UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

GUILHERME GROCHAU AZZI

Improving Conflict Detection in Double-Pushout Graph Transformation

Thesis presented in partial fulfillment of the requirements for the degree of Master of Computer Science

Advisor: Prof. Dr. Leila Ribeiro

Porto Alegre
July 2018

CIP — CATALOGING-IN-PUBLICATION

Azzi, Guilherme Grochau

Improving Conflict Detection in Double-Pushout Graph Transformation / Guilherme Grochau Azzi. – Porto Alegre: PPGC da UFRGS, 2018.

137 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2018. Advisor: Leila Ribeiro.

1. Graph transformation. 2. Double-pushout. 3. Static analysis. 4. Parallel independence. 5. Critical pair. 6. Initial conflict. 7. Category theory. 8. Adhesive category. 9. Subobject. I. Ribeiro, Leila. II. Título.

“The essential virtue of category theory
is as a discipline for making definitions,
the programmer’s main task in life.”

— DAVID RYDEHEARD

ACKNOWLEDGEMENTS

Writing this thesis was a considerable personal and professional achievement, and it was only possible due to the support and encouragement of several people. In particular, I would like to thank:

Prof. Leila Ribeiro for accepting and guiding me as a student. The opportunities you gave me, the examples you set and your ability to motivate students have greatly enriched my time at UFRGS. Your guidance on technical and professional matters was fundamental for the completion of this work, and will certainly be helpful for the remainder of my academic career.

Prof. Rodrigo Machado for all the help and fruitful discussions during the initial stages of my masters, helping me transition into the field of graph transformation. Your technical knowledge and constant excitement with interesting research remains an inspiration.

Prof. Andrea Corradini for the guidance near the final stages of my masters, even from afar. Thank you for the many Skype calls and prompt reviews of the papers on which we worked together. The clarity and wealth of experience you bring to research has deeply impacted my work. Without your help, this thesis would not be the same.

My colleagues for making my stay at UFRGS enjoyable and productive. Thanks for the many lunches, coffees and conversations about varied topics, academic or otherwise. Thanks especially to Jonas Bezerra, Andrei Costa and Leonardo Rodrigues for the many fruitful discussions of graph transformation, and for all the cooperation when developing Verigraph. Also many thanks to Marcelo Prates for all the tips regarding the experimental part of my work.

The many teachers and professors that sparked my interest in mathematics and computer science. Especially Profs. Paulo Blauth, Álvaro Moreira, Rafael Bordini and Sabine Glesner for introducing me to the theoretical aspects of computer science.

The governmental agencies CAPES and CNPq for the financial support.

My parents for their love, support and understanding throughout all these years. Thank you for being with me during the good and the bad, and for always helping me follow my dreams.

My friends and family for helping me keep my mind off work when I needed, and understanding my absence when I couldn’t.

ABSTRACT

Graph transformation is a useful framework for the specification, analysis and development of software, particularly within Model-Driven methodologies. In this setting, graphs or graph-like structures are used to represent states of a system, while its possible transitions are determined by transformation rules. This combines an intuitive visual representation with a rich theory and several verification techniques. Since the behaviour of these a rule-based models emerges from the interactions between rules, techniques for understanding such interactions are necessary. In this work, we focus on *conflicts*: situations where the application of a rule hinders the application of another. *Conflict detection* is then a static analysis technique that enumerates a finite but complete set of such situations, in the sense that every other conflict can be expressed in terms of some detected conflict. The most common approach to conflict detection is the enumeration of so-called critical pairs, and it was successfully applied in several contexts. Nevertheless, it still faces some issues related to scalability. These involve the running time of existing algorithms, but more fundamentally they stem from the amount of redundant potential conflicts that are enumerated. In this work, we take an important step towards improving conflict detection and reducing the redundancy of results. To this end, we develop a theory describing the root causes of conflicts in the form of *conflict essences*. Using lattice-theoretical techniques, we were also able to detect further sources of redundancy, identifying *irreducible essences* as suitable, less redundant subset. We also show that conflict essences are closely related to initial conflicts, a recently proposed subset of critical pairs, allowing their application to a wider variety of contexts. Moreover, simple algorithms for enumerating conflict and irreducible essences are provided. Finally, we present experimental evidence that initial conflicts and irreducible essences allow a significant reduction on the number of reported conflicts, when compared to the more traditional critical pairs, without loss of information. All of our results hold for categories of set-valued functors, a generalisation of graphs and graph structures, and sufficient conditions are identified for most of our results to hold in other adhesive categories.

Keywords: Graph transformation. double-pushout. static analysis. parallel independence. critical pair. initial conflict. category theory. adhesive category. subobject.

Melhorando a Detecção de Conflitos para Transformação de Grafos Algébrica

RESUMO

Transformação de grafos é uma teoria apropriada para a especificação, análise e desenvolvimento de software, particularmente em abordagens dirigidas a modelos. Nesse contexto, grafos ou estruturas semelhantes representam os estados de um sistema, enquanto as transições são determinadas por regras de reescrita. Assim, uma representação visual e intuitiva é combinada a uma vasta teoria, permitindo o uso de várias técnicas de verificação. Como o comportamento desses modelos baseados em regras emerge de suas interações, são necessárias técnicas para entendê-las. Nesse trabalho, focamos nos *conflitos* entre regras: situações em que a aplicação de uma regra impede a aplicação de outra. A análise estática denominada *detecção de conflitos* busca enumerar um conjunto finito dessas situações que seja completo, ou seja, tal que qualquer outro conflito possa ser expresso em termos de um dos conflitos detectados. Em particular, a enumeração de pares críticos foi aplicada com sucesso em vários contextos. Ainda assim, a ela encontra problemas relacionados à escalabilidade. Isso envolve o tempo de execução, mas fundamentalmente advém do grande número de conflitos potenciais redundantes que é identificado. Neste trabalho, damos um passo importante em direção a uma técnica mais eficiente para detecção de conflitos. Para isso, desenvolvemos uma teoria que descreve a causa principal dos conflitos na forma de *essências de conflito*. Usando teoria de reticulados, pudemos detectar mais fontes de redundância, identificando essências irredutíveis como um subconjunto apropriado e menos redundante. Também mostramos que essas são intimamente relacionadas aos conflitos iniciais, um subconjunto dos pares críticos proposto recentemente. Assim, a aplicação de conflitos iniciais se torna possível em novos contextos. Além disso, apresentamos algoritmos para enumerar essências de conflito, conflitos iniciais e essências irredutíveis. Por fim, apresentamos evidência empírica de que conflitos iniciais e essências irredutíveis reduzem significativamente o número de conflitos reportados, em relação aos pares críticos, sem perda de informação. Todos os resultados teóricos deste trabalho são válidos para categorias de funtores com codomínio na categoria de conjuntos, uma generalização de grafos e de *graph structures*. Também identificamos condições suficientes para que os principais resultados sejam válidos em outras categorias adesivas.

Palavras-chave: transformação de grafos, duplo-pushout, análise estática, independência paralela, par crítico, conflito inicial, teoria das categorias, categoria adesiva, subobjeto.

LIST OF ABBREVIATIONS AND ACRONYMS

DPO	Double-Pushout
IPO	Initial Pushout
NAC	Negative Application Condition
PB	Pullback
PO	Pushout
poset	Partially Ordered Set
UML	Unified Modelling Language
VK	Van Kampen

LIST OF SYMBOLS

$\mathbb{S}\mathrm{et}, \mathbb{G}\mathrm{raph}, \mathbb{C}, \mathbb{D} \ldots$	Categories
$A, B, G, H, X, Y \ldots$	Objects of a category (sets, graphs, typed graphs. . .)
$f : A \to B$	Morphism from A to B (function, graph morphism. . .)
$f : A \rightarrowtail B$	Monomorphism (e.g. injective function)
$f : A \hookrightarrow B$	Inclusion
$\mathrm{F} : \mathbb{C} \to \mathbb{D}$	Functor from $\mathbb{C}$ to $\mathbb{D}$
$f : \mathrm{F} \dot{\to} \mathrm{G}$	Natural transformation from F to G
$\mathscr{A} = (A_i)_{i \in I}$	Indexed family over the index set I
$f \circ g$	Composition of morphisms
$\rho = L \leftarrow K \to R$	Transformation rule
$m, n, \ldots$	Matches
$t : G \overset{\rho,m}{\Longrightarrow} H$	Transformation step with rule ρ and match m
$\mathrm{id}_X : X \to X$	Identity morphism for object X
$\mathbf{0}$	Initial object
$!_X : \mathbf{0} \to X$	Unique morphism from initial object to X
$X \cong Y$	Isomorphism relation on objects
$f(x)$	Application of function to an element x
$f(X)$	Image of set $X \subseteq \mathrm{dom}(f)$ by function f
$f^{-1}(y) = \{x \in \mathrm{dom}(f) \mid f(x) = y\}$	Preimage for a function f
$\mathrm{card}(S)$	Cardinality of set S
$\mathbf{Sub}(X)$	Poset of subobjects for object X
$\cup_{\mathbf{Sub}}, \cap_{\mathbf{Sub}}, \subseteq_{\mathbf{Sub}}$	Union, intersection and containment of subobjects
$\subseteq_L, \subset_L, \mathrel{\subset\!\!\!\cdot}_L$	Partial order, strict order and cover for poset L
$\cup_L, \cap_L$	Join (union) and meet (intersection) for lattice L

LIST OF FIGURES

Figure 2.1 Example state for the elevator control system. 27
Figure 2.2 Type graph for the elevator control system. 27
Figure 2.3 Rules for fulfilling requests. 28
Figure 2.4 Rules for moving an elevator up. 29
Figure 2.5 Rules for switching elevator direction. 30
Figure 2.6 Hasse diagram for lattice of subgraphs 35
Figure 2.7 Rule `move-up-NU` as a span 36
Figure 2.8 Match morphism for rule `move-up-NU` 36
Figure 2.9 Transformation step with rule `move-up-NU` 37
Figure 2.10 Non-applicable match with rule `deactivate` 38
Figure 2.11 Examples of parallel independence 39
Figure 2.12 Examples for the essential condition of parallel independence 40
Figure 2.13 Extension diagrams 42
Figure 2.14 Initial pushout of typed graphs 46
Figure 2.15 Deletion object for rule `move-up-NU`. 47

