightarrow,TG}$, Σ-formula Φ_K is equivalent to $\Phi_1[\hat{e}_1] \land \Phi_2[\hat{e}_2]$. This observation is especially important for an implementation, as it allows for constructing the formula component of the gluings at the syntactical level. More specifically, given two symbolic graphs $\langle A_1, \Phi_1 \rangle$ and $\langle A_2, \Phi_2 \rangle$, we construct symbolic graph $\langle K, \Phi_K \rangle$ with morphisms $e_1 : \langle A_1, \Phi_1 \rangle \to \langle K, \Phi_K \rangle$ and $e_2 : \langle A_2, \Phi_2 \rangle \to \langle K, \Phi_K \rangle$ such that $(e_1, e_2) \in \mathcal{E}'^{surj}_{\Leftrightarrow,TG}$ by first constructing E-graph K with jointly epimorphic E-graph morphisms e_1 and e_2. Subsequently we set Φ_K equal to $\Phi_1[\hat{e}_1] \land \Phi_2[\hat{e}_2]$ leading to $(e_1, e_2) \in \mathcal{E}'^{surj}_{\Leftrightarrow,TG}$.

Now we show that any pair $(e_1, e_2) \in \mathcal{E}'^{surj}_{\Leftrightarrow,TG}$ is indeed jointly epimorphic.

Lemma 7.3 (Any pair $(e_1, e_2) \in \mathcal{E}'^{surj}_{\Leftrightarrow,TG}$ is jointly epimorphic)**.**
Any morphism pair $(e_1, e_2) \in \mathcal{E}'^{surj}_{\Leftrightarrow,TG}$ is jointly epimorphic.

Proof. Given a pair $e_1 : \langle A_1, \Phi_1\rangle \to \langle K, \Phi_K\rangle$ and $e_2 : \langle A_2, \Phi_2\rangle \to \langle K, \Phi_K\rangle$ of typed symbolic graph morphisms with coproduct $(\langle A_{1+2}, \Phi_{1+2}\rangle, i_1, i_2)$ and induced morphism $e \in \mathcal{E}^{surj}_{\Leftrightarrow,TG}$, obtained according to Definition 7.1. For any morphism pair $g, h : \langle K, \Phi_K\rangle \to \langle C, \Phi_C\rangle$ in $\mathbb{TSG}_{\mathcal{D},TG}$, we have to show that if $g \circ e_i = h \circ e_i$, with $i = 1, 2$ then $g = h$.

$$\begin{tikzcd}
\langle A_1, \Phi_1\rangle \arrow[r, "i_1"] \arrow[dr, "e_1"'] & \langle A_{1+2}, \Phi_{1+2}\rangle \arrow[d, "e"] & \langle A_2, \Phi_2\rangle \arrow[l, "i_2"'] \arrow[dl, "e_2"] \\
& \langle K, \Phi_K\rangle \arrow[d, bend right, "g"'] \arrow[d, bend left, "h"] & \\
& \langle C, \Phi_C\rangle &
\end{tikzcd}$$

As we know from the coproduct that $e_1 = e \circ i_1$ and $e_2 = e \circ i_2$, this is equivalent to show that $g \circ e \circ i_i = h \circ e \circ i_i$, with $i = 1, 2$ implies $g = h$, which is a direct consequence of the fact that e is an epimorphism. □

The following construction for equivalent NACs from negative symbolic consistency constraints is the instantiation of Definition 5.20 for symbolic graphs. The construction is based on the fact that given a finite symbolic graph R^Φ and simple negative constraint $nc(N^\Phi)$ with an finite graph N^Φ, the set of jointly epimorphic morphism pairs $(R^\Phi \xrightarrow{y_i} Y_i^\Phi, N^\Phi \xrightarrow{c_i} Y_i^\Phi)$ is also finite. Thus, we can add for each pair of jointly epimorphic morphisms $(R^\Phi \xrightarrow{y_i} Y_i^\Phi, N^\Phi \xrightarrow{c_i} Y_i^\Phi)$ a simple NAC $nac_R(R^\Phi \xrightarrow{y_i} Y_i^\Phi)$ to NAC_R. Equivalent NACs for arbitrary negative symbolic constraints NC are derived by constructing the simple NACs for each simple negative symbolic constraint $nc(N^\Phi)$ in NC.

Definition 7.4 (Construction of NACs from negative constraints in $\mathbb{TSG}_{\mathcal{D},TG}$)**.** The construction of a NAC over typed symbolic graph R^Φ from a simple negative symbolic constraint $nc(N^\Phi)$ in $\mathbb{TSG}_{\mathcal{D},TG}$ is defined as

$$Acc_R(nc(N^\Phi)) = \bigcup_{i \in \mathbf{I}} \{nac_R(R^\Phi \xrightarrow{y_i} Y_i^\Phi)\},$$

where $\mathbf{I}$ ranges over all triples (Y_i^Φ, y_i, c_i) with morphisms $y_i : R^\Phi \to Y_i^\Phi$ and $c_i : N^\Phi \to Y_i^\Phi$ such that the pair $(y_i, c_i) \in \mathcal{E}'^{surj}_{\Leftrightarrow,TG}$.

$$\begin{tikzcd}
\langle R, \Phi_R\rangle \arrow[d, "y_i"] & \\
\langle Y_i, \Phi_{Yi}\rangle & \langle N, \Phi_N\rangle \arrow[l, "c_i"']
\end{tikzcd}$$

For a negative symbolic constraint NC, the construction is given by

$$Acc_R(NC) = \bigcup Acc_R(nc(N^\Phi)) \text{ for all } nc(N^\Phi) \in NC.$$

According to Definition 7.4, we can construct the extended symbolic production $\varrho = (p, \emptyset, Acc_R(NC))$ from a production $p = (L^\Phi \leftarrow K^\Phi \rightarrow R^\Phi)$ and a negative symbolic constraint NC. This construction is shown by the following example.

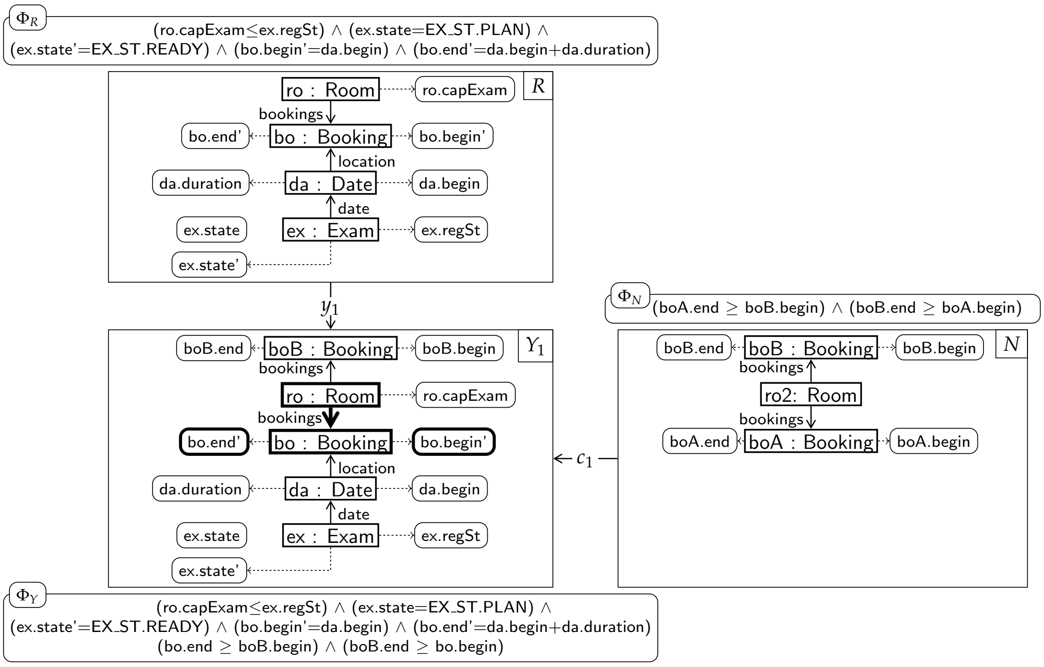


Figure 7.1: Simple right NAC $nac_R(R^\Phi \xrightarrow{y_1} Y_1^\Phi)$ for production projBookRoom and negative constraint NoCompetingBookings

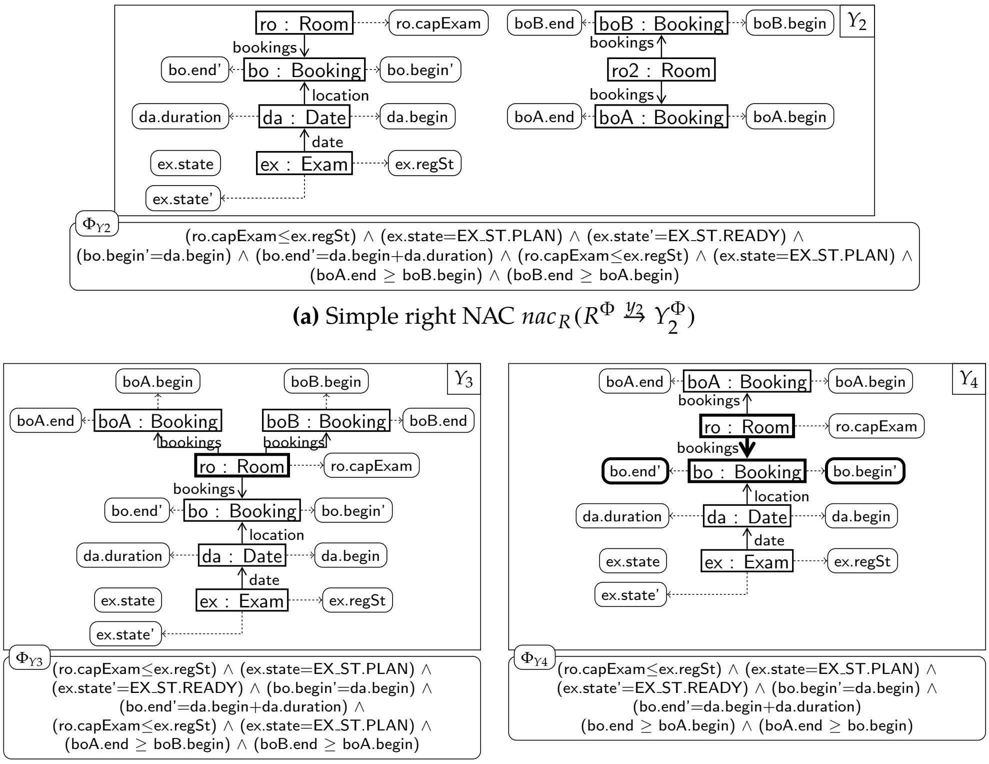
Example 7.5 (Construction of right NACs)**.**
This example presents the construction of the right NAC for production projBookRoom and symbolic negative constraint NoCompetingBookings originally presented in Figure 2.8.

Figure 7.1 shows the construction for one of the gluings in detail. More specifically, Figure 7.1 depicts the jointly epimorphic pair $(R^\Phi \xrightarrow{y_1} Y_1^\Phi, N^\Phi \xrightarrow{c_1} Y_1^\Phi)$, where the elements of R and N that are glued together are drawn bold in Y_1. Hence, the morphism y_1 is given by the correspondence of the node identifiers; morphism $c_1 : N \rightarrow Y_1$ is given by mapping ro2 to ro, boA to bo, and boB to boB. The mapping of the label nodes is determined by the following variable map

$$\hat{c}_1 : \begin{array}{cccc} \text{bo.begin'} & \text{bo.end'} & \text{boB.begin} & \text{boB.end} \\ \hline \text{boA.begin} & \text{boB.end} & \text{boB.begin} & \text{boB.end} \end{array} .$$

The Σ-formula Φ_{Y1} is defined by the conjunction $\Phi_R[\hat{y}_1] \wedge \Phi_N[\hat{c}_1]$. The resulting simple right NAC $nac_R(R^\Phi \xrightarrow{y_1} Y_1^\Phi)$ invalidates the application of production projBookRoom if the result contains another booking for Room ro1 with a time slot that overlaps with the time slot of the created Booking bo.

Figure 7.2 shows further gluings. The simple right NAC shown in Figure 7.2a is obtained by gluing R and N along the empty graph; simple right NAC shown


ro : Room
ro.capExam
bookings
bo.end'
bo : Booking
bo.begin'
location
da.duration
da : Date
da.begin
date
ex.state
ex : Exam
ex.regSt
ex.state'
boB.end
boB : Booking
boB.begin
bookings
ro2 : Room
bookings
boA.end
boA : Booking
boA.begin
Y2
ΦY2
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧ (ex.state'=EX_ST.READY) ∧
(bo.begin'=da.begin) ∧ (bo.end'=da.begin+da.duration) ∧ (ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(boA.end ≥ boB.begin) ∧ (boB.end ≥ boA.begin)

(a) Simple right NAC $nac_R(R^\Phi \xrightarrow{y_2} Y_2^\Phi)$

boA.begin
boB.begin
Y3
boA.end
boA : Booking
boB : Booking
boB.end
bookings
bookings
ro : Room
ro.capExam
bookings
bo.end'
bo : Booking
bo.begin'
location
da.duration
da : Date
da.begin
date
ex.state
ex : Exam
ex.regSt
ex.state'
ΦY3
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧
(bo.end'=da.begin+da.duration) ∧
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(boA.end ≥ boB.begin) ∧ (boB.end ≥ boA.begin)
boA.end
boA : Booking
boA.begin
Y4
bookings
ro : Room
ro.capExam
bookings
bo.end'
bo : Booking
bo.begin'
location
da.duration
da : Date
da.begin
date
ex.state
ex : Exam
ex.regSt
ex.state'
ΦY4
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧
(bo.end'=da.begin+da.duration)
(bo.end ≥ boA.begin) ∧ (boA.end ≥ bo.begin)

(b) Simple right NAC $nac_R(R^\Phi \xrightarrow{y_3} Y_3^\Phi)$

(c) Simple right NAC $nac_R(R^\Phi \xrightarrow{y_4} Y_4^\Phi)$

Figure 7.2: Construction of simple right NACs for production projBookRoom and negative constraint NoCompetingBookings

in Figure 7.2b is obtained by gluing R and N along ro. The simple right NAC in Figure 7.2c is obtained similar to $nac_R(R^\Phi \xrightarrow{y_1} Y_1^\Phi)$, but gluing boB to bo instead of gluing boA to bo. Note that there are many more gluings. However, we can dramatically reduce their number if we assume that all graphs are linear. This assumption is valid, as projective productions are assumed to be linear (i. e. they consist of linear graphs only); hence, the result of applying a linear production to a linear graph, is again a linear graph. Moreover, we may assume that a booking belongs to at most one room (containment association, see Figure 2.1).

Based on these results of Section 5.3, we can show that Definition 7.4 indeed leads to an equivalent negative application condition in the category $\mathbb{TSG}_{\mathcal{D},TG}$ in the following sense:

Theorem 7.6 (Construction of equivalent NACs in $\mathbb{TSG}_{\mathcal{D},TG}$)**.**
For any negative constraint NC and every graph $R^\Phi = \langle R, \Phi_R \rangle$ in $\mathbb{TSG}_{\mathcal{D},TG}$ with $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $n : R^\Phi \to H^\Phi$, we have

$$n \Vdash Acc_R(NC) \text{ iff } H^\Phi \Vdash NC.$$

Proof. This is a direct consequence of Theorem 5.21 and the fact that category $\mathbb{TSG}_{\mathcal{D},TG}$ is $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive (for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$, $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$), has binary coproducts as well as an $\mathcal{E}$–$\mathcal{N}$-factorization for $\mathcal{E} = \mathcal{E}^{surj}_{\Leftrightarrow,TG}$. □

7.2 Construction of Equivalent Left From right NACs

As shown in the previous section, given a production $p = (L^\Phi \leftarrow K^\Phi \rightarrow R^\Phi)$, we are able to construct an extended production $\varrho = (p, \emptyset, Acc_R(NC))$ with equivalent right NAC for any negative constraint NC. This construction guarantees that there is no direct transformation $G^\Phi \xRightarrow{\varrho @ m} H^\Phi$ that leads to an inconsistent result. However, in practice one has perform the transformation first, to decide afterwards whether the result satisfies the right NAC. Especially for security or safety critical applications it is inevitable to identify actions that lead to a constraint violation *before* actually executing them. In the following, we present the construction of equivalent symbolic left NACs from symbolic right NACs. As shown in Section 6.2, only functional projective productions provide the $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property, which is required for this construction. For this reason, we require in the following that all productions are typed functional projective.

Basically, an equivalent left NAC is derived from a right NAC by applying the production in reverse direction to each simple right NAC. The following construction is an instantiation of Definition 5.22 for typed functional projective productions.

Definition 7.7 (Construction of equivalent left NACs).
Given an extended production ϱ over typed functional projective production $p = (L^\Phi \leftarrow K^\Phi \rightarrow R^\Phi)$ with right negative application condition NAC_R. For a simple right NAC $nac_R(R^\Phi \xrightarrow{y} Y^\Phi) \in NAC_R$, let

$$shift_\varrho(nac_R((R^\Phi \xrightarrow{y} Y^\Phi)) = \{nac_L(L^\Phi \xrightarrow{x} X^\Phi)\}$$

be the singleton set constructed from $nac_R(R^\Phi \xrightarrow{y} Y^\Phi)$ as follows:

$$\begin{tikzcd}
\langle L, \Phi_L \rangle \arrow[d, "x"'] \arrow[dr, phantom, "(2)"] & \langle K, \Phi_L \rangle \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(1)"] & \langle R, \Phi_R \rangle \arrow[d, "y"] \\
\langle X, \Phi_X \rangle & \langle Z, \Phi_Z \rangle \arrow[l, "l'"] \arrow[r, "r'"'] & \langle Y, \Phi_Y \rangle
\end{tikzcd}$$

If the pair $r : \langle K, \Phi_L \rangle \rightarrow \langle R, \Phi_R \rangle$ and $y : \langle R, \Phi_R \rangle \rightarrow \langle Y, \Phi_Y \rangle$ has a pushout complement in $\mathbb{TSG}_{\mathcal{D},TG}$, choose $shift_\varrho(nac_R(R^\Phi \xrightarrow{y} Y^\Phi)) = \{nac_L(L^\Phi \xrightarrow{x} X^\Phi)\}$, where x is given by the POs (1) and (2); otherwise $shift_\varrho(nac_R(R^\Phi \xrightarrow{y} Y^\Phi)) = \emptyset$.

A left NAC from a right negative application condition NAC_R is obtained as follows:

$$shift_\varrho(NAC_R) = \bigcup shift_\varrho(nac_R(R^\Phi \xrightarrow{y_i} Y_i^\Phi)) \text{ for all } nac_R(R^\Phi \xrightarrow{y_i} Y_i^\Phi) \in NAC_R.$$

By Definition 7.7, we can derive extended production $\varrho' = (p, \mathit{shift}_\varrho(\mathit{NAC}_R), \emptyset)$ from any extended functional projective production $\varrho = (p, \emptyset, \mathit{NAC}_R)$ by simply shifting the right NACs to the left, as shown in the next example.

Example 7.8 (Construction of left from right NACs)**.**
Figure 7.3 shows the derivation of the simple left negative application condition $nac_L(L^\Phi \xrightarrow{x_1} X_1^\Phi)$ from the simple right negative application condition $nac_R(R^\Phi \xrightarrow{y_1} Y_1^\Phi)$, which was originally presented in Example 7.5. Note that production projBookRoom is functional projective . The resulting left negative application condition $nac_L(L^\Phi \xrightarrow{x_1} X_1^\Phi)$ prevents the production projBookRoom to be applied to a symbolic graph that still contains a booking, whose date is in conflict with the exam date.

The simple left NACs $nac_L(L^\Phi \xrightarrow{x_2} X_2^\Phi)$, $nac_L(L^\Phi \xrightarrow{x_3} X_3^\Phi)$, and $nac_L(L^\Phi \xrightarrow{x_4} X_4^\Phi)$ (shown in Figures 7.4a–7.4c) are derived in a similar manner from simple right NACs $nac_R(R^\Phi \xrightarrow{y_2} Y_2^\Phi)$, $nac_R(R^\Phi \xrightarrow{y_3} Y_3^\Phi)$, and $nac_R(R^\Phi \xrightarrow{y_4} Y_4^\Phi)$ (shown in Figures 7.2a–7.2c), respectively.

The extended functional projective production ϱ' is then defined as:

$$\varrho' = (p', \mathit{NAC}'_L, \emptyset), \text{ where } \mathit{NAC}'_L = \bigcup_{i \in \{1,2,3,4\}} \{nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi)\}.$$

Based on the results presented in Section 5.3, we are now able to show that the derived left NACs and right NACs are indeed equivalent in the following sense:

Theorem 7.9 (Equivalent left NACs for functional projective productions)**.**
Given an extended production $\varrho = (p, \mathit{shift}_\varrho(\mathit{NAC}_R), \mathit{NAC}_R)$ over functional projective production p and left NAC $\mathit{shift}_\varrho(\mathit{NAC}_R)$ derived from NAC_R according to Definition 7.7, then for all direct transformations $G^\Phi \xRightarrow{\varrho@m} H^\Phi$ via ϱ with match $m \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ and comatch $n \in \mathcal{M}^{inj}_{\Rightarrow,TG}$

$$m \Vdash \mathit{shift}_p(\mathit{NAC}_R) \text{ iff } n \Vdash \mathit{NAC}_R.$$

Proof. The proof follows directly from Theorem 5.23 and the fact that category $\mathbb{TSG}_{\mathcal{D},TG}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive (for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$, $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$), $\mathcal{M}^{inj}_{Func,TG}$ is a subclass of $\mathcal{M}^{inj}_{Proj,TG}$, as well as the fact that category $\mathbb{TSG}_{\mathcal{D},TG}$ has the $\mathcal{R}$–$\mathcal{N}$-PO–PB decomposition property for $\mathcal{R} = \mathcal{M}^{inj}_{Func,TG}$. □

Note that the equivalence of NAC_R and $\mathit{shift}_p(\mathit{NAC}_R)$ implies the equivalence of productions $\varrho = (p, \emptyset, \mathit{NAC}_R)$ and $\varrho' = (p, \mathit{shift}_\varrho(\mathit{NAC}_R), \emptyset)$; that is $G^\Phi \xRightarrow{\varrho@m} H^\Phi$ is a direct transformation if and only if $G^\Phi \xRightarrow{\varrho'@m} H^\Phi$ is a direct transformation.

7.3 Minimization of Symbolic Negative Application Conditions

Using the results of Theorem 7.6 and Theorem 7.9, we are able to construct an extended production $\varrho' = (p, \mathit{shift}_\varrho(\mathit{Acc}_R(\mathit{NC})), \emptyset)$ from any functional projective production p and negative constraint NC in the category $\mathbb{TSG}_{\mathcal{D},TG}$ such that for

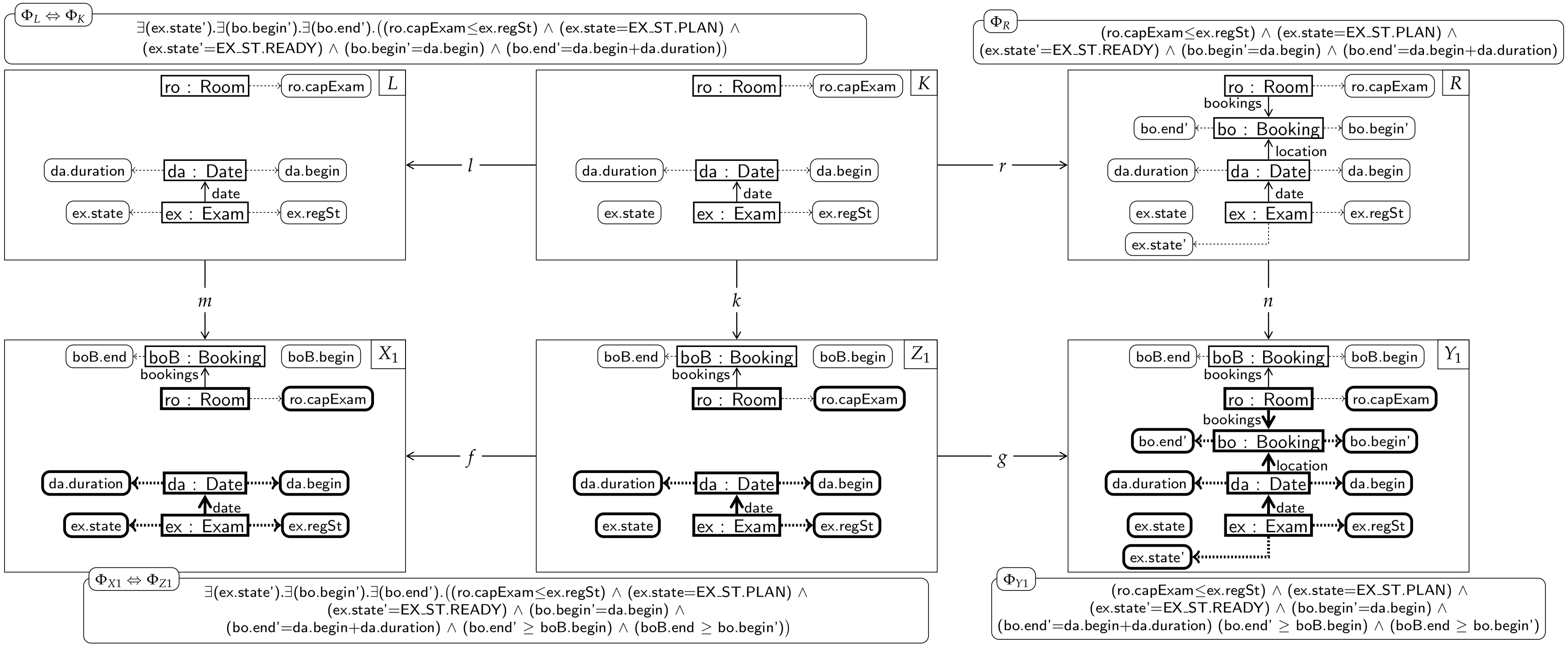
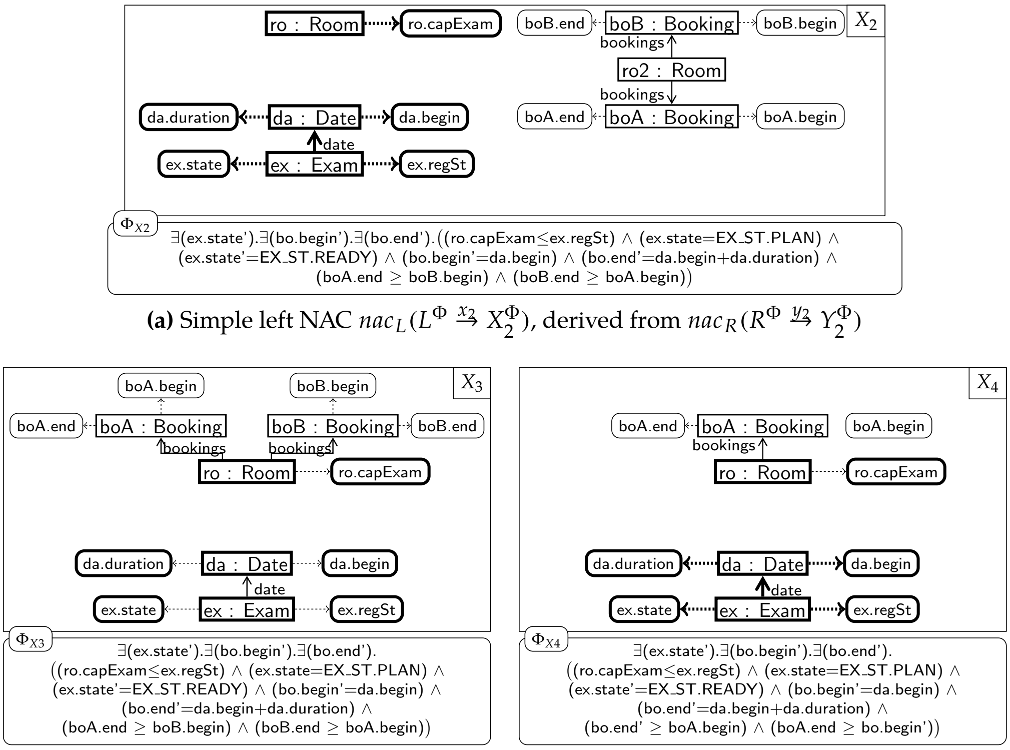

Φ_L ⇔ Φ_K
∃(ex.state').∃(bo.begin').∃(bo.end').((ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧ (bo.end'=da.begin+da.duration))
Φ_R
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧ (bo.end'=da.begin+da.duration)
L
ro : Room
ro.capExam
da.duration
da : Date
da.begin
date
ex.state
ex : Exam
ex.regSt
l
K
r
R
bookings
bo.end'
bo : Booking
bo.begin'
location
ex.state'
m
k
n
X1
boB.end
boB : Booking
boB.begin
Z1
f
g
Y1
Φ_X1 ⇔ Φ_Z1
∃(ex.state').∃(bo.begin').∃(bo.end').((ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧
(bo.end'=da.begin+da.duration) ∧ (bo.end' ≥ boB.begin) ∧ (boB.end ≥ bo.begin'))
Φ_Y1
(ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧
(bo.end'=da.begin+da.duration) (bo.end' ≥ boB.begin) ∧ (boB.end ≥ bo.begin')

Figure 7.3: Derivation of simple left NAC $nac_L(L^\Phi \xrightarrow{x_1} X_1^\Phi)$ from right NAC $nac_R(R^\Phi \xrightarrow{y_1} Y_1^\Phi)$


ro : Room
ro.capExam
boB.end
boB : Booking
boB.begin
X_2
bookings
ro2 : Room
bookings
da.duration
da : Date
da.begin
boA.end
boA : Booking
boA.begin
date
ex.state
ex : Exam
ex.regSt
Φ_{X2}
∃(ex.state').∃(bo.begin').∃(bo.end').((ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧ (bo.end'=da.begin+da.duration) ∧
(boA.end ≥ boB.begin) ∧ (boB.end ≥ boA.begin))
boA.begin
boB.begin
X_3
boA.end
boA : Booking
boB : Booking
boB.end
bookings
bookings
ro : Room
ro.capExam
da.duration
da : Date
da.begin
date
ex.state
ex : Exam
ex.regSt
Φ_{X3}
∃(ex.state').∃(bo.begin').∃(bo.end').
((ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧
(bo.end'=da.begin+da.duration) ∧
(boA.end ≥ boB.begin) ∧ (boB.end ≥ boA.begin))
X_4
boA.end
boA : Booking
boA.begin
bookings
ro : Room
ro.capExam
da.duration
da : Date
da.begin
date
ex.state
ex : Exam
ex.regSt
Φ_{X4}
∃(ex.state').∃(bo.begin').∃(bo.end').
((ro.capExam≤ex.regSt) ∧ (ex.state=EX_ST.PLAN) ∧
(ex.state'=EX_ST.READY) ∧ (bo.begin'=da.begin) ∧
(bo.end'=da.begin+da.duration) ∧
(bo.end' ≥ boA.begin) ∧ (boA.end ≥ bo.begin'))

(a) Simple left NAC $nac_L(L^\Phi \xrightarrow{x_2} X_2^\Phi)$, derived from $nac_R(R^\Phi \xrightarrow{y_2} Y_2^\Phi)$

(b) Simple left NAC $nac_L(L^\Phi \xrightarrow{x_3} X_3^\Phi)$, derived from $nac_R(R^\Phi \xrightarrow{y_3} Y_3^\Phi)$

(c) Simple left NAC $nac_L(L^\Phi \xrightarrow{x_4} X_4^\Phi)$, derived from $nac_R(R^\Phi \xrightarrow{y_4} Y_4^\Phi)$

Figure 7.4: Derived simple left NACs for production projBookRoom.

all direct transformations $G^\Phi \xRightarrow{\varrho'@m} H^\Phi$, the derived typed symbolic graph H^Φ is consistent with respect to NC. However, not all generated left NACs are required to ensure that extended production ϱ' is consistency preserving. In the following, we shall see how to reduce the number of generated application conditions. This is especially interesting from a practical point of view, as each application condition creates overhead when checking the applicability of a production.

Note that although we can construct equivalent left NACs from right NACs for functional projective transformations only, the results for minimizing NACs presented in the following also apply for projective productions, as it is irrelevant how the left NACs are obtained.

7.3.1 *Consistency Preserving Minimization of left NACs*

Recall, that consistency preservation (Definition 3.32) just requires the result of a direct transformation to be consistent if the symbolic graph was consistent before the transformation. Especially if we assume that the initial graph is consistent, it is sufficient to require that each production preserves consistency to ensure that inconsistent graphs are unreachable.

The following theorem shows that consistency preservation of an extended projective production ϱ with respect to a negative constraint NC is retained if we remove simple left NACs that are inconsistent with respect to NC.

Theorem 7.10 (Consistency preserving minimization of left NACs).
Given an extended production $\varrho = (p, NAC_L, \emptyset)$ that is consistency preserving with respect to a negative constraint NC, then any extended production $\varrho' = (p, NAC'_L, \emptyset)$ with left NAC

$$NAC'_L = NAC_L \backslash \{nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi)\} \text{ for a } nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi) \in NAC_L$$

is consistency preserving with respect to NC if $X_i^\Phi \not\Vdash NC$.

Proof. Given extended productions $\varrho = (p, NAC_L, \emptyset)$ and $\varrho' = (p, NAC'_L, \emptyset)$ with typed projective production $p = \langle \Phi_L, L \leftarrow K \rightarrow R, \Phi_R \rangle$, where negative application condition NAC'_L is defined as $NAC'_L = NAC_L \backslash \{nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi)\}$ such that $X_i^\Phi \not\Vdash NC$.

We prove this theorem by contradiction. To that end, suppose that extended production ϱ is consistency preserving with respect to NC, but ϱ' is not. Then there must exist a typed symbolic graph G^Φ and match $m : L^\Phi \rightarrow G^\Phi$ leading to direct transformation $G^\Phi \overset{\varrho'@m}{\Longrightarrow} H^\Phi$ via ϱ' such that $G^\Phi \Vdash NC$ and $H^\Phi \not\Vdash NC$. By assumption, the extended production ϱ is consistency preserving; hence, there must be a simple negative application condition in NAC_L that prevents the application of ϱ at match m.

$$\begin{tikzcd}
\langle X_i, \Phi_{Xi} \rangle \arrow[drr, "m'"] & & \langle L, \Phi_L \rangle \arrow[ll, "x_i"'] \arrow[d, "m"] \\
& & \langle G, \Phi_G \rangle \\
& \langle N, \Phi_N \rangle \arrow[uul, bend left, "c"] &
\end{tikzcd}$$

As $NAC'_L = NAC_L \backslash \{nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi)\}$, it must be the case that ϱ cannot be applied at match m because of $m \not\Vdash nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi)$, which implies the existence of a morphism $m' : X_i^\Phi \rightarrow G^\Phi$ such that $m = m' \circ x_i$. By assumption $X_i^\Phi \not\Vdash NC$, thus there is a simple negative constraint $nc(N^\Phi) \in NC$ with morphism $c : N^\Phi \rightarrow X_i^\Phi$. This means we can construct morphisms $(m' \circ c) : N^\Phi \rightarrow G^\Phi$, which is a contradiction as it implies that G^Φ is inconsistent with respect to NC. □

Example 7.11 (Consistency preserving minimization of left NACs).
Consider the left NAC with simple left NACs $nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi)$ for $i \in \{1, 2, 3, 4\}$ derived in Example 7.8 (see Figure 7.3 and Figure 7.4). If we take a closer look at the simple left NAC $nac_L(L^\Phi \xrightarrow{x_2} X_2^\Phi)$ we can see that it prevents the application of production projBookRoom if there is another room ro2 with a pair of bookings that have mutually overlapping time slots. This means, if $nac_L(L^\Phi \xrightarrow{x_2} X_2^\Phi)$ prevents the application of rule projBookRoom to a graph G^Φ, then G^Φ is already inconsistent with respect to negative constraint NoCompetingBookings. Similarly $nac_L(L^\Phi \xrightarrow{x_3} X_3^\Phi)$ prevents production projBookRoom to be applied to a room ro that already has a pair of bookings with mutually overlapping time slots.