Figure 4.1 Conflicting transformation steps of `move-up-NU` with itself 60
Figure 4.2 Causes of conflicts for `move-up-NU` as spans. 60
Figure 4.3 Lattice of subgraphs with irreducibles highlighted by shadows 64
Figure 4.4 Example of subobjects for a pair of typed graphs. 68

Figure 5.1 Overview of concepts related to conflicts and their root causes, with new concepts and results written in bold. 75
Figure 5.2 Overview of main results and categorical assumptions in Chapter 5. Results are drawn with solid lines while assumptions are drawn with dashed lines...76
Figure 5.3 Examples of disabling essences 78
Figure 5.4 Examples of disabling reason. 84
Figure 5.5 Extended transformation steps with distinct disabling reasons. 84
Figure 5.6 Potential disabling essences for $\rho_1 = \texttt{move-up-NU}$ and $\rho_2 = \texttt{deactivate}$, where $c_1 \in \mathbf{Ess}_{\mathrm{dbl}}(\rho_1, \rho_2)$ and $c_2 \in \mathbf{Ess}_{\mathrm{dbl}}(\rho_2, \rho_1)$. 87
Figure 5.7 Conflict essences for `move-up-NU`. 91

Figure 6.1 Disabling candidates for rules `move-up-ND` and `deactivate` 101

Figure 7.1 Log-log scatter plot of critical pairs vs. initial conflicts, for each pair of rules, with the fit of linear regression. 112
Figure 7.2 Scatter plot of the fraction of critical pairs that is initial vs. total number of critical pairs in logarithmic scale, for each pair of rules. 113
Figure 7.3 Scatter plot of conflict essences in logarithmic scale vs. irreducible conflict essences, for each pair of rules. 113
Figure 7.4 Scatter plot of disabling essences in logarithmic scale vs. irreducible disabling essences 114
Figure 7.5 Log-log scatter plot with running times of Algorithm 6.1 vs. filtering of irreducible essences, for each pair of rules. 115
Figure 7.6 Log-log scatter plot with running times of Algorithm 6.2 vs. filtering of irreducible essences, for each pair of rules. 116
Figure 7.7 Log-log scatter plot with running time of Algorithm 6.1 vs. checked overlappings of the left-hand sides, for each pair of rules. 116

Figure 7.8 Log-log scatter plot with running time of Algorithm 6.2 vs. checked overlappings of the left-hand sides and deletion object, for each pair of rules..... 117
Figure 7.9 Log-log scatter plot comparing the number of overlappings checked by Algorithm 6.2 with the total number of overlappings of the left-hand sides, for each pair of rules. 117
Figure 7.10 Log-log scatter plot comparing the time spent enumerating conflict and disabling essences, for each pair of rules........ 118

LIST OF TABLES

Table 4.1 Main results of Chapter 4 61

Table 6.1 Numbers of jointly epic pairs for small sets. 98

Table 7.1 Overview of transformation systems used as input for the experiment. 111

CONTENTS

1 INTRODUCTION..........21
1.1 Problem..........22
1.2 Goals..........23
1.3 Outline..........23
2 ALGEBRAIC GRAPH TRANSFORMATION..........25
2.1 Modelling an Elevator System..........25
2.2 Categories of Graphs and Adhesive Categories..........30
2.3 Rules, Matches and Transformations..........35
2.4 Conflicts and Parallel Independence..........38
2.5 Conflict Detection..........41
2.5.1 Critical Pairs..........42
2.5.2 Initial Conflicts..........44
2.6 Initial Pushouts..........45
3 FUNCTOR CATEGORIES AS GENERALIZED GRAPHS..........51
4 GENERALIZED SUBOBJECTS..........59
4.1 Lattice Theory..........62
4.2 Generalised Subobjects..........65
4.2.1 Subobjects for Monic Spans..........66
4.2.2 Subobjects for Pairs of Objects..........67
5 CHARACTERIZING CONFLICTS..........75
5.1 Disabling and Conflict Essences..........76
5.1.1 Conflict Essence and Extension..........80
5.1.2 Comparing with Previous Work..........82
5.2 Potential Essences..........85
5.2.1 Potential Essences and Initial Conflicts..........87
5.2.2 Irreducible Conflict Essences..........90
6 ENUMERATING POTENTIAL CONFLICTS..........93
6.1 Enumerating Conflict Essences and Initial Conflicts..........94
6.1.1 Performance Bottleneck..........96
6.2 Enumerating Disabling Essences..........98
6.3 Finding Irreducible Essences..........105
7 EXPERIMENTAL EVALUATION..........109
7.1 Methods..........110
7.2 Results..........112
8 RELATED WORK..........119
8.1 Algebraic Approaches to Graph Transformation..........119
8.2 Categories for Algebraic Graph Transformation..........120
8.2.1 Variations of Adhesivity..........120
8.2.2 Attributed Graphs and Structures..........122
8.3 Conflict Detection in the Double-Pushout Approach..........123
8.3.1 Applications of Conflict Detection..........124
8.4 Verification Techniques for Graph Transformation..........125
9 CONCLUSIONS..........127
9.1 Contributions..........127
9.2 Future Work..........129
REFERENCES..........131

1 INTRODUCTION

Software is becoming increasingly ubiquitous and complex. No longer confined to simple managerial tasks, it now controls many important parts of our lives, performing crucial tasks in banks, cars, trains, aeroplanes and telecommunications, among many others. Many lives and much of the economy rely on software, leading to significant damages when it doesn't work as intended. Thus, understanding the behaviour of software and ensuring its *correctness* remains a very relevant topic.

An important tool to understand and verify software are *models*. By abstracting from the full complexity of software, they allow engineers and analysts to focus on the most relevant aspects for the problem at hand. Indeed, in the Model-Driven approach to software development, models are defined in domain-specific modelling languages tailored to their particular application domain (SCHMIDT, 2006). When equipped with a formal semantics, it is also possible to *analyse* a model to extract summarised information or even prove that is has certain properties.

This thesis is concerned with models based on *graph transformation*, which have a visual and intuitive representation along with strong formal underpinnings (ROZENBERG, 1997). Each *state* of the system is modelled as a graph or similar structure, representing entities by nodes and their current relationships by edges. The *behaviour* of the system is then specified with transformation rules, which express conditions for a certain event to occur, and the modified state after its occurrence. That is, they provide pre- and post-conditions as snapshots of the system state, or part of it.

For many applications, the usual notion of graphs is not sufficient. We often want to ascribe *types* to the nodes or edges, to distinguish between different kinds of entities and relationships. Some of the entities may contain boolean or numeric *attributes*. Thus, variations on the notion of graph are used for different applications.

In order to avoid redefining the theory for each notion of graph, the *algebraic approach* to graph transformation is based on notions of category theory (EHRIG et al., 2006). It abstracts from graphs and mappings between them to objects and morphisms of arbitrary categories, as long as they satisfy certain assumptions. This allows the main definitions and results, stated and proven for arbitrary categories, to be instantiated for various notions of "transformed object". This includes several variations of graphs, such as graphs with types, subtypes, labels or attributes, but also other structures such as Petri nets, algebraic signatures and transformation rules themselves.

Algebraic graph transformation has found many applications, in particular the Double-Pushout (DPO) approach. In the context of Model-Driven Engineering, it underlies approaches to model transformation such as triple graph grammars (KLAR et al., 2010), and model transformation tools like Henshin (ARENDT et al., 2010).

Having a formal semantics, graph transformation allows several analysis techniques for understanding the behaviour of a model. This thesis is concerned with the detection of *conflicts*, that is, situations where rules interfere with each other in a particular way: the application of a rule may hinder the application of another. In this context, the most common technique is enumeration of *critical pairs*, which are minimal contexts in which a pair of rules conflict. Besides helping to understand interactions between rules, critical pairs can be used to investigate confluence, which is an important property for several systems.

Indeed, many practical applications of critical pair analysis were reported in the literature. For example, it has was used to understand interactions between features in Software Product Lines by Jayaraman et al. (2007), and between refactorings of UML models by Mens, Taentzer and Runge (2007). Taentzer et al. (2010) used it as a basis for model versioning. It was also used to validate medical guidelines by Cota et al. (2017) and requirement models for software by various authors (ERMEL et al., 2011; HAUSMANN; HECKEL; TAENTZER, 2002; OLIVEIRA JR. et al., 2014).

1.1 Problem

Despite the various applications of critical pair analysis, its computation is sometimes infeasible. The most pressing issue is the large number of critical pairs, even for relatively small rules. Lambers et al. (2018) identified some sources of redundancy that occur in critical pairs, proposing *initial conflicts* as a suitable subset. It was proven complete in the sense that every critical pair is the embedding of an initial conflict into a larger context. Their existence, however, was only proven for the particular case of typed graphs. To the best of our knowledge, this result wasn't yet generalised for other graph models. Moreover, as will be shown in this thesis, some redundancy may still be left within the initial conflicts.

A second issue is that manually determining the root cause of a conflict is often difficult. Lambers, Ehrig and Orejas (2008) formally characterised *conflict reasons*, but these may contain elements unrelated to the conflict and lack a direct connection to the definition of conflicts. Furthermore, their relation to initial conflicts has not been reported, to the best of our knowledge.

1.2 Goals

The main goal of this thesis is to improve the detection of conflicts between rules. This involves characterising root causes, enumerating all causes for conflicts between a pair of rules, and reducing redundancy of results. Specific goals of this thesis are the following.

G1 Identify a class of categories where initial conflicts exist, including typed graphs.

G2 Formally characterise the root causes of conflicting transformations with a clear connection to initial conflicts.

G3 Formally characterise all possible causes of conflicts for pairs of rules.

G4 Design algorithms for enumerating initial conflicts. They should be defined in terms of categorical constructions, such that they work in all categories identified for G1.

1.3 Outline

Chapter 2 An introduction to Double-Pushout graph transformation is presented. We begin with an informal introduction along with a running example for this thesis. The necessary background of category theory is then reviewed, as well as the theory of Double-Pushout transformation.

Chapter 3 Categories of set-valued functors are proposed as a graph model. It is shown that they are suitably general, subsuming several commonly used categories including those of typed graphs. Nevertheless, they are shown to have important properties for the remainder of this thesis, while still allowing some proofs to be carried out set-theoretically.

Chapter 4 In category theory, subobjects generalise the set-theoretical notion of subset. We further generalise the notion of subobject, characterising in particular the subobjects for pairs of objects and for cospans of monomorphisms. These will serve as a basis for the formal characterisation of root causes of conflicts. By showing that each pair of objects and each span of monomorphisms has a *lattice* of subobjects, we allow the use of lattice-theoretical techniques in subsequent chapters.

Chapter 5 Conflict and disabling essences are proposed as a formal characterisation for the root causes of conflicts between two transformations. Sets of potential conflict and disabling essences for pairs of rules are also characterised. We prove several important properties of essences, also relating them to conflict reasons and initial conflicts. Finally, redundant conflict and disabling essences are detected, and irreducible essences are proposed to avoid this redundancy.

Chapter 6 Algorithms for enumerating conflict and disabling essences are designed, using categorical constructions as a basis. The performance and scalability of the proposed algorithms is briefly discussed.

Chapter 7 The concepts and algorithms proposed in this thesis are experimentally evaluated. In particular, we investigate how the numbers of critical pairs, conflict essences and irreducible essences relate to each other. We also validate some claims about the performance of the algorithms proposed in the previous chapter.

Chapter 8 An overview of the related work is presented, including other approaches of algebraic graph transformation, classes of categories where these approaches are commonly instantiated, applications of conflict detection and other verification techniques.

Chapter 9 Presents concluding remarks. It summarises the main contributions of this thesis and suggests future directions of research.

2 ALGEBRAIC GRAPH TRANSFORMATION

This chapter reviews most of the background necessary for this thesis, including some topics in category theory and the theory of Double-Pushout (DPO) graph transformation. Further background in lattice theory is introduced in Chapter 4, where it will be necessary.

This is not intended as an introduction to category theory, assuming familiarity with its basic concepts. In particular, readers should be familiar with functors, natural transformations, limits, colimits and the category of sets with total functions. Suitable introductions to category theory can be found in the books by Riehl (2017) and Adámek, Herrlich and Strecker (2009).

The first part of this chapter provides an informal introduction to graph transformation, presenting also the running example for this thesis: a system controlling one or more elevators in a building. Subsequent sections review the theory of DPO graph transformation, introducing some concepts of category theory as they are needed. The final subsection introduces the concept of initial pushout, an important construction used throughout this thesis, providing known and novel results. In particular, we relate initial pushouts to subobjects.

2.1 Modelling an Elevator System

As a motivating example, we will model a system controlling one or more elevators in a building. This is based on the model proposed by Lambers (2009), but lifting two restrictions: the system may control multiple elevators, and movement may occur between arbitrary floors, instead of always involving a main stop. We begin by informally describing the requirements of the system.

Example 2.1 (Requirements for an Elevator Control System)**.** The *elevator control system* is intended for buildings where a series of floors is connected by one or more adjacent elevators, all of which service the same floors. Users of the system expect to be transported between two arbitrary (but distinct) floors.

All floors are equipped with buttons for requesting that an elevator stop at this floor so that people can get into the elevator and be transported to another floor. Most floors contain two buttons, which indicate the direction of the request, that is, whether the users expect to reach a floor above or below their current floor. The highest and lowest floors contain only one button, since only one direction is possible. Note that different users may simultaneously request both directions for the same floor. Each elevator is equipped with a series of buttons identifying each of the floors. By pressing such a button, users signal that they expect to be taken to the corresponding floor.

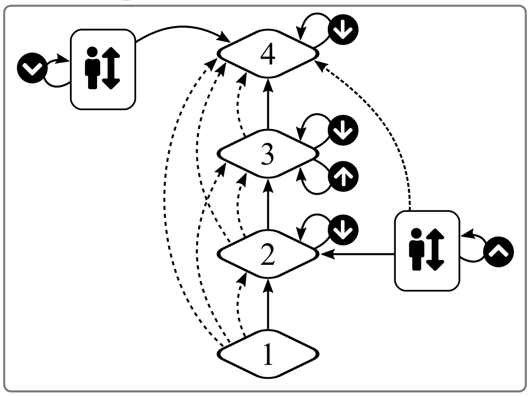
Each elevator is always moving in a particular direction: either up or down. Moreover, an elevator should only move if there are unfulfilled internal or external requests at floors in its current direction. *Internal requests* are caused by pressing a button inside the elevator, while *external requests* are caused by pressing buttons on each floor, and are classified according to their direction. When moving, an elevator should visit each floor in succession, stopping whenever a floor has one or more appropriate requests and fulfilling all of them. Internal requests are always appropriate for stopping, while external requests are only appropriate if their direction is consistent with the current direction of motion. Thus, an elevator moving up will not stop due to an external request for moving down.

At each point in time, the *system state* can be suitably represented by a graph, where nodes and edges are ascribed *types* to indicate which kind of entity or relation they represent.

Example 2.2 (Types for the Elevator Control System)**.** There are two kinds of entities in the elevator system: `elevator` (denoted by ⇅) and `floor` (denoted by ⬭). Edges between floors may have two different types: `next-up` (denoted by solid edges) and `below` (denoted by dashed edges), indicating the relative positions of the floors. Similarly, a solid edge from an elevator to a floor may have type `at` (denoted by solid edges), indicating its current location, or type `req-stop` (represented by dashed edges), indicating an internal request for the source elevator to stop at the target floor. Note that self-loop edges can be interpreted as predicates of an entity, instead of relations. Then self-loop edges on an elevator indicate its direction of motion, having the type `going-up` (denoted by ⏶) or `going-down` (denoted by ⏷). Self-loops on floors indicate the existence of external requests, with type `req-up` (denoted by ⬆) or `req-down` (denoted by ⬇).

Example 2.3 (State of Elevator Control System)**.** Figure 2.1 depicts an example state for a system controlling two elevators along four floors, where the labels on the floors are merely illustrative and not part of the graph. Note that there are several edges of type `below`, since it is a transitive relation. One of the elevators is located on the second floor and moving up, while the other is on the fourth floor and moving down. The elevator on the second floor has an internal request for stopping on the fourth floor, while the second and fourth floors have external requests for going down. Moreover, the third floor has two external requests, for either direction.

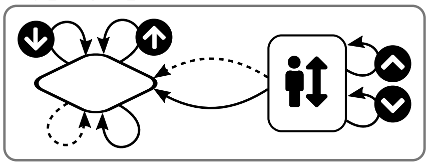
Figure 2.1: Example state for the elevator control system.



A common and concise representation for the types of nodes and edges is a *type graph*, where each node and edge represents a distinct type. Type graphs are similar in nature to UML class diagrams, while the graphs that represent system states are analogous to object diagrams (OBJECT MANAGEMENT GROUP, 2017).

Example 2.4 (Type Graph for Elevator Control System)**.** The type graph for the elevator control system is depicted in Fig. 2.2, containing the types described in Example 2.1.

Figure 2.2: Type graph for the elevator control system.



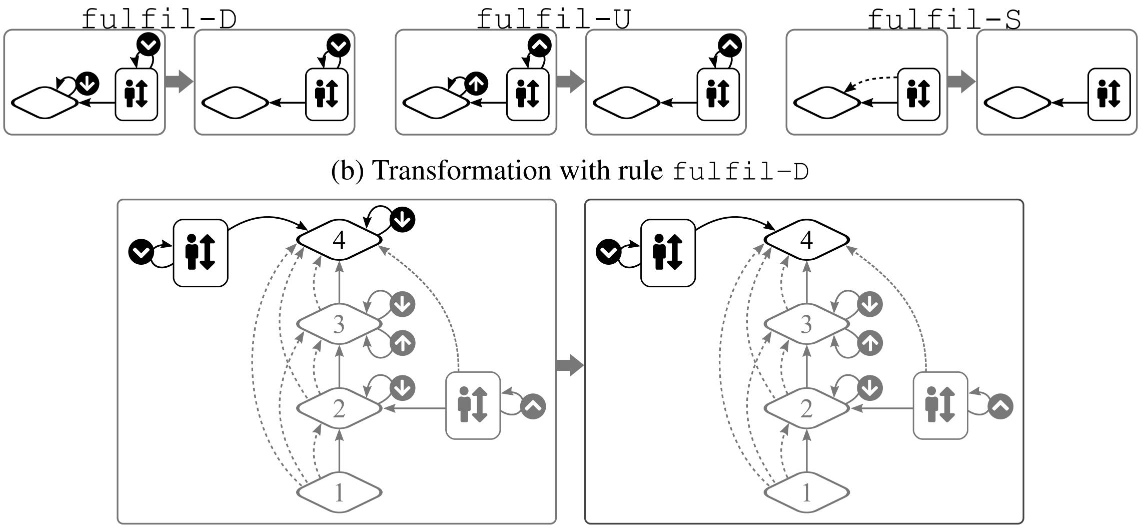
So far, only the static aspects of the system have been described. In order to specify its behaviour, we use transformation *rules*, each consisting of two graphs that describe pre- and post-conditions in a local way. If a system state contains the pre-condition as a subgraph, we refer to this situation as a *match* for the rule. Then the rule may be applied at this match, replacing the appropriate subgraph with the post-condition.