According to Theorem 7.10 we may remove simple left NACs $nac_L(L^\Phi \xrightarrow{x_2} X_2^\Phi)$ and $nac_L(L^\Phi \xrightarrow{x_3} X_3^\Phi)$ such that the resulting extended production

$$\varrho'' = (p', NAC''_L, \emptyset), \text{ with } NAC''_L = \{nac_L(L^\Phi \xrightarrow{x_1} X_1^\Phi), nac_L(L^\Phi \xrightarrow{x_4} X_4^\Phi)\}$$

remains consistency preserving.

7.3.2 *Minimization of Subsumed left NACs*

Hence after removing those NACs that are not required for preserving consistency, there might be NACs that are subsumed by other NACs and, thus, may be removed without altering the semantics of a production. Basically, a *subsumed* NAC is a simple negative application condition for which there exists an other simple application condition that is a subgraph of the corresponding NAC.

Definition 7.12 (Subsumed NACs).
Given a negative application condition NAC_L over symbolic graph $\langle L, \Phi_L \rangle$ then a simple negative application condition $nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi) \in NAC_L$ is *subsumed* by an other application condition $nac_L(L^\Phi \xrightarrow{x_j} X_j^\Phi) \in NAC_L$ with $X_i \neq X_j$ if there exists an $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $s : \langle X_j, \Phi_{Xj} \rangle \to \langle X_i, \Phi_{Xi} \rangle$ such that $s \circ x_j = x_i$.

Removing subsumed simple negative application conditions from a NAC does not alter its semantics:

Proposition 7.13 (Subsumed NACs).
Given a negative application condition NAC_L over the symbolic graph $\langle L, \Phi_L \rangle$ with simple negative application conditions $nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi), nac_L(L^\Phi \xrightarrow{x_j} X_j^\Phi) \in NAC_L$ such that $nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi)$ is subsumed by $nac_L(L^\Phi \xrightarrow{x_j} X_j^\Phi)$. Then for any $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $m : \langle L, \Phi_L \rangle \to \langle G, \Phi_G \rangle$, we have $m \Vdash NAC_L$ if and only if $m \Vdash NAC'_L$, where

$$NAC'_L = NAC_L \backslash \{nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi)\}.$$

Proof. Without loss of generality we may assume that

$$NAC_L = \{nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi), nac_L(L^\Phi \xrightarrow{x_j} X_j^\Phi)\} \text{ and } NAC'_L = \{nac_L(L^\Phi \xrightarrow{x_j} X_j^\Phi)\}.$$

Now assume that $nac_L(L^\Phi \xrightarrow{x_j} X_j^\Phi)$ subsumes $nac_L(L^\Phi \xrightarrow{x_i} X_i^\Phi)$; that is, there exists a morphism $s : \langle X_j, \Phi_{Xj} \rangle \to \langle X_i, \Phi_{Xi} \rangle$. We proof the theorem by contradiction.

Case 1. Given an $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $m : \langle L, \Phi_L \rangle \to \langle G, \Phi_G \rangle$ such that $m \Vdash NAC_L$ and $m \nVdash NAC'_L$. Thus, there must be an $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $p_j : \langle X_j, \Phi_{Xj} \rangle \to \langle G, \Phi_G \rangle$, such that $p_j \circ x_j = m$. As $nac_L(L^\Phi \xrightarrow{x_j} X_j^\Phi)$ is also in NAC_L, we have $m \nVdash NAC_L$ which is a contradiction.

Case 2. Given any $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $m : \langle L, \Phi_L \rangle \to \langle G, \Phi_G \rangle$ such that $m \nVdash NAC_L$ and $m \Vdash NAC'_L$. Hence, we have the following cases:

a) There exists an $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $p_j : \langle X_j, \Phi_X j \rangle \to \langle G, \Phi_G \rangle$ such that $p_j \circ x_j = m$. Thus $m \nVdash NAC'_L$, which is a contradiction.

b) There exists an $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $p_i : \langle X_i, \Phi_i\rangle \to \langle G, \Phi_G\rangle$ such that $p_i \circ x_i = m$; hence, there is a morphism $(p_i \circ s) : \langle X_j, \Phi_{Xj}\rangle \to \langle G, \Phi_G\rangle$ with $p_i \circ s \circ x_j = m$. Thus $m \not\models NAC'_L$, which is a contradiction.

$$\begin{tikzcd}
\langle X_i, \Phi_{Xi}\rangle \arrow[dr, "p_i"'] & \langle L, \Phi_L\rangle \arrow[l, "x_i"'] \arrow[r, "x_j"] \arrow[d, "m"] & \langle X_j, \Phi_{Xj}\rangle \arrow[ll, bend right, "s"'] \\
& \langle G, \Phi_G\rangle &
\end{tikzcd}$$

□

In the following, we define *essential* NACs, which are basically those NACs that remain after removing consistency guaranteeing and subsumed NACs:

Definition 7.14 (Essential NAC).
Given a set of negative constraints *NC* and an application condition NAC_L over a symbolic graph $\langle L, \Phi_L\rangle$, NAC_L is *essential* if all simple negative application condition in NAC_L are consistency preserving but not consistency guaranteeing with respect to *NC* and there does not exist a simple negative application condition in NAC_L that is subsumed by an other application condition in NAC_L.

Example 7.15 (Subsumed and essential NACs).
Consider the extended production

$$\varrho'' = (p', NAC''_L, \emptyset), \text{ with } NAC''_L = \{nac_L(L^\Phi \xrightarrow{x_1} X_1^\Phi), nac_L(L^\Phi \xrightarrow{x_4} X_4^\Phi)\}$$

after the minimization performed in Example 7.11. It can be seen that symbolic graphs $\langle X_1, \Phi_{X1}\rangle$ and $\langle X_4, \Phi_{X4}\rangle$ are isomorphic; consequently, there exists an symbolic graph isomorphism $s : \langle X_1, \Phi_{X1}\rangle \to \langle X_4, \Phi_{X4}\rangle$. Moreover, we have that $s \circ x_1 = x_4$ and $s^{-1} \circ x_4 = x_1$. Hence, according to Proposition 7.13, we may remove either $nac_L(L^\Phi \xrightarrow{x_1} X_1^\Phi)$ or $nac_L(L^\Phi \xrightarrow{x_4} X_4^\Phi)$ from NAC''_L, and in both cases, the resulting negative application conditions

$$NAC^*_L = \{nac_L(L^\Phi \xrightarrow{x_1} X_1^\Phi)\} \text{ and } NAC^\diamond_L = \{nac_L(L^\Phi \xrightarrow{x_4} X_4^\Phi)\}$$

are essential.

8

CONFLICT DETECTION AND RESOLUTION

In this chapter we present our results on conflict detection and resolution for attributed graph transformation systems. Intuitively, two transformations of the same graph have no conflict if the result of executing them in parallel is the same as executing them serially. Accordingly, two transformation are in conflict if the results of executing them in parallel differs from the result of a serialized execution. A conflict of two transformation for the same graph can be *resolved* if the outputs of first transformations can be joined again, i. e., there are transformations that lead to the same result.

The main contributions of this chapter is the extension of our results presented in [KDL^{+}15] to local confluence for projective graph transformation on arbitrary symbolic graphs. Moreover, we propose local confluence modulo normal form equivalence to increase the precision of the confluence analysis.

We begin with introducing the different properties for characterizing conflicts and their resolution, and discuss their application to conflict analysis in Section 8.1. In Section 8.2 we present the notions required to lift conflict analysis from the transformation level to the production level. In Section 8.3, we present our main contribution, namely the Local Confluence Theorem for projective graph transformation systems. For the rest of this chapter we assume that the category $\mathbb{TSG}_{\mathcal{D},TG}$ is defined by a symbolic type graph TG^{Φ} and a Σ-structure $\mathcal{D}$.

8.1 Conflicts and Conflict Resolution

As mentioned before, two transformations of the same graph are not in conflict if the result of executing them in parallel is the same as executing them serially. This property is captured by the concept of *parallel independence*. The idea of parallel independence is shown in Figure 8.1a; that is, two parallel independent transformations of the same graph can be executed in parallel, and the result is the same as executing the transformations arbitrarily serialized. For two direct transformations without negative application conditions, this is the case if none of the involved transformations deletes an element that is in the match of the other.

However, in many cases it is sufficient that a pair of parallel dependent (i. e., not parallel independent) transformation can be joined again; that is, there exist other transformations leading from results of the diverging transformations to the same result. This brings us to the concept of conflict resolution. *Conflict resolution* is usually captured by the concept of *confluence*. Figures 8.1b–8.1d show different forms of confluence that can be used to characterize different strategies for conflict resolution.

$$\begin{tikzcd}
 & G^\Phi \arrow[dl, Rightarrow, "p_1@m_1"'] \arrow[dr, Rightarrow, "p_2@m_2"] \arrow[dd, Rightarrow, "p_1+p_2" description] & \\
H_1^\Phi \arrow[dr, Rightarrow, "p_2@m_2"'] & & H_2^\Phi \arrow[dl, Rightarrow, "p_1@m_1"] \\
 & H_3^\Phi &
\end{tikzcd}$$

(a) Parallel Independence

$$\begin{tikzcd}
 & G^\Phi \arrow[dl, Rightarrow, "*"'] \arrow[dr, Rightarrow, "*"] & \\
H_1^\Phi \arrow[dr, Rightarrow, "*"] & & H_2^\Phi \arrow[dl, Rightarrow, "*"'] \\
 & H_3^\Phi &
\end{tikzcd}$$

(b) Confluence

$$\begin{tikzcd}
 & G^\Phi \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & \\
H_1^\Phi \arrow[dr, Rightarrow, "*"] & & H_2^\Phi \arrow[dl, Rightarrow, "*"'] \\
 & H_3^\Phi &
\end{tikzcd}$$

(c) Local Confluence

$$\begin{tikzcd}
 & G^\Phi \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & \\
H_1^\Phi \arrow[dr, Rightarrow, "0..1"] & & H_2^\Phi \arrow[dl, Rightarrow, "0..1"'] \\
 & H_3^\Phi &
\end{tikzcd}$$

(d) Subcommutativity

Figure 8.1: Commutativity Properties

In the context of typed projective graph transformation systems, *confluence* means if whenever typed symbolic graph G^Φ can be transformed into typed symbolic graphs H_1^Φ and H_2^Φ, there are transformations leading to the graph H_3^Φ, as shown in Figure 8.1b. Note that $H_1^\Phi \overset{*}{\Longrightarrow} H_3^\Phi \overset{*}{\Longleftarrow} H_2^\Phi$ means that there exist the following sequences of direct transformations

$$H_1^\Phi\Phi \simeq A_0^\Phi \Longrightarrow \ldots \Longrightarrow A_n^\Phi \simeq H_3^\Phi \simeq B_m^\Phi \Longleftarrow \ldots \Longleftarrow B_0^\Phi \simeq H_2^\Phi$$

for some $n \geq 0$ and $m \geq 0$ (see Definition 3.24). Hence, $\overset{*}{\Longrightarrow}$ is a relation on isomorphisms classes of (typed) symbolic graphs.

A weaker form of confluence is local confluence, shown in Figure 8.1c. In case of local confluence we require that H_1^Φ and H_2^Φ can be transformed to H_3^Φ when H_1^Φ and H_2^Φ are obtained by a direct transformation from G^Φ. As shown by Newmann, local confluence implies (global) confluence if the given transformation system is terminating [New42]. Moreover, if a graph transformation system is (locally) confluent and terminating it is functional; that is, given a set of productions and a start graph, then applying the productions as long as possible leads always to the same result, independently from the actual sequence in which the productions are applied. Functional behaviour is an important property in the context of model transformations, where we often expect that the result of transforming a model is the same for each run.

However, for analysing reactive systems (e. g. our running example) local confluence seems inadequate as those systems are nonterminating by design. Nevertheless, conflict resolution for nonterminating systems can be performed by requiring subcommutativity, shown in Figure 8.1d, as subcommutativity implies confluence without requiring termination [EEKR99]. Basically, two direct transformation $H_1^\Phi \Longleftarrow G^\Phi \Longrightarrow H_2^\Phi$ are *subcommutative* if there are transformations $H_1^\Phi \overset{0..1}{\Longrightarrow} H_3^\Phi \overset{0..1}{\Longleftarrow} H_2^\Phi$, whereas $H_1^\Phi \overset{0..1}{\Longrightarrow} H_3^\Phi$ means that either H_1^Φ is isomorphic to H_3^Φ (i. e. $H_1^\Phi \simeq H_3^\Phi$) or there is a direct transformation $H_1^\Phi \Longrightarrow H_3^\Phi$.

8.1.1 *Independence, Local Confluence and Subcommutativity*

In the following, we give formal definitions for parallel independence, local confluence, and subcommutativity for typed projective graph transformation systems.

Basically, two direct transformations of the same graph are parallel dependent if one transformation deletes an element that is in the match of the other.

Definition 8.1 (Parallel independence for **TPGTS**)**.**
Let **TPGTS** be a typed projective graph transformation system, then two direct transformations

$$\langle H_1, \Phi_{H1}\rangle \overset{p_1@m_1}{\Longleftarrow} \langle G, \Phi_G\rangle \overset{p_2@m_2}{\Longrightarrow} \langle H_2, \Phi_{H2}\rangle,\ p_1, p_2 \in \mathbf{P}$$

are parallel independent if there exist $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphisms

$$i : \langle L_1, \Phi_{L1}\rangle \to \langle D_2, \Phi_{D2}\rangle \text{ and } j : \langle L_2, \Phi_{L2}\rangle \to \langle D_1, \Phi_{D1}\rangle$$

such that $g_2 \circ i = m_1$ and $g_1 \circ j = m_2$.

$$\begin{tikzcd}
\langle R_1, \Phi_{R1}\rangle \arrow[d, "n_1"] & \langle K_1, \Phi_{K1}\rangle \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "k_1"] & \langle L_1, \Phi_{L1}\rangle \arrow[dr, "m_1"] \arrow[drrr, "i"] & & \langle L_2, \Phi_{L2}\rangle \arrow[dl, "m_2"'] \arrow[dlll, "j"'] & \langle K_2, \Phi_{K2}\rangle \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "k_2"] & \langle R_2, \Phi_{R2}\rangle \arrow[d, "n_2"] \\
\langle H_1, \Phi_{H1}\rangle & \langle D_1, \Phi_{D1}\rangle \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & \langle G, \Phi_G\rangle & & \langle D_2, \Phi_{D2}\rangle \arrow[ll, "g_2"'] \arrow[r, "h_2"] & \langle H_2, \Phi_{H2}\rangle
\end{tikzcd}$$

The existence of morphisms $i : \langle L_1, \Phi_{L1}\rangle \to \langle D_2, \Phi_{D2}\rangle$ and $j : \langle L_2, \Phi_{L2}\rangle \to \langle D_1, \Phi_{D1}\rangle$ and $g_2 \circ i = m_1$ and $g_1 \circ j = m_2$ ensures that all elements in the matches m_1 and m_2 are not deleted by the other production, respectively.

The next theorem states that any parallel independent pair of direct transformations is subcommutative and, therefore, confluent. This property is interesting, as checking parallel independence is computationally less complex than checking subcommutativity or local confluence. Hence, as we shall see later, parallel independence serves as a first filter criterion for subsequent confluence analysis steps.

Theorem 8.2 (Parallel Local Church–Rosser Theorem for **TPGTS**)**.**
Let **TPGTS** be a typed projective graph transformation system and let

$$\langle H_1, \Phi_{H1}\rangle \overset{p_1@m_1}{\Longleftarrow} \langle G, \Phi_G\rangle \overset{p_2@m_2}{\Longrightarrow} \langle H_2, \Phi_{H2}\rangle,\ p_1, p_2 \in \mathbf{P},$$

be two parallel independent direct transformations then there is a typed symbolic graph $\langle H_3, \Phi_{H3}\rangle$ and direct transformations

$$\langle H_1, \Phi_{H1}\rangle \overset{p_2@m_2'}{\Longrightarrow} \langle H_3, \Phi_{H3}\rangle \overset{p_1@m_1'}{\Longleftarrow} \langle H_2, \Phi_{H2}\rangle.$$

Proof. This is a direct consequence of Theorem 5.25 and the fact that typed projective graph transformation systems are $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive (Corollary 6.8). □

Example 8.3 (Parallel dependent pair of direct transformations)**.**
Figure 8.2 shows an example for a pair of parallel dependent direct transformation via productions unregExam (left) and regExam (right), originally introduced in Chapter 2. However, to be able to print the critical pair on a single page,

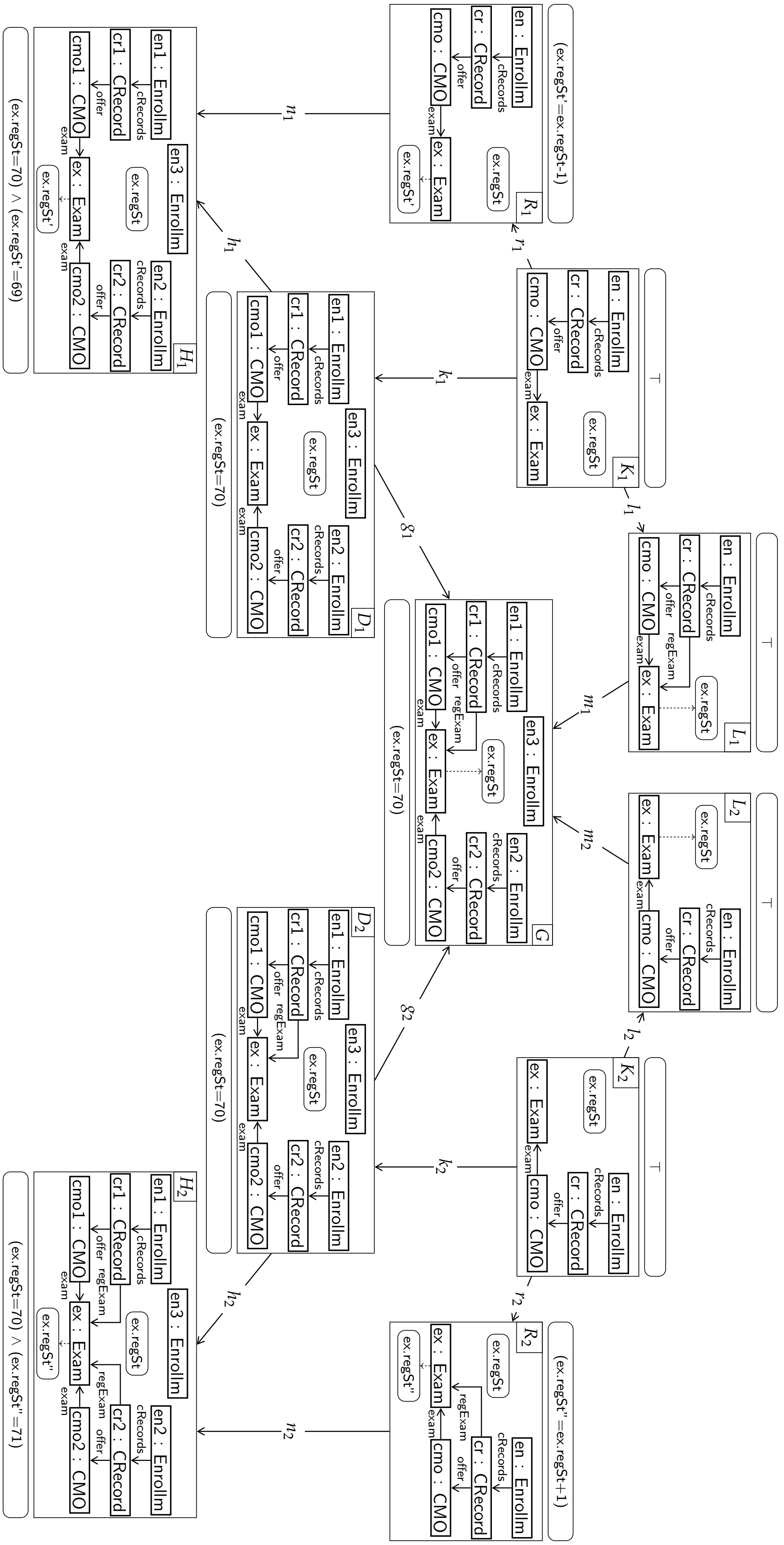


Figure 8.2: Parallel dependent pair of direct transformations via productions unregExam (left) and regExam (right)

we had to simplify the productions. More specifically, we removed all variables and corresponding expressions, except those related to the ex.regSt attribute. Moreover, we abbreviated Enrollment by Enroll, CourseRecord by CRecord, and CoModOffer by CMO.

Production unregExam is given by $\langle L_1, \Phi_{L1}\rangle \leftarrow \langle K_1 \Phi_{K1}\rangle \rightarrow \langle R_1, \Phi_{R1}\rangle$. The production takes an enrollment (en : Enroll) and an examination (ex : Exam) as input. To unregister from an exam the link regExam is deleted and the number of registrations (ex.regSt) is decremented by one.

Production regExam is given by the span $\langle L_2, \Phi_{L2}\rangle \leftarrow \langle K_2 \Phi_{K2}\rangle \rightarrow \langle R_2, \Phi_{R2}\rangle$. The production takes also an enrollment (en : Enroll) and an examination (ex : Exam) as input. The registration is performed by creating the link regExam and incrementing the number of registrations (ex.regSt) by one.

Both productions are typed projective productions as

$$\exists(\mathsf{ex.regSt}').(\mathsf{ex.regSt}' = \mathsf{ex.regSt} - 1) \Leftrightarrow \top$$

and

$$\exists(\mathsf{ex.regSt}'').(\mathsf{ex.regSt}'' = \mathsf{ex.regSt} + 1) \Leftrightarrow \top,$$

assuming the Σ-structure of natural numbers with addition and subtraction, defined as usual. In Figure 8.2 both productions are applied to symbolic graph $\langle G, \Phi_G\rangle$. More specifically, production unregExam is applied to enrollment en1 and examination ex; production regExam is applied to enrollment en2 an examination ex. Symbolic graph $\langle G, \Phi_G\rangle$ contains an additional enrollment en3 that exemplarily represent all elements of $\langle G, \Phi_G\rangle$ that are not required for applying the productions (note that $\langle G, \Phi_G\rangle$ usually comprises hundreds of exams, and thousands of enrollments).

As both direct transformations changes the value of ex.regSt, they are in conflict. More specifically, both direct transformations are parallel dependent, as there does not exists morphisms $i : \langle L_1, \Phi_{L1}\rangle \rightarrow \langle D_2, \Phi_{D2}\rangle$ and $j : \langle L_2, \Phi_{L2}\rangle \rightarrow \langle D_1, \Phi_{D1}\rangle$, because both direct transformations delete the label edge between ex : Exam and ex.regSt.

Remark 8.4 (Checking parallel independence)**.**
Note that it is sufficient (and necessary) to ensure only the presence of a E-graph $\mathcal{M}^{inj}$-morphisms $i : L_1 \rightarrow D_2$ and $j : L_2 \rightarrow D_1$, in order to show parallel independence. Recall that morphisms $g_1 : \langle D_1, \Phi_{D1}\rangle \rightarrow \langle G, \Phi_G\rangle$ and $g_2 : \langle D_2, \Phi_{D2}\rangle \rightarrow \langle G, \Phi_G\rangle$ in the definition of parallel independence (see Definition 8.1) are in $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$ (closure of $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$ under pushouts); hence, $\Phi_{D1}[\hat{g}_1] \Leftrightarrow \Phi_G \Leftrightarrow \Phi_{D2}[\hat{g}_2]$ is valid. Consequently, as $\Phi_G \Rightarrow \Phi_{L1}[\hat{m}_1]$ and $\Phi_G \Rightarrow \Phi_{L2}[\hat{m}_2]$, also $\Phi_{D2} \Rightarrow \Phi_{L1}[\hat{i}]$ and $\Phi_{D1} \Rightarrow \Phi_{L2}[\hat{j}]$ for all typed E-graph $\mathcal{M}^{inj}_{TG}$-morphisms $i : L_1 \rightarrow D_2$ and $j : L_2 \rightarrow D_1$, such that $g_2 \circ i = m_1$ and $g_1 \circ j = m_2$. This property is especially relevant regarding an implementation. In Chapter 9 we use this property to guarantee the soundness of our implementation.

In order to show that the conflict shown in the previous example can be resolved, we need to give a formal definition for local confluence and subcommutativity for typed projective graph transformation systems.

Definition 8.5 (Local confluence and subcommutativity for **TPGTS**).
Let **TPGTS** be a typed projective graph transformation system, a pair of direct transformations

$$\langle H_1, \Phi_{H1} \rangle \xLeftarrow{p_1@m_1} \langle G, \Phi_G \rangle \xRightarrow{p_2@m_2} \langle H_2, \Phi_{H2} \rangle,\ p_1, p_2 \in \mathbf{P}$$

is *locally confluent* if there are transformations

$$\langle H_1, \Phi_{H1} \rangle \overset{*}{\Longrightarrow} \langle H_3, \Phi_{H3} \rangle \overset{*}{\Longleftarrow} \langle H_2, \Phi_{H2} \rangle$$

via productions in **P**.

A pair of direct transformations

$$\langle H_1, \Phi_{H1} \rangle \xLeftarrow{p_1@m_1} \langle G, \Phi_G \rangle \xRightarrow{p_2@m_2} \langle H_2, \Phi_{H2} \rangle,\ p_1, p_2 \in \mathbf{P}$$

is *subcommutative* if there are transformations

$$\langle H_1, \Phi_{H1} \rangle \overset{0..1}{\Longrightarrow} \langle H_3, \Phi_{H3} \rangle \overset{0..1}{\Longleftarrow} \langle H_2, \Phi_{H2} \rangle$$

via productions in **P**.

A typed projective graph transformation system **TPGTS** *is locally confluent (subcommutative)* if each pair of direct transformations via productions in **P** is locally confluent (subcommutative).

One might expect that the two parallel direct transformations of the previous example are subcommutative. Unfortunately, this is not the case as illustrated by the next example.

Example 8.6 (Subcommutativity).
The upper part of Figure 8.3 shows the application of the productions unregExam (i. e., production p_1) and regExam (i. e., production p_2) to symbolic graph $\langle G, \Phi_G \rangle$ via matches m_1 and m_2, as shown in Example 8.3. The result are the symbolic graphs $\langle H_1, \Phi_{H1} \rangle$ and $\langle H_2, \Phi_{H2} \rangle$, respectively. The lower part of Figure 8.3 shows the application of productions regExam (i. e., production p_2) to symbolic graphs $\langle H_1, \Phi_{H1} \rangle$ and unregExam (i. e., production p_1) to symbolic graph and $\langle H_2, \Phi_{H2} \rangle$. The matches m_2' and m_1' are indicated by the bold elements in $\langle H_1, \Phi_{H1} \rangle$ and $\langle H_2, \Phi_{H2} \rangle$, respectively. The results are the graphs $\langle H_3, \Phi_{H3} \rangle$ and $\langle H_4, \Phi_{H4} \rangle$. Although $\langle H_3, \Phi_{H3} \rangle$ and $\langle H_4, \Phi_{H4} \rangle$ look very similar (i. e., for graph nodes, graph edges and label edges and the value of the ex.regSt* attribute), they are not isomorphic, because of the label nodes ex.regSt′ and ex_regSt″ represent different values (i. e. ex.regSt′=69 and ex.regSt″=71). Hence, the diagram shown in Figure 8.3 is not subcommutative.

The problem illustrated by Example 8.6 is that from a practical point of view, we are often interested only in the values of the variables that are assigned to a graph node when comparing two graphs. However, if we check whether the results are isomorphic, we also take the values of the auxiliary variables (i. e., the variables not assigned to a graph node; see Definition 4.10) into account. These auxiliary

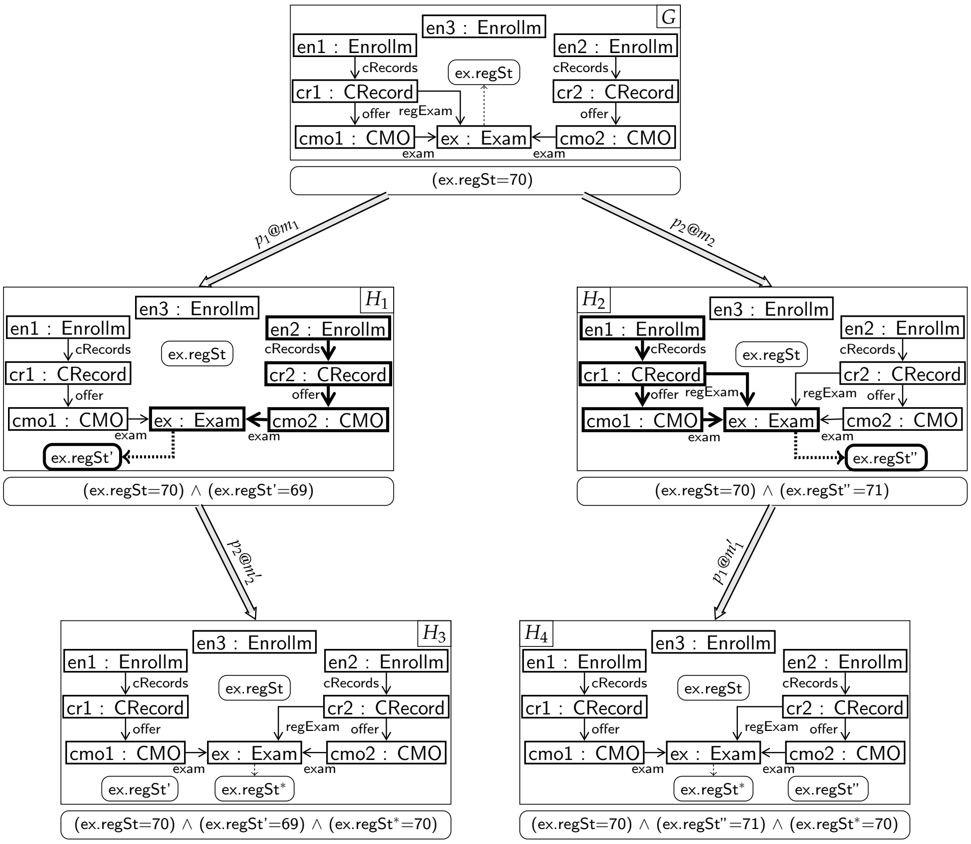


Figure 8.3: Example for nonsubcommutative direct transformations

variables may differ, although the values of the nonauxiliary variables are the same as shown in the previous example.

8.1.2 *Local Confluence Modulo Normal Form Equivalence*

To overcome this problem, we propose normal form equivalence to compare two symbolic graphs. Basically, two symbolic graphs are normal form equivalent if their normal forms (see Definition 4.10) are isomorphic.

Definition 8.7 (Normal form equivalence of symbolic graphs (≡))**.**
Two symbolic graphs $\langle G_1, \Phi_{G1}\rangle, \langle G_2, \Phi_{G2}\rangle$ in $\mathbb{TSG}_{\mathcal{D},TG}$ are *normal form equivalent* (denoted as $\langle G_1, \Phi_{G1}\rangle \equiv \langle G_2, \Phi_{G2}\rangle$) if their normal forms are isomorphic, i. e., $nor(\langle G_1, \Phi_{G1}\rangle) \simeq nor(\langle G_2, \Phi_{G2}\rangle)$.

Note that ≡ is an equivalence relation on symbolic graphs.

Based on the notion of normal form equivalence, we can reformulate Definition 8.5 leading to the notions of local confluence and subcommutativity modulo normal form equivalence. Basically a pair of direct transformation *is locally confluent (or subcommutative) modulo normal form equivalence*, if there are transformations such that the resulting graphs are normal form equivalent.

Definition 8.8 (Local confluence and subcommutativity modulo ≡)**.**
Let **TPGTS** be a typed projective graph transformation system. A pair of direct transformations

$$\langle H_1, \Phi_{H1}\rangle \xLeftarrow{p_1@m_1} \langle G, \Phi_G\rangle \xRightarrow{p_2@m_2} \langle H_2, \Phi_{H2}\rangle,\ p_1, p_2 \in \mathbf{P}$$

is *local confluent modulo* ≡ if there are transformations

$$\langle H_1, \Phi_{H1}\rangle \overset{*}{\Longrightarrow} \langle H_3, \Phi_{H3}\rangle \text{ and } \langle H_2, \Phi_{H2}\rangle \overset{*}{\Longrightarrow} \langle H_4, \Phi_{H4}\rangle$$

via productions in **P**, such that $\langle H_3, \Phi_{H3}\rangle \equiv \langle H_4, \Phi_{H4}\rangle$.

A pair of direct transformations

$$\langle H_1, \Phi_{H1}\rangle \xLeftarrow{p_1@m_1} \langle G, \Phi_G\rangle \xRightarrow{p_2@m_2} \langle H_2, \Phi_{H2}\rangle,\ p_1, p_2 \in \mathbf{P}$$

is *subcommutative modulo* ≡ if there are transformations

$$\langle H_1, \Phi_{H1}\rangle \xRightarrow{0..1} \langle H_3, \Phi_{H3}\rangle \equiv \langle H_4, \Phi_{H4}\rangle \xLeftarrow{0..1} \langle H_2, \Phi_{H2}\rangle$$

via productions in **P**.

In the following example, we show that subcommutativity modulo normal form equivalence leads to the expected result.