Example 2.5 (Fulfilling Requests)**.** When an elevator is moving down and reaches a floor with an external `req-down`, it should stop and fulfil this request. This is encoded by rule `fulfil-D`, which is depicted in Fig. 2.3a with the pre-condition drawn on the left and the post-condition on the right. It essentially deletes the `req-down` edge. Analogous rules `fulfil-U` and `fulfil-S` are also shown in Fig. 2.3a, fulfilling `req-up` requests when the elevator is moving up and `req-stop` requests regardless of direction.

Rule `fulfil-D` may be applied to the state of Example 2.3 as shown in Fig. 2.3b, where the subgraph corresponding to the pre-condition is shown in black and the rest of the state, in grey.

Figure 2.3: Rules for fulfilling requests.

(a) Rules

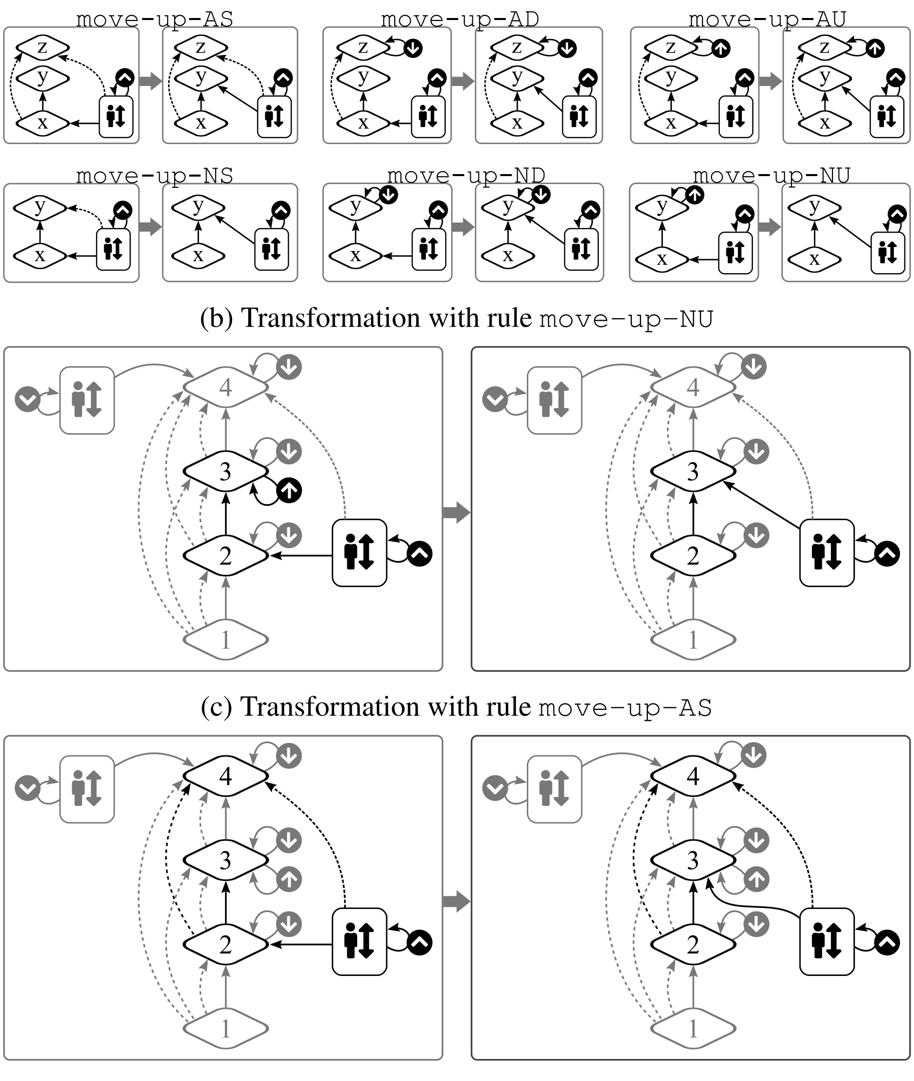


(b) Transformation with rule `fulfil-D`

Example 2.6 (Moving an Elevator Up)**.** Any elevator that is moving up should keep moving up as long as there are requests in higher floors. In this case, it should always proceed to the next floor up. Note that this can occur with a request of any kind: `req-up`, `req-down` or `req-stop`. Moreover, the request may be on the next floor up, or on a floor further above. Thus, we have six different pre-conditions for moving up, and we need to model this with six different rules. They will be named with the following convention: all rules are prefixed with `move-up-`, followed by a letter indicating the location of the request (`N` for the next floor up, `A` for a floor further above) and then a letter indicating the kind of request (`U` or `D` for external requests going up or down, and `S` for internal requests to stop at a floor). These rules are shown in Fig. 2.4a, all of them deleting the `at` edge for the current position of the elevator, then creating a new `at` edge to the floor above. Node labels in Fig. 2.4a are not part of the graphs themselves, but used to indicate how they are preserved by rules. Note that in rules `move-up-NU` and `move-up-NS`, the elevator reaches a floor with requests that can be fulfilled. We have chosen to immediately fulfil such requests, thus these rules delete the corresponding edges.

Several of these rules can be applied to the system state of Example 2.3 to move the elevator from the second floor. The application of rule `move-up-NU` is shown in Fig. 2.4b, immediately fulfilling an external request in the third floor. The application of `move-up-AS` is shown in Fig. 2.4c, where no request is yet fulfilled, requiring a subsequent application of `fulfil-U`.

Figure 2.4: Rules for moving an elevator up.



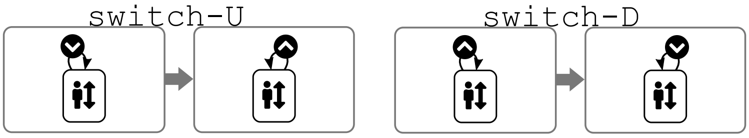
(a) Rules

(b) Transformation with rule `move-up-NU`

(c) Transformation with rule `move-up-AS`

Note that, with the rules defined in Example 2.6, an elevator may move out of a floor without fulfilling all appropriate requests. In order to avoid this, the rule would have to be extended with a *Negative Application Condition* (NAC), which forbids the application of a rule in the presence of additional elements (HABEL; HECKEL; TAENTZER, 1996). Similarly, one may use *constraints* to encode invariants that should hold in every system state (EHRIG et al., 2004). Both NACs and constraints, however, are outside the scope of this thesis. Integrating them to our main contributions is an important direction

Figure 2.5: Rules for switching elevator direction.



for future work.

Beside the rules mentioned above, a model for the elevator control system needs rules for moving elevators down, analogous to those that move it up, and for switching its direction of motion, shown in Fig. 2.5. A complete model consists of the type graph from Example 2.4, an initial state such as Example 2.3 and all the rules defined in this section.

2.2 Categories of Graphs and Adhesive Categories

The concept of graph used in algebraic graph transformation is generally that of a *directed multigraph*, that is, allowing parallel edges. This is also known as a quiver.

Definition 2.7 (Graph)**.** A *graph* $G = (V, E, s, t)$ has sets V of nodes and E of edges, along with source and target functions $s, t : E \to V$.

Functions, as mappings between sets, can be generalized to mappings between graphs. These are called *graph morphisms*, mapping nodes (edges) of a graph to nodes (edges) of another. Since graph morphisms have well-defined notions of domain, codomain and composition, they determine a category.

Definition 2.8 (Category of Graphs)**.** A *graph morphism* $f : G \to G'$ is a pair of functions $f = (f_V : V \to V', f_E : E \to E')$ that preserve incidence, that is, $f_V \circ s = s' \circ f_E$ and $f_V \circ t = t' \circ f_E$. Graphs and graph morphisms determine the *category of graphs*, denoted $\mathbb{G}$raph.

As seen in the previous section, we often want to ascribe types to the nodes and edges of a graph. Recall also that the collection of types can be expressed as a *type graph* T, where each node and edge represents a type. Then graph morphisms provide a suitable way to formalize typing: fixing a type graph T, a morphism $g : G \to T$ can be interpreted as a typing of G, or as a T-typed graph. A natural notion of type-preserving morphism also arises.

Definition 2.9 (Category of T-typed Graphs)**.** Let T be an arbitrary but fixed graph, referred to as the *type graph*. Then a *T-typed graph* is a morphism $g : G \to T$. Given typed graphs $g : G \to T$ and $g' : G' \to T$, a *type-preserving morphism* is just a graph morphism $f : G \to G'$ such that $g' \circ f = g$, i.e. diagram (2.1) commutes. This determines the category $\mathbb{G}\mathrm{raph}_T$ of T-typed graphs with type-preserving morphisms, which is essentially the slice category $\mathbb{G}\mathrm{raph} \downarrow T$.

$$\begin{tikzcd}
G \arrow[rr, "f"] \arrow[dr, "g"'] & & G' \arrow[dl, "g'"] \\
& T &
\end{tikzcd} \tag{2.1}$$

Remark 2.10. Typing will often be left implicit, denoting a typed graph $g : G \to T$ by G.

The categories of (typed) graphs have a lot of structure, which allows the definition of rules and transformations. For example, they are complete and cocomplete as they have all small limits and colimits, in particular all pullbacks and pushouts. Here we recall some well-known properties of pushouts and pullbacks, which will be used in subsequent proofs.

Fact 2.11 (Composition)**.** Assume diagram (2.2) commutes.

(i) If squares ① and ② are pullbacks, then rectangle ① + ② is also a pullback.

(ii) If squares ① and ② are pushouts, then rectangle ① + ② is also a pushout.

Fact 2.12 (Decomposition)**.** Assume diagram (2.2) commutes.

(i) If rectangle ① + ② and square ② are pullbacks then square ① is also a pullback.