Example 8.9 (Subcommutativity modulo normal form equivalence)**.**
Figure 8.4 shows the direct transformations as presented in Example 8.6, but instead of requiring $\langle H3, \Phi_{H3}\rangle$ and $\langle H4, \Phi_{H4}\rangle$ to be isomorphic (as in Exam-

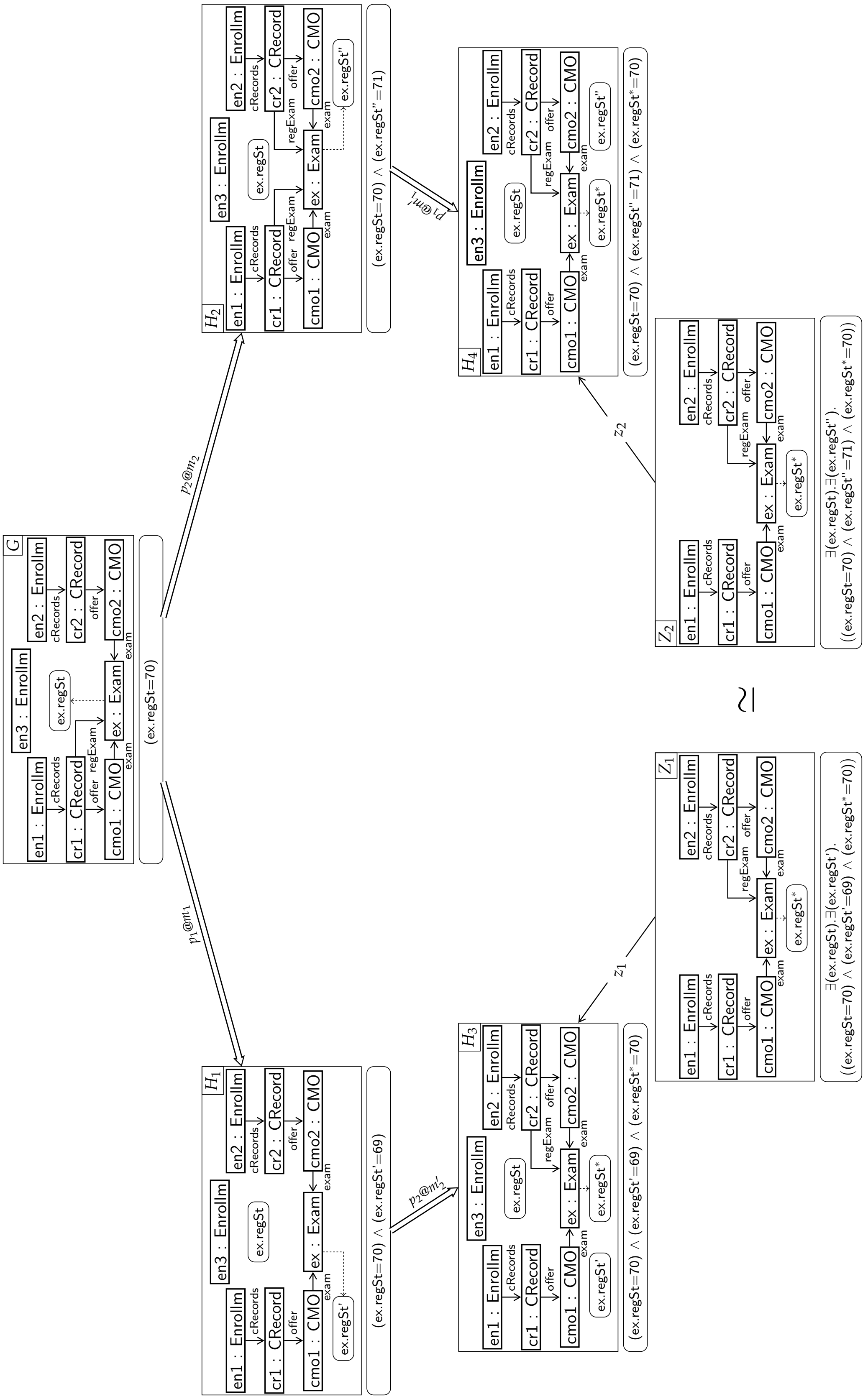

G
en1 : Enrollm
en2 : Enrollm
en3 : Enrollm
cRecords
cr1 : CRecord
cr2 : CRecord
offer
regExam
cmo1 : CMO
cmo2 : CMO
ex : Exam
exam
ex.regSt
(ex.regSt=70)
$p_1@m_1$
$p_2@m_2$
H_1
ex.regSt'
(ex.regSt=70) ∧ (ex.regSt'=69)
H_2
ex.regSt''
(ex.regSt=70) ∧ (ex.regSt''=71)
$p_2@m_2'$
$p_1@m_1'$
H_3
ex.regSt*
(ex.regSt=70) ∧ (ex.regSt'=69) ∧ (ex.regSt*=70)
H_4
(ex.regSt=70) ∧ (ex.regSt''=71) ∧ (ex.regSt*=70)
z_1
z_2
Z_1
∃(ex.regSt).∃(ex.regSt').
((ex.regSt=70) ∧ (ex.regSt'=69) ∧ (ex.regSt*=70))
Z_2
∃(ex.regSt).∃(ex.regSt'').
((ex.regSt=70) ∧ (ex.regSt''=71) ∧ (ex.regSt*=70))
≅

Figure 8.4: Subcommutativity modulo normal form equivalence for the direct transformation via the productions unregExam and regExam

ple 8.6), we require that $\langle H3, \Phi_{H3}\rangle$ and $\langle H4, \Phi_{H4}\rangle$ are normal form equivalent. To this end we, derive the normal forms $\langle Z_1, \Phi_{Z1}\rangle = nor(\langle H3, \Phi_{H3}\rangle)$ and $\langle Z_2, \Phi_{Z2}\rangle = nor(\langle H4, \Phi_{H4}\rangle)$ with induced morphisms z_1 and z_2 as shown in Figure 8.4. The normal forms $\langle Z_1, \Phi_{Z1}\rangle$ and $\langle Z_2, \Phi_{Z2}\rangle$ are isomorphic; hence, $\langle H3, \Phi_{H3}\rangle$ and $\langle H4, \Phi_{H4}\rangle$ are normal form equivalent. Consequently, the pair of transformations $\langle H_1, \Phi_{H1}\rangle \overset{p_1@m_1}{\Longleftarrow} \langle G, \Phi_G\rangle \overset{p_2@m_2}{\Longrightarrow} \langle H_2, \Phi_{H2}\rangle$ is subcommutative modulo $\equiv$.

8.2 Conflict Detection by Critical Pair Analysis

Up until now we discussed conflicts and confluence on the transformation level. In this section we generalize this analysis to productions using critical pair analysis. The basic idea of critical pairs is very similar to the construction of application conditions from constraints. Instead of verifying local confluence (subcommutativity) for all possible transformation, it is sufficient to ensure local confluence (subcommutativity) for only some *minimal contexts* that are built by all jointly surjective gluings of the left-hand sides of the involved productions. If the pair of direct transformation that results from applying the pair to a minimal context is parallel dependent it is a *critical pair*.

Before we define critical pairs we first introduce other results that allow for extending transformations to larger contexts. The definitions and theorems in this section are instantiations of the results in Chapter 5 to projective graph transformation systems.

8.2.1 *Embedding and Extension*

We begin with the definition of extension diagrams for typed projective transformations. Basically, an *extension diagram* describes how to extend a transformation $t : \langle G_0, \Phi_{G_0}\rangle \overset{*}{\Longrightarrow} \langle G_n, \Phi_{G_n}\rangle$ to a transformation $t' : \langle G'_0, \Phi'_{G_0}\rangle \overset{*}{\Longrightarrow} \langle G'_n, \Phi'_{G_n}\rangle$ via an extension morphisms $k_0 : \langle G_0, \Phi_{G_0}\rangle \to \langle G'_0, \Phi'_{G_0}\rangle$, where t' and t are transformations via the same sequence of productions

Definition 8.10 (Extension diagram for **TPGTS**).
Let **TPGTS** be a typed projective graph transformation system, then diagram (1) is an *extension diagram* over transformation $t : \langle G_0, \Phi_{G_0}\rangle \overset{*}{\Longrightarrow} \langle G_n, \Phi_{G_n}\rangle$ and *extension morphism* $k_0 : \langle G_0, \Phi_{G_0}\rangle \to \langle G'_0, \Phi'_{G_0}\rangle$, $k_0 \in \mathcal{M}^{inj}_{\Rightarrow,TG}$, where t and t' are transformations via the same sequence of productions $p_0, \ldots, p_n \in \mathbf{P}$ with matches $(m_0, \ldots, m_{n-1})$ and $(k_0 \circ m_0, \ldots, k_{n-1} \circ m_{n-1})$, respectively, given by the double pushout diagrams on the right.

$$\begin{tikzcd}
\langle G_0,\Phi_{G_0}\rangle \arrow[r, Rightarrow, "t"{description}, "*"{pos=1}] \arrow[d, "k_0"'] \arrow[dr, phantom, "(1)"] & \langle G_n,\Phi_{G_n}\rangle \arrow[d, "k_n"] \\
\langle G'_0,\Phi'_{G_0}\rangle \arrow[r, Rightarrow, "t'"{description}, "*"{pos=1}] & \langle G'_n,\Phi'_{G_n}\rangle
\end{tikzcd}$$

$$\begin{tikzcd}
\langle L_i,\Phi_{L_i}\rangle \arrow[d, "m_i"] \arrow[dr, phantom, "(PO)"] & \langle K_i,\Phi_{K_i}\rangle \arrow[l, "l_i"'] \arrow[r, "r_i"] \arrow[d, "s_i"] \arrow[dr, phantom, "(PO)"] & \langle R_i,\Phi_{R_i}\rangle \arrow[d, "n_i"] \\
\langle G_i,\Phi_{G_i}\rangle \arrow[d, "k_i"] \arrow[dr, phantom, "(PO)"] & \langle D_i,\Phi_{D_i}\rangle \arrow[l, "g_i"'] \arrow[r, "h_i"] \arrow[d, "d_i"] \arrow[dr, phantom, "(PO)"] & \langle G_{i+1},\Phi_{G_{i+1}}\rangle \arrow[d, "k_{i+1}"] \\
\langle G'_i,\Phi'_{G_i}\rangle & \langle D'_i,\Phi'_{D_i}\rangle \arrow[l, "g'_i"'] \arrow[r, "h'_i"] & \langle G'_{i+1},\Phi'_{G_{i+1}}\rangle
\end{tikzcd}$$

The following definition of a derived span describes how to combine the changes of a transformation $t : \langle G_0,\Phi_{G_0}\rangle \overset{*}{\Longrightarrow} \langle G_n,\Phi_{G_n}\rangle$ (i. e., a sequence of direct transformations) into a a single transformation step $t : \langle G_0,\Phi_{G_0}\rangle \Longrightarrow \langle G_n,\Phi_{G_n}\rangle$. In this way any transformation can be treated as a direct transformation.

Definition 8.11 (Derived span for typed projective transformations)**.**
Let **TPGTS** be a typed projective graph transformation system, the derived span of a direct projective transformation $\langle G,\Phi_G\rangle \overset{p@m}{\Longrightarrow} \langle H,\Phi_H\rangle$ with $p \in \mathbf{P}$, is given by

$$der(\langle G,\Phi_G\rangle \overset{p@m}{\Longrightarrow} \langle H,\Phi_H\rangle) = (\langle G,\Phi_G\rangle \leftarrow \langle D,\Phi_D\rangle \rightarrow \langle H,\Phi_H\rangle).$$

Given a transformation sequence

$$t : \langle G_0,\Phi_{G_0}\rangle \overset{*}{\Longrightarrow} \langle G_{n-1},\Phi_{G_{n-1}}\rangle \Longrightarrow \langle G_n,\Phi_{G_n}\rangle,$$

via productions in **P**, and with derived spans

$$s_1 = der(\langle G_0,\Phi_{G_0}\rangle \overset{*}{\Longrightarrow} \langle G_{n-1},\Phi_{G_{n-1}}\rangle) = (\langle G_0,\Phi_{G_0}\rangle \leftarrow \langle D',\Phi'_D\rangle \rightarrow \langle G_{n-1},\Phi_{G_{n-1}}\rangle)$$
$$s_2 = der(\langle G_{n-1},\Phi_{G_{n-1}}\rangle \Longrightarrow \langle G_n,\Phi_{G_n}\rangle) = (\langle G_{n-1},\Phi_{G_{n-1}}\rangle \leftarrow \langle D'',\Phi''_D\rangle \rightarrow \langle G_n,\Phi_{G_n}\rangle)$$

as shown in the following diagram:

$$\begin{tikzcd}[column sep=small]
 & & & \langle D,\Phi_D\rangle \arrow[llld, "d_0"'] \arrow[ld, "v"'] \arrow[rd, "w"] \arrow[rrrd, "d_n"] & & & \\
\langle G_0,\Phi_{G_0}\rangle & & \langle D',\Phi'_D\rangle \arrow[ll, "g_0"'] \arrow[r, "g_{n-1}"] & \langle G_{n-1},\Phi_{G_{n-1}}\rangle \arrow[u, phantom, "(1)"] & \langle D'',\Phi''_D\rangle \arrow[l, "f_{n-1}"'] \arrow[rr, "f_n"] & & \langle G_n,\Phi_{G_n}\rangle
\end{tikzcd}$$

The derived span

$$der(t) = \langle G_0,\Phi_{G_0}\rangle \overset{d_0}{\leftarrow} \langle D,\Phi_D\rangle \overset{d_n}{\rightarrow} \langle G_n,\Phi_{G_n}\rangle,$$

of transformation sequence t is given by the composition of derived spans s_1 and s_2 via pullback (1), where $d_0 = g_0 \circ v$ and $d_n = f_n \circ w$.

Note that a derived span is unique up to isomorphism and does not depend on the order of the pullback constructions. For more details see Remark 5.28.

Based on the notion of initial pushouts we can now define consistency, and show that consistency is sufficient (Theorem 8.13) and necessary (Theorem 8.14) to guarantee the existence of an extension diagram.

Definition 8.12 (Consistency for **TPGTS**).
Let **TPGTS** be a typed projective graph transformation system. Given a transformation $t : \langle G_0, \Phi_{G_0}\rangle \overset{*}{\Longrightarrow} \langle G_n, \Phi_{G_n}\rangle$ via productions in **P**, where $der(t) = (\langle G_0, \Phi_{G_0}\rangle \leftarrow \langle D, \Phi_D\rangle \rightarrow \langle G_n, \Phi_{G_n}\rangle)$.

$$\begin{tikzcd}
\langle B, \Phi_B\rangle \arrow[r, "b_0"] \arrow[d] \arrow[rr, bend left, "b"] \arrow[dr, phantom, "(1)"] & \langle G_0, \Phi_{G_0}\rangle \arrow[d, "k_0"] & \langle D, \Phi_D\rangle \arrow[l, "d_0"'] \arrow[r, "d_n"] & \langle G_n, \Phi_{G_n}\rangle \\
\langle C, \Phi_C\rangle \arrow[r] & \langle G'_0, \Phi'_{G_0}\rangle & &
\end{tikzcd}$$

A morphism $k_0 : \langle G_0, \Phi_{G_0}\rangle \rightarrow \langle G'_0, \Phi'_{G_0}\rangle$, $k_0 \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ is *consistent* with respect to transformation t if there exists an quasi $(\mathcal{L}, \mathcal{N})$-initial pushout (1) over k_0 and a morphism $b \in \mathcal{M}^{inj}_{Proj,TG}$ with $d_0 \circ b = b_0$.

Theorem 8.13 (Embedding Theorem for **TPGTS**).
Let **TPGTS** be a typed projective graph transformation system. Given transformation $t : \langle G_0, \Phi_{G_0}\rangle \overset{*}{\Longrightarrow} \langle G_n, \Phi_{G_n}\rangle$ via productions in **P** and an $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphism $k_0 : \langle G_0, \Phi_{G_0}\rangle \rightarrow \langle G'_0, \Phi'_{G_0}\rangle$ such that k_0 is consistent with respect to transformation t, then there is an extension diagram over t and k_0.

Proof. This is a direct consequence of Theorem 5.30 and the fact that category $\mathbb{TSG}_{\mathcal{D},TG}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive and has $(\mathcal{L}, \mathcal{N})$-initial pushouts (for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$, $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$). □

Theorem 8.14 (Extension Theorem for **TPGTS).**
Let **TPGTS** be a typed projective graph transformation system, given a transformation $t : \langle G_0, \Phi_{G_0}\rangle \overset{*}{\Longrightarrow} \langle G_n, \Phi_{G_n}\rangle$ via productions in **P**, with derived span

$$der(t) = (\langle G_0, \Phi G_0\rangle \xleftarrow{d_0} \langle D, \Phi_D\rangle \xrightarrow{d_n} \langle G_n, \Phi_{G_n}\rangle)$$

and extension diagram (1)

$$\begin{tikzcd}
\langle B, \Phi_B\rangle \arrow[r, "b_0"] \arrow[d] \arrow[dr, phantom, "(2)"] & \langle G_0, \Phi_{G_0}\rangle \arrow[r, Rightarrow, "t", "*"'] \arrow[d, "k_0"] \arrow[dr, phantom, "(1)"] & \langle G_n, \Phi_{G_n}\rangle \arrow[d, "k_n"] \\
\langle C, \Phi_C\rangle \arrow[r] & \langle G_0', \Phi_{G_0}'\rangle \arrow[r, Rightarrow, "t'", "*"'] & \langle G_n', \Phi_{G_n}'\rangle
\end{tikzcd}$$

with $(\mathcal{L}, \mathcal{N})$-initial pushout (2), then we have the following:

a) k_0 is consistent with respect to transformation t.

b) There is a direct transformation $\langle G_0', \Phi_{G_0}'\rangle \overset{der(t)@k_0}{\Longrightarrow} \langle G_n', \Phi_{G_n}'\rangle$ given by the following double pushout diagram.

$$\begin{tikzcd}
\langle G_0, \Phi_{G_0}\rangle \arrow[d, "k_0"] \arrow[dr, phantom, "(3)"] & \langle D, \Phi_D\rangle \arrow[l, "d_0"'] \arrow[r, "d_n"] \arrow[d, "h"] \arrow[dr, phantom, "(4)"] & \langle G_n, \Phi_{G_n}\rangle \arrow[d, "k_n"] \\
\langle G_0', \Phi_{G_0}'\rangle & \langle D', \Phi_D'\rangle \arrow[l] \arrow[r] & \langle G_n', \Phi_{G_n}'\rangle
\end{tikzcd}$$

c) There are quasi $(\mathcal{L}, \mathcal{N})$-initial pushouts (5) and (6).

$$\begin{tikzcd}
\langle B, \Phi_B\rangle \arrow[r, "b"] \arrow[d] \arrow[dr, phantom, "(5)"] & \langle D, \Phi_D\rangle \arrow[d, "h"] \\
\langle C, \Phi_C\rangle \arrow[r] & \langle D', \Phi_D'\rangle
\end{tikzcd}
\qquad
\begin{tikzcd}
\langle B, \Phi_B\rangle \arrow[r, "d_n \circ b"] \arrow[d] \arrow[dr, phantom, "(6)"] & \langle G_n, \Phi_{G_n}\rangle \arrow[d, "k_n"] \\
\langle C, \Phi_C\rangle \arrow[r] & \langle G_n', \Phi_{G_n}'\rangle
\end{tikzcd}$$

Proof. This is a direct consequence of Theorem 5.31 and the fact that category $\mathbb{TSG}_{\mathcal{D},TG}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive and has $(\mathcal{L}, \mathcal{N})$-initial pushouts (for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$, $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$). □

8.2.2 *Critical Pairs and Completeness*

Now we define symbolic critical pairs and show that symbolic critical pairs are complete; that is, for any pair of parallel dependent transformations, there exists a critical pair that can be extended to the pair of parallel dependent transformations.

Definition 8.15 (Symbolic critical pairs for **TPGTS**).
Let **TPGTS** be a typed projective graph transformation system, then a *critical pair* of **TPGTS** is defined as a pair of parallel depended direct transformations

$$\langle P_1, \Phi_{P1}\rangle \xLeftarrow{p_1@o_1} \langle K, \Phi_K\rangle \xRightarrow{p_2@o_2} \langle P_2, \Phi_{P2}\rangle$$

with $p_1, p_2 \in \mathbf{P}$ such that the morphism pair (o_1, o_2) is in $\mathcal{E}'^{surj}_{\Leftrightarrow,TG}$.

Lemma 8.16 (Completeness of symbolic critical pairs for **TPGTS**).
Let **TPGTS** be a typed projective graph transformation system, then the symbolic critical pairs of **TPGTS** are complete. This means that for each pair of parallel dependent direct transformations

$$\langle H_1, \Phi_{H1}\rangle \xLeftarrow{p_1@m_1} \langle G, \Phi_G\rangle \xRightarrow{p_2@m_2} \langle H_2, \Phi_{H2}\rangle,$$

with $p_1, p_2 \in \mathbf{P}$, there exists a symbolic critical pair

$$\langle P_1, \Phi_{P1}\rangle \xLeftarrow{p_1@o_1} \langle K, \Phi_K\rangle \xRightarrow{p_2@o_2} \langle P_2, \Phi_{P2}\rangle$$

with the following extension diagrams (1) and (2) over extension morphism m.

$$\begin{tikzcd}
\langle P_1, \Phi_{P1}\rangle \arrow[d] \arrow[dr, phantom, "(1)"] & \langle K, \Phi_K\rangle \arrow[l, Rightarrow] \arrow[r, Rightarrow] \arrow[d, "m"] \arrow[dr, phantom, "(2)"] & \langle P_2, \Phi_{P2}\rangle \arrow[d] \\
\langle H_1, \Phi_{H1}\rangle & \langle G, \Phi_G\rangle \arrow[l, Rightarrow] \arrow[r, Rightarrow] & \langle H_2, \Phi_{H2}\rangle
\end{tikzcd}$$

Proof. This is a direct consequence of Lemma 5.33 and the fact that category $\mathbb{TSG}_{\mathcal{D},TG}$ is $(\mathcal{L}, \mathcal{R}, \mathcal{N})$-adhesive, has binary coproducts, $\mathcal{E}$–$\mathcal{N}$ factorization and the $\mathcal{L}$–$\mathcal{N}$-pushout–pullback decomposition property for $\mathcal{L} = \mathcal{M}^{bij}_{\Leftrightarrow,TG}$, $\mathcal{R} = \mathcal{M}^{inj}_{Proj,TG}$, $\mathcal{N} = \mathcal{M}^{inj}_{\Rightarrow,TG}$, and $\mathcal{M}^{bij}_{TG}$. □

Example 8.17 (Extension diagram for symbolic critical pairs).
Figure 8.6 shows the extension diagram for a critical pair derived for productions unregExam and regExam and the parallel dependent direct transformations presented in Example 8.3. The minimal context $\langle K, \Phi_K\rangle$ is obtained by gluing $\langle L_1, \Phi_{L1}\rangle$ and $\langle L_2, \Phi_{L2}\rangle$ together, whereas the glued elements are drawn bold. The formula component Φ_K is the conjunction of Φ_{L1} and Φ_{L2} (see Definition 7.1); hence, the morphisms pair (o_1, o_2) is jointly epimorphic (see Lemma 7.3).

The critical pair $\langle P_1, \Phi_{P1}\rangle \xLeftarrow{p_1@o_1} \langle K, \Phi_K\rangle \xRightarrow{p_2@o_2} \langle P_2, \Phi_{P2}\rangle$ is obtained by applying the productions unregExam (i. e., $p_1 = \langle \Phi_{L1}, L_1 \leftarrow K_1 \rightarrow R_1, \Phi_{R1}\rangle$) and regExam (i. e., $p_2 = \langle \Phi_{L2}, L_2 \leftarrow K_2 \rightarrow R_2, \Phi_{R2}\rangle$) to the minimal context $\langle K, \Phi_K\rangle$.

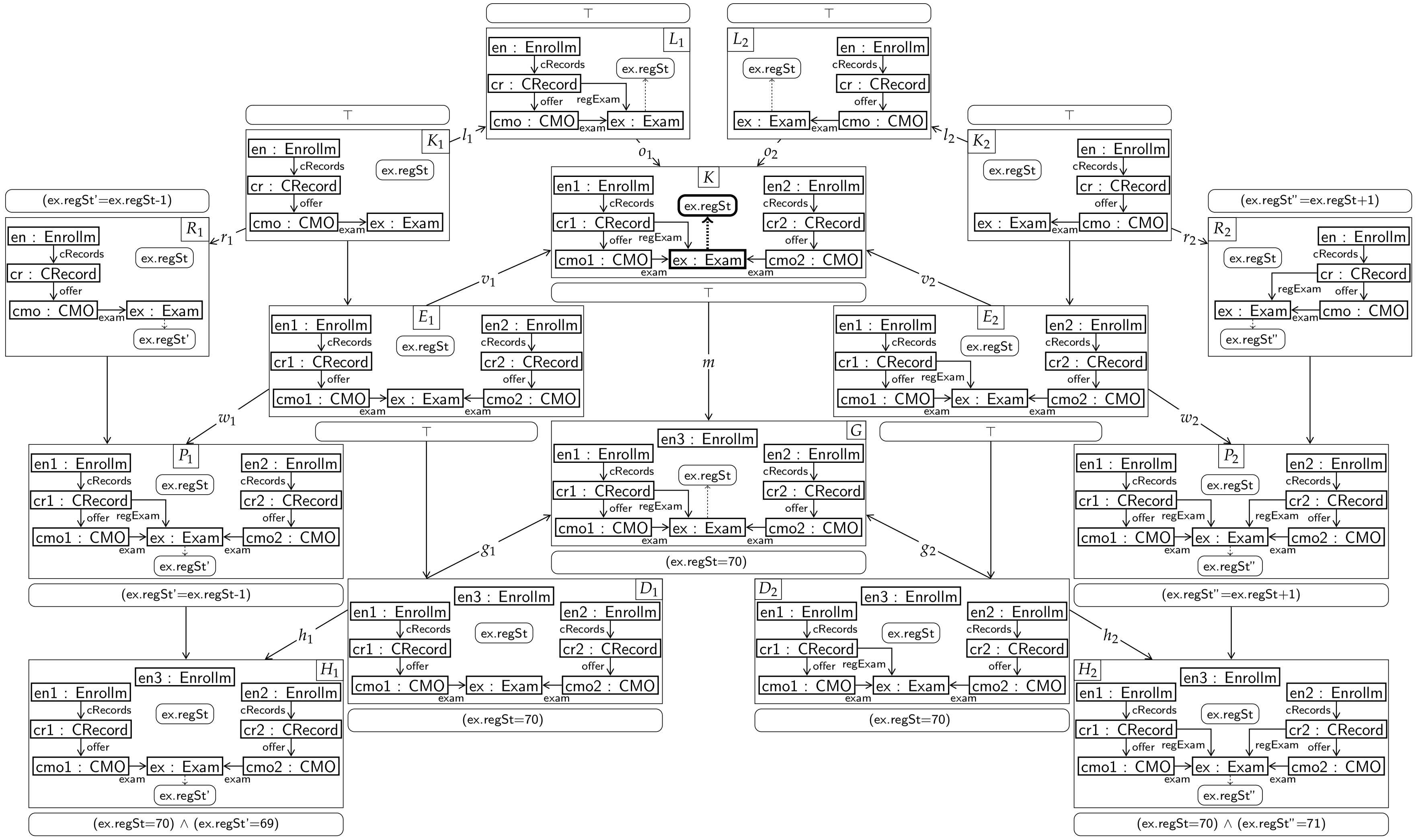


Figure 8.5: Example for an extension diagram for a symbolic critical pair (shown in the middle) of productions registerStudentForExam and unRegisterStudentForExam (shown on top) and parallel dependent direct transformations presented in Example 8.3 (shown on the bottom).

> On the bottom of Figure 8.6 the parallel dependent pair of direct transformations $\langle H_1, \Phi_{H1}\rangle \xLeftarrow{p_1@m_1} \langle G, \Phi_G\rangle \xRightarrow{p_2@m_2} \langle H_2, \Phi_{H2}\rangle$ is depicted, which was originally introduced in Example 8.3. The the matches m_1 and m_2 are given by $m_1 = m \circ o_1$ and $m_1 = m \circ o_2$. Hence, the diagram shown in Figure 8.6 is an example for an extension diagram in the sense of Lemma 8.16 as all rectangles are pushouts.

8.3 Conflict Resolution by Critical Pair Analysis

In this section we present the main result of this chapter, namely the *Local Confluence Theorem Modulo Normal Form Equivalence for typed projective transformations systems*.

On may expect that local confluence (subcommutativity) of all critical pairs ensures local confluence of the graph transformation system. Unfortunately, this is not the case as shown in [Plu93]. Nevertheless, a slightly more restricted version of local confluence, called strict confluence, is sufficient to ensure local confluence [Plu93]. Intuitively, a critical pair $\langle P_1, \Phi_{P1}\rangle \xLeftarrow{p_1@o_1} \langle K, \Phi_K\rangle \xRightarrow{p_2@o_2} \langle P_2, \Phi_{P2}\rangle$ is *strict local confluent* if it is local confluent, i. e., there are transformations $t_3 : \langle P_1, \Phi_{P1}\rangle \overset{*}{\Longrightarrow} \langle P_3, \Phi_{P3}\rangle$, $t_4 : \langle P_2, \Phi_{P2}\rangle \overset{*}{\Longrightarrow} \langle P_3, \Phi_{P3}\rangle$, and all elements of $\langle K, \Phi_K\rangle$ that are not deleted are mapped to the same elements in $\langle P_3, \Phi_{P3}\rangle$. In this way it is ensured that transformations $\langle P_1, \Phi_{P1}\rangle \xRightarrow{p_2@o'_1} \langle P_3, \Phi_{P_3}\rangle \xLeftarrow{p_1@o'_1} \langle P_2, \Phi_{P2}\rangle$ can be extended to transformations $\langle H_1, \Phi_{H1}\rangle \xRightarrow{p_2@m'_1} \langle H_3, \Phi_{H_3}\rangle \xLeftarrow{p_1@m'_1} \langle P_2, \Phi_{P2}\rangle$.

In order to define *strict local confluence modulo normal form equivalence* we first have to define the class of *normal form preserving morphisms* $\mathcal{I}^{inj}_{Proj,TG}$.

Definition 8.18 (The class $\mathcal{I}^{inj}_{Proj,TG}$-morphisms)**.**
The class $\mathcal{I}^{inj}_{Proj,TG}$ is given as the subclass of $\mathcal{M}^{inj}_{Proj,TG}$-morphisms that are bijective for graph nodes and all kind of edges, and injective for label nodes.

The class $\mathcal{I}^{inj}_{Proj,TG}$ is called the class of *normal form preserving morphisms* as for any morphisms $f : \langle A, \Phi_A\rangle \to \langle B, \Phi_B\rangle$ in $\mathcal{I}^{inj}_{Proj,TG}$ we have that $\langle A, \Phi_A\rangle$ and $\langle B, \Phi_B\rangle$ have isomorphic normal forms.

In the following, we only define *strict local confluence modulo normal form equivalence*. We do not give an explicit definition of *strict subcommutativity modulo normal form equivalence* as the corresponding definition (and later the corresponding proof) can be obtained as a special case of strict local confluence modulo normal form equivalence.

Definition 8.19 (Strict local confluence modulo ≡)**.**
Let **TPGTS** be a typed projective graph transformation system, a critical pair

$$t_1 : \langle K, \Phi_K\rangle \xRightarrow{p_1@o_1} \langle P_1, \Phi_{P1}\rangle,\ t_2 : \langle K, \Phi_K\rangle \xRightarrow{p_2@o_2} \langle P_2, \Phi_{P2}\rangle, p_1, p_2 \in \mathbf{P}$$

of **TPGTS** with derived spans $der(t_1) = (\langle K, \Phi_K\rangle \xleftarrow{v_1} \langle E_1, \Phi_{E1}\rangle \xrightarrow{w_1} \langle P_1, \Phi_{P1}\rangle)$ and $der(t_2) = (\langle K, \Phi_K\rangle \xleftarrow{v_2} \langle E_2, \Phi_{E2}\rangle \xrightarrow{w_2} \langle P_2, \Phi_{P2}\rangle)$ is *strict local confluent modulo* ≡, if the following holds:

a) *Modulo confluence.* The critical pair is local confluent modulo ≡, i. e., there are transformations $t_3 : \langle P_1, \Phi_{P1}\rangle \overset{*}{\Longrightarrow} \langle P_3, \Phi_{P3}\rangle$ and $t_4 : \langle P_2, \Phi_{P2}\rangle \overset{*}{\Longrightarrow} \langle P_4, \Phi_{P4}\rangle$, via productions in **P**, such that $\langle P_3, \Phi_{P3}\rangle \equiv \langle P_4, \Phi_{P4}\rangle$.

b) *Strictness.* Let $der(t_3) = (\langle P_1, \Phi_{P1}\rangle \xleftarrow{v_3} \langle E_3, \Phi_{E3}\rangle \xrightarrow{w_3} \langle P_3, \Phi_{P3}\rangle)$, and $der(t_4) = (\langle P_2, \Phi_{P2}\rangle \xleftarrow{v_4} \langle E_4, \Phi_{E4}\rangle \xrightarrow{w_4} \langle P_4, \Phi_{P4}\rangle)$ be the derived spans of transformations t_3 and t_4; let $\langle N, \Phi_N\rangle$ be the pullback object of pullback (1). Then, there exists a $\langle Z, \Phi_Z\rangle$ with morphisms $z_1, z_2 \in \mathcal{I}^{inj}_{Proj,TG}$ and morphisms $y_3, y_4, y_5 \in \mathcal{M}^{inj}_{Proj,TG}$ such that (2), (3), (4), and (5) commute.

$$\begin{tikzcd}
 & & & \langle K, \Phi_K\rangle & & & \\
 & \langle E_1, \Phi_{E1}\rangle \arrow[urr, "v_1"] \arrow[dl, "w_1"'] & & (1) & & \langle E_2, \Phi_{E2}\rangle \arrow[ull, "v_2"'] \arrow[dr, "w_2"] & \\
\langle P_1, \Phi_{P1}\rangle & & (2) & \langle N, \Phi_N\rangle \arrow[ull, "y_1"] \arrow[urr, "y_2"'] \arrow[dll, "y_3"'] \arrow[drr, "y_4"] \arrow[ddd, "y_5"] & (3) & & \langle P_2, \Phi_{P2}\rangle \\
 & \langle E_3, \Phi_{E3}\rangle \arrow[ul, "v_3"] \arrow[dr, "w_3"'] & (4) & & (5) & \langle E_4, \Phi_{E4}\rangle \arrow[ur, "v_4"'] \arrow[dl, "w_4"] & \\
 & & \langle P_3, \Phi_{P3}\rangle & & \langle P_4, \Phi_{P4}\rangle & & \\
 & & & \langle Z, \Phi_Z\rangle \arrow[ul, "z_1"] \arrow[ur, "z_2"'] & & &
\end{tikzcd}$$

Example 8.20.
Figure 8.6 shows that the critical pair $\langle P_1, \Phi_{P1}\rangle \xLeftarrow{p_1@o_1} \langle K, \Phi_K\rangle \xRightarrow{p_2@o_2} \langle P_2, \Phi_{P2}\rangle$ (originally presented in Example 8.17) is subcommutative modulo normal form equivalence. To actually show that $\langle P_1, \Phi_{P1}\rangle \xLeftarrow{p_1@o_1} \langle K, \Phi_K\rangle \xRightarrow{p_2@o_2} \langle P_2, \Phi_{P2}\rangle$ is subcommutative, we have to find transformations $t_3 : \langle P_1, \Phi_{P1}\rangle \xRightarrow{0..1} \langle P_3, \Phi_{P3}\rangle$, $t_4 : \langle P_2, \Phi_{P2}\rangle \xRightarrow{0..1} \langle P_4, \Phi_{P4}\rangle$ and a symbolic graph $\langle Z, \Phi_Z\rangle$ with $\mathcal{I}^{inj}_{Proj,TG}$-morphisms $z_1 : \langle Z, \Phi_Z\rangle \rightarrow \langle P_3, \Phi_{P3}\rangle$ and $z_2 : \langle Z, \Phi_Z\rangle \rightarrow \langle P_4, \Phi_{P4}\rangle$ such that (2), (3), (4), and (5) commutes. As shown in Figure 8.6 there is indeed such a pair of transformations and symbolic graph $\langle Z, \Phi_Z\rangle$. Note that $\langle Z, \Phi_Z\rangle$ is isomorphic to the projection of $\langle P_3, \Phi_{P3}\rangle$ to Z' (or $\langle P_4, \Phi_{P4}\rangle$ to Z'), where Z' is the subgraph of P_3 (P_4) obtained by removing all auxiliary variables from P_3 (P_4) that have no preimage in N via $w_3 \circ y_3$ ($w_4 \circ y_4$).