(ii) If rectangle ① + ② and square ① are pushouts then square ② is also a pushout.

$$\begin{tikzcd}
A \arrow[r] \arrow[d] \arrow[dr, phantom, "\textcircled{1}"] & B \arrow[r] \arrow[d] \arrow[dr, phantom, "\textcircled{2}"] & C \arrow[d] \\
D \arrow[r] & E \arrow[r] & F
\end{tikzcd} \tag{2.2}$$

Fact 2.13 (Trivial Pullbacks/Pushouts)**.** If the square in diagram (2.3) commutes, having h monic and g an isomorphism, then it is a pullback. Dually, if the square in diagram (2.4) commutes having g epic and h an isomorphism, then it is a pushout.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, tail, "h"] \\
C \arrow[r, "k"'] & D
\end{tikzcd} \tag{2.3}$$

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, two heads, "g"'] & B \arrow[d, "h"] \\
C \arrow[r, "k"'] & D
\end{tikzcd} \tag{2.4}$$

Fact 2.14 (Pullbacks Preserve Monomorphisms)**.** If the square in diagram (2.3) is a pullback and h is monic, then g is also monic.

Besides having all pullbacks and pushouts, in categories of (typed) graphs the pushouts along monos are well-behaved with respect to pullbacks, interacting similarly to unions and intersections. This behaviour is captured formally in the definition of a Van Kampen square, which was discussed at length by Lack and Sobocinski (2005).

Definition 2.15 (Van Kampen Square)**.** A square (2.5) is a *Van Kampen (VK) square* if it is a pushout and it satisfies the following condition, for every cube as below where the bottom face corresponds to (2.5) and the back faces are pullbacks: the top face is a pushout if and only if the front faces are pullbacks.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "h"] \\
C \arrow[r, "k"'] & D
\end{tikzcd} \tag{2.5}$$

$$\begin{tikzcd}
& A' \arrow[dl] \arrow[rrd] \arrow[dd] & & \\
B' \arrow[dd] \arrow[rrd] & & & C' \arrow[dl] \arrow[dd] \\
& A \arrow[dl] \arrow[rrd] & D' \arrow[dd] & \\
B \arrow[rrd] & & & C \arrow[dl] \\
& & D &
\end{tikzcd} \tag{2.6}$$

Definition 2.16. A *pushout along mono* is a pushout square (2.5) such that f or g is monic.

Lemma 2.17 (LACK; SOBOCINSKI, 2005)**.** In the categories $\mathbb{S}\mathrm{et}$, $\mathbb{G}\mathrm{raph}$ and $\mathbb{G}\mathrm{raph}_T$ for any type graph T, all pushouts along monos are VK squares.

It turns out that having all pushouts along monos be VK squares ensures many of the properties necessary for graph transformation. Then a suitable class of categories in this context are adhesive categories, defined by Lack and Sobocinski (2005).

Definition 2.18 (Adhesive Category)**.** A category $\mathbb{C}$ is *adhesive* if and only if:

(i) It has all pullbacks.

(ii) It has all pushouts along monos.

(iii) All pushouts along monos are VK squares.

When studying adhesive categories, Lack and Sobocinski (2005) proved many useful properties for the theory of DPO transformation, some of which we now recall.

Theorem 2.19 (LACK; SOBOCINSKI, 2005)**.** Categories $\mathbb{S}\mathrm{et}$, $\mathbb{G}\mathrm{raph}$ and $\mathbb{G}\mathrm{raph}_T$ are adhesive, for any type graph T.

Importantly, many standard constructions of categories preserve adhesivity, as shown by Lack and Sobocinski (2005). Adhesivity is also related to the concept of elementary topos, a class of categories widely used for geometry, logic and the foundations of mathematics (MACLANE; MOERDIJK, 2012). Indeed, being adhesive is strictly weaker than being an elementary topos.

Theorem 2.20 (LACK; SOBOCINSKI, 2005)**.** Let $\mathbb{C}$ and $\mathbb{D}$ be adhesive categories, and let $\mathbb{X}$ be an arbitrary category.

(i) The product category $\mathbb{C} \times \mathbb{D}$ is adhesive.

(ii) The (co)slice categories $\mathbb{C} \downarrow C$ and $C \downarrow \mathbb{C}$ are adhesive, for any object C of $\mathbb{C}$.

(iii) The functor category $\mathbb{C}^{\mathbb{X}}$ is adhesive.

Theorem 2.21 (LACK; SOBOCINSKI, 2005)**.** Every elementary topos is adhesive.

The following properties are widely used in the theory of graph transformation. They are often called High-Level Replacement (HLR) properties.

Theorem 2.22 (LACK; SOBOCINSKI, 2005)**.** All adhesive categories have the following *HLR properties*.

(i) Pushouts along monos are also pullbacks: if square (2.7) is a pushout with monic l or k, then it is also a pullback.

(ii) Pushouts preserve monos: if square (2.7) is a pushout with monic l (or k), then g (or m) is also monic.

(iii) Uniqueness of pushout complements: given morphisms l and m as in diagram (2.7), if l is monic and there are k and g making diagram (2.7) a pushout, then these morphisms are unique up to isomorphism.

(iv) PO-PB decomposition: if rectangle $\textcircled{1} + \textcircled{2}$ of diagram (2.8) is a pushout and square $\textcircled{2}$ is a pullback, where morphisms l and f are monic, then squares $\textcircled{1}$ and $\textcircled{2}$ are pushouts.

$$\begin{tikzcd}
K \arrow[r, "l"] \arrow[d, "k"'] & L \arrow[d, "m"] \\
D \arrow[r, "g"'] & G
\end{tikzcd} \tag{2.7}$$

$$\begin{tikzcd}
K \arrow[r, "k"] \arrow[d, tail, "l"] \arrow[dr, phantom, "\textcircled{1}"] & D \arrow[r, "k'"] \arrow[d, "g"] \arrow[dr, phantom, "\textcircled{2}"] & D' \arrow[d, "g'"] \\
L \arrow[r, "m"'] & G \arrow[r, tail, "f"'] & G'
\end{tikzcd} \tag{2.8}$$

Another fundamental property of adhesive categories is related to the concept of *subobject*, a generalization of notions like subset and subgraph. Indeed, in adhesive categories, the subobjects of any arbitrary object behave like subsets: they have unions and intersections which distribute over each other. This is because sets of subobjects are distributive lattices, a topic that will be further discussed in Chapter 4.

Definition 2.23 (Subobject)**.** Let X be an object of a category $\mathbb{C}$. Two monos $a : A \rightarrow X$ and $b : B \rightarrow X$ are *isomorphic* when there exists an isomorphism $f : A \rightarrow B$ making diagram (2.9) commute, i.e. with $b \circ f = a$. A *subobject* of X is an isomorphism class of monos with codomain X. Moreover, $\mathbf{Sub}(X)$ is the *partially ordered set of subobjects* for X, where $a \subseteq_X b$ if and only if there exists a mono $f : A \rightarrowtail B$ making diagram (2.9) commute.

$$\begin{tikzcd}
A \arrow[rr, "f"] \arrow[dr, tail, "a"'] & & B \arrow[dl, tail, "b"] \\
& X &
\end{tikzcd} \tag{2.9}$$

Theorem 2.24 (LACK; SOBOCINSKI, 2005)**.** In an adhesive category, $\mathbf{Sub}(X)$ is a distributive lattice for any object X. Then it is equipped with the following notions, given subobjects $a, b \in \mathbf{Sub}(X)$.

(i) The *intersection* $a \cap_X b$ is obtained from a pullback over a and b. That is, given pullback ① from diagram (2.10), $a \cap_X b \cong x \circ i_X = y \circ i_Y$.

(ii) The *union* $a \cup_X b$ is obtained from a pushout over the intersection. That is, given pullback ① from diagram (2.10) and pushout ② from diagram (2.11), then the mediating morphism $u : U \rightarrow X$ is monic, and $a \cup_X b \cong u$.

(iii) The *top* or maximum subobject is $\top \cong \mathrm{id}_X \in \mathbf{Sub}(X)$.

$$\begin{tikzcd}
& A \arrow[dr, "a"] & \\
I \arrow[ur, "i_X"] \arrow[dr, "i_Y"'] & ① & X \\
& B \arrow[ur, "b"'] &
\end{tikzcd} \tag{2.10}$$

$$\begin{tikzcd}
& A \arrow[dr, "a_U"] \arrow[drr, bend left, "a"] & & \\
I \arrow[ur, "i_X"] \arrow[dr, "i_Y"'] & ② & U \arrow[r, dotted, tail, "u"] & X \\
& B \arrow[ur, "b_U"'] \arrow[urr, bend right, "b"'] & &
\end{tikzcd} \tag{2.11}$$

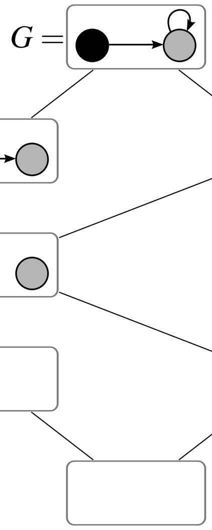
Example 2.25 (Lattice of Subgraphs)**.** Fig. 2.6 shows the lattice of subobjects for a particular graph, which is exactly its lattice of subgraphs.

Note that the lattice of subobjects of Example 2.25 has a bottom element, which is the empty graph. In fact, this is a suitable generalisation for the notion of "emptiness"

Remark 2.26. The bottom subobject $\bot \in \mathbf{Sub}(X)$ can be seen as the "empty subobject", when it exists. This is consistent with the categorical characterisation of emptiness using strict initial objects.

Definition 2.27. An object $\mathbf{0}$ is *initial* in a category $\mathbb{C}$ if, for each object X of $\mathbb{C}$, there is a unique arrow $!_X : \mathbf{0} \rightarrow X$. An initial object $\mathbf{0}$ is *strict* if, for any arrow $f : X \rightarrow \mathbf{0}$, the domain X is also initial.