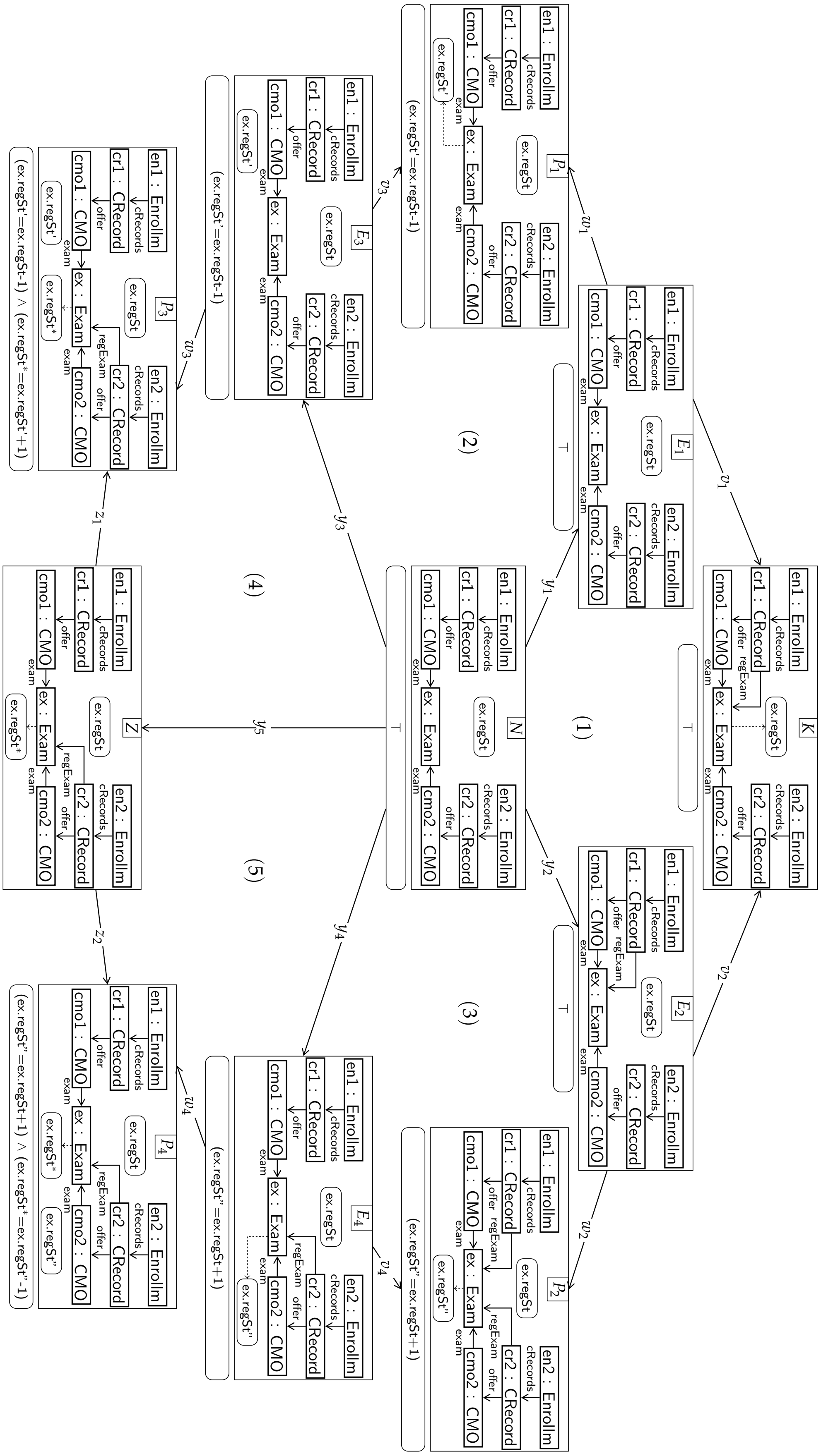


Figure 8.6: Application of Definition 8.19 to show that the critical pair of Example 8.17 is subcommutative.

At the end of this chapter, we shall see a method for constructing the most general $\langle Z, \Phi_Z\rangle$, whereas most general means that if the method fails to construct a $\langle Z, \Phi_Z\rangle$, then there does not exists any symbolic graph $\langle Z, \Phi_Z\rangle$ with $\mathcal{I}^{inj}_{Proj,TG}$-morphisms $z_1 : \langle Z, \Phi_Z\rangle \rightarrow \langle P_3, \Phi_{P3}\rangle$ and $z_2 : \langle Z, \Phi_Z\rangle \rightarrow \langle P_4, \Phi_{P4}\rangle$ such that (4), and (5) commutes. To this end, we need to give an alternative definition for the construction of normal forms from arbitrary symbolic graphs in terms of the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph. While the notion of normal forms given in Definition 4.11 is of more constructive nature, the following definition in terms of smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph has a more declarative character. However, we shall see that both definitions are equivalent.

Definition 8.21 (Smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph)**.**

Let $\langle B, \Phi_B\rangle$ be a typed symbolic graph in $\mathbb{TSG}_{\mathfrak{D},TG}$. An $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of $\langle B, \Phi_B\rangle$ is any typed symbolic graph $\langle A, \Phi_A\rangle$ in $\mathbb{TSG}_{\mathfrak{D},TG}$ for which there exists an $\mathcal{I}^{inj}_{Proj,TG}$-morphisms $f : \langle A, \Phi_A\rangle \rightarrow \langle B, \Phi_B\rangle$.

An $\mathcal{I}^{inj}_{Proj,TG}$-subgraph $\langle A, \Phi_A\rangle$ is the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of $\langle B, \Phi_B\rangle$ if for any $\mathcal{I}^{inj}_{Proj,TG}$-subgraph $\langle C, \Phi_C\rangle$ of $\langle B, \Phi_B\rangle$ with $\mathcal{I}^{inj}_{Proj,TG}$-morphism $h : \langle C, \Phi_C\rangle \rightarrow \langle B, \Phi_B\rangle$, there exists an $\mathcal{I}^{inj}_{Proj,TG}$-morphism $g : \langle A, \Phi_A\rangle \rightarrow \langle C, \Phi_C\rangle$ such that $h \circ g = f$.

$$\begin{tikzcd}
\langle A, \Phi_A\rangle \arrow[rr, "f" description] \arrow[dr, "g"'] \arrow[dr, phantom, "(1)", shift left=6] & & \langle B, \Phi_B\rangle \\
& \langle C, \Phi_C\rangle \arrow[ur, "h"'] &
\end{tikzcd}$$

To actually prove that the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of a graph $\langle B, \Phi_B\rangle$ is indeed isomorphic to the normal form of $\langle B, \Phi_B\rangle$, we first need to show that $\mathcal{I}^{inj}_{Proj,TG}$-morphisms are closed under composition and decomposition.

Lemma 8.22 ($\mathcal{I}^{inj}_{Proj,TG}$ is closed under composition and decomposition)**.**
$\mathcal{I}^{inj}_{Proj,TG}$ is closed under composition and decomposition.

Proof. For untyped E-graphs, this is a consequence of the fact that E-graph morphisms can be composed (decomposed) componentwise in $\mathfrak{Set}$, and isomorphisms and monomorphisms are closed under composition and decomposition in $\mathfrak{Set}$. For typed E-graphs this follows from Definition 3.16. The composition and decomposition property for the formula component is a direct consequence of the fact that the formula component for $\mathcal{I}^{inj}_{Proj,TG}$-morphisms is the same as for $\mathcal{M}^{inj}_{Proj,TG}$-morphisms and $\mathcal{M}^{inj}_{Proj,TG}$ is closed under composition and decomposition. □

Lemma 8.23 (Smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph and normal form)**.**
Let $\langle B, \Phi_B\rangle$ be a typed symbolic graph in $\mathbb{TSG}_{\mathcal{D},TG}$, then a typed symbolic graph $\langle A, \Phi_A\rangle$ in $\mathbb{TSG}_{\mathcal{D},TG}$ is the normal form of $\langle B, \Phi_B\rangle$ if and only if $\langle A, \Phi_A\rangle$ is the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of $\langle B, \Phi_B\rangle$.

Proof. For technical reasons we begin with the only if direction.

Only if. Given typed symbolic graph $\langle A, \Phi_A\rangle \simeq nor(\langle B, \Phi_B\rangle)$ with induced morphism $f : \langle A, \Phi_A\rangle \to \langle B, \Phi_B\rangle$. We have to show that $\langle A, \Phi_A\rangle$ is the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of $\langle B, \Phi_B\rangle$, which is equivalent to prove that $f \in \mathcal{I}^{inj}_{Proj,TG}$ and for any $\mathcal{I}^{inj}_{Proj,TG}$-subgraph $\langle C, \Phi_C\rangle$ of $\langle B, \Phi_B\rangle$ with $h : \langle C, \Phi_C\rangle \to \langle B, \Phi_B\rangle$, $h \in \mathcal{I}^{inj}_{Proj,TG}$, there exists a $g : \langle A, \Phi_A\rangle \to \langle C, \Phi_C\rangle$, $g \in \mathcal{I}^{inj}_{Proj,TG}$ such that $h \circ g = f$.

$$\begin{tikzcd}
\langle A, \Phi_A\rangle \arrow[rr, "f"] \arrow[dr, "g"'] \arrow[rr, phantom, "(1)"', shift right=4] & & \langle B, \Phi_B\rangle \\
& \langle C, \Phi_C\rangle \arrow[ur, "h"'] &
\end{tikzcd}$$

It is easy to see that $f \in \mathcal{I}^{inj}_{Proj,TG}$, as it is constructed as the projection of $\langle B, \Phi_B\rangle$ to A, where A is obtained by just removing all auxiliary variables from B. Now, we show that there exists a typed E-graph morphism g defined by $g = h^{-1} \circ f$, where $h^{-1} : B^* \to C$ is the inverse of h with domain $B^* = h(C)$. We have to verify that $g = h^{-1} \circ f$ is defined for all elements in A. As h is in $\mathcal{I}^{inj}_{Proj,TG}$ (i. e., bijective for graph nodes and all kind of edges), this trivially holds for graph nodes and all kind of edges. For the label nodes, we have to show that $f_X(X_A) \subseteq h_X(X_C)$. By definition $f_X(X_A) = B_X \backslash aux(B)$. Moreover, h is a bijection on label edges; Therefore, at least the label nodes in $B_X \backslash aux(B)$ that are assigned via a label edge must be in $h_X(C_X)$. Consequently, $f_X(X_A) \subseteq h_X(X_C)$; so $g = h^{-1} \circ f$ is a valid E-graph morphism such that (1) commutes.

It remains to show that $g \in \mathcal{I}^{inj}_{Proj,TG}$. By definition, $h \in \mathcal{I}^{inj}_{Proj,TG}$; $\mathcal{I}^{inj}_{Proj,TG}$ is a subclass of $\mathcal{M}^{inj}_{Proj,TG}$. Hence, from the projection property of h follows that g is a symbolic graph morphism. From $f, h \in \mathcal{I}^{inj}_{Proj,TG}$ and the closure of $\mathcal{I}^{inj}_{Proj,TG}$ under decomposition we obtain $g \in \mathcal{I}^{inj}_{Proj,TG}$.

If. Let $\langle B, \Phi_B\rangle$ be a typed symbolic graph in $\mathbb{TSG}_{\mathcal{D},TG}$ and $\langle A, \Phi_A\rangle$ the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of $\langle B, \Phi_B\rangle$ with morphism $f : \langle A, \Phi_A\rangle \to \langle B, \Phi_B\rangle$, $f \in \mathcal{I}^{inj}_{Proj,TG}$. We have to show that $\langle A, \Phi_A\rangle$ is isomorphic to the normal form of $\langle B, \Phi_B\rangle$. Consider the following diagram:

$$\begin{tikzcd}
\langle A, \Phi_A\rangle \arrow[rr, "f"] \arrow[dr, bend left, "g"] & & \langle B, \Phi_B\rangle \\
& \langle C, \Phi_C\rangle \arrow[ul, bend left, "d"] \arrow[ur, "h"'] &
\end{tikzcd}$$

Suppose that $\langle C, \Phi_C \rangle$ is the normal form of $\langle B, \Phi_B \rangle$ with induced morphism $h : \langle C, \Phi_C \rangle \to \langle B, \Phi_B \rangle$, constructed according to Definition 4.11. From the previous part of the proof, we know that $h \in \mathcal{I}^{inj}_{Proj,TG}$. As $\langle A, \Phi_A \rangle$ is the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of $\langle B, \Phi_B \rangle$ there is an $\mathcal{I}^{inj}_{Proj,TG}$-morphism $g : \langle A, \Phi_A \rangle \to \langle C, \Phi_C \rangle$ such that

$$f = h \circ g. \tag{8.1}$$

From $f \in \mathcal{I}^{inj}_{Proj,TG}$, we know from the previous part of this proof that there exists morphisms $d : \langle C, \Phi_C \rangle \to \langle A, \Phi_A \rangle$ such that

$$h = f \circ d. \tag{8.2}$$

By inserting Equation (8.2) into Equation (8.1), we obtain $f = f \circ d \circ g$ which is equivalent to $f \circ id_A = f \circ d \circ g$, hence $id_A = d \circ g$. By inserting Equation (8.1) into Equation (8.2), we obtain $h = h \circ g \circ d$ which is equivalent to $h \circ id_C = h \circ g \circ d$, hence $id_C = g \circ d$. Consequently, g is an isomorphism with inverse $g^{-1} = h$. So $\langle A, \Phi_A \rangle$ and $\langle C, \Phi_C \rangle$ are isomorphic. □

As mentioned before, all morphisms in $\mathcal{I}^{inj}_{Proj,TG}$ are normal form preserving which we prove next.

Lemma 8.24 ($\mathcal{I}^{inj}_{Proj,TG}$-morphisms preserve normal forms)**.**
Any $\mathcal{I}^{inj}_{Proj,TG}$-morphism preserves normal forms; that is, given a morphism $f : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$, if $f \in \mathcal{I}^{inj}_{Proj,TG}$, then $nor(\langle A, \Phi_A \rangle) \simeq nor(\langle B, \Phi_B \rangle)$ and the following diagram commutes, where g and h are the morphisms induced by constructing the normal forms of $\langle A, \Phi_A \rangle$ and $\langle B, \Phi_B \rangle$ according to Definition 4.11, respectively.

$$\begin{tikzcd}
\langle A, \Phi_A \rangle \arrow[rr, "f"] & \arrow[d, phantom, "(1)"] & \langle B, \Phi_B \rangle \\
& nor(\langle A, \Phi_A \rangle) \simeq nor(\langle B, \Phi_B \rangle) \arrow[ul, "g"] \arrow[ur, "h"'] &
\end{tikzcd}$$

Proof. Consider the following diagram, where $\langle A', \Phi'_A \rangle$ and $\langle B', \Phi'_B \rangle$ are the normal forms of $\langle A, \Phi_A \rangle$ and $\langle B, \Phi_B \rangle$ with morphisms $g : \langle A', \Phi'_A \rangle \to \langle A, \Phi_A \rangle$ and $h : \langle B', \Phi'_B \rangle \to \langle B, \Phi_B \rangle$, $g, h \in \mathcal{I}^{inj}_{Proj,TG}$.

Given morphism $f : \langle A, \Phi_A \rangle \to \langle B, \Phi_B \rangle$, $f \in \mathcal{I}^{inj}_{Proj,TG}$, we have to show that $\langle A', \Phi'_A \rangle$ and $\langle B', \Phi'_B \rangle$ are isomorphic.

$$\begin{tikzcd}
& \langle A, \Phi_A \rangle \arrow[rr, "f"] & & \langle B, \Phi_B \rangle \\
\langle A', \Phi'_A \rangle \arrow[ur, "g"] \arrow[rr, bend left, "s"] & & \langle B', \Phi'_B \rangle \arrow[ul, "r"'] \arrow[ur, "h"'] \arrow[ll, bend left, "t"] &
\end{tikzcd}$$

As $\langle B', \Phi'_B \rangle$ is the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of $\langle B, \Phi_B \rangle$ (see Lemma 8.23) and morphism $f \in \mathcal{I}^{inj}_{Proj,TG}$, there exists (according to Definition 8.21) an $\mathcal{I}^{inj}_{Proj,TG}$-morphism $r : \langle B', \Phi'_B \rangle \to \langle A, \Phi_A \rangle$ such that

$$h = f \circ r. \tag{8.3}$$

From $f, g \in \mathcal{I}^{inj}_{Proj,TG}$ and the closure of $\mathcal{I}^{inj}_{Proj,TG}$ under composition, we obtain $(f \circ g) \in \mathcal{I}^{inj}_{Proj,TG}$. Hence, there is an $\mathcal{I}^{inj}_{Proj,TG}$-morphism $t : \langle B', \Phi'_B \rangle \to \langle A', \Phi'_A \rangle$ with

$$h = f \circ g \circ t, \tag{8.4}$$

as $\langle B', \Phi'_B \rangle$ is the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of $\langle B, \Phi_B \rangle$. Similarly, using the smallest $\mathcal{I}^{inj}_{Proj,TG}$-subgraph property of $\langle A', \Phi'_A \rangle$ with $r \in \mathcal{I}^{inj}_{Proj,TG}$, we obtain $\mathcal{I}^{inj}_{Proj,TG}$-morphism $s : \langle A', \Phi'_A \rangle \to \langle B', \Phi'_B \rangle$ such that

$$g = r \circ s. \tag{8.5}$$

Combining Equations (8.3) and (8.4) leads to $f \circ r = f \circ g \circ t$, which implies

$$r = g \circ t. \tag{8.6}$$

By inserting Equations (8.5) into (8.6) and (8.6) into (8.5) we obtain

$$r = r \circ s \circ t \tag{8.7}$$

$$g = g \circ t \circ s. \tag{8.8}$$

Rewriting Equations (8.7) and (8.8) leads to $id_{B'} = s \circ t$ and $id_{A'} = t \circ s$. Hence, $\langle A', \Phi'_A \rangle$ and $\langle B', \Phi'_B \rangle$ are isomorphic. □

As a last property, we require that $\mathcal{I}^{inj}_{Proj,TG}$-morphisms are closed under pushouts and pullbacks along $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphisms, shown next.

Lemma 8.25 (Closure of $\mathcal{I}^{inj}_{Proj,TG}$ under pushouts and pullbacks).
$\mathcal{I}^{inj}_{Proj,TG}$ is closed under pushouts and pullbacks along $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphisms.

Proof. For untyped E-graphs this is a direct consequence of the fact that pushouts in $\mathbb{EG}$ can be constructed componentwise in $\mathfrak{Set}$ and the fact that isomorphism and monomorphism in $\mathfrak{Set}$ are closed under pushouts. The closure property for typed E-graphs follows from Fact 3.17.a. The closure property for the Σ-formula component is a direct consequence of the fact that the formula component for $\mathcal{I}^{inj}_{Proj,TG}$-morphisms is the same as for $\mathcal{M}^{inj}_{Proj,TG}$-morphisms and $\mathcal{M}^{inj}_{Proj,TG}$ is closed under pushouts.

The proof for the closure of $\mathcal{I}^{inj}_{Proj,TG}$ under pullbacks along $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphisms can be obtained similarly. □

Now we are able to proof the main result of this chapter, namely the Local Confluence Theorem modulo normal form equivalence for typed projective transformations systems.

Theorem 8.26 (Local Confluence Theorem modulo normal form equivalence)**.**

Let **TPGTS** = $((\mathbb{TSG}_{\mathcal{D},TG}, \mathcal{M}^{bij}_{\Leftrightarrow,TG}, \mathcal{M}^{inj}_{Proj,TG}, \mathcal{M}^{inj}_{\Rightarrow,TG}), \mathbf{P})$ be a typed projective graph transformation system, then **TPGTS** is locally confluent modulo ≡ if all its critical pairs are strictly confluent modulo ≡.

Proof. Assume that all critical pairs of **TPGTS** are strictly confluent modulo ≡. Given a pair of direct transformations

$$\langle H_1, \Phi_{H1}\rangle \overset{p_1@m_1}{\Longleftarrow} \langle G, \Phi_G\rangle \overset{p_2@m_2}{\Longrightarrow} \langle H_2, \Phi_{H2}\rangle$$

via projective productions $p_1, p_2 \in \mathbf{P}$, we have to show that there exist transformations

$$t'_3 : \langle H_1, \Phi_{H1}\rangle \overset{*}{\Longrightarrow} \langle H_3, \Phi_{H3}\rangle,\ t'_4 : \langle H_2, \Phi_{H2}\rangle \overset{*}{\Longrightarrow} \langle H_4, \Phi_{H4}\rangle$$

via productions in **P** such that $\langle H_3, \Phi_{H3}\rangle \equiv \langle H_4, \Phi_{H4}\rangle$.

If the given pair is parallel independent, then, according to Theorem 8.2, there are transformations t'_3 and t'_4 such that $\langle H_3, \Phi_{H3}\rangle$ and $\langle H_4, \Phi_{H4}\rangle$ are isomorphic and, therefore, also normal form equivalent.

If the given pair is parallel dependent we show in part (i) that there exists transformations t'_3 and t'_4; in part (ii) we show that the results $\langle H_3, \Phi_{H3}\rangle$ and $\langle H_4, \Phi_{H4}\rangle$ are normal form equivalent.

The first part of this proof is an adaption of the proof of Theorem 6.28 in [EEPT06].

i). Suppose the given pair is parallel dependent, then Lemma 8.16 implies the existence of a critical pair $\langle P_1, \Phi_{P1}\rangle \overset{p_1@o_1}{\Longleftarrow} \langle K, \Phi_K\rangle \overset{p_2@o_2}{\Longrightarrow} \langle P_2, \Phi_{P_2}\rangle$ with extension diagrams (20) and (21):

$$\begin{tikzcd}
 & & & \langle G, \Phi_G\rangle \arrow[dddlll, Rightarrow, "p_1@m_1"'] \arrow[dddrrr, Rightarrow, "p_2@m_2"] & & & \\
 & & & & & & \\
 & (20) & & \langle K, \Phi_K\rangle \arrow[uu, "m"'] \arrow[dll, Rightarrow, "p_1@o_1"'] \arrow[drr, Rightarrow, "p_2@o_2"] & & (21) & \\
\langle H_1, \Phi_{H1}\rangle & \langle P_1, \Phi_{P1}\rangle \arrow[l, "q_1"'] \arrow[dr, Rightarrow, "t_3", "*"'] & & & & \langle P_2, \Phi_{P2}\rangle \arrow[r, "q_2"] \arrow[dl, Rightarrow, "t_4"', "*"] & \langle H_2, \Phi_{H2}\rangle \\
 & & \langle P_3, \Phi_{P3}\rangle & \equiv & \langle P_4, \Phi_{P4}\rangle & &
\end{tikzcd}$$

By assumption, this critical pair is strictly confluent modulo ≡, which leads to transformations

$$t_3 : \langle P_1, \Phi_{P1}\rangle \overset{*}{\Longrightarrow} \langle P_3, \Phi_{P3}\rangle,\ t_4 : \langle P_2, \Phi_{P2}\rangle \overset{*}{\Longrightarrow} \langle P_4, \Phi_{P4}\rangle,$$

and symbolic graphs $\langle N, \Phi_N\rangle$ and $\langle Z, \Phi_Z\rangle$ with morphisms shown below, such that (1) is a pullback and (2), (3), (4), and (5) commute:

$$\begin{tikzcd}
 & & & & \langle G,\Phi_G\rangle & & & & \\
 & \langle D_1,\Phi_{D1}\rangle \arrow[urrr, "g_1"] \arrow[ddl, "h_1"'] & & (11) & \langle K,\Phi_K\rangle \arrow[u, "m"] & (12) & & \langle D_2,\Phi_{D2}\rangle \arrow[ulll, "g_2"'] \arrow[ddr, "h_2"] & \\
 & (10) & \langle E_1,\Phi_{E1}\rangle \arrow[ul, "s_1"] \arrow[urr, "v_1"] \arrow[dl, "w_1"] & & (1) & & \langle E_2,\Phi_{E2}\rangle \arrow[ur, "s_2"'] \arrow[ull, "v_2"'] \arrow[dr, "w_2"'] & (13) & \\
\langle H_1,\Phi_{H1}\rangle & \langle P_1,\Phi_{P1}\rangle \arrow[l, "q_1"'] & & (2) & \langle N,\Phi_N\rangle \arrow[ull, "y_1"'] \arrow[urr, "y_2"] \arrow[dll, "y_3"'] \arrow[drr, "y_4"] \arrow[ddd, "y_5"] & (3) & & \langle P_2,\Phi_{P2}\rangle \arrow[r, "q_2"] & \langle H_2,\Phi_{H2}\rangle \\
 & & \langle E_3,\Phi_{E3}\rangle \arrow[ul, "v_3"] \arrow[dr, "w_3"] & (4) & & (5) & \langle E_4,\Phi_{E4}\rangle \arrow[ur, "v_4"'] \arrow[dl, "w_4"'] & & \\
 & & & \langle P_3,\Phi_{P3}\rangle & & \langle P_4,\Phi_{P4}\rangle & & & \\
 & & & & \langle Z,\Phi_Z\rangle \arrow[ul, "z_1"] \arrow[ur, "z_2"'] & & & &
\end{tikzcd}$$

Morphism class $\mathcal{M}^{bij}_{\Leftrightarrow,TG}$ is a subclass of $\mathcal{M}^{inj}_{Proj,TG}$, so $v_1, v_2, v_3, v_3 \in \mathcal{M}^{bij}_{\Leftrightarrow,TG}$ implies $v_1, v_2, v_3, v_3 \in \mathcal{M}^{inj}_{Proj,TG}$. From $v_1, v_2 \in \mathcal{M}^{inj}_{Proj,TG}$, pullback (1), and the closure of $\mathcal{M}^{inj}_{Proj,TG}$ under pullbacks it follows that $y_1, y_2 \in \mathcal{M}^{inj}_{Proj,TG}$. The commutativity of (2) and (3) together with $v_3, w_1, y_1, v_4, w_2, y_2 \in \mathcal{M}^{inj}_{Proj,TG}$ and the closure of $\mathcal{M}^{inj}_{Proj,TG}$ under composition and decomposition implies $y_3, y_4 \in \mathcal{M}^{inj}_{Proj,TG}$. Similarly, the commutativity of (4) and $w_3, y_3, z_1 \in \mathcal{M}^{inj}_{Proj,TG}$ as well as the commutativity of (5) and $w_4, y_4, z_2 \in \mathcal{M}^{inj}_{Proj,TG}$ implies $y_5 \in \mathcal{M}^{inj}_{Proj,TG}$.

Now consider the following diagram in which (6) is a quasi $(\mathcal{L}, \mathcal{N})$-initial pushout over m; (10) and (11) are as in the previous diagram.

$$\begin{tikzcd}
\langle B,\Phi_B\rangle \arrow[r, "b"] \arrow[rr, bend left, "b_1"] \arrow[d, "m'"'] \arrow[dr, phantom, "(6)"] & \langle K,\Phi_K\rangle \arrow[d, "m"] \arrow[dr, phantom, "(11)"] & \langle E_1,\Phi_{E1}\rangle \arrow[l, "v_1"'] \arrow[r, "w_1"] \arrow[d, "s_1"] \arrow[dr, phantom, "(10)"] & \langle P_1,\Phi_{P1}\rangle \arrow[d, "q_1"] \\
\langle C,\Phi_C\rangle \arrow[r, "c"] \arrow[rr, bend right, "c_1"'] & \langle G,\Phi_G\rangle & \langle D_1,\Phi_{D1}\rangle \arrow[l, "g_1"'] \arrow[r, "h_1"] & \langle H_1,\Phi_{H1}\rangle
\end{tikzcd}$$

The initiality of (6), applied to pushout (11), leads to unique morphisms $b_1, c_1 \in \mathcal{M}^{inj}_{Proj,TG}$ such that $v_1 \circ b_1 = b$ and $g_1 \circ c_1 = c$. Moreover, Lemma 5.19 implies that also (17) and (18) are quasi $(\mathcal{L}, \mathcal{N})$-initial pushouts over s_1 and q_1, respectively.

$$\begin{tikzcd}
\langle B,\Phi_B\rangle \arrow[r, "b_1"] \arrow[d, "m'"'] \arrow[dr, phantom, "(17)"] & \langle E_1,\Phi_{E1}\rangle \arrow[r, "w_1"] \arrow[d, "s_1"] \arrow[dr, phantom, "(10)"] & \langle P_1,\Phi_{P1}\rangle \arrow[d, "q_1"] \\
\langle C,\Phi_C\rangle \arrow[r, "c_1"] & \langle D_1,\Phi_{D1}\rangle \arrow[r, "h_1"] & \langle H_1,\Phi_{H1}\rangle
\end{tikzcd}
\qquad
\begin{tikzcd}
\langle B,\Phi_B\rangle \arrow[r, "w_1 \circ b_1"] \arrow[d, "m'"'] \arrow[dr, phantom, "(18)"] & \langle P_1,\Phi_{P1}\rangle \arrow[d, "q_1"] \\
\langle C,\Phi_C\rangle \arrow[r, "h_1 \circ c_1"] & \langle H_1,\Phi_{H_1}\rangle
\end{tikzcd}$$

Analogously, we obtain morphisms $b_2, c_2 \in \mathcal{M}^{inj}_{Proj,TG}$ with $v_2 \circ b_2 = b$ from (12) and (13).

$$\begin{tikzcd}
 & & & & & \langle C,\Phi_C\rangle \arrow[ddl, "c"'] & & & \\
 & & & & & \langle B,\Phi_B\rangle \arrow[u, "m'"'] \arrow[ddl, "b"] \arrow[dddlll, "b_1"] \arrow[dddr, "b_2"] \arrow[ddddl, "b'"] \arrow[dddddlll, "b_3"'] \arrow[dddddr, bend right, "b_4"] & & & \\
 & & & & \langle G,\Phi_G\rangle & & & & \\
 & \langle D_1,\Phi_{D1}\rangle \arrow[urrr, "g_1"] \arrow[ddl, "h_1"'] & & (11) & \langle K,\Phi_K\rangle \arrow[u, "m"] & (12) & & \langle D_2,\Phi_{D2}\rangle \arrow[ulll, "g_2"'] \arrow[ddr, "h_2"] & \\
 & (10) & \langle E_1,\Phi_{E1}\rangle \arrow[ul, "s_1"] \arrow[urr, "v_1"] \arrow[dl, "w_1"] & & (1) & & \langle E_2,\Phi_{E2}\rangle \arrow[ur, "s_2"'] \arrow[ull, "v_2"'] \arrow[dr, "w_2"] & (13) & \\
\langle H_1,\Phi_{H1}\rangle & \langle P_1,\Phi_{P1}\rangle \arrow[l, "q_1"'] & & (2) & \langle N,\Phi_N\rangle \arrow[ull, "y_1"'] \arrow[urr, "y_2"] \arrow[dll, "y_3"] \arrow[drr, "y_4"'] \arrow[dddd, "y_5"] & & (3) & \langle P_2,\Phi_{P2}\rangle \arrow[r, "q_2"] & \langle H_2,\Phi_{H2}\rangle \\
 & (9) & \langle E_3,\Phi_{E3}\rangle \arrow[ul, "v_3"'] \arrow[dl, "s_3"'] \arrow[ddr, "w_3"'] & & & & \langle E_4,\Phi_{E4}\rangle \arrow[ur, "v_4"] \arrow[dr, "s_4"] \arrow[ddl, "w_4"] & (14) & \\
 & \langle D_3,\Phi_{D3}\rangle \arrow[uul, "g_3"] \arrow[dddrr, "h_3"'] & & (4) & & (5) & & \langle D_4,\Phi_{D4}\rangle \arrow[uur, "g_4"'] \arrow[dddl, "h_4"] & \\
 & & (8) & \langle P_3,\Phi_{P3}\rangle \arrow[dd, "q_3"'] & & \langle P_4,\Phi_{P4}\rangle \arrow[ddr, "q_4"] & (15) & & \\
 & & & & \langle Z,\Phi_Z\rangle \arrow[ul, "z_1"'] \arrow[ur, "z_2"] & & & & \\
 & & & \langle H_3,\Phi_{H3}\rangle & & & \langle H_4,\Phi_{H4}\rangle & &
\end{tikzcd}$$

By using the universal property of pullback (1) with $v_1 \circ b_1 = b = v_2 \circ b_2$, we obtain unique morphism $b' : \langle B, \Phi_B\rangle \to \langle N, \Phi_N\rangle$ with $y_1 \circ b' = b_1$ and $y_2 \circ b' = b_2$. By $b_1, y_1 \in \mathcal{M}^{inj}_{Proj,TG}$ and the decomposition property of $\mathcal{M}^{inj}_{Proj,TG}$, we can conclude that also $b' \in \mathcal{M}^{inj}_{Proj,TG}$. To show the consistency of q_1 with respect to transformation t_3, with quasi $(\mathcal{L}, \mathcal{N})$-initial pushout (18), we have to construct $b_3 : \langle B, \Phi_B\rangle \to \langle E_3, \Phi_{E3}\rangle$, $b_3 \in \mathcal{M}^{inj}_{Proj,TG}$, such that $v_3 \circ b_3 = w_1 \circ b_1$. This holds for $b_3 = y_3 \circ b'$. Moreover, as $y_3, b' \in \mathcal{M}^{inj}_{Proj,TG}$, so $b_3 \in \mathcal{M}^{inj}_{Proj,TG}$ (using the composition property of $\mathcal{M}^{inj}_{Proj,TG}$-morphisms). In the same way we can show the consistency of q_2 with respect to t_4. Finally, by Theorem 8.13 we get extension diagrams (22) and (23).

$$\begin{tikzcd}
 & & & \langle G,\Phi_G\rangle \arrow[ddlll, Rightarrow, "p_1@m_1"'] \arrow[ddrrr, Rightarrow, "p_2@m_2"] & & & \\
 & & (20) & \langle K,\Phi_K\rangle \arrow[u, "m"] \arrow[dll, Rightarrow, "p_1@o_1"'] \arrow[drr, Rightarrow, "p_2@o_2"] & (21) & & \\
\langle H_1,\Phi_{H1}\rangle \arrow[ddr, Rightarrow, "t_3'"', "*" near end] & \langle P_1,\Phi_{P1}\rangle \arrow[l, "q_1"'] \arrow[dr, Rightarrow, "t_3", "*" near end] & & & & \langle P_2,\Phi_{P2}\rangle \arrow[r, "q_2"] \arrow[dl, Rightarrow, "t_4"', "*"' near end] & \langle H_2,\Phi_{H2}\rangle \arrow[ddl, Rightarrow, "t_4'", "*"' near end] \\
 & (22) & \langle P_3,\Phi_{P3}\rangle \arrow[dl, "q_3"'] & \equiv & \langle P_4,\Phi_{P4}\rangle \arrow[dr, "q_4"] & (23) & \\
 & \langle H_3,\Phi_{H3}\rangle & & & & \langle H_4,\Phi_{H4}\rangle &
\end{tikzcd}$$

ii). For the second part we have to show that $\langle H_3, \Phi_{H3}\rangle \equiv \langle H_4, \Phi_{H4}\rangle$. To this end, consider the following diagram.

$$\begin{tikzcd}
 & & & \langle B, \Phi_B\rangle \arrow[dlll, "w_3 \circ b_3"'] \arrow[ddl, "b_z"] \arrow[dd, "m'"] \arrow[drr, "w_4 \circ b_4"] & & \\
\langle P_3, \Phi_{P3}\rangle \arrow[dd, "q_3"] & & & & & \langle P_4, \Phi_{P4}\rangle \arrow[dd, "q_4"] \\
 & (7) & \langle Z, \Phi_Z\rangle \arrow[ull, "z_1"] \arrow[urrr, "z_2"] \arrow[dd, "z_3"] & \langle C, \Phi_C\rangle \arrow[dlll, "h_3 \circ c_3"'] \arrow[ddl, "c_z^*"] \arrow[drr, "h_3 \circ c_4"] & (16) & \\
\langle H_3, \Phi_{H3}\rangle & & & & & \langle H_4, \Phi_{H4}\rangle \\
 & & \langle Z', \Phi'_Z\rangle \arrow[ull, "z'_1"] \arrow[urrr, "z'_2"'] & & &
\end{tikzcd}$$

First we choose morphism $b_z : \langle B, \Phi_B\rangle \to \langle Z, \Phi_Z\rangle$ such that $b_z = y_5 \circ b'$. Then, we construct the diagonal face as the pushout of b_2 and m'. By morphisms $q_3 \circ z_1$ and $c_3 \circ h_3$ and the universal property of the diagonal pushout we obtain unique morphism $z'_1 : \langle Z', \Phi'_Z\rangle \to \langle H_3 \Phi_{H3}\rangle$. As the back left face is an pushout (i. e., a quasi $(\mathcal{L}, \mathcal{N})$-initial pushout; see Theorem 8.14.c), we may conclude that (7) is a pushout by decomposing the back left pushout with the diagonal pushout. Similarly, we may obtain pushout (16). It still remains to show that $\langle H_3, \Phi_{H3}\rangle \equiv \langle H_4, \Phi_{H4}\rangle$, which is equivalent to show that $nor(\langle H_3, \Phi_{H3}\rangle) \simeq nor(\langle H_4, \Phi_{H4}\rangle)$. To this end, consider the following diagram.

$$\begin{tikzcd}
\langle P_3, \Phi_{P3}\rangle \arrow[d, "q_3"] \arrow[drr, phantom, "(7)"] & & \langle Z, \Phi_Z\rangle \arrow[ll, "z_1"'] \arrow[rr, "z_2"] \arrow[d, "z'"] \arrow[drr, phantom, "(16)"] & & \langle P_4, \Phi_{P4}\rangle \arrow[d, "q_4"] \\
\langle H_3, \Phi_{H3}\rangle & & \langle Z', \Phi'_Z\rangle \arrow[ll, "z'_1"'] \arrow[rr, "z'_2"] & & \langle H_4, \Phi_{H4}\rangle \\
 & \langle X', \Phi'_X\rangle \arrow[ul, "x_1"] \arrow[ur, "x_0"] \arrow[urrr, "x_2"'] & & &
\end{tikzcd}$$

From pushouts (7) and (16) and $q_3, q_4 \in \mathcal{M}^{inj}_{\Rightarrow,TG}$ follows $z' \in \mathcal{M}^{inj}_{\Rightarrow,TG}$. By definition $z_1, z_2 \in \mathcal{I}^{inj}_{Proj,TG}$, so $z'_1, z'_2 \in \mathcal{I}^{inj}_{Proj,TG}$ as $\mathcal{I}^{inj}_{Proj,TG}$ is closed under pushouts along $\mathcal{M}^{inj}_{\Rightarrow,TG}$-morphisms. Now we construct $\langle X', \Phi'_X\rangle$ as the normal form of $\langle Z', \Phi'_Z\rangle$. As $z'_1, z'_2 \in \mathcal{I}^{inj}_{Proj,TG}$ we can conclude from Lemma 8.24 that $\langle X', \Phi'_X\rangle$ is the normal form of $\langle H_3, \Phi_{H3}\rangle$ and $\langle H_4, \Phi_{H4}\rangle$; hence, $nor(\langle H_3, \Phi_{H3}\rangle)$ and $nor(\langle H_4, \Phi_{H4}\rangle)$ are isomorphic. □

To construct the most general $\langle Z, \Phi_Z\rangle$ we use the fact that any finite symbolic graph has finitely many $\mathcal{I}^{inj}_{Proj,TG}$-subgraphs.