Figure 2.6: Hasse diagram for lattice of subgraphs



Example 2.28. In $\mathbb{S}\mathrm{et}$, the empty set is a strict initial object. In $\mathbb{G}\mathrm{raph}$, the empty graph is a strict initial object. In the category of sets and partial functions, the empty set is also initial, but it is not strict: there is a partial function $f : \{1\} \to \varnothing$, but $\{1\}$ is not initial.

Fact 2.29. If any initial object is strict, all initial objects are, since they are isomorphic. Furthermore, if $\mathbf{0}$ is strict, then every arrow $!_X : \mathbf{0} \to X$ is monic.

Fact 2.30. If category $\mathbb{C}$ has a strict initial object $\mathbf{0}$, then for each object X of $\mathbb{C}$ the bottom subobject $\perp \in \mathbf{Sub}(X)$ is determined by the strict initial object. That is, $\perp \cong\, !_X \in \mathbf{Sub}(X)$.

2.3 Rules, Matches and Transformations

In this section, we formally introduce the fundamental concepts of the Double-Pushout (DPO) approach to graph transformation. An important aspect of this approach is that the definitions and theorems are stated in categorical terms. Then, the entire theory can be instantiated for different notions of “transformed object”, instead of just graphs, as long these objects live in a category with the necessary properties. For a more detailed discussion, we refer to the book by Ehrig et al. (2006).

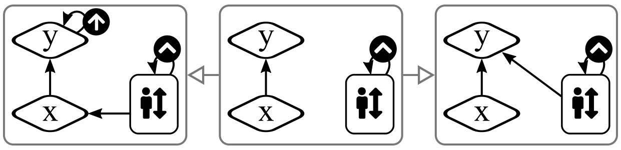
We begin by giving a categorical account of *rules*, which must not only contain objects representing the pre- and post-conditions, but also characterise what the rule deletes, creates and preserves. Thus, besides having a pair of objects, a rule also contains their intersection, defining what is preserved, along with embeddings into the pre- and post-condition objects. The part of the pre-condition outside the intersection is then deleted, while creation is characterised analogously with the post-condition.

Definition 2.31. A *span* is a pair of morphisms $A \xleftarrow{f} C \xrightarrow{g} B$ with shared domain. Dually, a *cospan* is a pair of morphisms $A \xrightarrow{f} C \xleftarrow{g} B$ with shared codomain.

Definition 2.32 (Rule)**.** A *rule* is a span $\rho = L \overset{l}{\hookleftarrow} K \overset{r}{\rightarrowtail} R$, with monic l and r. We call L and R the *left-* and *right-hand sides*, respectively, while K is called the *interface*.

Example 2.33. Several rules for moving an elevator up were described in Example 2.6. Their left- and right-hand sides were given, but interfaces were omitted. In Fig. 2.7, the rule `move-up-NU` is given as a span. Nodes of type `floor` are mapped according to their labels, while the remainder of the mappings is uniquely determined.

Figure 2.7: Rule `move-up-NU` as a span

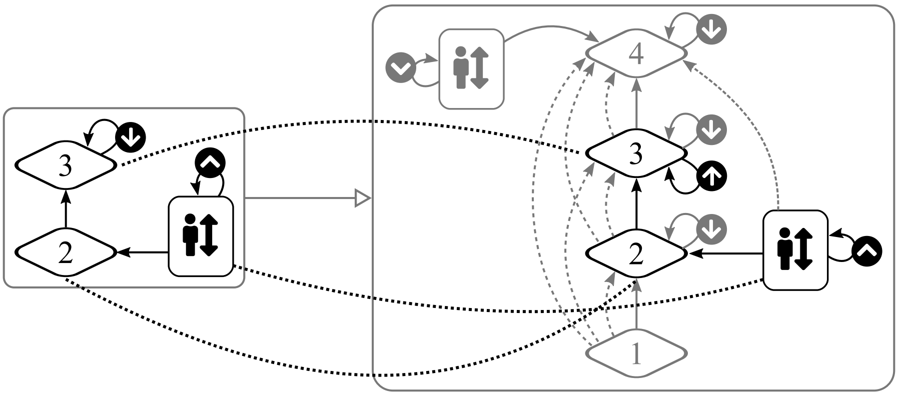


Recall that, in order for a rule to be applied, it requires a *match* in the system state G. That is, there must be a subgraph isomorphic to the pre-conditions graph L. This can be formalised as a mono $m : L \rightarrowtail G$, which represents a subobject of the system state.

Definition 2.34 (Match)**.** Let $\rho = L \overset{l}{\hookleftarrow} K \overset{r}{\rightarrowtail} R$ be a rule and G an arbitrary object, representing the system state. A *match for ρ in G* is a monomorphism $m : L \rightarrowtail G$.

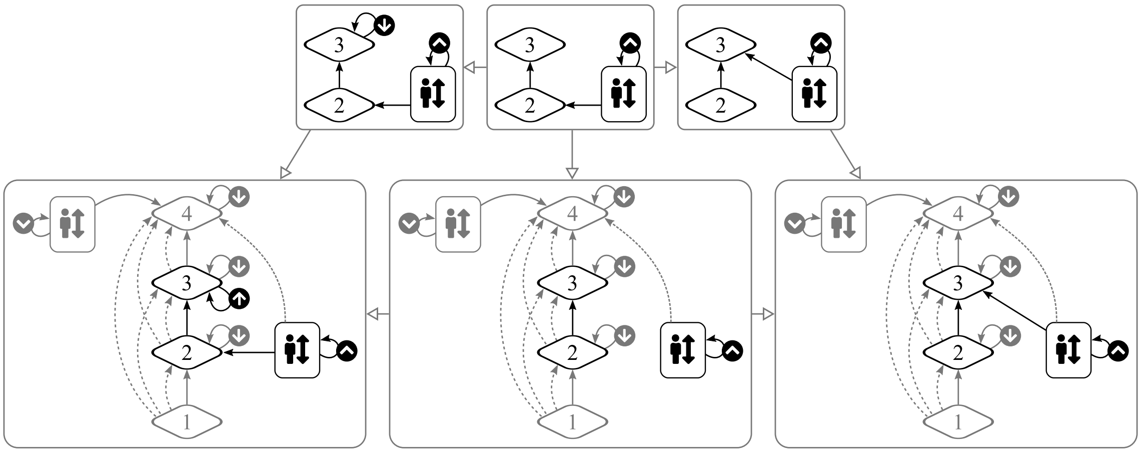
Example 2.35. Recall the match for rule `move-up-NU`, as shown in Example 2.6. In Fig. 2.8, it is presented as a graph morphism from the left-hand side to the system state, where the mapping of nodes is presented with dotted lines. The mapping of edges is omitted for readability, since it is uniquely determined from the nodes.

Figure 2.8: Match morphism for rule `move-up-NU`



The application of a rule at a match is then defined using pushouts as a gluing construction, as discussed extensively by Ehrig et al. (2006).

Figure 2.9: Transformation step with rule `move-up-NU`



Definition 2.36 (Transformation)**.** Given a match $m : L \rightarrowtail G$ for rule ρ, a *transformation step* $t : G \overset{\rho,m}{\Longrightarrow} H$ corresponds to a diagram like (2.12), where both squares are pushouts. In an adhesive category, the left pushout is unique up to isomorphism when it exists, guaranteeing that the transformation caused by a particular match is deterministic.

$$\begin{tikzcd}
L \arrow[d, tail, "m"'] & K \arrow[l, tail, "l"'] \arrow[r, tail, "r"] \arrow[d, tail, "k"] & R \arrow[d, tail, "m'"] \\
G & D \arrow[l, tail, "l'"] \arrow[r, tail, "r'"'] & H
\end{tikzcd} \tag{2.12}$$

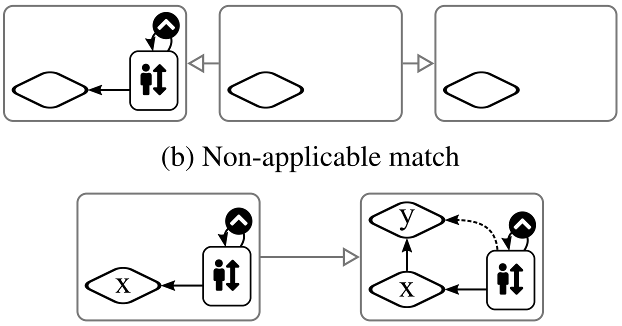
Example 2.37. Recall the rule `move-up-NU`, given in Example 2.33, and its match given in Example 2.35. The corresponding step is show in Fig. 2.9. Note that the pushout square on the left deletes the necessary elements, while the pushout on the right creates.

Note that not every match determines a transformation step, since the existence of a pushout complement is not guaranteed.

Example 2.38 (Non-Applicable Match)**.** Consider an office building that, in order to save electricity, deactivates some of the elevators at night and on weekends. Assume this behaviour was modelled with a rule `deactivate` as shown in Fig. 2.10a, which deletes the corresponding `elevator` node. The match shown in Fig. 2.10b does *not* determine a transformation step. In fact, the necessary pushout complement does not exist because it would have to delete the `elevator` node, but preserve its incident `req-stop` edge. This problem is commonly referred to as a *dangling edge*.

Finally, when modelling with graph transformation, we usually assemble a *transformation system* by combining a set of rules with an initial state.

Figure 2.10: Non-applicable match with rule `deactivate`



(a) Rule `deactivate`

(b) Non-applicable match

Definition 2.39 (Transformation System)**.** A *transformation system* consists of a set $\mathscr{R}$ of rules and an object G_0 representing the initial state.