Definition 8.27 (Construction of the most general $\langle Z, \Phi_Z\rangle$).
Given $\mathcal{M}^{inj}_{Proj,TG}$-morphisms $y_3 : \langle N, \Phi_N\rangle \to \langle E_3, \Phi_{E3}\rangle$ and $w_3 : \langle E_3, \Phi_{E3}\rangle \to \langle P_3, \Phi_{P3}\rangle$ with finite symbolic graph $\langle P_3, \Phi_{P3}\rangle$ as given in Definition 8.19. Let

$\mathbf{Z} = \{z'_1 : \langle Z'_1, \Phi'_{Z1}\rangle \rightarrow \langle P_3 \Phi_{P3}\rangle, \ldots, z'_n : \langle Z'_n, \Phi'_{Zn}\rangle \rightarrow \langle P_3, \Phi_{P3}\rangle\}$ be the finite set consisting of all $\mathcal{I}^{inj}_{Proj,TG}$-morphisms for all $\mathcal{I}^{inj}_{Proj,TG}$-subgraphs $\langle Z'_i, \Phi'_{Zi}\rangle$ of $\langle P_3, \Phi_{P3}\rangle$ such that $w_3 \circ y_3 = z'_i \circ y'_{5,i}$ for all $i \in \{1, \ldots, n\}$, as shown in the diagram below.

$$\begin{tikzcd}
 & & & \langle N,\Phi_N\rangle & \\
\langle E_3,\Phi_{E3}\rangle & (4) & \langle Z'_1,\Phi'_{Z1}\rangle & & \\
\langle P_3,\Phi_{P3}\rangle & {} & & \langle Z'_2,\Phi'_{Z2}\rangle & \\
 & & & & \langle Z'_n,\Phi'_{Zn}\rangle
\arrow[from=1-4, to=2-1, "y_3"']
\arrow[from=1-4, to=2-3, "y'_{5,1}"']
\arrow[from=1-4, to=3-4, "y'_{5,2}"]
\arrow[from=1-4, to=4-5, "y'_{5,n}"]
\arrow[from=2-1, to=3-1, "w_3"']
\arrow[from=2-3, to=3-1, "z'_1"']
\arrow[from=3-4, to=3-1, "z'_2"]
\arrow[from=3-2, to=3-1, dotted]
\arrow[from=4-5, to=3-1, "z'_n"]
\end{tikzcd}$$

Symbolic graph $\langle Z, \Phi_Z\rangle$ is then derived by iterated pullback constructions as shown in the diagram below, where $(2')$, $(3')$, ..., (n') are pullbacks. Pullback $(1')$ (not shown in the diagram below) is the trivial pullback obtained by intersecting $\langle Z'_1, \Phi'_{Z1}\rangle$ with itself, leading to $\langle Z_1, \Phi_{Z1}\rangle \simeq \langle Z'_1, \Phi'_{Z1}\rangle$.

$$\begin{tikzcd}
 & & & \langle N,\Phi_N\rangle & & & \\
\langle E_3,\Phi_{E3}\rangle & (4) & \langle Z_1,\Phi_{Z1}\rangle & & \langle Z_2,\Phi_{Z2}\rangle & & \\
 & & (2') & \langle Z'_2,\Phi'_{Z2}\rangle & & {} & \\
\langle P_3,\Phi_{P3}\rangle & {} & {} & (3') & & & \langle Z,\Phi_Z\rangle \\
 & & & & & (n') & \\
 & & & & \langle Z'_n,\Phi'_{Zn}\rangle & &
\arrow[from=1-4, to=2-1, "y_3"']
\arrow[from=1-4, to=2-3, "y_{5,1}"']
\arrow[from=1-4, to=3-4, "y'_{5,2}"]
\arrow[from=1-4, to=2-5, "y_{5,2}"]
\arrow[from=1-4, to=6-5, "y'_{5,n}"]
\arrow[from=1-4, to=4-7, bend left, "y_{5,n}"]
\arrow[from=2-1, to=4-1, "w_3"']
\arrow[from=2-3, to=4-1, "z'_1"']
\arrow[from=3-4, to=4-1, "z'_2"]
\arrow[from=4-2, to=4-1, dotted]
\arrow[from=6-5, to=4-1, "z'_n"]
\arrow[from=2-5, to=2-3, "a_2"']
\arrow[from=2-5, to=3-4, "b_2"]
\arrow[from=3-6, to=2-5, dotted, "a_3"]
\arrow[from=3-6, to=4-3, dotted, "b_3"]
\arrow[from=4-7, to=3-6, dotted, "a_n"]
\arrow[from=4-7, to=6-5, "b_n"]
\end{tikzcd}$$

For each pullback (i') the morphism $y_{5,i} : \langle N, \Phi_N\rangle \rightarrow \langle Z_i, \Phi_{Zi}\rangle$ is constructed by applying the universal pullback property of pullback (i') to morphisms $y_{5,i-1} : \langle N, \Phi_N\rangle \rightarrow \langle Z_{i-1}, \Phi_{Zi-1}\rangle$ and $y'_{5,i} : \langle N, \Phi_N\rangle \rightarrow \langle Z'_i, \Phi'_{Zi}\rangle$. After n pullbacks the result is the symbolic graph $\langle Z, \Phi_Z\rangle$ with morphism $z_1 : \langle Z, \Phi_Z\rangle \rightarrow \langle P_3, \Phi_{P3}\rangle$ defined as $z_1 = z'_n \circ b_n$. As $\mathcal{I}^{inj}_{Proj,TG}$ is closed under pullbacks, morphism $a_i, b_i, z'_i \in \mathcal{I}^{inj}_{Proj,TG}$ for $i \in \{1, .., n\}$ so $z_1 = z'_n \circ b_n$.

Note that for any symbolic graph $\langle Z'_i, \Phi'_{Zi}\rangle$ there is a symbolic $\mathcal{I}^{inj}_{Proj,TG}$-morphism $z^*_i : \langle Z, \Phi_Z\rangle \rightarrow \langle Z'_i, \Phi'_{Zi}\rangle$.

Lemma 8.28 (Construction of the most general $\langle Z, \Phi_Z \rangle$)**.**
Let $\langle Z, \Phi_Z \rangle$ with $\mathcal{I}^{inj}_{Proj,TG}$-morphism $z_1 : \langle Z, \Phi_Z \rangle \rightarrow \langle P_3, \Phi_{P3} \rangle$ be the symbolic graph constructed according to Definition 8.27, then $\langle Z, \Phi_Z \rangle$ is the most general in the following sense. If no morphism $z_2 : \langle Z, \Phi_Z \rangle \rightarrow \langle P_4, \Phi_{P4} \rangle$ exists such that $z_2 \circ y_5 = w_4 \circ y_4$, then there does not exists an other $\langle Z', \Phi'_Z \rangle$ with $\mathcal{I}^{inj}_{Proj,TG}$-morphisms $z'_1 : \langle Z', \Phi'_Z \rangle \rightarrow \langle P_3, \Phi_{P3} \rangle$ and $z'_2 : \langle Z', \Phi'_Z \rangle \rightarrow \langle P_4, \Phi_{P4} \rangle$, as well as $\mathcal{M}^{inj}_{Proj,TG}$-morphism $y'_5 : \langle N, \Phi_N \rangle \rightarrow \langle Z', \Phi'_Z \rangle$ such that $z'_1 \circ y'_5 = w_3 \circ y_3$ and $z'_2 \circ y'_5 = w_4 \circ y_4$.

$$\begin{tikzcd}
 & & \langle N, \Phi_N \rangle \arrow[dll, "y_3"'] \arrow[drr, "y_4"] \arrow[d, "y_5"] & & \\
\langle E_3, \Phi_{E3} \rangle \arrow[dr, "w_3"'] & (4) & \langle Z, \Phi_Z \rangle \arrow[dl, "z_1"'] \arrow[dr, dashed, "z_2"] & (5) & \langle E_4, \Phi_{E4} \rangle \arrow[dl, "w_4"] \\
 & \langle P_3, \Phi_{P3} \rangle & & \langle P_4, \Phi_{P4} \rangle &
\end{tikzcd}$$

Proof. We show this lemma by contradiction. To this end, consider the diagram shown below, where $\langle Z, \Phi_Z \rangle$ with $z_1 \circ y_5 = w_3 \circ y_3$ is the symbolic graph constructed according to Definition 8.27. Let us assume that there does not exists morphism $z_2 : \langle Z, \Phi_Z \rangle \rightarrow \langle P_4, \Phi_{P4} \rangle$ but there exists an other graph $\langle Z', \Phi'_Z \rangle$ with $\mathcal{I}^{inj}_{Proj,TG}$-morphisms $z'_1 : \langle Z', \Phi'_Z \rangle \rightarrow \langle P_3, \Phi_{P3} \rangle$, $z'_2 : \langle Z', \Phi'_Z \rangle \rightarrow \langle P_4, \Phi_{P4} \rangle$, and $\mathcal{M}^{inj}_{Proj,TG}$-morphism $y'_5 : \langle N, \Phi_N \rangle \rightarrow \langle Z', \Phi'_Z \rangle$ such that $z'_1 \circ y'_5 = w_3 \circ y_3$ and $z'_2 \circ y'_5 = w_4 \circ y_4$. Hence, $\langle Z', \Phi'_Z \rangle$ is an $\mathcal{I}^{inj}_{Proj,TG}$-subgraph of $\langle P_4, \Phi_{P3} \rangle$. Consequently, $z'_1 : \langle Z', \Phi'_Z \rangle \rightarrow \langle P_3, \Phi_{P3} \rangle$ must be in **Z** and there must be a morphism $z^* : \langle Z, \Phi_Z \rangle \rightarrow \langle Z', \Phi'_Z \rangle$ such that $z'_2 \circ z^* \circ y_5 = w_4 \circ y_4$ which is a contradiction.

$$\begin{tikzcd}
 & & \langle N, \Phi_N \rangle \arrow[dll, "y_3"'] \arrow[drr, "y_4"] \arrow[d, "y_5"] & & \\
\langle E_3, \Phi_{E3} \rangle \arrow[ddr, "w_3"'] & & \langle Z, \Phi_Z \rangle \arrow[ddl, "z_1"'] \arrow[dd, "z^*"] & & \langle E_4, \Phi_{E4} \rangle \arrow[ddl, "w_4"] \\
 & & & & \\
 & \langle P_3, \Phi_{P3} \rangle & & \langle P_4, \Phi_{P4} \rangle & \\
 & & \langle Z', \Phi'_Z \rangle \arrow[ul, "z'_1"] \arrow[ur, "z'_2"'] & &
\end{tikzcd}$$

□

9

TOOL SUPPORT AND EVALUATION

To show that the presented theoretical result are applicable in practice, all techniques presented in Chapter 7 and Chapter 8 were implemented resulting in the Symbolic Graph Analysis and Verification (SyGrAV) tool prototype. In this chapter we give an overview on our efforts and insights gained when implementing SyGrAV and analysing the CMS case study introduced in Chapter 2. More specifically, in Section 9.1, we give an overview on the SyGrAV tool prototype. In Sections 9.2 and 9.3 we present the measurement results and insights gained when applying the constant enforcement and conflict analysis techniques to the CMS case study. The chapter concludes with discussing the measurement results and giving some directions for further improvements.

9.1 The Symbolic Graph Analysis and Verification Framework

The majority of SyGrAV is realized using the model transformation and metacase tool (i. e., a tool for building tools) eMoflon [LAS14]. Hence, almost all components are realized in Java and conform to the Eclipse Modeling Framework (EMF).

Figure 9.1 depicts the basic structure of SyGrAV, which consists of the following main components:

1. The Symbolic Graphs and Morphisms component provides an Eclipse Modeling Framework (EMF) compliant metamodel for symbolic graph productions in terms of symbolic graphs and morphisms. This component facilitates interfacing between SyGrAV and other graph transformation tools. As a proof of concept we have realized a transformation to translate graph transformation specifications from the eMoflon tool to SyGrAV. In the current version we support symbolic graphs as presented in this thesis, i. e., node attributed symbolic graphs with first-order formulas. Basically, arbitrary first-order formulas can be specified; however, the actual support is constrained by the capabilities of the solver used within the symbolic graph pattern matching component.

2. The Symbolic Graph Pattern Matching component is conceptually the most challenging part, as it draws on solving a combination of two problems that are themselves subject to intensive research, namely graph pattern matching (also known as the subgraph isomorphism problem) and satisfiability checking of first-order formulas over multiple background theories. However, by combining specialized of the shelf solvers for each problem, respectively, we were able to reduce the implementation efforts and increase the efficiency and reliability

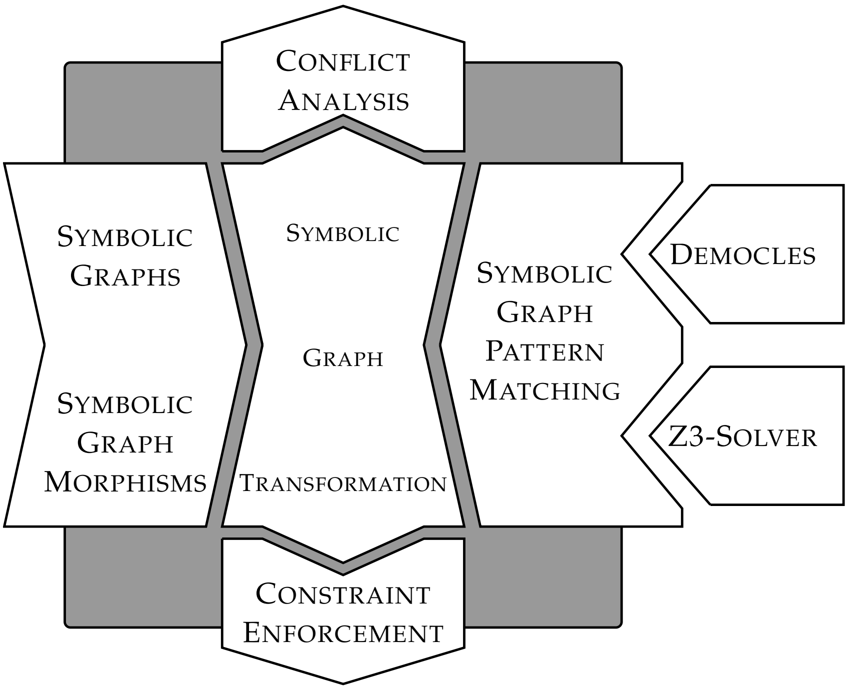


Figure 9.1: The Symbolic Graph Analysis and Verification Framework (SyGrAV)

of the implementation at the same time. For the current implementation we combined the Democles pattern matching engine with the Z3 SMT solver.

a) Democles is a local search based pattern matching engine that is currently developed at the Fachgebiet Echtzeitsysteme, Technische Universität Darmstadt, Germany [VAS12]. Note that the subgraph isomorphism problem is NP-complete (or polynomial assuming a fixed size pattern graph). For this reason, Democles uses heuristics and a search plan driven strategy in order to optimize execution times of the pattern matching process.

b) Z3 is a well-established satisfiability modulo theories (SMT) solver, which is widely used in several projects [dMB08]. Z3 is interfaced by using the SMT-LIB[1] format, which defines common input and output language for SMT solvers. Hence, Z3 may easily be replaced with any other SMT solver that supports the SMT-LIB format. Z3 supports first-order formulas with equality over various background theories, including nonlinear integer and real arithmetic. Note that satisfiability of problems using these FOL fragments is undecidable in general. Nevertheless, this does not prevent Z3 from finding an answer in many cases.

In combination, also symbolic graph pattern matching is undecidable. Hence, in contrast to pure graph pattern matching, we have to handle the case that there is an E-graph morphisms but Z3 is not able to decide whether the morphism is a symbolic graph morphism. In such a case Z3 returns unknown. In the current implementation those cases are basically treated as there is no morphism. As we

[1] http://smtlib.cs.uiowa.edu/

shall see later, this treatment guarantees the soundness of our implementations for constraint enforcement and conflict analysis.

3. The Symbolic Graph Transformation component encapsulates the basic categorical constructions for symbolic graphs such as pushouts and pullbacks, but also more complex compound functionalities such as the computation of direct transformations via double pushouts and the construction of all possible gluings of two graphs. The Symbolic Graph Transformation component relies on the Symbolic Graphs and Morphisms and Symbolic Graph Pattern Matching components. Note that this component is not a library for generic categorical construction as, e. g., proposed in [MS10]; in fact, it offers specific categorical constructions optimized for symbolic graphs.

4. The Constraint Enforcement component realizes the translation of graph constraints to left application conditions. The current version supports in addition to negative constraints and application conditions (as presented in Chapter 7), also arbitrary nested conditions as, e. g., proposed in [EGH$^+$14].

5. The Conflict Analysis component encapsulates the critical pair and confluence analysis techniques for symbolic graphs as presented in Chapter 8. The current version supports construction of symbolic critical pairs and subcommutativity analysis modulo normal form equivalence for projective graph transformation systems. Currently, productions with application conditions are not supported.

Summing up, the SyGrAV framework is a toolbox for static analysis and verification of attributed graph transformations. The main component to facilitate this techniques is the Symbolic Graph Pattern Matching component, which allows for finding symbolic graph morphisms between arbitrary symbolic graphs and not only for finding matches of symbolic graphs in instance models (i. e., definite or grounded symbolic graphs). As we shall see in the following sections, SyGrAV performs quite well for symbolic graphs consisting of a couple of elements, i. e., for symbolic graphs as they appear in productions. However, SyGrAV is not intended as a model transformation tool to transform large models (i. e., grounded or definite symbolic graphs) consisting of thousands of elements.

9.2 Support for Enforcing Symbolic Graph Constraints

In the following, we first give an overview on the constraint enforcement capabilities of SyGrAV in Section 9.2.1. Subsequently we present the measurement results collected by actually verifying the campus management system case study in Section 9.2.2. Finally, we argue that the current implementation is sound.

9.2.1 *Support for Enforcing Symbolic Graph Constraints in SyGrAV*

The current implementation of the constraint enforcement techniques is very close to the theoretical results presented in Chapter 7. Accordingly, the overall process from a set of constraints to a rule with only essential NACs comprises the following steps:

Generation of right application conditions (presented in Section 7.1). Although we presented only the generation of *negative* right application condition from *negative* constraints in Section 7.1, the current version of SyGrAV also supports the generation of right application conditions from arbitrary nested constraints. The current implementation is based on the procedure proposed in [EGH+14]. However, both procedures (i. e., the one presented in Section 7.1 and the one presented in [EGH+14]) rely on the generation of all possible gluings of two graphs. As the number of possible gluings grows exponentially with the size of the involved graphs, this procedure itself does not scale. However, a more important aspect is to keep the number of generated gluings as small as possible, as all succeeding steps have to be performed on the outcomes of this step, i. e., for each gluing. During our experiments, it turned out that the restriction to linear gluings (i. e. gluings that are linear symbolic graphs, see Definition 4.8) dramatically reduces their number; that is, we require that if two graph nodes are glued together all their label nodes have to be glued together, too (if present).

right to left application conditions (presented in Section 7.2). Similar to the previous step SyGrAV supports the transformation of arbitrarily nested right application conditions to left application conditions. The current implementation follows closely the procedure proposed in [EGH+14].

Minimization of left NACs (presented in Section 7.3). While the construction of left application condition is purely syntactically (also for the formula components), the minimization of application conditions requires semantic reasoning about arbitrary nested application conditions which is undecidable in general (still for pure graphs without attribute conditions). For this reason, SyGrAV currently supports only the minimization of negative application conditions. The minimization is performed as described in Section 7.3; that is, in a first step all *consistency guaranteeing* left negative application conditions are removed (see Section 7.3.1). The result is a set rules with only *consistency preserving* left NACs. Subsequently, for each rule all subsumed left NACs are removed (see Section 7.3.2). The result is a set of rules that carry only *essential* left NACs.

9.2.2 *Performance Evaluation*

The results of running the construction of left application conditions for the campus management system are summarized in Table 9.1. The graph transformation system comprises 18 productions, which are all listed in the Appendix A. The productions are denoted using the compact notation originally introduced in Chapter 2. However, to establish a connection between the notation used for symbolic graphs and productions in Chapters 3–8, all attribute expressions are given as a first-order formula depicted below the productions. Note that primed variables appear only in the RHS graph and are implicitly assumed to be existentially quantified in the LHS formulas.

The campus management system example comprises 90 negative constraints, where 7 negative constraints are user defined (see Appendix B). The remaining 83 negative constraints were automatically generated from the cardinality and containment restrictions imposed by the metamodel. For example, consider Figure 9.2

Table 9.1: Overall measurement results

#Rules	#Constraints	Time NC to Pre NAC	Time Minimize	Time Overall
18	90	3.66 sec	6.48 sec	10.13 sec

that shows the negative constraints generated for the containment association date of cardinality 0..1 from the class Exam to the class Date (see Figure 2.1). Negative constraints (a) and (b) are generated from the fact that association date has cardinality 0..1; hence, there must not exist two links of type date from an Exam ex to two different dates or to the same date. Note that constraint (b) is owed to the fact that we support only injective symbolic morphisms. Negative constraint (c) is generated from the fact that association date defines a containment relation; that is, an object of type Date is contained in at most one other object. However, as according to the CMS metamodel (see Figure 2.1) dates may also be contained in instances of class Lecture, additionally negative constraint (c) is generated. The negative constraints for the other association are generated similarly. Note that the generated constraints do not impose any restriction on the attributes which is represented by a formula component equivalent to ⊤ (i. e., *true*).

The overall process for generating all left application conditions from those 90 constraints for all 18 productions requires about 10 seconds, where the most time (i. e., 6.5 sec) was spent for minimizing and 3.7 sec were spent for generating all left NACs. The measurements were performed on a Windows machine with a core i-7-2600-3.4GHz CPU and 8 GB memory. For the measurement the overall procedure was run 20 times. In order to compensate the just-in-time optimization performed by the java virtual machine, only the last 15 runs were considered to calculate the average values shown in Table 9.1. The standard deviation of the overall runtime with respect to the last 15 runs was below 200ms.

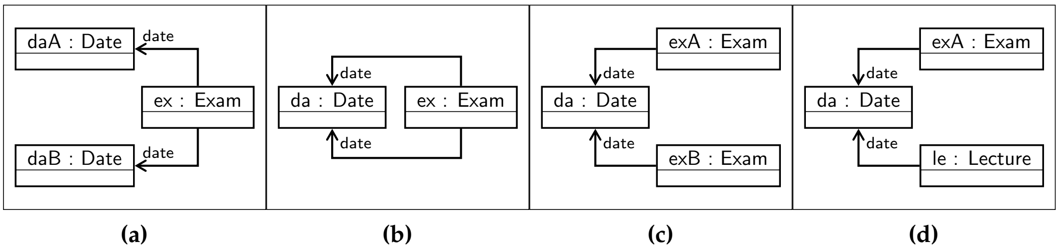


Figure 9.2: Negative constraints generated from the containment association *date* of cardinality 0..1 from the class Exam to the class Date.

Table 9.2 shows the measurement results for each production separately. The first column contains the name of the productions followed by the sum of the graph nodes and edges contained in the left-hand side of the production. As we only consider linear gluings, we did not take the number of label nodes and edges into account as they do not influence the number of gluings. As mentioned, all productions can be found in the appendix. The second column contains the number of generated right application conditions, and column 3, 4, and 5 contain the number of generated left application conditions before minimization, after removing the consistency guaranteeing NACs, and after removing subsumed NACs for each production, respectively. One interesting result is, that most of the generated left NACs are consistency guaranteeing and thus are removed during minimization. By also removing the subsumed NACs the number of remaining essential NACs is quite small. More specifically, from the 2043 right application conditions generated for all productions (see last row of Table 9.2), just 24 remain after minimization. Although, the minimization step is not necessary form a theoretical point of view (i. e., no constraint can be violated by applying a production), these results show that the minimization step is important from a practical point of view to not degrade the performance when applying a production by hundreds of unnecessary application condition checks.

As shown in Table 9.2 the number of right NACs and, consequently, also the runtimes are mainly influenced by the number of contained graph elements.

Table 9.2: Detailed measurement results

Production (#graphElements)	#NACs				Time [ms]					
	Post NACs	Pre NACs			NCs to Pre NACs			Minimizing		
	All	All	Preserving	Essential	NC to Post	Post to Pre	Overall	Guaranteeing	Subsumed	Overall
transResFail (12)	212	212	0	0	178	91	269	625	0	625
transResPas (12)	212	212	0	0	176	85	261	619	0	619
updateEx (9)	191	191	2	1	165	83	248	714	8	722
regExam (7)	187	187	3	2	178	89	267	570	8	578
unregExam (8)	184	184	0	0	172	72	244	544	0	544
updateLect (9)	163	163	0	0	161	60	221	544	0	544
regCMO (6)	140	80	4	2	170	53	223	302	17	319
setExam (6)	139	139	6	3	162	50	212	746	43	789
setLecture (7)	126	126	4	2	158	44	202	579	15	594
regTMO (7)	103	76	2	1	163	35	198	162	7	169
regThesis (5)	83	59	2	1	157	27	184	144	6	150
bookRoom (4)	75	52	4	2	154	21	175	150	17	167
updateDate (3)	62	62	3	2	154	11	165	145	6	151
obtDeg (2)	49	36	2	1	154	11	165	97	6	103
setDate (4)	41	41	7	4	153	8	161	115	18	133
uploadRes (1)	38	38	6	3	152	6	158	118	29	147
closeExam (2)	19	19	0	0	152	3	155	53	0	53
resetCMO (1)	19	19	0	0	150	3	153	71	0	71
Sum (105)	2043	1896	45	24	2909	752	3661	6298	180	6478

9.2.3 *Soundness of the Conflict Enforcement Procedure*

Although, symbolic graph pattern matching is undecidable, the implemented constraint enforcement procedure is sound in the sense that after running the constraint enforcement procedure for a set of constraints and productions, the resulting extended productions are consistency preserving. To show this, we have to argue that for the involved steps either (i) no symbolic graph pattern matching is required, or (ii) if symbolic graph pattern matching is required, we have to argue that not recognizing a morphism (although there exists one) does not lead to the loss of a left application condition.

Construction of right application conditions. As shown in Remark 7.2 the construction of the formula component for the gluings is performed on the on the syntactical level. Hence, there is no need to invoke Z3.

Construction of left application conditions. As mentioned in Remark 6.13 to decide whether an right application condition can be shifted along a functional projective production, as well as the shift construction itself do not require any reasoning on the semantics of the involved formulas.

Minimization of left NACs In order to argue that minimization procedure is sound, we have to show that not recognizing a symbolic graph morphism (although there exists one) does not lead to a removal of an essential NAC. By considering the procedures presented in Section 7.3 one can see that a NAC is only removed if there either exists a symbolic graph morphism from a negative constraint to the corresponding NAC or a symbolic graph morphism from an other NAC to the corresponding NAC. Hence, in both cases not capturing a morphism does not lead to the removal of a NAC. Consequently, the minimization procedure is sound also for undecidable background theories. However, we cannot guarantee that the set of application condition is minimal (also in case of negative application conditions); that is, after minimization there might remain negative application conditions that are not essential.

Although we could not observe such a case when analyzing the CMS case study, it is reasonable that such case become more likely especially for difficult problems over background theories such as nonlinear real arithmetic. However, note that all generated left application condition are sufficient and necessary in the sense that none of the generated application condition blocks an application of a production that would lead to a consistent graph.

9.3 Support for Conflict Analysis

In the following, we present the capabilities fo conflict analysis of SyGrAV. In Section 9.3.1, we report on our experiences gained when analysing the CMS case study with SyGrAV. The measurement results collected by actually analysing the CMS case study are presented in Section 9.3.2. Finally, we show that the implemented conflict analysis procedure is sound, also for undecidable background theories.

9.3.1 *Conflict Detection and Resolution with SyGrAV*

In the current version, SyGrAV supports conflict detection by critical pair analysis and conflict resolution by subcommutativity analysis as presented in Chapter 8.

In the following, we present how the proposed techniques can be used to analyze our CMS system to detect and correct unintended conflicts. Table 9.3 summarizes the results of our efforts. More specifically, Table 9.3a shows the nonresolvable conflicts (marked by an ✗) of the initial campus management system specification called CMS. Table 9.3b shows the nonresolvable conflicts of the corrected campus management system specification called CMS'. The primed productions denote those productions that were actually changed to mitigate a conflict. The complete list of all productions (i. e., initial as well as the corrected ones) can be found in the

Table 9.3: Comparison of nonresolvable conflicts (✗) of the initial and the improved GTS specifications CMS and CMS′, respectively.

	bookRoom	uploadRes	setDate	updateDate	transResPas	transResFail	closeExam	regExam	regCMO	unregExam	regTMO	regThesis	obtDeg	setLecture	setExam	resetCMO	updateLect	updateEx
bookRoom	✗																	
uploadRes		✗																
setDate																		
updateDate	✗																	
transResPas																		
transResFail																		
closeExam					✗	✗												
regExam	✗				✗	✗												
regCMO						✗			✗									
unregExam					✗	✗												
regTMO						✗												
regThesis					✗	✗												
obtDeg						✗												
setLecture														✗				
setExam															✗			
resetCMO																		
updateLect																	✗	
updateEx					✗	✗		✗		✗							✗	✗

(a) Nonresolvable conflicts of the initial GTS specification CMS

	bookRoom’	uploadRes	setDate	updateDate	transResPas’	transResFail’	closeExam	regExam’	regCMO	unregExam’	regTMO	regThesis	obtDeg	setLecture	setExam	resetCMO	updateLect	updateEx’
bookRoom’	✗																	
uploadRes		✗																
setDate																		
updateDate	✗																	
transResPas’																		
transResFail’																		
closeExam					✗	✗												
regExam’																		
regCMO						✗			✗									
unregExam’																		
regTMO						✗												
regThesis					✗	✗												
obtDeg						✗												
setLecture														✗				
setExam															✗			
resetCMO																		
updateLect																	✗	
updateEx’																	✗	✗

(b) Nonresolvable conflicts of the corrected GTS specification CMS′

appendix. We were able mitigate 9 nonresolvable conflicts; that is, from initially 25 nonresolvable conflicts for CMS to 16 for CMS'. As expected, we were not able to mitigate all conflicts, as this would imply that the system is confluent; consequently, every (or none) enrollment would lead to a degree regardless the actual results achieved for the exams. However, we were able to discover and mitigate some problematic conflicts.

Our studies with SyGrAV lead to the following workflow:

1. **Running the conflict analysis.** To run this step a graph transformation system has to be provided containing the productions that have to be analyzed. During this step the conflict analysis is performed on all pairs of productions. The result is an analysis report for each pair. Accordingly, a pair has
   - *no conflict*; that is, the pair is parallel independent
   - *a conflict that can be resolved*; that is, the pair is parallel dependent but is subcommutative.
   - *a conflict that can not be resolved*; that is, the pair has either a real conflict that can not be resolved or SyGrAV (i. e., more specifically the Z3 solver) was unable decide the validity of a formula during the analysis. In order to guarantee the soundness of our approach such a case is reported as nonresolvable even if the involved productions actually do not constitute a nonresolvable conflict (details follow in Section 9.3.3). For each nonresolvable conflict the reason is given by means of the corresponding critical pair.
2. **Manually analysing the critical pairs.** As the critical pairs contain the minimal contexts that lead to the conflict they provide useful hints how to adapt the spec-

ification to mitigate the conflict. During our work with SyGrAV we established the following characterization of nonresolvable conflicts.

- *Intended conflict.* As mentioned before the campus management system is intentionally nonconfluent. Hence, there are intended conflicts that shall not be resolved. However, by inspecting the minimal contexts it can be ensured that there are no unintended reasons for a conflict. For example, the updateDate and bookRoom task are in conflict, as according to the life cycle of an examination artifact (Figure 2.4) the date of an exam may only be updated as long as no room is booked. By inspecting the critical pairs we can ensure that there is no additional reason for the conflict of these tasks. Especially, those reasons caused by the concurrent interaction of two or more artifacts are of interest.
- *Impossible conflict.* By inspecting the critical pairs we often encountered the case, where a captured conflict relied on the existence of a minimal context that should not occur in normal system operation. In such cases the conflict can be mitigated by adding a negative constraint. For example, the reason that implies the existence of two current Semesters (i. e., two semesters both with current attribute value equal to true).
- *Unintended conflict.* This kind of conflict characterizes all conflicts that are unintentionally and can only be solved by adapting the involved productions.
- *Undecidable conflict.* Such a conflict usually appears if during the conflict analysis the Z3 solver was not able to decide the validity of a formula.
- *Lack of expressiveness conflict.* We were not able to mitigate some conflicts due to the missing support of application conditions for conflict detection.

After manually analysing the critical pairs and refining the specification, the process is repeated until the desired results are obtained.

In the following, we demonstrate this process by mitigating the conflict between bookRoom and regExam. The result of running the conflict analysis is a single critical pair that is not subcommutative. The corresponding minimal context is shown in Figure 9.3. By inspecting this minimal context, we can see that the left-hand sides of the rules are glued along the Exam ex. Consequently, the reason for the conflict is related to a conflicting manipulation of attributes located in ex : Exam. The only attribute variable that is accessed by both productions is ex.regSt. Consequently, it is sufficient to consider only the part of the formulas that are related to the variables ex.regSt or ex.regSt'. Accordingly, the problematic parts are $(\mathsf{ro.capExam} \leq \mathsf{ex.regSt})$ of bookRoom and $(\mathsf{ex.regSt}' = \mathsf{ex.regSt} + 1)$ of regExam. More specifically, to apply the production bookRoom the number of registrations for the exam (i. e., the value of ex.regSt) must not exceed the exam capacity of the room (i. e., the value of ro.capExam). However, the application of production regExam increments the value of ex.regSt by one. Hence, if the exam capacity of the room and the number of registered students are equal (i. e., $\mathsf{ex.regSt} = \mathsf{ro.capExam}$) the production bookRoom cannot be applied to the same room after applying regExam. Consequently, first applying bookRoom and then regExam leads to a different result than applying the production the other way around. Note that Z3 provides an counterexample if

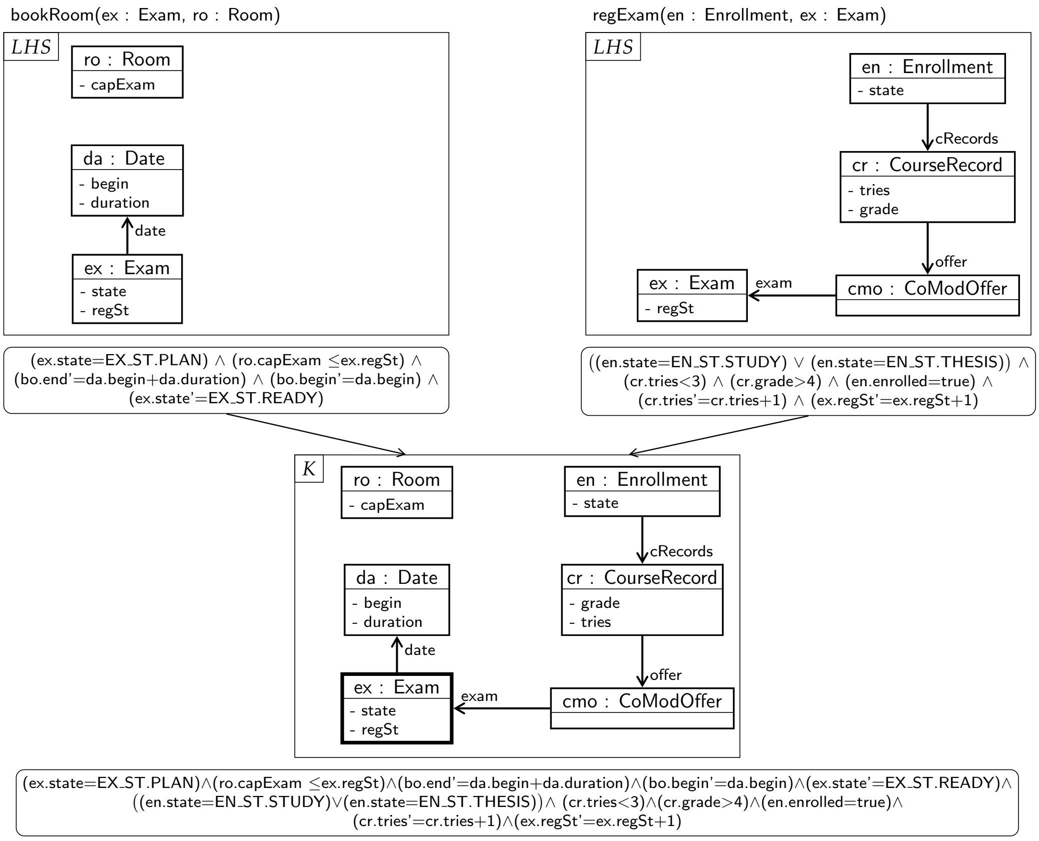

bookRoom(ex : Exam, ro : Room)
LHS
ro : Room
- capExam
da : Date
- begin
- duration
date
ex : Exam
- state
- regSt
regExam(en : Enrollment, ex : Exam)
LHS
en : Enrollment
- state
cRecords
cr : CourseRecord
- tries
- grade
offer
ex : Exam
- regSt
exam
cmo : CoModOffer
(ex.state=EX_ST.PLAN) ∧ (ro.capExam ≤ex.regSt) ∧ (bo.end'=da.begin+da.duration) ∧ (bo.begin'=da.begin) ∧ (ex.state'=EX_ST.READY)
((en.state=EN_ST.STUDY) ∨ (en.state=EN_ST.THESIS)) ∧ (cr.tries<3) ∧ (cr.grade>4) ∧ (en.enrolled=true) ∧ (cr.tries'=cr.tries+1) ∧ (ex.regSt'=ex.regSt+1)
K
ro : Room
- capExam
en : Enrollment
- state
cRecords
da : Date
- begin
- duration
cr : CourseRecord
- grade
- tries
date
offer
ex : Exam
- state
- regSt
exam
cmo : CoModOffer
(ex.state=EX_ST.PLAN)∧(ro.capExam ≤ex.regSt)∧(bo.end'=da.begin+da.duration)∧(bo.begin'=da.begin)∧(ex.state'=EX_ST.READY)∧ ((en.state=EN_ST.STUDY)∨(en.state=EN_ST.THESIS))∧ (cr.tries<3)∧(cr.grade>4)∧(en.enrolled=true)∧ (cr.tries'=cr.tries+1)∧(ex.regSt'=ex.regSt+1)

Figure 9.3: The minimal context of bookRoom and regExam that leads to a conflict that is not subcommutative.

two formulas are note equivalent. Basically, these counterexample might be used to assist with finding the reason for the conflict. However, this is currently not implemented.

According to the aforementioned characterization, this conflict is an unintended conflict. Hence, we have to adapt the involved productions. There are several possibilities to synchronize the involved productions. We decided us to assign a global schedule to each semester to ensure that productions bookRoom and regExam cannot be applied at the same time. To this end, we augmented the class Semester with the attributes semBegin, semEnd, regBegin, and regEnd determining the begin and end of the semester as well as the begin and end of the corresponding examination registration period, respectively. The intended schedule of the tasks bookRoom and regExam is illustrated in Figure 9.4; that is, registrations for an examination should only be possible during the registration period of the semester, whereas a room may only be booked for an examination after the registration period. In this way, bookRoom and regExam cannot be applied at the same time. Moreover, scheduling the task bookRoom after the registration period also ensures that the booked room provides sufficient capacity (i. e., seats) to conduct the examination.

In order to realize this scheduling we need access to the current time. To this end, we introduce the class System that has an attribute currentTime carrying the actual time. The refined productions bookRoom' and regExam' are shown in Figure 9.5a

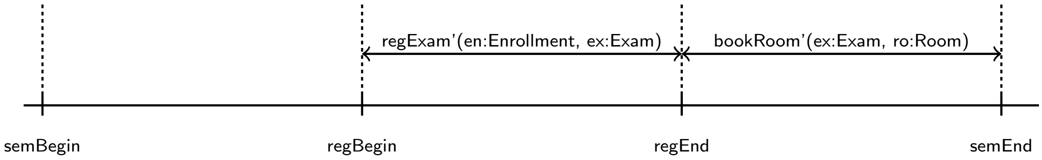


Figure 9.4: Schedule for the bookRoom' and regExam' tasks.

and Figure 9.5b, respectively. Both refined productions now look up the current semester (sem : Semester) and the system timer (sys : System). By adding the corresponding predicates to the formula (the middle line), production bookRoom' is only applicable if the current time is larger than the value for the semester begin the (sys.currentTime > sem.regEnd), whereas (sys.currentTime > sem.regBegin) ∧ (sys.currentTime < sem.regEnd) ensures that production regExam' is only applicable during the registration period of the current semester.

Finally, to get things working, we have to tell the analysis framework that System is a singleton class, otherwise we may have a gluing with two Systems with potentially different values for the currentTime attribute. This is achieved by adding the negative constraint shown in Figure 9.6 to the specification.

The other production is refined in the same manner according to the schedule shown in Figure 9.7. The complete list of productions (i. e., the initial and corrected versions) and negative constraints can be found in the appendix.

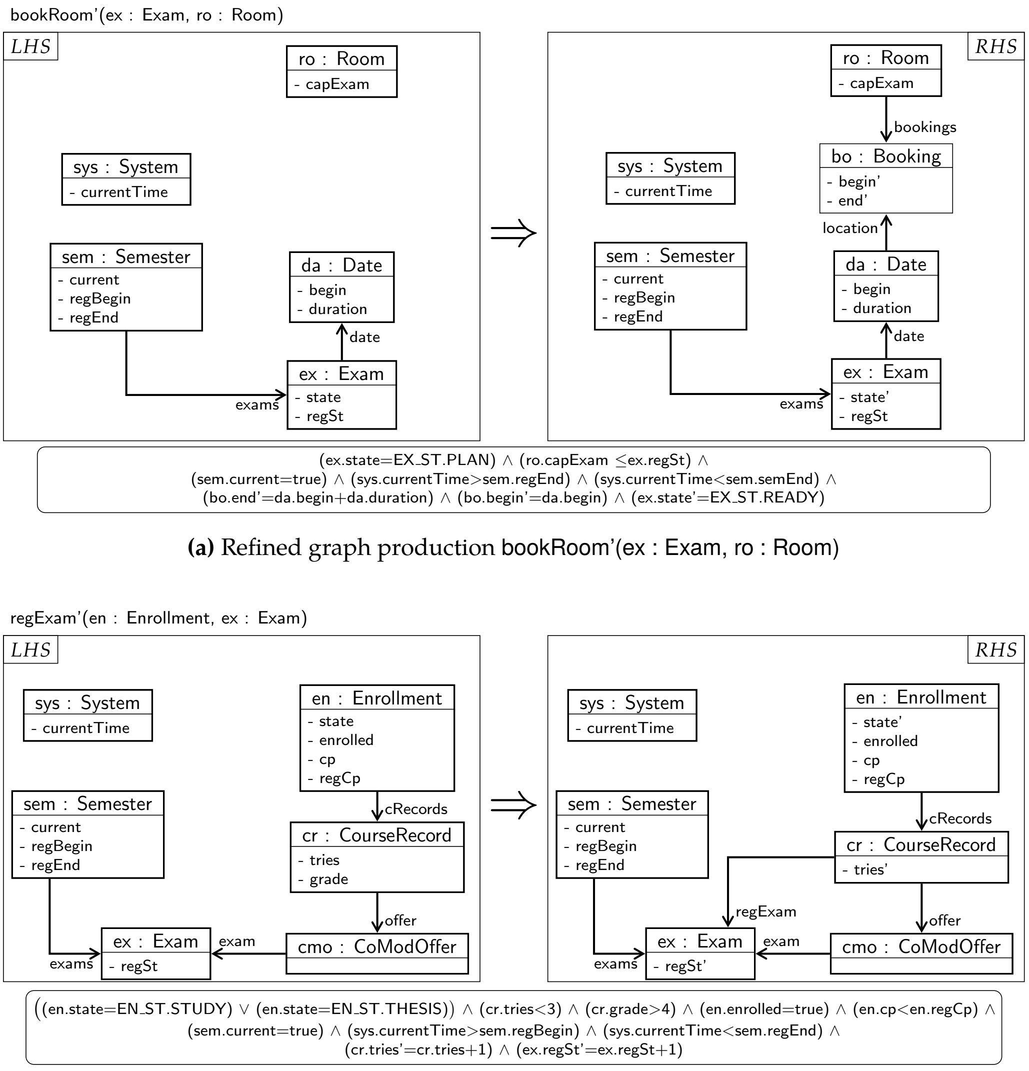
Table 9.4 shows the remaining nonresolvable conflicts characterized according to the aforementioned characterization. It can be seen that most conflicts are intended conflicts (I). However, there are two lack of expressiveness conflicts (LE) and one undecidable conflict (U). The lack of expressiveness conflict between the production closeExam and the productions transResPas' and transResFail' can be resolved with a NAC that requires that all results have been transferred before the exam can be closed. However, application conditions are not supported by the current implementation ot the conflict analysis process.

9.3.2 *Performance Evaluation*

In the following, we give an overview on the runtime results for performing conflict analysis with the SyGrAV framework for the campus management system example, whereas the measurement setup is the same as for constraint enforcement. Table 9.5a compares the overall measurement results of the conflict analysis for initial version (CMS) and improved version (CMS′) of the campus management system.

The results are interpreted as follows:

Generation of all minimal contexts. The minimal contexts are generated by all possible gluings of the left hand sides of the productions. For this step, the same procedure as for the generation of right application conditions is used; that is, only linear gluings are considered (i. e. gluings that are linear symbolic graphs, see Definition 4.8). The number and the time for generating all minimal contexts for all pairs of productions for CMS and CMS′ are shown in the corresponding row. It can be seen that modifications to correct the specification increase the number of minimal contexts by a factor of approximately 3, which is a result of increasing the



(a) Refined graph production bookRoom'(ex : Exam, ro : Room)

(b) Refined graph production regExam'(en : Enrollment, ex : Exam)

Figure 9.5: Refined graph productions bookRoom' and regExam'

number of elements in left-hand sides of some productions. A more remarkable fact is that although the number of minimal contexts increased by a factor of 3 the calculation time increased by a factor of nearly 9. Hence, (at least in the specific case) it seems that the runtime of the implemented procedure is quadratic in the number of generated gluings.

Calculating consistent minimal contexts. In order to reduce the number of minimal contexts, in this step those minimal contexts that are inconsistent with respect to a negative constraint are removed. The result is a set of *consistent minimal contexts*. Surprisingly, although 3 times more minimal contexts were constructed for CMS' (compared to the number constructed for CMS) the amount of consistent minimal contexts is only slightly larger.

As shown in Table 9.5a, by this step the number of minimal contexts that have to be considered for conflict analysis can be dramatically reduced. More specifically, in case of CMS by 92% and in case of CMS' by 97%. Hence, filtering inconsistent

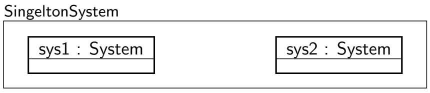


Figure 9.6: Negative constraints to forbid the existence of two instances of class System.

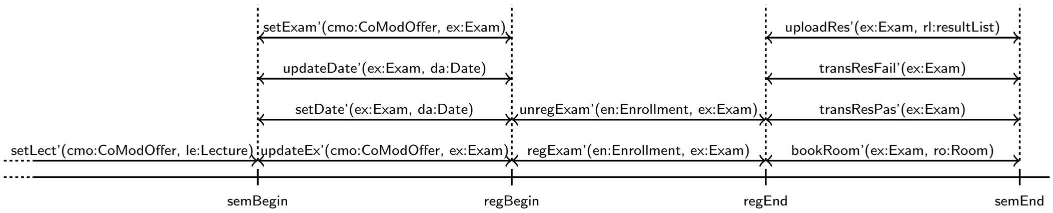

setExam'(cmo:CoModOffer, ex:Exam)
updateDate'(ex:Exam, da:Date)
setDate'(ex:Exam, da:Date)
setLect'(cmo:CoModOffer, le:Lecture)
updateEx'(cmo:CoModOffer, ex:Exam)
unregExam'(en:Enrollment, ex:Exam)
regExam'(en:Enrollment, ex:Exam)
uploadRes'(ex:Exam, rl:resultList)
transResFail'(ex:Exam)
transResPas'(ex:Exam)
bookRoom'(ex:Exam, ro:Room)
semBegin
regBegin
regEnd
semEnd

Figure 9.7: Overall schedule for the tasks CMS′

minimal contexts can be considered as the key measure to reduce the runtime of the overall conflict analysis procedure.

Critical pair analysis. In this step all critical pairs are built as explained in Section 8.2. The result is a set of pairs of parallel dependent direct transformations. As expected, improving the specification leads to fewer conflicts (i. e., critical pairs).

Subcommutativity analysis. In this step subcommutativity modulo normal form equivalence is checked for each critical pair. As the number of critical pairs for CMS′ is smaller than for CMS, the time for subcommutativity analysis, as well as the number of nonsubcommutative critical pairs for CMS′ is smaller than for CMS.

Table 9.5b shows the relative amount of time spent for *symbolic graph pattern matching*, which is the sum of the time spent for pure *graph pattern matching* and the amount of time required for *solving the involved first-order formulas*. In both cases (i. e. for CMS and CMS′) more than 80% of the runtime is spent for symbolic graph pattern matching. However, while for analysing CMS most of the time (i. e., 59%) is spent for first-order logic solving, the situation is turned for analysing CMS′; that is, most of time (i. e., 50%) is spent for graph pattern matching. This can be explained with the huge difference in the number of generated minimal contexts for CMS and CMS′, and the efforts required to filter them. Recall, to check whether a symbolic graph $\langle K, \Phi_K \rangle$ is inconsistent with respect to a negative constraint $nc(\langle N, \Phi_N \rangle)$, we have to find a symbolic graph morphisms $c : \langle N, \Phi_N \rangle \to \langle K, \Phi_K \rangle$; that is, an E-graph morphisms $c : N \to K$ such that such that $(\Phi_K \Rightarrow \Phi_N[\hat{c}])$ is valid. As mentioned in Section 9.2 the majority of negative constraints are generated from the metamodel, which means they are of the form $nc(\langle N, \top \rangle)$ (recall, $\top$ means *true*). Hence, finding a symbolic graph morphism $c : \langle N, \top \rangle \to \langle K, \Phi_K \rangle$ reduces to pure graph pattern matching as $(\Phi_K \Rightarrow \top)$ is trivially valid. Hence, regarding our example, filtering inconsistent minimal contexts requires manly pure graph pattern matching.

Table 9.6 shows the overall runtime results for conflict analysis for each pair of productions measured in milliseconds. Additionally, we denoted after each production the number of graph elements (i. e., the sum of graph nodes and edges) contained in the left-hand side. It can be seen that the overall time required for conflict analysis is influenced by (i) the similarity of the involved productions, (ii) the the number of graph elements, and (iii) the complexity of the involved formulas. The correlation between the runtime and similarity of the involved productions is

Table 9.4: Corrected version CMS′ with nonresolvable conflicts where **I** denotes an intended conflict, **LE** a lack of expressiveness conflict, and **U** an undecidable conflict.

	bookRoom'	uploadRes	setDate	updateDate	transResPas'	transResFail'	closeExam	regExam'	regCMO	unregExam'	regTMO	regThesis	obtDeg	setLecture	setExam	resetCMO	updateLect	updateEx'
bookRoom'	I																	
uploadRes		I																
setDate																		
updateDate	I																	
transResPas'																		
transResFail'																		
closeExam					LE	LE												
regExam'																		
regCMO						I			I									
unregExam'																		
regTMO						I												
regThesis					U	I												
obtDeg						I												
setLecture														I				
setExam															I			
resetCMO																		
updateLect																	I	
updateEx'																	I	I

the most significant. It can be seen that in almost all cases analysing the production with itself requires the most time, which is not surprising as two similar graphs have potentially more overlappings as less similar graphs. Also the relation between runtime and size of the involved productions is reflected by the measurements. The relation between the complexity of the involved formulas and the runtime is quite more harder to grasp. However, during our experiments it has become apparent that Z3 has problems with `if then else` expressions (see for example production transResFail in Appendix A).

9.3.3 *Soundness of the Conflict Analysis Procedure*

Similar to the constraint enforcement procedure, we can guarantee the soundness of our conflict analysis procedure.

We begin with arguing that the critical pair analysis procedure is sound; that is, if according to the procedure two productions p_1 and p_2 are nonconflicting, then for all symbolic graph $\langle G, \Phi_G \rangle$ any pair of direct transformations

$$\langle H_1, \Phi_{H1} \rangle \overset{p_1@m_1}{\Longleftarrow} \langle G, \Phi_G \rangle \overset{p_2@m_2}{\Longrightarrow} \langle H_2, \Phi_{H2} \rangle,$$

via productions p_1 and p_2 and matches m_1 and m_2 is parallel independent. To show this, we have to argue that for the involved steps either(i) no symbolic graph pattern matching is required, or (ii) if symbolic graph pattern matching is required, we have

Table 9.5: Overview of the measurement results of the conflict analysis for the initial version (CMS) and corrected version (CMS′) of the campus management system.

	CMS		CMS'	
	Quantity	Time	Quantity	Time
Minimal Contexts	2794	4.1 s	9441	34.2 s
Consistent Minimal Contexts	209	14.4 s	211	23.8 s
Critical Pairs	144	2.2 s	110	2.7 s
Nonsubcommutative Critical Pairs	36	21.8 s	24	15.3 s
Sum		42.6 s		76 s

(a)

	CMS	CMS'
Graph Pattern Matching	27%	50%
First Order Logic Solving	59%	33%
Symbolic Graph Pattern Matching	86%	83%

(b)

Table 9.6: Pairwise runtime required for overall conflict analysis in *milliseconds*

	bookRoom' (7)	uploadRes (2)	setDate (2)	updateDate (2)	transResPas' (15)	transResFail' (15)	closeExam (1)	regExam' (10)	regCMO (7)	unregExam' (11)	regTMO (7)	regThesis (5)	obtDeg (3)	setLecture (6)	setExam (6)	resetCMO (1)	updateLect (9)	updateEx' (10)
bookRoom' (7)	517																	
uploadRes (2)	22	191																
setDate (2)	65	11	220															
updateDate (2)	301	11	49	220														
transResPas' (15)	126	218	12	12	3918													
transResFail' (15)	121	208	12	12	3822	3926												
closeExam (1)	22	50	11	11	1143	1138	85											
regExam' (10)	96	135	147	147	1578	1581	143	995										
regCMO (7)	10	0	0	0	1985	1550	0	594	1033									
unregExam' (11)	96	130	141	141	1649	1651	140	1017	520	1066								
regTMO (7)	10	0	0	0	1322	1295	0	217	194	257	181							
regThesis (5)	10	0	0	0	1330	1285	0	98	156	145	73	122						
obtDeg (3)	10	0	0	0	1288	1118	0	78	67	125	62	15	115					
setLecture (6)	73	0	0	0	1441	1425	0	341	145	379	5	1	2	979				
setExam (6)	98	129	141	141	1556	1538	138	431	143	465	5	1	1	602	963			
resetCMO (1)	10	0	0	0	1270	1263	0	175	108	216	5	1	1	45	48	62		
updateLect (9)	74	0	0	0	1457	1442	0	353	152	389	5	1	1	1450	601	16	3148	
updateEx' (10)	452	329	386	387	1504	1493	332	351	175	398	5	1	1	575	928	17	878	1906

to argue that not recognizing a morphism (although there exists one) does not lead to the loss of a critical pair.

Construction of all minimal contexts. As shown in Remark 7.2 the construction of the formula component for the gluings is performed on the on the syntactical level. Hence, there is no need to invoke Z3.

Filtering minimal contexts. It is easy to see that missing a morphism from a negative constraint to a minimal context during the filtering process does not lead to the removal of a consistent minimal context.

Constructing direct derivations (for minimal contexts). As mentioned in Remark 6.18, the construction of a direct transformation for a projective production with a given match can be performed purely syntactically also for the formula component. Hence, there is no need to invoke Z3 for constructing direct derivation (provided that a match of the left-hand side is given).

Checking parallel independence (dependence). As mentioned in Remark 8.4, only pure E-graph matching is required to decide whether a pair of transformations is parallel independent or not. Consequently, the critical pair analysis procedure is sound.

In order to argue that our conflict resolution procedure based on subcommutativity analysis (see Definition 8.19) is sound, we basically have to consider two situation where symbolic graph pattern matching is required. Assume, given a critical pair

$$\langle H_1, \Phi_{H1}\rangle \xLeftarrow{p_1@m_1} \langle G, \Phi_G\rangle \xRightarrow{p_2@m_2} \langle H_2, \Phi_{H2}\rangle,$$

to show (or refute) subcommutativity of the critical pair the procedure has to construct direct transformations $t_3 : \langle P_1, \Phi_{P1}\rangle \xRightarrow{p_3@m_3} \langle P_3, \Phi_{P3}\rangle$ and $t_4 : \langle P_2, \Phi_{P2}\rangle \xRightarrow{p_4@m_4} \langle P_4, \Phi_{P4}\rangle$. To this end, the matches m_3 and m_4 have to be looked up, which requires symbolic graph pattern matching. However, if the recognition of one of the matches fails (i. e., it is falsely not recognized as a symbolic graph morphism), we potentially miss a direct transformation that potentially would resolve the conflict, but does not lead to a conflict that is falsely resolved.

The second situation where we need symbolic graph pattern matching is to decide whether the results of the transformation $t_3 : \langle P_1, \Phi_{P1}\rangle \xRightarrow{p_3@m_3} \langle P_3, \Phi_{P3}\rangle$ and $t_4 : \langle P_2, \Phi_{P2}\rangle \xRightarrow{p_4@m_4} \langle P_4, \Phi_{P4}\rangle$ are equivalent modulo normal form. To this end the procedure constructs the most general symbolic graph $\langle Z, \Phi_Z\rangle$ with morphism $z_1 : \langle Z, \Phi_Z\rangle \to \langle P_3, \Phi_{P3}\rangle$ from $\langle P_3, \Phi_{P3}\rangle$. In order to show that (4) and (5) commutes (see Definition 8.19), we have to find symbolic $\mathcal{I}^{inj}_{Proj,TG}$-morphism z_2 and symbolic $\mathcal{M}^{inj}_{Proj,TG}$ morphism y_5. However, if the recognition of one of the morphisms fails (i. e., it is falsely not recognized as a symbolic graph morphism), the procedure potentially misses a $\langle Z, \Phi_Z\rangle$ that possibly would resolve the conflict, but does not lead to a conflict that is falsely resolved.

9.4 Threats to Validity

By providing the SyGrAV prototype we have shown that the theoretical results obtained in Chapters 4–8 can be implemented.

We conducted experiments on a case study from the enterprise modeling domain, including 18 productions that were inspired by the real workflows of the campus management system actually used at Technische Universität Darmstadt. During our experiments it turned out that the minimizing and filtering steps are the key measures to apply the proposed constraint enforcement and conflict analysis techniques to reasonably realistic problems. Regarding the scalability of the approach, the complexity of the underlying analysis problem is mainly caused by the size and number of the productions (and graph constraints) under consideration. While we expect that the number of productions is much larger in an industrial size scenarios, the number of elements per production in our running example is quite representative for medium size productions (according to our experiences with model transformation). However, to finally assess the significance of the measurements with respect to larger scenarios, further experiments have to be conducted. Nevertheless, by taking into account that for industrial size verification problems, runtimes of several days on large multiprocessor computers are acceptable, it seem reasonable that the proposed techniques are also applicable to those problems.

By introducing (functional) projective graph transformation and thereby omitting the need for an infinite number of label nodes to represent attribute values, we were able to provide an implementation that is very close to our theoretical construction, that were shown to be correct. We have shown that the implemented procedures are sound, also if the used fragment of first-order logic is undecidable. All results that were obtained for the CMS case study were manually revised and checked for plausibility. Additionally, we have a test suite with several (smaller) examples whose validity were checked manually. However, SyGrAV is still a prototype and we cannot exclude the presence of bugs.

Threats to external validity may arise from the usage of off-the-shelf SMT solver and pattern matching capabilities. However, Z3 is a well-established SMT solver which is widely used in many projects and known for producing reliable results. Democles is actually (since mid of 2015) the main pattern matching engine used in the eMoflon tool and has reached during this time an adequate degree of reliability.

10

RELATED WORK

In the following, we compare the new concepts presented in this thesis with other existing approaches.

10.1 Transformation of Attributed Graph Structures

We begin with comparing projective graph transformation with other existing concepts for the transformation of attributed graph structures. As there are various attribution concepts for graph transformation we focus on those approaches that have a formal foundation, which are mainly those that are based on the algebraic double pushout approach.

Symbolic and lazy graph transformations. Symbolic graphs were first introduced in [Ore08] to define attributed graph constraints. Subsequently these results were extended to symbolic graph transformation in [OL10b]. As mentioned in Chapter 3, transformations via symbolic productions enjoy the properties of adhesive transformation systems. However, as discussed in Section 3.5 transformations via symbolic productions are improper for transforming nongrounded symbolic graphs. In [OL12] lazy graph transformation is proposed to overcome these limitations. Similar, to projective productions the left-hand side morphism of a lazy production has to be in $\mathcal{M}^{bij}_{\Leftrightarrow}$, whereas the right-hand morphism of a lazy production is only required to be in $\mathcal{M}^{inj}_{\Rightarrow}$ (note, we require the right-hand side morphism to be in $\mathcal{M}^{inj}_{Proj}$ in case of projective productions, and $r \in \mathcal{M}^{inj}_{Func}$ in case of functional projective productions). Hence, lazy graph productions provide more expressive power than projective and functional projective productions; that is, a transformation via a lazy graph production may further constrain the values of existing label nodes, whereas in case of projective and functional projective graph transformations only the values of created label nodes may be constrained. Unfortunately, this gain of expressiveness leads to the loss of the HLR properties for lazy graph productions, which build the basis for proving the correctness for consistency enforcement and conflict analysis techniques. Accordingly, projective and functional projective transformation systems can be considered as a compromise between symbolic and lazy graph transformation; that is, (functional) projective graph productions are not only restricted to transformations of grounded symbolic graphs (such as symbolic graph productions), but still retain the properties required to apply the existing results for constraint enforcement and conflict analysis techniques.

Transformation approaches based on algebra attributed graphs. Basically, we can distinguish between two algebra based approaches for graph attribution. In

[LKW93] both the graph structure and the attributes are coded as an algebra. In [HKT02] the algebra is embedded into the graph. More specifically, an algebra attributed graph is seen as a pair formed by an E-graph and an algebra, to define values for the label nodes. In [Ehr03] it is shown that both approaches are equivalent, up to a certain point. In [EPT04] it was shown that algebra attributed graphs fit into the framework of adhesive high-level replacement (HLR) systems; thus providing a formal foundation for graph transformation including all its basic results. However, although this representation is theoretically satisfactory, including the algebra in the graph structure leads to potentially infinite graphs. This is especially problematic with respect to an implementation, as the theoretical results can not be directly transferred to an implementation, as real systems rely on finite data structures in general. In [OL10b] it is shown that every algebra attributed graph can be coded as a symbolic graph, whereas the converse is not true. Hence, symbolic graph transformations and, consequently, also (functional) projective graph transformations, are expressively more powerful than algebra attributed graph transformations.

Transformation approaches based on partially labeled Graphs. Basically, a partially labeled graph is a graph together with a partial label function to assign labels from a label alphabet to the graph nodes (and edges). In contrast to algebra and symbolic attributed graphs the labels are not coded in the graph structure, which eliminates the need for infinite graphs. Moreover, attributed graphs based on partially labeled graphs circumvent some inconveniences of algebra and symbolic attributed graphs. For example, every node has by definition at most one value for each attribute. In case of algebra and symbolic attributed graphs this can only be achieved by additional negative constraints. Also the typing concept of attributed graphs based on partially labeled graphs is more elegant compared with algebra and symbolic attributed graphs. For algebra and symbolic attributed graphs label nodes are typed twice, i. e., by the type graph and the signature of the algebra. In case of partially labeled graphs the typing can de done at either level; for example, untyped graphs with typed attributes may be defined [PH15].

In [HP12a] a category for attributed graphs based on partially labeled graphs is proposed; it is shown that the category is $(\mathcal{M}, \mathcal{N})$-adhesive. Thus, all results obtained for $(\mathcal{M}, \mathcal{N})$-adhesive transformations systems directly apply to this approach. However, although this approach enjoys the aforementioned advantages of partially labeled graph, it is currently limited to simple replacement of attribute values and does not support computations on attribute values [PH15].

In [Gol12] a general attribution concept for $\mathcal{M}$-adhesive transformation systems is presented. In [PH15] it was shown that this concept is related to $(\mathcal{M}, \mathcal{N})$-adhesive transformation systems. Similar to the approach presented in [HP12a, PH15] only the replacement of attribute values is currently supported.

In [Plu09] *graph programs* are proposed, which provide an other attribution concept based on partially labeled graphs. In contrast to the approach presented in [PH15], graph programs allow for computations on attribute values. The main drawback of this approach is that it does not fit into the framework of adhesive transformation systems. Hence, all results that are direct consequences from the HLR-properties, need to be verified separately.

10.2 Verification of consistency constraints

In the following, we relate our results with respect to other existing approaches for verifying consistency constraints.

The construction of application conditions from constraints was initially introduced for plain graphs in [HW95]. Subsequently the approach was generalized to high-level structures within the framework of $\mathcal{M}$-adhesive categories in [EEHP06]. To this end, the underlying category has to provide (in addition to the HLR properties) some extra properties referred to as HLR$^+$properties.

In [DV14] we have shown that the symbolic graph transformation systems (originally introduced in [OL10b]) provide these HLR$^+$properties; thus, the results for constructing equivalent precondition application conditions from graph constraints obtained in the context of high-level transformation system [EEHP06] also apply for symbolic graph constraints and transformations via symbolic graph productions. Compared with the notion of functional projective productions presented in this thesis, symbolic graph productions are expressively less powerful, as every symbolic production is also a functional projective production, but not vice versa. However, from a practical point of view, this increase in expressive power is very small. Nevertheless, in contrast to symbolic graph productions, functional projective productions are suitable for transforming nongrounded symbolic graphs; thus the underlying computational model is more close to real implementations.

As mentioned in the previous section transformations via lazy graph productions provide more expressive power than transformation via projective and functional projective productions. However, as shown in Chapter 7, projective productions fail to provide the required properties to transform post- into equivalent precondition application conditions. As a direct consequence, also lazy productions fail to provide these properties.

In [EEPT06] it was shown that algebra attributed graph transformation systems provide the HLR$^+$properties required for transforming constraints to equivalent application conditions. As mentioned in the previous section the approach comes with some technical difficulties that arise from the conceptual complexity of combining graphs with algebras. As shown in [Ore08] algebra attributed graph constraints provide less expressive power than symbolic graph constraints.

The construction of precondition application conditions from graph constraints in the framework of partially labeled graphs and graph transformation systems was shown [HP12a].

For graph programs the construction of precondition application conditions from graph constraints was proven in [PP12]. In [PP14] graph constraints are extended to make them equivalently expressive to monadic second-order logic on graphs; a construction for preconditions for these assertions is provided, too. As mentioned in the previous chapter, graph programs do not fit into the framework of adhesive transformation systems. Hence, all constructions were verified separately. Moreover, other results obtained in the framework high-level transformations systems (e. g., for conflict analysis) cannot directly be transferred to graph programs.

An approach that also considers constraints with an expressive power equivalent to second order monadic logic is presented in [HR10]. In contrast to graph programs this approach is defined for adhesive transformation systems; thus, these results may be transferred to (functional) projective graph transformation systems with reasonable efforts.

In [AHRT14] and [RAB^{+}15] the transformation of OCL constraints into graph constraints is considered. As these results are defined for adhesive transformation systems, they likely apply also to functional projective transformations system;

10.3 Conflict Detection and Resolution for Attributed Graph Transformations

The only related approaches that provenly provide the properties to perform conflict analysis are symbolic graph transformation systems and algebra attributed graph transformation system.

From a theoretical perspective, all results presented in Chapter 8 are also valid for symbolic graph productions, as every symbolic graph production is also a projective production. However, from a practical point of view, performing confluence analysis with symbolic productions does not lead to satisfactory results. Recall that symbolic productions are in general not useful when applied to nongrounded symbolic graphs. However, for confluence analysis we have to transform minimal contexts, which are generally nongrounded. Lazy symbolic graph productions [OL12] circumvent these shortcomings by allowing the creation of label nodes. Moreover, transformation via lazy productions may further constrain the values of existing label nodes. Unfortunately, this gain in expressive power leads to the loss of the locality property; that is, further constraining the values of existing label nodes potentially affects the values of label nodes that are not in the match of the production. This problem is illustrated in Figure 10.1. The lazy production shown on top of Figure 10.1 further restricts the value of existing label node x by adding constraint (x = 5). Thus, the production is not a projective production. The direct transformation derived by applying this production to a symbolic graph $\langle G, \Phi_G \rangle$ is shown on the bottom of Figure 10.1. In addition to a label node x the graph G contains an other label node y, whose value is given by (x = y). Applying the production to symbolic graph $\langle G, \Phi_G \rangle$ results in symbolic graph $\langle H, \Phi_H \rangle$ with $\Phi_H \Leftrightarrow (x = y) \wedge (x = 5)$. Hence, in addition to setting te value of x equal to 5, the application implicitly sets also the value of y equal to 5, although label node y was not in the match. Without this locality property, the Embedding and Extension

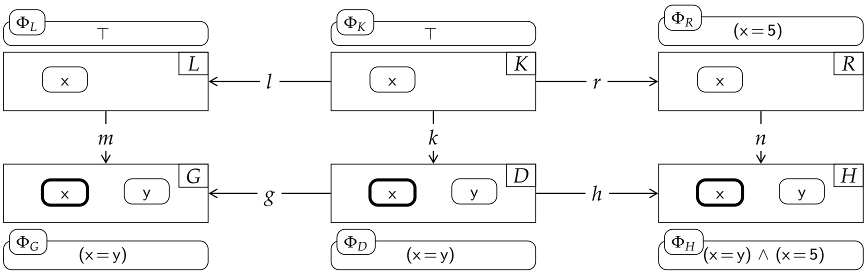


Figure 10.1: Example of a transformation via a lazy production which has nonlocal effects.