2.4 Conflicts and Parallel Independence

An important tool for understanding the behaviour of a transformation system is the notion of *parallel independence*, which ensures that two transformation steps do not interfere with each other. Essentially, steps $H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ are parallel independent when there exist residual matches $m'_1 : L_1 \to H_2$ and $m'_2 : L_2 \to H_1$ as well as steps $H_1 \overset{\rho_2,m'_2}{\Longrightarrow} H$ and $H_2 \overset{\rho_1,m'_1}{\Longrightarrow} H$ reaching the same state[1]. If they are not parallel independent, it is said they are *in conflict*.

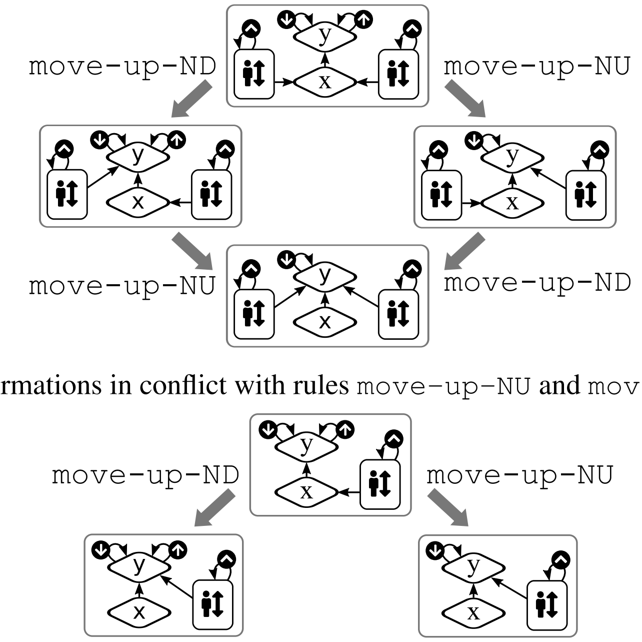
Example 2.40. Recall the rules `move-up-NU` and `move-up-ND` from Example 2.6, which move an elevator up into a floor with a request of type `req-up` or `req-down`, respectively. Fig. 2.11a shows their application to a state with two floors and two elevators, each of the rules acting on a different elevator. These transformations are parallel independent, as the converging steps of Fig. 2.11a shows.

Example 2.41. A different case is shown in Fig. 2.11b, where the rules `move-up-NU` and `move-up-ND` act on the same elevator. These transformations disable each other mutually: the existence of residual matches is hindered when the `at` edge is deleted.

The intuition behind parallel independence is that the elements matched by either rule are preserved by the other, and thus still available after the application. Ehrig (1978) formalised this in set-theoretical terms as follows: steps $(t_1, t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ are

[1] Note that this is related to, but distinct from, *local confluence*. The latter allows the transformations $H_1 \Longrightarrow^* H$ and $H_2 \Longrightarrow^* H$ to contain *multiple* steps, possibly involving other rules of the system.

Figure 2.11: Examples of parallel independence



(a) Parallel independent transformations with rules `move-up-NU` and `move-up-ND`

(b) Transformations in conflict with rules `move-up-NU` and `move-up-ND`

parallel independent when the intersection of the matches is exactly the intersection of preserved elements: $m_1(L_1) \cap m_2(L_2) = m_1(l_1(K_1)) \cap m_2(l_2(K_2))$. In order to prove that this implies the existence of $H_1 \overset{\rho_2,m_2'}{\Longrightarrow} H$ and $H_2 \overset{\rho_1,m_1'}{\Longrightarrow} H$, Ehrig (1978) has shown it equivalent to Definition 2.42. Being stated in categorical terms, it was then taken as the standard definition of parallel independence, as described in the foundational texts by Corradini et al. (1997) and Ehrig et al. (2006).

Definition 2.42. A pair of transformation steps $(t_1, t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ satisfies the *standard condition of parallel independence* if and only if there exist monos $q_{12} : L_1 \rightarrowtail D_2$ and $q_{21} : L_2 \rightarrowtail D_1$ making diagram (2.13) commute.

$$\begin{tikzcd}
R_1 \arrow[d, "n_1"'] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "k_1"'] & L_1 \arrow[dr, "m_1"] \arrow[drrr, dotted, tail, "q_{12}"' near end] & & L_2 \arrow[dl, "m_2"'] \arrow[dlll, dotted, tail, "q_{21}" near end] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "k_2"] & R_2 \arrow[d, "n_2"] \\
H_1 & D_1 \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & G & & D_2 \arrow[ll, "g_2"'] \arrow[r, "h_2"] & H_2
\end{tikzcd} \tag{2.13}$$

More recently, Corradini et al. (2018) noted that, although this categorical formulation is equivalent to the set-theoretical version, its intuitive meaning is different: the former emphasises the existence of residual matches, while the latter emphasises the preservation of required elements. To capture the latter intuition categorically, the *essential condition* was proposed, using pullbacks as a categorical generalisation of intersections. Corradini et al. (2018) proved this is equivalent to the standard condition in adhesive categories. In this thesis, we take the essential condition as the definition of parallel independence.

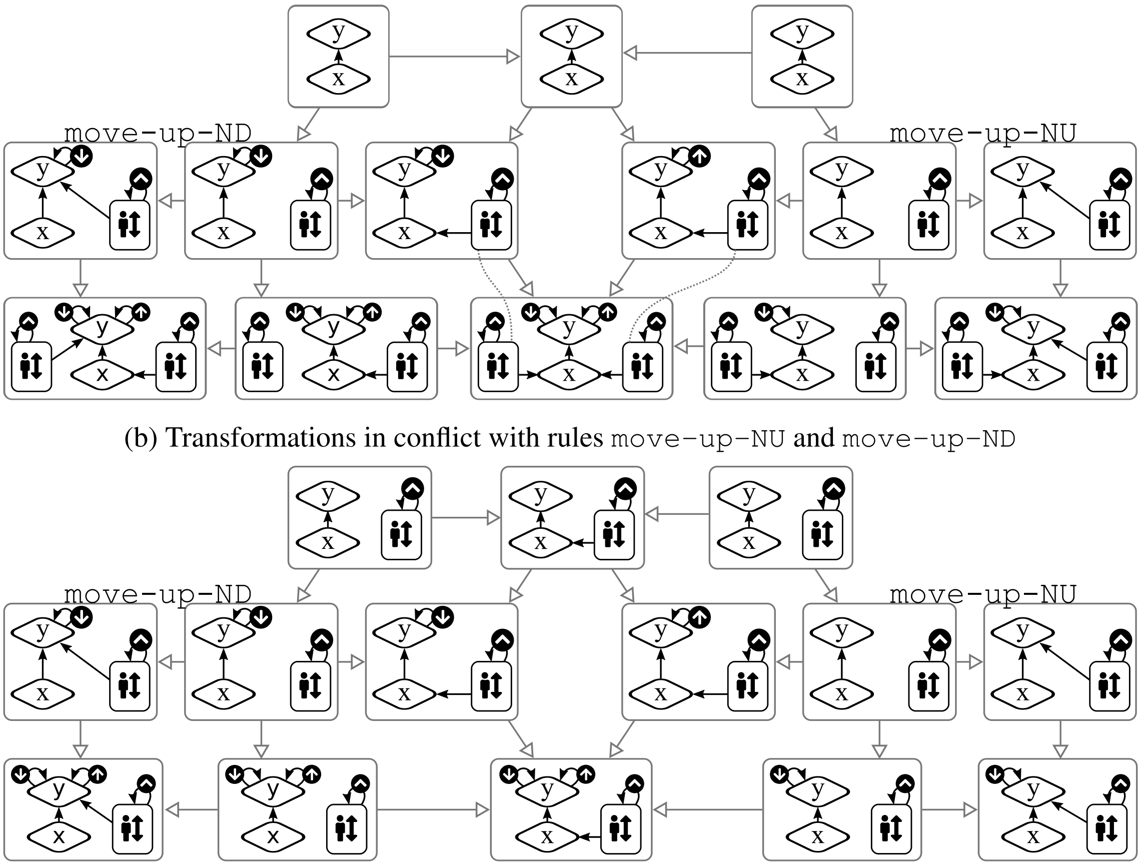
Definition 2.43 (Parallel Independence)**.** Transformation steps $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ are *parallel independent* if and only if, given pullbacks ①, ② and ③ as in diagram (2.14), the arrows $K_1L_2 \to L_1L_2$ and $L_1K_2 \to L_1L_2$ are isomorphisms.[2]

$$\begin{tikzcd}
 & K_1L_2 \arrow[rr, "\cong"] \arrow[d] \arrow[dr, phantom, "\lrcorner"] \arrow[r, phantom, "\textcircled{2}" near end, shift right=4] & & L_1L_2 \arrow[dl, "p_1"'] \arrow[dr, "p_2"] \arrow[d, phantom, "\vee"] & & L_1K_2 \arrow[ll, "\cong"'] \arrow[d] \arrow[dl, phantom, "\llcorner"] \arrow[l, phantom, "\textcircled{3}" near end, shift left=4] & \\
R_1 \arrow[d, "n_1"'] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "k_1"'] & L_1 \arrow[dr, "m_1"] \arrow[rr, phantom, "\textcircled{1}"] & & L_2 \arrow[dl, "m_2"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "k_2"] & R_2 \arrow[d, "n_2"] \\
H_1 & D_1 \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & G & & D_2 \arrow[ll, "g_2"'] \arrow[r, "h_2"] & H_2
\end{tikzcd} \qquad (2.14)$$

Remark 2.44. If t_1 and t_2 are not parallel independent, we say they are *in conflict*. When $K_1L_2 \to L_1L_2$ is not an isomorphism, we say t_1 *disables* t_2; when $L_1K_2 \to L_1L_2$ is not an isomorphism, we say t_2 disables t_1.

Figure 2.12: Examples for the essential condition of parallel independence

(a) Parallel independent transformations with rules `move-up-NU` and `move-up-ND`



(b) Transformations in conflict with rules `move-up-NU` and `move-up-ND`

[2] Since l_1 and l_2 are monos, this is equivalent to requiring existence of arrows $L_1L_2 \to K_1$ and $L_1L_2 \to K_2$ making the resulting triangles commute. However, this simpler condition is not helpful for the characterisation of conflicts to be proposed in Chapter 5.