Theorems (Theorem 8.13 and Theorem 8.14) as well as the completeness lemma for symbolic critical pairs (Lemma 8.16) are not valid anymore, as they are based on the assumption that transformations have only local effects.

The results for conflict detection and resolution are also valid for algebra attributed graphs [EEPT06]. However, the results for conflict detection do not apply for arbitrary algebra attributed graph productions; that is, only productions, where the left-hand sides are attributed by merely variables are permitted. However, this further restricts the expressive power of algebra attributed graph productions.

Local confluence for productions with negative application conditions in the context of adhesive transformation systems is studied in [LEO06, LEPO08]. The results were generalized to nested application conditions in [EGH^{+}12]. As these results are formalized within the framework of adhesive transformation systems, we are optimistic that it is possible to provide similar proofs for functional projective transformation systems.

11

CONCLUSIONS

Graph transformation with its formal foundations and its broad range of theoretical results constitutes an effective framework for the specification, analysis, and verification of software systems. Nevertheless, almost all realistic systems for which graph-based modeling is appropriate incorporate primitive data in terms of attributes. Although there exists a broad spectrum of theoretical results for consistency enforcement and conflict analysis techniques, there is currently rather limited tool support for these techniques with respect to attributed graph structures. Accordingly, the main objective of this thesis was to close this gap by developing a formal framework for the static verification of attributed graph transformation systems, with the aim to deliver an implementation. We identified the need for potentially infinite graphs as the main obstacle that prevents an implementation. As for an implementation the underlying data structures need to be finite, it is rather difficult to argue that an implementation of the developed verification techniques preserves the properties of the related theory. More concretely, it is by no means trivial to show that an implementation with finite representations of infinite graph structures preserves the correctness and completeness proofs of its underlying graph transformation theory.

Accordingly, The main contributions of this thesis are (i) the development of a formal framework for attributed graph transformations that does not rely on infinite graphs, (ii) the proofs of the results for consistency enforcement and conflict analysis in this framework, and (iii) the realization of the developed theoretical concepts leading to the Symbolic Graph Analysis and Verification (SyGrAV) tool prototype. Moreover, we assessed the practical applicability of the theoretical results by evaluating the tool prototype by means of a case study.

In the following, we first summarize the contributions of this thesis in detail in Section 11.1. Subsequently, we discuss in Section 11.2 the practical relevance of our findings. This chapter concludes with providing directions for future improvements and research (Section 11.3).

11.1 Contributions

The basis for all contributions provided in this thesis, was to identify the classes of projection and functional projection morphisms. These morphism classes provide the basis to formalize attributed graph transformation systems without the need for potentially infinite graph data structures. At the same time, these morphisms classes retain the characteristics required for consistency enforcement and conflict analysis techniques.

Based on these morphism classes we developed a formal framework for attributed graph transformation systems including the following contributions:

$(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive categories and transformation systems. To prove the fundamental results of the double pushout approach for projective graph transformation systems we introduced the new concept of $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive categories and transformation systems in Chapter 5. Moreover, we have shown that basic results obtained for HLR-categories are also valid for $(\mathcal{L},\mathcal{R},\mathcal{N})$-adhesive categories.

Enforcing Symbolic Graph Constraints. Chapter 7 provides the proofs of the additional properties required to construct equivalent application conditions from symbolic graph constraints. Moreover, we have shown that the construction of equivalent precondition application conditions is valid for functional projective transformation rules. Finally, we provided minimization procedures to reduce the number of generated negative application conditions. Our experiments have shown that most of the generated application condition are not necessary to preserve consistency. Accordingly, they are removed by the minimization procedures. Hence, the minimization procedures are important from a practical point of view to not degrade the performance of applying a production by hundreds of unnecessary application condition checks.

Conflict Detection and Resolution. Chapter 8 provides the proofs required to transfer the well-known results for conflict detection (by parallel dependence analysis) and conflict resolution (by local confluence analysis) to projective graph transformation systems. It turned out that the standard local confluence analysis approach, which performs well for graph transformation without attributes, does not lead to the desired results when applied to projective graph transformation systems. This problem was solved by introducing *local confluence modulo normal form equivalence*. Additionally, we have shown tat the results of the Local Confluence Theorem remain valid for local confluence modulo normal form equivalence.

Tool support and evaluation We implemented all theoretical results obtained in this thesis leading to the Symbolic Graph Analysis and Verification framework. In Chapter 9 we used SyGrAV to conduct experiments on a case study from the enterprise modeling domain. The measured run times obtained during our experiments are quite promising. However, to finally assess the significance of the measurements with respect to larger scenarios, further experiments have to be conducted.

11.2 Practical Relevance

An other important metric regarding the practical applicability of our approach is its expressive power. In general, there is always a trade-off between the expressive power of a language and the properties that can be verified. Accordingly, it was necessary to impose certain restrictions on the allowed attribute expression in order enable consistency enforcement and conflict analysis techniques. These restrictions led to the notions of projective and functional projective graph transformations. In the following, we discuss the implications of these restrictions concerning the realization of static verification support for current state of the art graph transformation tools. To this end, we analyzed the permissible attribute conditions in the graph transformation tools Henshin [ABJ+10], Viatra2 [BDH+15], GrGen.net [GBG+06], and eMoflon [LAS14]. It turned out that all tools permit attribute expression comparable to the expressive power of first-order logic without quantifiers. Moreover, all tools require new attribute values to be defined in terms of functions by the existing attribute values, which matches exactly the restriction imposed by functional projective transformation rules. Consequently, the SyGrAV tool prototype can be considered as a major step towards extending current state of the art graph transformation tools with static verification capabilities. The term "major step" is owed to the fact that the considered graph transformation tools provide further language features such as inheritance and amalgamation, which are in fact orthogonal to attribution but currently not supported by SyGrAV. This leads us to possible direction for future work.

11.3 Future Directions

As mentioned the SyGrAV tool prototype can be considered as a major step towards offering support for static verification for current state of the art graph transformation tools. Nevertheless, additional language features need to be integrated to achieve a comprehensive support.

An important language extension are application conditions. Local confluence analysis for graph transformation systems with negative application conditions was first presented in [LEO06, LEPO08] in the context of adhesive transformation systems. The results were generalized to nested application conditions in [EGH^{+}12]. Hence, to establish support for conflict analysis with application conditions the formal requirements to apply these techniques need to be verified. Towards an implementation of these techniques the main challenge is to establish capabilities for reasoning over application conditions. This problem is undecidable for arbitrary nested application conditions, but decidable for a certain subset including propositional expressions over negative and positive application conditions [Pen08]. Hence, it should be possible (with reasonable efforts) to provide an implementation for conflict analysis with negative and positive application conditions.

Another concept is type inheritance, which is especially important for the object-oriented approach to metamodeling. We are optimistic that the results for consistency enforcement presented in [TR05] as well as the results for conflict analysis presented in [GLEO12] for algebra attributed graphs with type inheritance can be transferred to functional projective graph transformation systems.

Another popular language extension are amalgamated graph transformations, which allows for the definition of transformations whose size is determined at transformation time by means of the actual model characteristics. In this way it is possible to express *for each loops* by graph transformation rules. Recently an approach for conflict analysis and an algorithm for conflict detection for amalgamated graph transformations was proposed [TG15, BT16]. A next step might be to study if (or to which extent) our results can be combined with the results for conflict resolution of amalgamated transformations.

Other future directions aim at improving current tool support with respect to expressive power and performance. Concerning the expressive power of SyGrAV, we plan to integrate support for sequence based datatypes such as strings. Currently only integer and real numbers as well as bitvectors (for finite domain datatypes) and enumerations are supported. Z3 supports sequences since version 4.4.2, including operations for concatenation, comparison as well as predicates on the length of strings. However, in version 4.4.2 the Z3 Java API has some major bugs. Nevertheless, if these problems are fixed it should be possible (with negligible efforts), to provide support for strings, too.

As mentioned before SyGrAV is currently in the prototype stage. Accordingly, we expect that the performance of SyGrAV can be considerably increased by fine tuning several components. Despite this, there are still proposals for optimizing the conflict resolution procedure itself (e. g., [LEO08]). However, it is still an open question whether these optimizations indeed lead to an considerable performance boost when implemented.

BIBLIOGRAPHY

[AA10] Rainer Alt and Gunnar Auth. Campus management system. *Business & Information Systems Engineering*, 2(3):187–190, 2010. (Cited on page 7.)

[ABJ+10] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced concepts and tools for in-place EMF model transformations. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*, volume 6394 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2010. (Cited on page 2 and 184.)

[AHPZ07] Karl Azab, Annegret Habel, Karl-Heinz Penneman, and Christian Zuckschwerdt. ENFORCe: A system for ensuring formal correctness of high-level programs. In *In Graph Based Tools (GraBaTs'06), Electronic Communications of the EASST*, pages 82–93, 2007. (Cited on page 2.)

[AHRT14] Thorsten Arendt, Annegret Habel, Hendrik Radke, and Gabriele Taentzer. From core OCL invariants to nested graph constraints. In Giese and König [GK14], pages 97–112. (Cited on page 180.)

[BCW12] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-driven software engineering in practice*. Morgan & Claypool Publishers, 2012. (Cited on page 1.)

[BDH+15] Gábor Bergmann, István Dávid, Ábel Hegedüs, Ákos Horváth, István Ráth, Zoltán Ujhelyi, and Dániel Varró. Viatra 3: A reactive model transformation platform. In Dimitris S. Kolovos and Manuel Wimmer, editors, *Theory and Practice of Model Transformations - 8th International Conference, ICMT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 20-21, 2015. Proceedings*, volume 9152 of *Lecture Notes in Computer Science*, pages 101–110. Springer, 2015. (Cited on page 2 and 184.)

[BGH+07] Kamal Bhattacharya, Cagdas Evren Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2007. (Cited on page 8.)

[BT16] Kristopher Born and Gabriele Taentzer. An algorithm for the critical pair analysis of amalgamated graph transformations. In Rachid

Echahed and Mark Minas, editors, *Graph Transformation - 9th International Conference, ICGT 2016, in Memory of Hartmut Ehrig, Held as Part of STAF 2016, Vienna, Austria, July 5-6, 2016, Proceedings*, volume 9761 of *Lecture Notes in Computer Science*, pages 118–134. Springer, 2016. (Cited on page 185.)

[CEM$^+$06] Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg, editors. *Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17-23, 2006, Proceedings*, volume 4178 of *Lecture Notes in Computer Science*. Springer, 2006. (Cited on page 190 and 191.)

[Día13] Vicente García Díaz. *Advances and Applications in Model-Driven Engineering*. IGI Global, 2013. (Cited on page 1.)

[DKL$^+$16] Frederik Deckwerth, Géza Kulcsár, Malte Lochau, Gergely Varró, and Andy Schürr. Conflict detection for edits on extended feature models using symbolic graph transformation. In Julia Rubin and Thomas Thüm, editors, *Proceedings 7th International Workshop on Formal Methods and Analysis in Software Product Line Engineering, FMSPLE@ETAPS 2016, Eindhoven, The Netherlands, April 3, 2016.*, volume 206 of *EPTCS*, pages 17–31, 2016. (Cited on page 3.)

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. (Cited on page 160.)

[DV14] Frederik Deckwerth and Gergely Varró. Attribute handling for generating preconditions from graph constraints. In Giese and König [GK14], pages 81–96. (Cited on page 51, 119, and 179.)

[EEHP06] Hartmut Ehrig, Karsten Ehrig, Annegret Habel, and Karl-Heinz Pennemann. Theory of constraints and application conditions: From graphs to high-level structures. *Fundam. Inform.*, 74(1):135–166, 2006. (Cited on page 28, 36, 119, and 179.)

[EEKR99] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation: Vol. 2: Applications, Languages, and Tools*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999. (Cited on page 132.)

[EEKR12] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Graph Transformations - 6th International Conference, ICGT 2012, Bremen, Germany, September 24-29, 2012. Proceedings*, volume 7562 of *Lecture Notes in Computer Science*. Springer, 2012. (Cited on page 191.)

[EEPR04] Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors. *Graph Transformations, Second International Conference, ICGT 2004, Rome, Italy, September 28 - October 2, 2004, Proceedings*, volume 3256 of *Lecture Notes in Computer Science*. Springer, 2004. (Cited on page 190.)

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2006. (Cited on page 12, 23, 46, 75, 87, 96, 97, 98, 117, 153, 179, and 181.)

[EFT94] H.-D. Ebbinghaus, J. Flum, and Wolfgang Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer, 1994. (Cited on page 38 and 44.)

[EGH10] Hartmut Ehrig, Ulrike Golas, and Frank Hermann. Categorical frameworks for graph transformation and HLR systems based on the DPO approach. *Bulletin of the EATCS*, 102:111–121, 2010. (Cited on page 31.)

[EGH$^+$12] 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. *Fundam. Inform.*, 118(1-2):35–63, 2012. (Cited on page 87, 181, and 185.)

[EGH$^+$14] 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. *Mathematical Structures in Computer Science*, 24(4), 2014. (Cited on page 161 and 162.)

[EGSW07] Gregor Engels, Baris Güldali, Christian Soltenborn, and Heike Wehrheim. Assuring consistency of business process models and web services using visual contracts. In Andy Schürr, Manfred Nagl, and Albert Zündorf, editors, *Applications of Graph Transformations with Industrial Relevance, Third International Symposium, AGTIVE 2007, Kassel, Germany, October 10-12, 2007, Revised Selected and Invited Papers*, volume 5088 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2007. (Cited on page 1.)

[EH86] Hartmut Ehrig and Annegret Habel. *Graph grammars with application conditions*. Springer, 1986. (Cited on page 36.)

[EHKP90] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, and Francesco Parisi-Presicce. From graph grammars to high level replacement systems. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science, 4th International Workshop, Bremen, Germany, March 5-9, 1990, Proceedings*, volume 532 of *Lecture Notes in Computer Science*, pages 269–291. Springer, 1990. (Cited on page 31.)

[EHPP04] Hartmut Ehrig, Annegret Habel, Julia Padberg, and Ulrike Prange. Adhesive high-level replacement categories and systems. In Ehrig et al. [EEPR04], pages 144–160. (Cited on page 31.)

[Ehr03] Hartmut Ehrig. *Attributed graphs and typing: Relationship between different representations*. Techn. Univ. Berlin, Fakultät IV, Elektrotechnik und Informatik, 2003. (Cited on page 178.)

[EHRT08] Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, and Gabriele Taentzer, editors. *Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings*, volume 5214 of *Lecture Notes in Computer Science*. Springer, 2008. (Cited on page 192 and 193.)

[EPS73] Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph-grammars: An algebraic approach. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 167–180. IEEE Computer Society, 1973. (Cited on page 31.)

[EPT04] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental theory for typed attributed graph transformation. In Ehrig et al. [EEPR04], pages 161–177. (Cited on page 178.)

[ERRS10] Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors. *Graph Transformations - 5th International Conference, ICGT 2010, Enschede, The Netherlands, September 27 - - October 2, 2010. Proceedings*, volume 6372 of *Lecture Notes in Computer Science*. Springer, 2010. (Cited on page 192.)

[FG98] Mark S. Fox and Michael Grüninger. Enterprise modeling. *AI Magazine*, 19(3):109–121, 1998. (Cited on page 1.)

[Fow02] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002. (Cited on page 7.)

[Gal85] Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row Publishers, Inc., 1985. (Cited on page 38.)

[GBEG14] Karsten Gabriel, Benjamin Braatz, Hartmut Ehrig, and Ulrike Golas. Finitary -adhesive categories. *Mathematical Structures in Computer Science*, 24(4), 2014. (Cited on page 73 and 80.)

[GBG$^+$06] Rubino Geiß, Gernot Veit Batz, Daniel Grund, Sebastian Hack, and Adam Szalkowski. GrGen: A fast spo-based graph rewriting tool. In Corradini et al. [CEM$^+$06], pages 383–397. (Cited on page 2 and 184.)

[GK14] Holger Giese and Barbara König, editors. *Graph Transformation - 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22-24, 2014. Proceedings*, volume 8571 of *Lecture Notes in Computer Science*. Springer, 2014. (Cited on page 187, 188, and 193.)

[GLEO12] Ulrike Golas, Leen Lambers, Hartmut Ehrig, and Fernando Orejas. Attributed graph transformation with inheritance: Efficient conflict detection and local confluence analysis using abstract critical pairs. *Theor. Comput. Sci.*, 424:46–68, 2012. (Cited on page 185.)

[Gol12] Ulrike Golas. A general attribution concept for models in $\mathcal{M}$-adhesive transformation systems. In Ehrig et al. [EEKR12], pages 187–202. (Cited on page 178.)

[HKT02] Reiko Heckel, Jochen Küster, and Gabriele Taentzer. Towards automatic translation of UML models into semabtic domains. In *APPLIGRAPH Workshop on Applied Graph Transformation*, pages 11–22, 2002. (Cited on page 2 and 178.)

[HP12a] Annegret Habel and Detlef Plump. $\mathcal{M}, \mathcal{N}$-adhesive transformation systems. In Ehrig et al. [EEKR12], pages 218–233. (Cited on page 31, 33, 73, 80, 87, 178, and 179.)

[HP12b] Annegret Habel and Detlef Plump. $\mathcal{M}, \mathcal{N}$-adhesive transformation systems (long version). `http://formale-sprachen.informatik.uni-oldenburg.de/pub/index.html`, 2012. (Cited on page 93.)

[HR10] Annegret Habel and Hendrik Radke. Expressiveness of graph conditions with variables. *ECEASST*, 30, 2010. (Cited on page 179.)

[HW95] Reiko Heckel and Annika Wagner. Ensuring consistency of conditional graph rewriting - a constructive approach. *Electr. Notes Theor. Comput. Sci.*, 2:118–126, 1995. (Cited on page 87, 119, and 179.)

[KDL+15] Géza Kulcsár, Frederik Deckwerth, Malte Lochau, Gergely Varró, and Andy Schürr. Improved conflict detection for graph transformation with attributes. In Arend Rensink and Eduardo Zambon, editors, *Proceedings Graphs as Models, GaM 2015, London, UK, 11-12 April 2015.*, volume 181 of *EPTCS*, pages 97–112, 2015. (Cited on page 3 and 131.)

[LAS14] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Developing emoflon with emoflon. In Davide Di Ruscio and Dániel Varró, editors, *Theory and Practice of Model Transformations - 7th International Conference, ICMT 2014, Held as Part of STAF 2014, York, UK, July 21-22, 2014. Proceedings*, volume 8568 of *Lecture Notes in Computer Science*, pages 138–145. Springer, 2014. (Cited on page 2, 159, and 185.)

[LEO06] Leen Lambers, Hartmut Ehrig, and Fernando Orejas. Conflict detection for graph transformation with negative application conditions. In Corradini et al. [CEM+06], pages 61–76. (Cited on page 181 and 185.)

[LEO08] Leen Lambers, Hartmut Ehrig, and Fernando Orejas. Efficient conflict detection in graph transformation systems by essential critical pairs. *Electr. Notes Theor. Comput. Sci.*, 211:17–26, 2008. (Cited on page 186.)

[LEPO08] Leen Lambers, Hartmut Ehrig, Ulrike Prange, and Fernando Orejas. Embedding and confluence of graph transformations with negative application conditions. In Ehrig et al. [EHRT08], pages 162–177. (Cited on page 181 and 185.)

[LKW93] Michael Löwe, Martin Korf, and Annika Wagner. An algebraic framework for the transformation of attributed graphs. In *Term Graph Rewriting: Theory and Practice*, pages 185–199. Jhon Wiley and Sons Ltd., 1993. (Cited on page 178.)

[LS04] Stephen Lack and Pawel Sobocinski. Adhesive categories. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2987 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2004. (Cited on page 31.)

[MS10] Mark Minas and Hans Jürgen Schneider. Graph transformation by computational category theory. In Gregor Engels, Claus Lewerentz, Wilhelm Schäfer, Andy Schürr, and Bernhard Westfechtel, editors, *Graph Transformations and Model-Driven Engineering - Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*, volume 5765 of *Lecture Notes in Computer Science*, pages 33–58. Springer, 2010. (Cited on page 161.)

[NC03] Anil Nigam and Nathan S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003. (Cited on page 8 and 10.)

[New42] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942. (Cited on page 132.)

[OL10a] Fernando Orejas and Leen Lambers. Delaying constraint solving in symbolic graph transformation. In Ehrig et al. [ERRS10], pages 43–58. (Cited on page 59.)

[OL10b] Fernando Orejas and Leen Lambers. Symbolic attributed graphs for attributed graph transformation. *ECEASST*, 30, 2010. (Cited on page 3, 45, 47, 48, 49, 50, 51, 53, 109, 177, 178, and 179.)

[OL12] Fernando Orejas and Leen Lambers. Lazy graph transformation. *Fundam. Inform.*, 118(1-2):65–96, 2012. (Cited on page 3, 177, and 180.)

[Ore08] Fernando Orejas. Attributed graph constraints. In Ehrig et al. [EHRT08], pages 274–288. (Cited on page 177 and 179.)

[Pen08] Karl-Heinz Pennemann. An algorithm for approximating the satisfiability problem of high-level conditions. *Electr. Notes Theor. Comput. Sci.*, 213(1):75–94, 2008. (Cited on page 185.)

[PH15] Christoph Peuser and Annegret Habel. Attribution of graphs by composition of $\mathcal{M}, \mathcal{N}$-adhesive categories. In Detlef Plump, editor, *Proceedings of the 6th International Workshop on Graph Computation Models co-located with the 8th International Conference on Graph Transformation (ICGT 2015) part of the Software Technologies: Applications and Foundations (STAF 2015) federation of conferences, L'Aquila, Italy, July 20, 2015.*, volume 1403 of *CEUR Workshop Proceedings*, pages 66–81. CEUR-WS.org, 2015. (Cited on page 33 and 178.)

[Plu93] Detlef Plump. Term graph rewriting; hypergraph rewriting: critical pairs and undecidability of confluence. pages 201–213. John Wiley and Sons Ltd., Chichester, UK, 1993. (Cited on page 146.)

[Plu09] Detlef Plump. The graph programming language GP. In Symeon Bozapalidis and George Rahonis, editors, *Algebraic Informatics, Third International Conference, CAI 2009, Thessaloniki, Greece, May 19-22, 2009, Proceedings*, volume 5725 of *Lecture Notes in Computer Science*, pages 99–122. Springer, 2009. (Cited on page 178.)

[PP12] Christopher M. Poskitt and Detlef Plump. Hoare-style verification of graph programs. *Fundam. Inform.*, 118(1-2):135–175, 2012. (Cited on page 179.)

[PP14] Christopher M. Poskitt and Detlef Plump. Verifying monadic second-order properties of graph programs. In Giese and König [GK14], pages 33–48. (Cited on page 179.)

[PW15] Francesco Parisi-Presicce and Bernhard Westfechtel, editors. *Graph Transformation - 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings*, volume 9151 of *Lecture Notes in Computer Science*. Springer, 2015. (Cited on page 193 and 194.)

[RAB$^+$15] Hendrik Radke, Thorsten Arendt, Jan Steffen Becker, Annegret Habel, and Gabriele Taentzer. Translating essential OCL invariants to nested graph constraints focusing on set operations. In Parisi-Presicce and Westfechtel [PW15], pages 155–170. (Cited on page 180.)

[RET11] Olga Runge, Claudia Ermel, and Gabriele Taentzer. AGG 2.0 - new features for specifying and analyzing algebraic graph transformations. In Andy Schürr, Dániel Varró, and Gergely Varró, editors, *Applications of Graph Transformations with Industrial Relevance - 4th International Symposium, AGTIVE 2011, Budapest, Hungary, October 4-7,* *2011, Revised Selected and Invited Papers*, volume 7233 of *Lecture Notes in Computer Science*, pages 81–88. Springer, 2011. (Cited on page 2.)

[TG15] Gabriele Taentzer and Ulrike Golas. Towards local confluence analysis for amalgamated graph transformation. In Parisi-Presicce and Westfechtel [PW15], pages 69–86. (Cited on page 185.)

[TR05] Gabriele Taentzer and Arend Rensink. Ensuring structural constraints in graph-based models with type inheritance. In Maura Cerioli, editor, *Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3442 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2005. (Cited on page 185.)

[VAS12] Gergely Varró, Anthony Anjorin, and Andy Schürr. Unification of compiled and interpreter-based pattern matching techniques. In Antonio Vallecillo, Juha-Pekka Tolvanen, Ekkart Kindler, Harald Störrle, and Dimitrios S. Kolovos, editors, *Modelling Foundations and Applications - 8th European Conference, ECMFA 2012, Kgs. Lyngby, Denmark, July 2-5, 2012. Proceedings*, volume 7349 of *Lecture Notes in Computer Science*, pages 368–383. Springer, 2012. (Cited on page 160.)

[WVvdA⁺09] Moe Thandar Wynn, H. M. W. Verbeek, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Business process verification - finally a reality! *Business Proc. Manag. Journal*, 15(1):74–92, 2009. (Cited on page 1 and 14.)

CURRICULUM VITAE

Frederik Deckwerth

Education

2012–2016 **Ph.D**, *Real-Time Systems Lab, Data Systems Technology Institute, Eletrical Engineering & Information Engineering, Technische Universität Darmstadt*, Germany.

2004–2012 **Dipl.-Ing.**, *Technische Universität Darmstadt*, Germany.
Diplom-Ingenieur für Elektro- und Informationstechnik.

A

ALL PRODUCTIONS OF THE CMS CASE STUDY

A.1 Production bookRoom

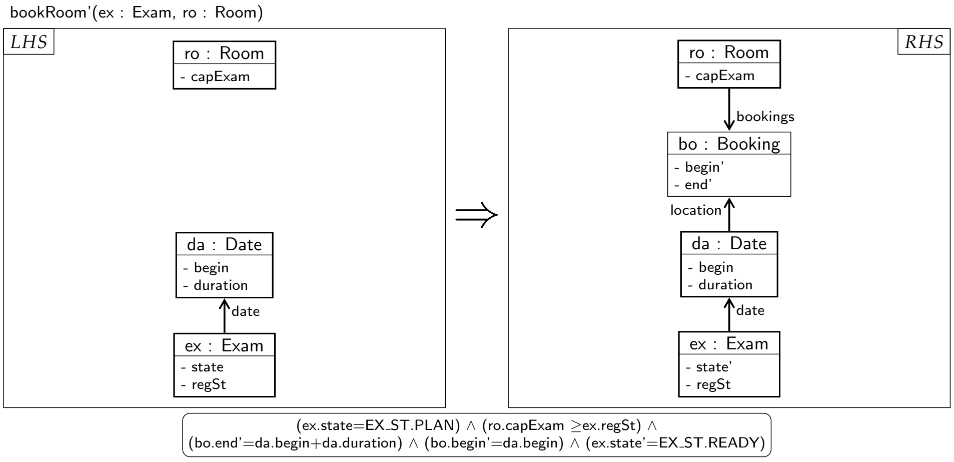


Figure A.1: Production bookRoom(ex : Exam, ro : Room) takes an Exam ex and a Room ro. It is applicable if Exam ex is in the PLAN state, has a Date assigned, and the exam capacity of Room ro is smaller or equal to the number of registered students. The production is applied by creating a Booking bo and assigning it to Room ro. The value of bo.begin is set equal to da.begin. The value of bo.end is set equal to da.begin+da.duration.

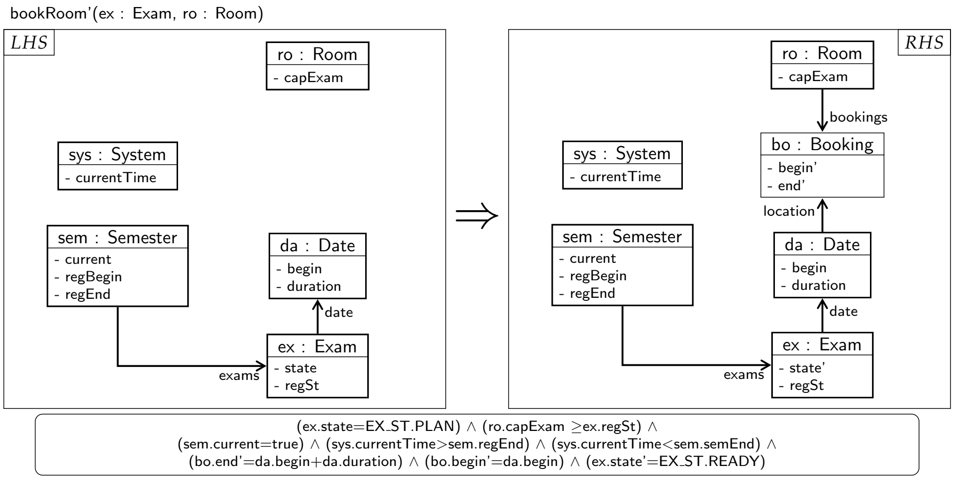


Figure A.2: In contrast to bookRoom, the production bookRoom'(ex : Exam, ro : Room) is only applicable after the registration period of the current Semester sem has ended (sys.currntTime > sem.regEnd), but before sem has ended (sys.currentTime < sem.semEnd).

A.2 Production uploadRes

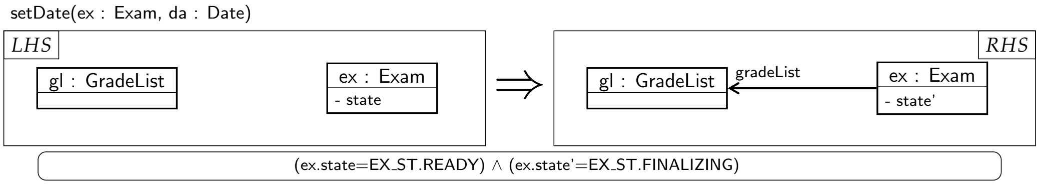

setDate(ex : Exam, da : Date)
LHS
gl : GradeList
ex : Exam
- state
⇒
RHS
gl : GradeList
gradeList
ex : Exam
- state'
(ex.state=EX_ST.READY) ∧ (ex.state'=EX_ST.FINALIZING)

Figure A.3: The Production uploadRes(ex . Exam, da : Date) adds a given grade list to the Exam ex containing the results for the examiniation. The production is applicable if Exam ex is in the READY state. By applying production uploadRes ex.state is changed to FINALIZING

A.3 Production setDate

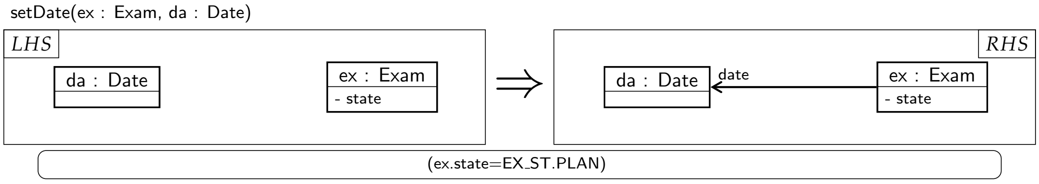


Figure A.4: Production setDate(ex : Exam, da : Date) assigns a Date da to Exam ex.

A.4 Production updateDate

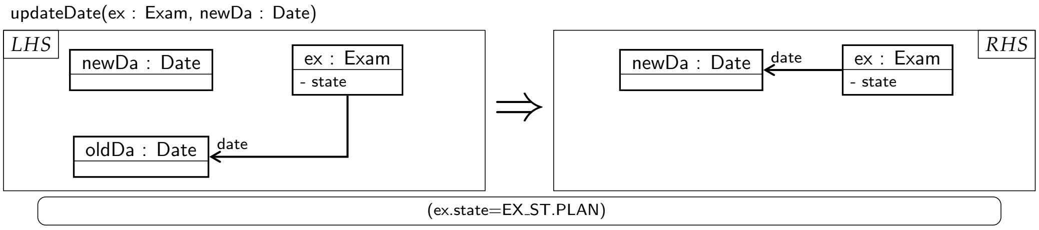


Figure A.5: Production updateDate(ex Exam, newDa : Date) updates the date of an Exam ex.

A.5 Production transResPas

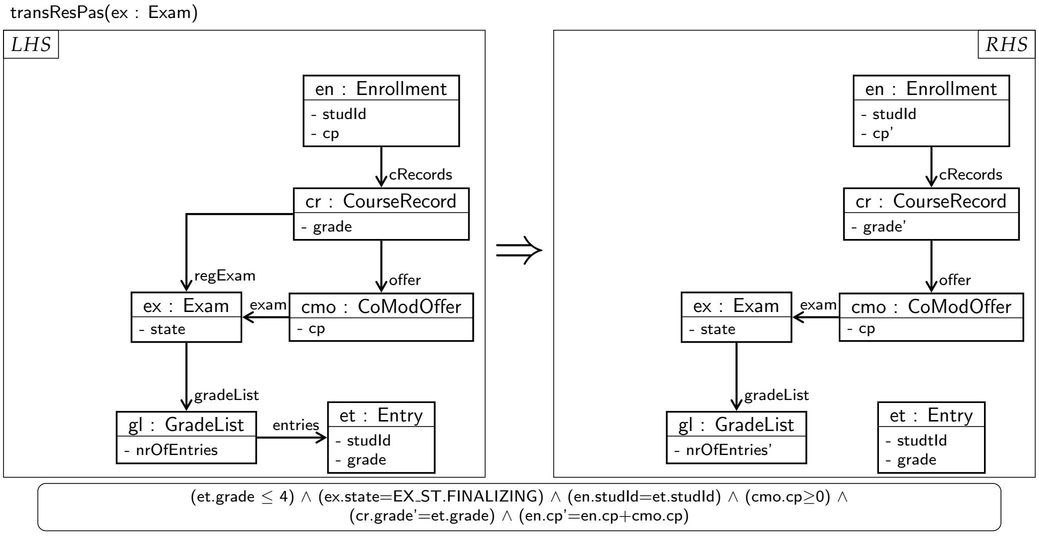
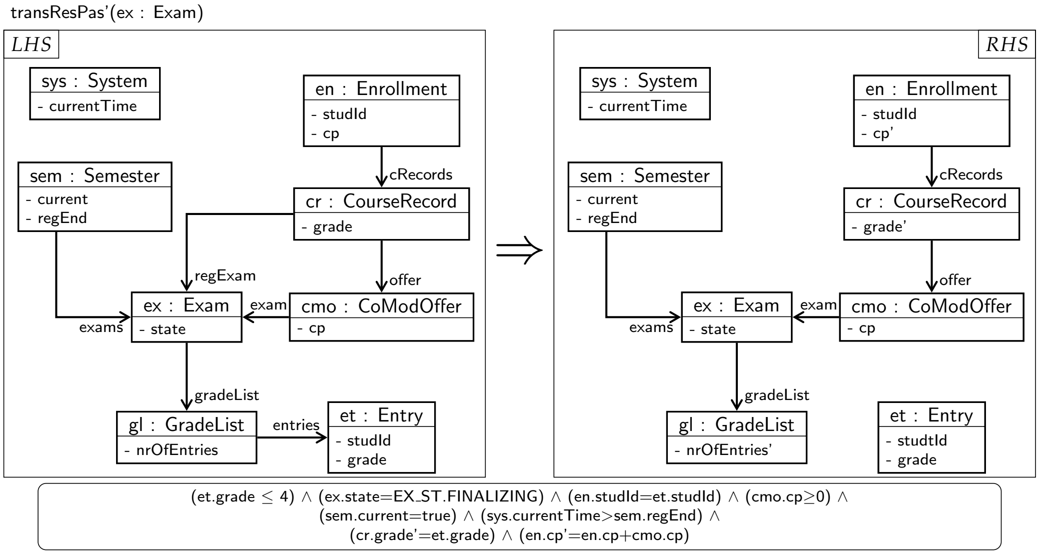


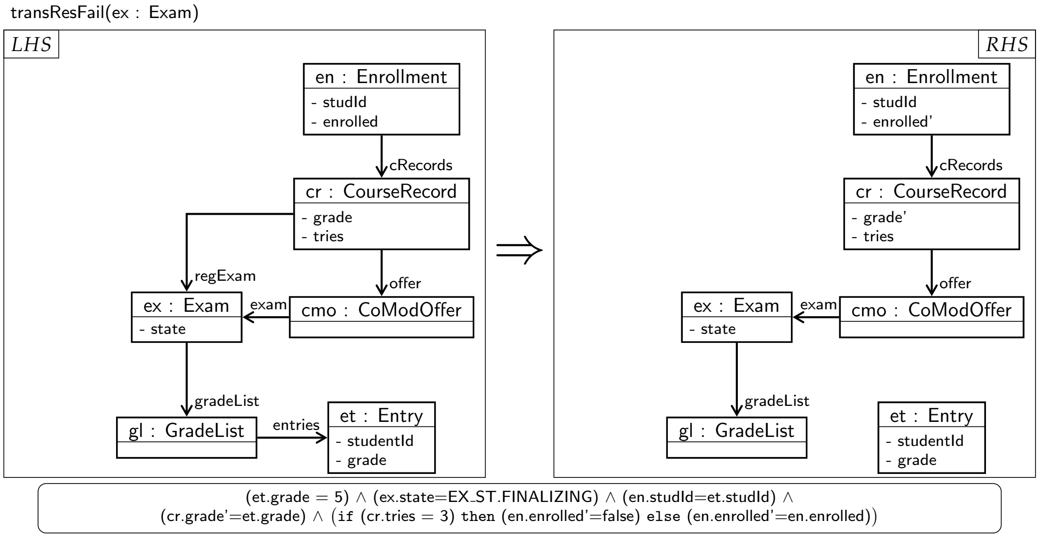
Figure A.6: Production transResPas(ex : Exam) is intended to transfer a result for an exam stored in Entry et to the corresponding CourseRecord cr. The production transfers only results whose grades (et.grade) are smaller or equal 4 (i. e., passed). To this end the production looks up the corresponding Enrollment en such that en.studId=et.studId. The production can only be applied if the Exam ex is in the FIALIZING state. The condition cmo.cp≥0 ensures that the credit point that can be obtained for the course are not negative. The condition is required to guide the solver ensuring that that en.cp'geqen.cp. By applying the production the corresponding the grade stored in the entry is written to the corresponding CourseRecord cr (i. e. cr.grade'=et.grade), and the obtained credit points are incremented by the number of credit points granted for the course (i. e.. en.cp'=en.cp+cmp.cp). Additionally, the link entires is deleted.



$(\text{et.grade} \leq 4) \wedge (\text{ex.state=EX\_ST.FINALIZING}) \wedge (\text{en.studId=et.studId}) \wedge (\text{cmo.cp}\geq 0) \wedge$
$(\text{sem.current=true}) \wedge (\text{sys.currentTime}>\text{sem.regEnd}) \wedge$
$(\text{cr.grade'=et.grade}) \wedge (\text{en.cp'=en.cp+cmo.cp})$

Figure A.7: Production transResPas'(ex : Exam) is similar to production transResPas(ex : Exam) but can only be applied after the registration period of the current semester has ended (i. e., sys.currentTime>sem.regEnd).

A.6 Production transResFail



$(\text{et.grade} = 5) \wedge (\text{ex.state=EX\_ST.FINALIZING}) \wedge (\text{en.studId=et.studId}) \wedge$
$(\text{cr.grade'=et.grade}) \wedge (\texttt{if}\ (\text{cr.tries} = 3)\ \texttt{then}\ (\text{en.enrolled'=false})\ \texttt{else}\ (\text{en.enrolled'=en.enrolled}))$

Figure A.8: Production transResFail(ex :Exam) is similar to production transResPas, but in contrast to transResPas only entries with a grade equal to 5 (i. e., the exam was failed) are transferred. Hence if the number of tries (cr.tries) is equal to 3, the Enrollment en is exmatriculated (i. e., en.enrolled is set to false).

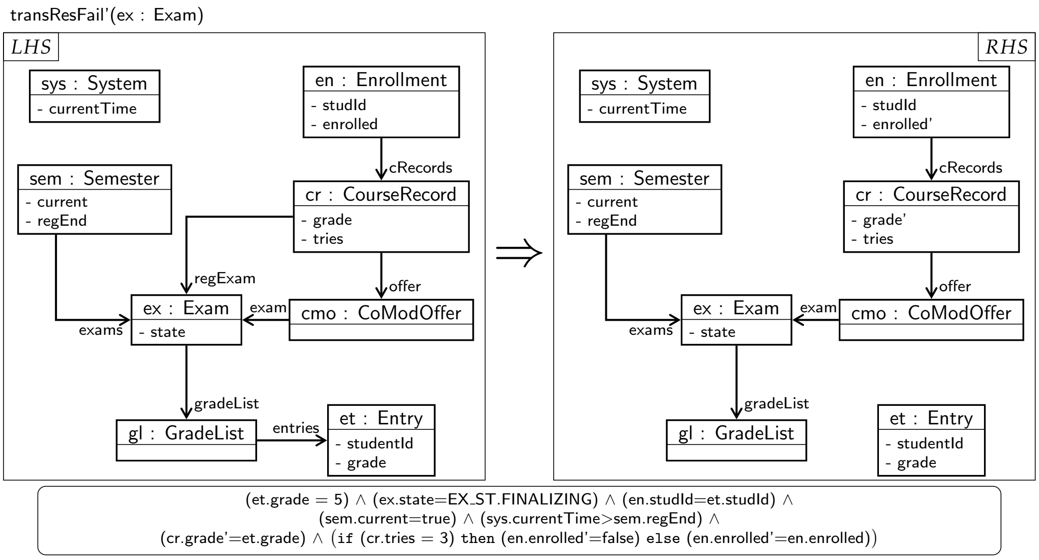


Figure A.9: Production transResFail'(ex : Exam) is similar to production transResFail(ex : Exam) but can only be applied after the registration period of the current semester has ended (i. e., sys.currentTime>sem.regEnd).

A.7 Production closeExam

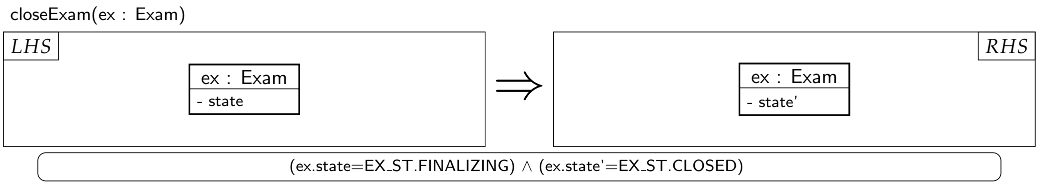


Figure A.10: Production closeExam(ex : Exam) closes an by changing ex.state from FINALIZING to CLOSED.

A.8 Production regExam

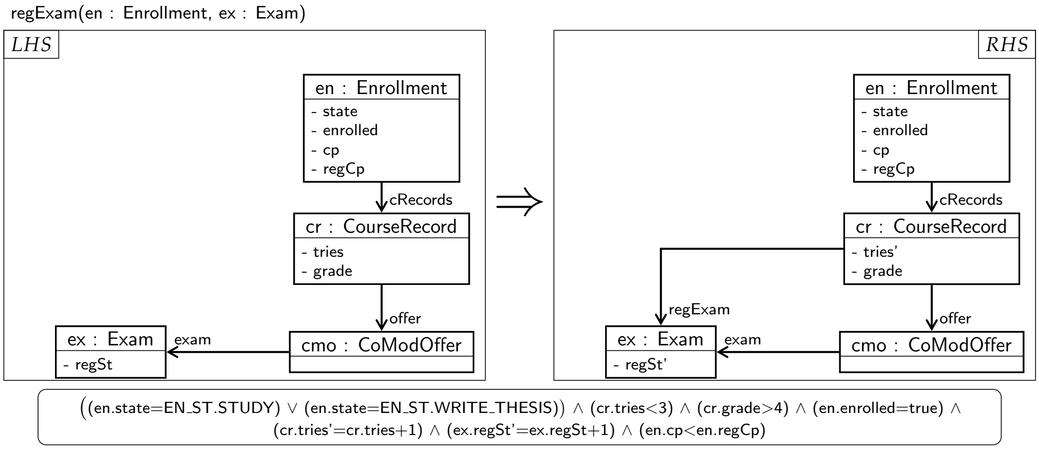


Figure A.11: Production regExam(en : Enrollment, ex : Exam) registers a given Enrollment en to an Exam ex, by creating link regExam from the CourseRecord cr to the examination ex. Note that the production can only be applied if a RourseRecord for the corresponding course module offer (cmo : CoModOffer) exists. Moreover, Enrollment en as to be in the the STUDY or WRITE_THESIS state, the number of tries (cr.tries) has to be lower than 3, and the Exam ex must not be passed before (i. e., cr.grade>4). Additionally the Enrollment en must be enrolled (i. e. en.enrolled=true). By applying the production the number of tries recorded in the course record cr and the number of registered students ex.regSt are incremented by one.

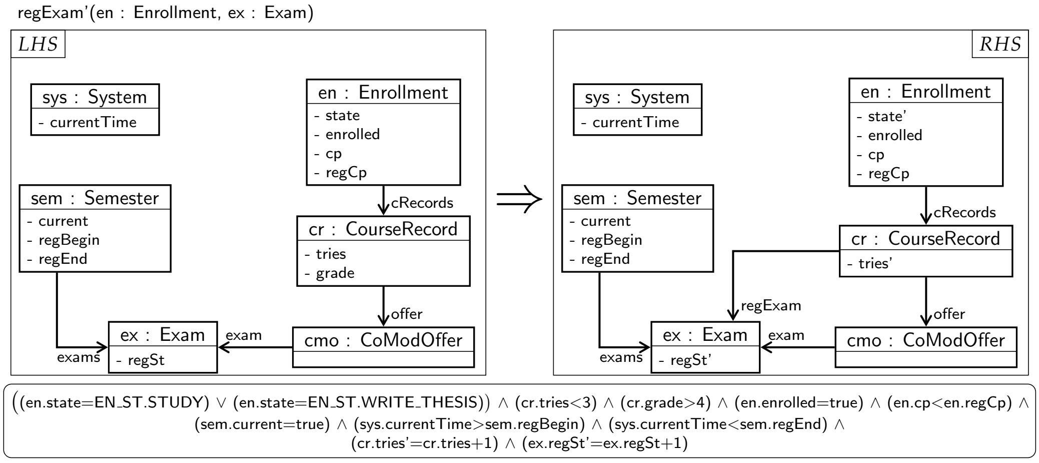


Figure A.12: Production regExam'(en : Enrollment, ex : Exam) requires in addition to production regExam(en : Enrollment, ex : Exam) that the currentTime is between the begin (sem.regBegin) and end (sem.regEnd) of the registration period of the current Semester sem.

A.9 Production regCMO

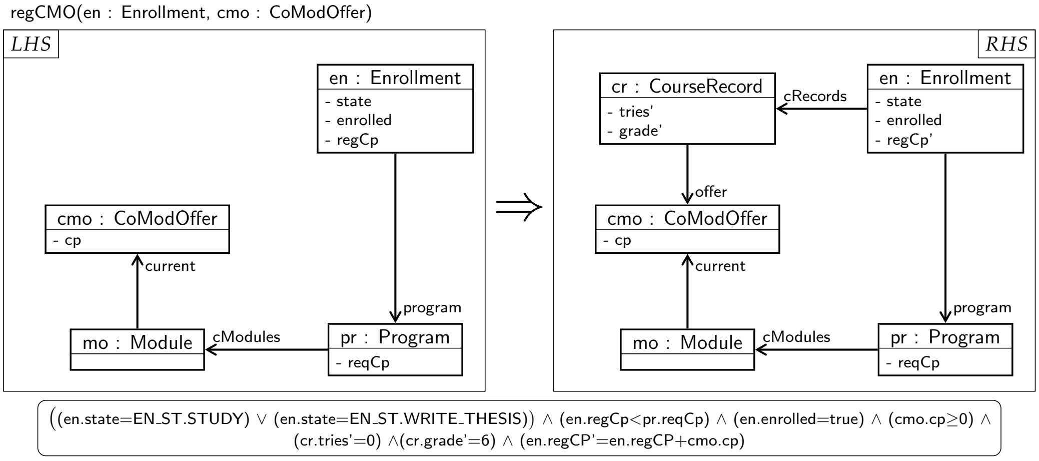


Figure A.13: Production regCMO(en : Enrollment, cmo : CoModOffer) registers Enrollment en to a given course module offer cmo of the Program pr. To this end a CourseRecord cr is crated and assigned by link offer to course module offer cmo. The production is only applicable if the Enrollment en is in the STUDY or WRITE_THESIS state; en.enrolled=true.

A.10 Production unregExam

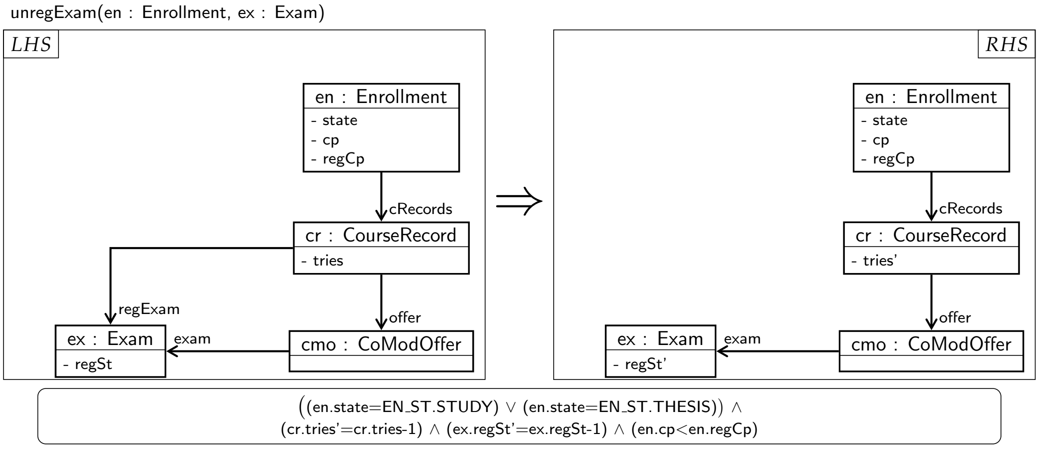


Figure A.14: Production unregExam(en : Enrollment, ex : Exam) takes as input an Enrollment en and an Exam ex. By applying the production, the link regExam from cr : CourseRecord to ex : Ex is removed, as well as the number of tries (cr.tries) and the number of registrations (ex.regSt) are decremented by one.

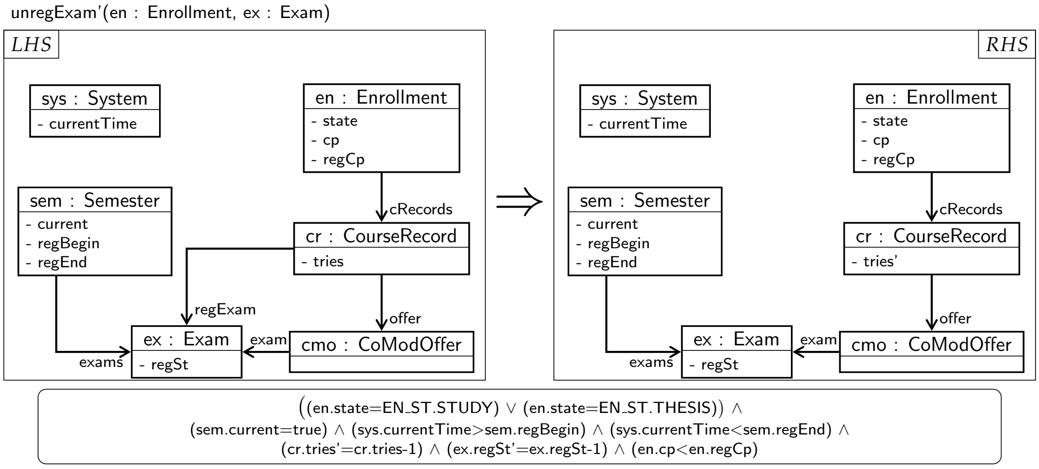


Figure A.15: Production unregExam'(en : Enrollment, ex : Exam) is the corrected version of unregExam Production unregExam' requires in addition to production unregExam that the currentTime is between the begin (sem.regBegin) and end (sem.regEnd) of the registration period of the current Semester sem.

A.11 Production regTMO

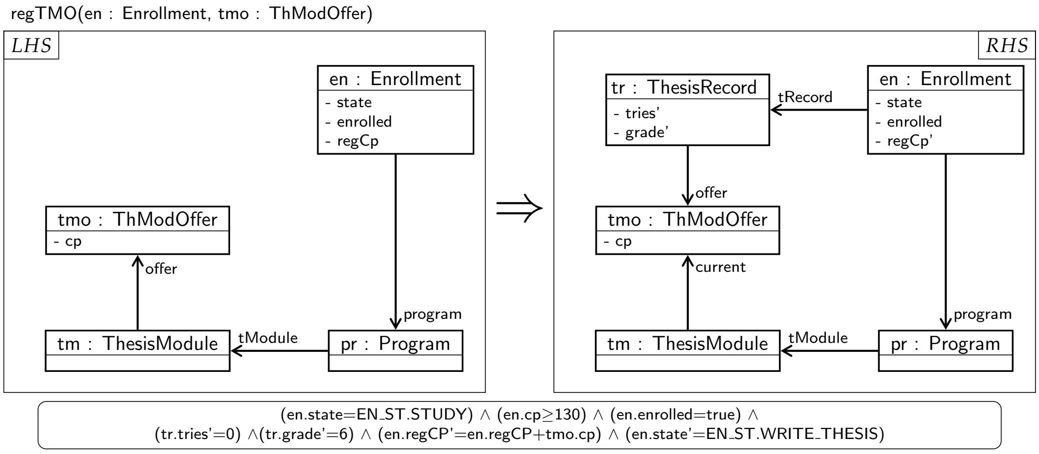


Figure A.16: The production regTMO(en : Enrollment, tmo : ThModOffer) registers an student represented by its enrollment for a thesis module offer (ThModOffer). To this end, the corresponding thesis module offer tmo has to be part of the thesisModule tm of the enrolled Program pr. Moreover, the Enrollment en has to be in the STUDY; the number of archived credit points en.cp must be larger or equal to 130. By registering for a thesis module offer a ThesisRecord tr is created and assigned to Enrollment en and thesis module offer tm. The values for number of tries tr.tries is initialized with value 0; the value for the grade tr.grade is initialized value 6. The number of registered *cp en.regCP* stored in the Enrollment en is incremented by the value of tmo.cp; that is, the number of credit points that can be obtained for a thesis. After applying the production, the Enrollment en is in the WRITE_THESIS state.

A.12 Production regThesis

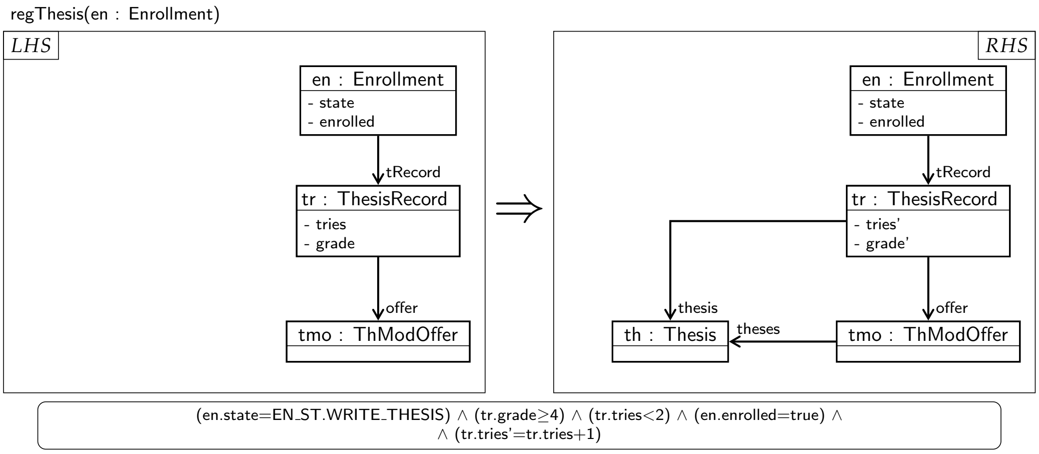


Figure A.17: An Enrollment en can be registered for a Thesis th if it has assigned a ThesisRecord, no already passed thesis exists (tr.grade≥4), the number of tries (tr.tries) must be smaller than two, and the corresponding student is enrolled (en.enrolled=true). By registering for a thesis the number fo tries is incremented by one.

A.13 Production obtDeg

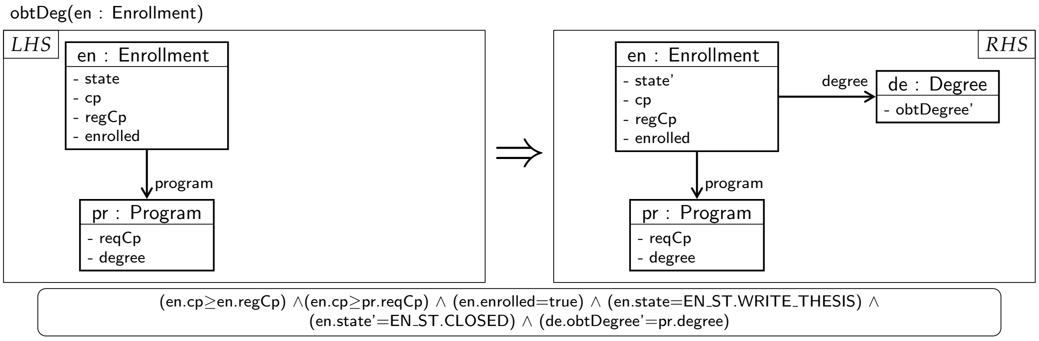


Figure A.18: The production obtDeg assigns a Degree de to and Enrollment en. A degree can only be obtained if number of obtained credit points en.cp are larger or equal to the number of registered credit points en.regCp and the number of registered credit points en.regCp must be larger or equal to the number of credit points pr.reqCp required for the registered Program pr. The enrollment must be valid,i. e., en.enrolled=true and in the WRITE_THESIS state.

A.14 Production setLecture

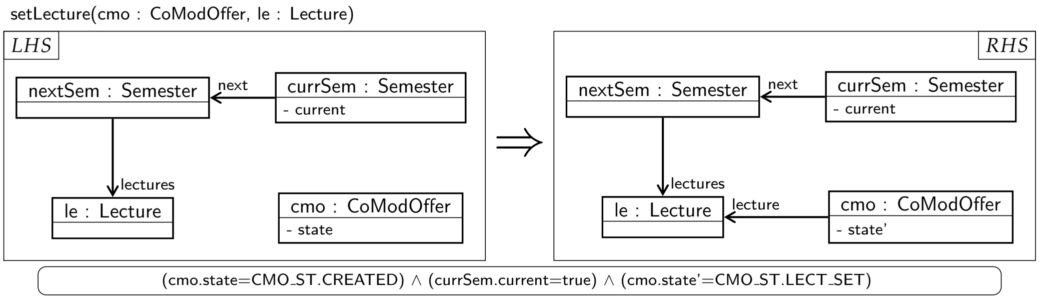


Figure A.19: The production setLecture sets the lecture for a course module offer cmo the first time, i. e., the course module offer was newly CREATED. To this end, a new link lecture is created to the given lecture le.

A.15 Production updateLect

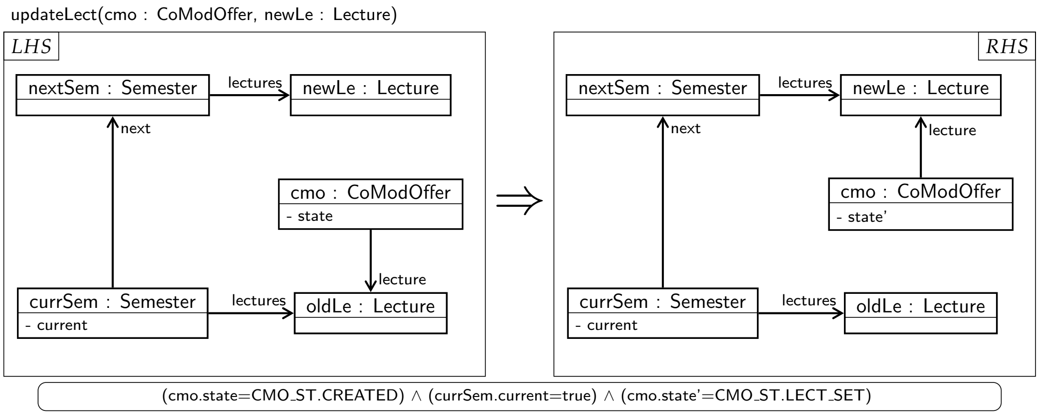


Figure A.20: The production updateLect is intended to update the lecture for an already existing course module offer. To this end, the link lecture is redirected from the old lecture oldLe to a given new lecture newLe that has to be contained in the next semester nextSem. The production is applicable only if the the course module offer is in RESET state. After updating the lecture the course module offer is in the state LECT_UPDATED.

A.16 Production setExam

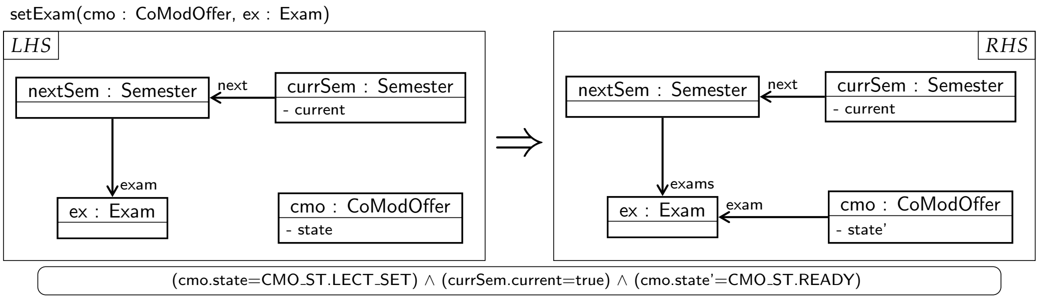


Figure A.21: The production setExam sets a given examination ex for a given course module offer cmo. The production can only be applied if examination ex is in the next semester nextSem and cmo is in state LECT_SET. By applying the production the state of the course module offer is change to READY.

A.17 Production updateEx

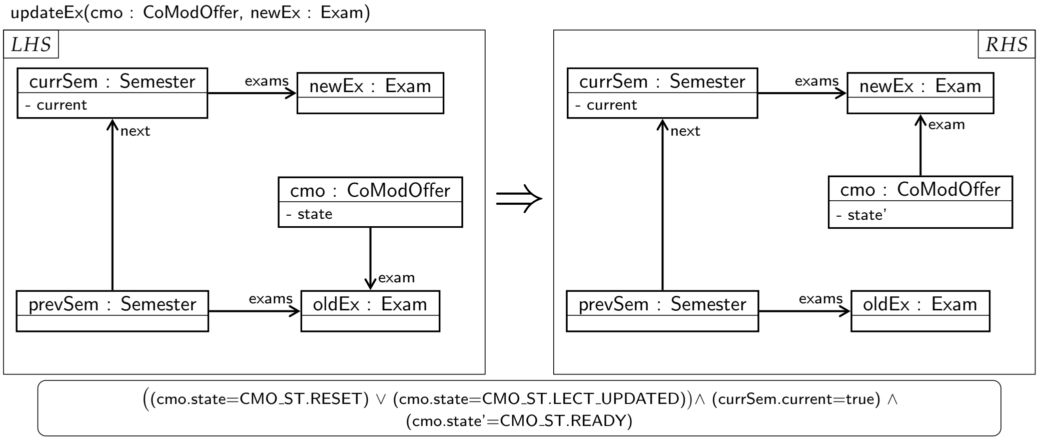


Figure A.22: The production updateEx(cmo : CoModOffer, newEx : Exam) is applied to an course module offer (cmo : CoModOffer) to set a new exam (newEx : Exam). To this end, the link exam is redirected from the old examination (oldEx : Exam) to the new examination (newEx : Exam)is removed and a new link of type exam. The new examination newEX has to be contained in the current semester (as (currSem.current=true)), whereas the old exam is assumed to be in the previous semester prevSem. The production can only be applied if the course module offer cmo is in the RESET or LECT_UPDATED state, whereas the new state (i. e., cmo.state') is set to READY.

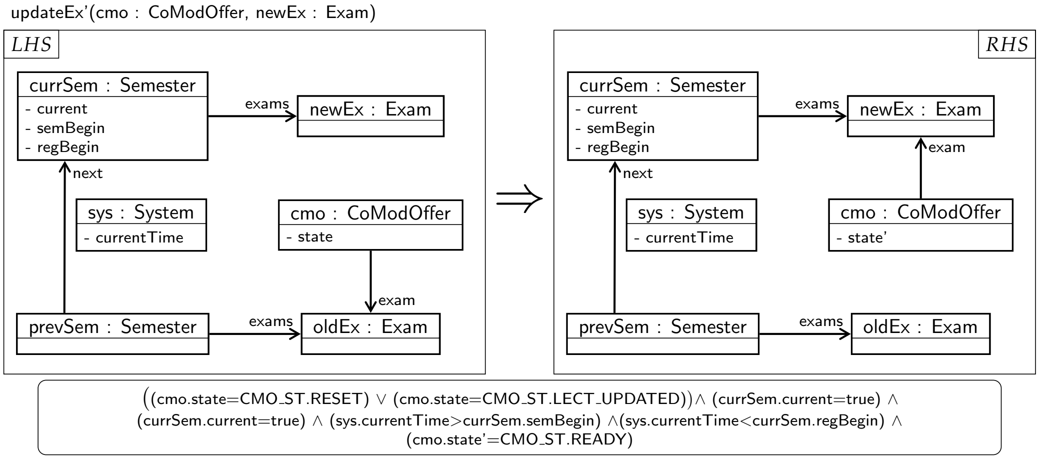


Figure A.23: The production updateEx'(cmo : CoModOffer, newEx : Exam) is similar to production updateEx(cmo : CoModOffer, newEx : Exam) except that production updateEx' can only be applied after the begin of the current semester and before the begin of the current registration period.

A.18 Production resetCMO

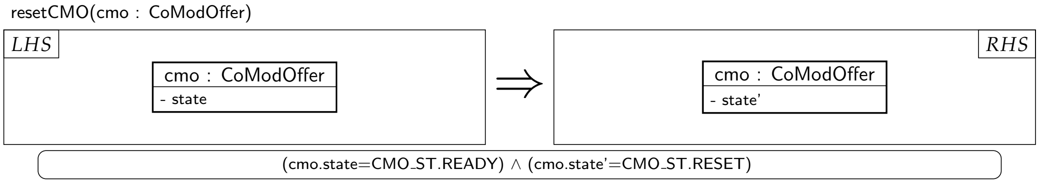

resetCMO(cmo : CoModOffer)
LHS
cmo : CoModOffer
- state
⇒
RHS
cmo : CoModOffer
- state'
(cmo.state=CMO_ST.READY) ∧ (cmo.state'=CMO_ST.RESET)

Figure A.24: The production resetCMO resets a course module offer by changing the state from READY to RESET.

B

USER DEFINED NEGATIVE CONSTRAINTS

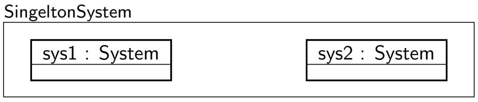


Figure B.1: The negative constraint SingeltonSystem ensured that consistent instance models only contain one instance of class System. This is required to ensure that there is only one currentTime value. Alternatively we might define a constraint that requires that there do not exists a pair of Systems with different values for currentTime.

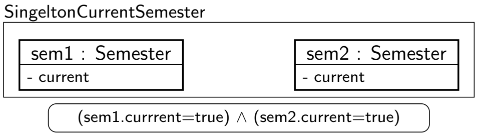


Figure B.2: Negative constraint SigeltonCurrentSemester declares that any instance model that contains two semesters whose current attribute is set to true (i. e. (sem1.currrent=true) and (sem2.current=true)) is inconsistent.

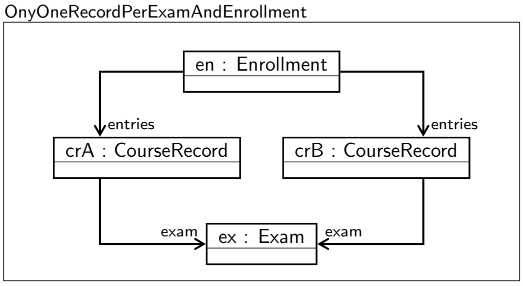


Figure B.3: Negative constraint OnyOneRecordPerExamAndEnrollment ensures that any Enrollment has at most one CourseRecord for an Exam.

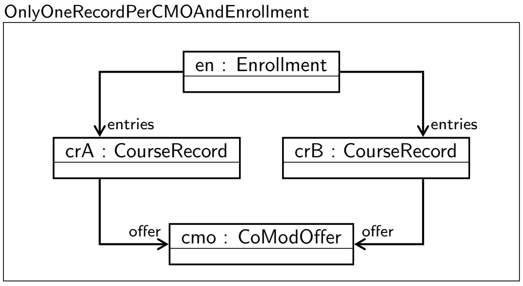


Figure B.4: Negative constraint OnlyOneRecordPerCMOAndEnrollment ensures that any Enrollment has at most one CourseRecord for an course module offer (CoModOffer).

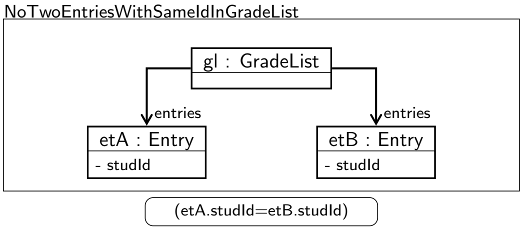


Figure B.5: Negative constratin NoTwoEntriesWithSameIdInGradeList ensures that there are two Entries for the same student (i. e. studId) in a GradeList.

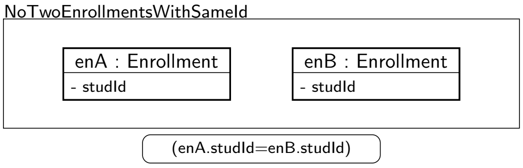


Figure B.6: Negative constraint NoTwoEnrollmentsWithSameId ensures that no two Enrollments with same studId exist.

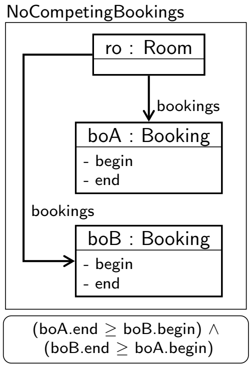


Figure B.7: Negative constraint NoCompetingBookings ensures that a Room does not have two Bookings with overlapping time slots.