Example 2.45. Recall the rules `move-up-NU` and `move-up-ND` from Example 2.6, which move an elevator up into a floor with a request of type `req-up` or `req-down`, respectively. Fig. 2.12a shows their application to a state with two floors and two elevators, each of the rules acting on a different elevator. These steps are parallel independent, as Fig. 2.12a shows: computing the pullbacks, the expected arrows are isomorphisms.

Example 2.46. A different case is shown in Fig. 2.12b, where the rules `move-up-NU` and `move-up-ND` act on the same elevator. We would not expect them be parallel independent, which is indeed the case: neither of the necessary arrows are isomorphisms, since the `at` edge is deleted. Thus, the transformation steps are in conflict. In fact, they disable each other mutually.

2.5 Conflict Detection

The concepts introduced in the previous section describe the conflicts between two particular *transformation steps*. When analysing a transformation system, however, one requires an overview of all *potential* conflicts for each pair of *rules*. However, since the set of potential conflicts is usually infinite, enumerating them is not viable.

The problem of conflict detection is then to find a *finite set* of conflicts while still expressing all possible interactions. This is formally grounded on the *extension* of transformations into larger contexts, such that extended transformations have essentially the same behaviour as the original ones. Thus, the finite set of conflicts produced by conflict detection must generate all possible conflicts by extending into larger contexts.

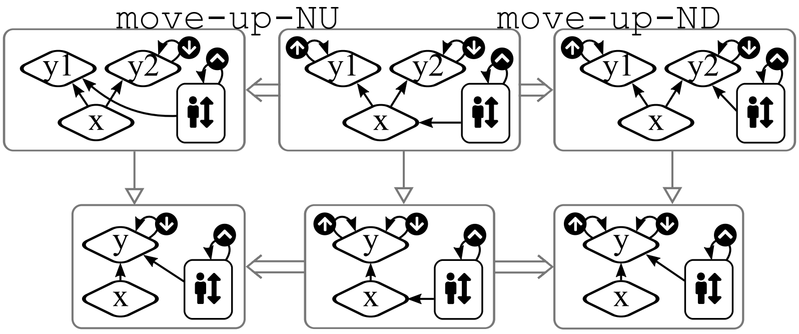
Definition 2.47 (Extension)**.** An *extension diagram* over transformation step $\bar{t} : \overline{G} \overset{\rho,\overline{m}}{\Longrightarrow} \overline{H}$ and *extension morphism* $e : \overline{G} \to G$ is a diagram like (2.15) where $m = f \circ \overline{m}$ is monic and there is some $t : G \overset{\rho,m}{\Longrightarrow} H$ defined by the four pushout squares of diagram (2.16). When extension diagram (2.15) exists, we say $\bar{t}$ is *embedded into* t, and t is an *extension* of $\bar{t}$.

$$\begin{tikzcd}
\overline{G} \arrow[r, Rightarrow, "\bar{t}"] \arrow[d, "e"'] & \overline{H} \arrow[d, "f"] \\
G \arrow[r, Rightarrow, "t"] & H
\end{tikzcd} \qquad (2.15)$$

$$\begin{tikzcd}
L \arrow[d, tail, "\overline{m}"'] \arrow[dr, phantom, "\urcorner", very near start] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, tail, "\overline{k}"] & R \arrow[d, tail, "\overline{n}"] \arrow[dl, phantom, "\ulcorner", very near start] \\
\overline{G} \arrow[d, "e"'] \arrow[dr, phantom, "\urcorner", very near start] & \overline{D} \arrow[l, "\overline{g}"'] \arrow[r, "\overline{h}"] \arrow[d, "d"] & \overline{H} \arrow[d, "f"] \arrow[dl, phantom, "\ulcorner", very near start] \\
G & D \arrow[l, "g"'] \arrow[r, "h"] & H
\end{tikzcd} \qquad (2.16)$$

Example 2.48. Figure 2.13 presents extension diagrams with the rules `move-up-NU` and `move-up-ND` from Example 2.6.

Figure 2.13: Extension diagrams



Problem 2.49 (Conflict Detection)**.** Given rules ρ_1 and ρ_2, find a *finite* set S of conflicts such that, for every conflict $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ there exists some pair $(\overline{t_1},\overline{t_2}) \in S$ that can be extended into (t_1,t_2) as in diagram (2.17).

$$\begin{tikzcd}
\overline{H_1} \arrow[d] & \overline{G} \arrow[l, Leftarrow, "\overline{t_1}"'] \arrow[r, Rightarrow, "\overline{t_2}"] \arrow[d, dashed, "f"] & \overline{H_2} \arrow[d] \\
H_1 & G \arrow[l, Leftarrow, "t_1"'] \arrow[r, Rightarrow, "t_2"] & H
\end{tikzcd} \tag{2.17}$$

The problem of conflict detection is usually formulated differently, in terms of a *completeness theorem* for a particular set of conflicts (e.g. Theorem 2.56). We introduce this formulation to emphasise the fact that different solutions may exist.

2.5.1 Critical Pairs

The most common solution to conflict detection is given by *critical pairs*, adapted from the context of term rewriting. Critical pairs are pairs of conflicting transformation steps in a *minimal context*, where the state is completely covered by the matches. This is formalised categorically as a jointly epic pair of matches.

Definition 2.50 (Jointly Epic)**.** A pair of morphisms $X \xrightarrow{f} Z \xleftarrow{g} Y$ with same codomain is *jointly epic* when $h_1 \circ f = h_2 \circ f$ and $h_1 \circ g = h_2 \circ g$ implies $h_1 = h_2$ for all $h_1, h_2 : Z \to Z'$.

$$\begin{tikzcd}
X \arrow[dr, "f"] & & \\
& Z \arrow[r, shift left, "h_1"] \arrow[r, shift right, "h_2"'] & Z' \\
Y \arrow[ur, "g"'] & &
\end{tikzcd} \tag{2.18}$$

Fact 2.51. Every pushout is jointly epic.

Fact 2.52. In $\mathbb{Set}$, $\mathbb{Graph}$ and $\mathbb{Graph}_T$, a pair of morphisms $X \xrightarrow{f} Z \xleftarrow{g} Y$ is jointly epic if and only if it is jointly surjective, that is, when $f(X) \cup g(Y) = Z$.

Definition 2.53 (Critical Pair)**.** A *critical pair* is a conflict $(t_1, t_2) : H_1 \overset{\rho_1, m_1}{\Longleftarrow} G \overset{\rho_2, m_2}{\Longrightarrow} H_2$ such that the matches (m_1, m_2) are jointly epic.

Example 2.54. The conflicting transformation steps caused by rules `move-up-NU` and `move-up-ND` given in Example 2.46 are a critical pair. On the other hand, the steps caused by rules `move-up-NU` and `move-up-AS` given in Example 2.6 are not a critical pair. Although they are in conflict, they are not in a minimal context. In fact, many elements of the system state are outside the images of the matches, including the first floor, the elevator on the left and most edges of type `below`.

Inspecting critical pairs is viable because, given rules with finite left-hand sides, the set of critical pairs is also finite. Enumerating critical pairs for every pair of rules is called *critical pair analysis* and is the basis for many applications which will be presented in Chapter 8. On the other hand, inspecting critical pairs is guaranteed to uncover all potential conflicts because every pair of conflicting transformation steps is the *extension* of a critical pair, having essentially the same effect in an enlarged context. This holds in categories with unique epi-mono pair factorisations, that is, where every pair of matches can be uniquely decomposed into a jointly epic pair followed by a monomorphism.

Definition 2.55. A category has *unique epi-mono pair factorisation* when, given morphisms m_1 and m_2 as in diagram (2.19), there exists a unique jointly epic pair $(\overline{m_1}, \overline{m_2})$ and monomorphism f making diagram (2.19) commute, up to isomorphism.

$$\begin{tikzcd}
L_1 \arrow[drr, bend left, "m_1"] \arrow[dr, "\overline{m_1}"'] & & \\
& \overline{G} \arrow[r, dashed, "f"] & G \\
L_2 \arrow[urr, bend right, "m_2"'] \arrow[ur, "\overline{m_2}"] & &
\end{tikzcd} \tag{2.19}$$

Theorem 2.56 (Completeness of Critical Pairs)**.** In any adhesive category with unique epi-mono pair factorisation, for every pair $(t_1, t_2) : H_1 \overset{\rho_1, m_1}{\Longleftarrow} G \overset{\rho_2, m_2}{\Longrightarrow} H_2$ of conflicting transformation steps there exists a critical pair $(\overline{t_1}, \overline{t_2}) : \overline{H_1} \overset{\rho_1, \overline{m_1}}{\Longleftarrow} \overline{G} \overset{\rho_2, \overline{m_2}}{\Longrightarrow} \overline{H_2}$ that can be embedded into (t_1, t_2) as in diagram (2.20).

$$\begin{tikzcd}
\overline{H_1} \arrow[d] & \overline{G} \arrow[l, Rightarrow, "\overline{t_1}"'] \arrow[r, Rightarrow, "\overline{t_2}"] \arrow[d, dashed, "f"] & \overline{H_2} \arrow[d] \\
H_1 & G \arrow[l, Rightarrow, "t_1"'] \arrow[r, Rightarrow, "t_2"] & H
\end{tikzcd} \tag{2.20}$$

Proof. Shown, for example, by Ehrig et al. (2006). □

2.5.2 Initial Conflicts

Despite being finite, sets of critical pairs are often very large. Indeed, there is often redundancy such as in Fig. 2.13, where both the upper and lower pairs of steps are critical pairs, but one is an extension of the other. In order to curb this redundancy, Lambers et al. (2018) proposed a notion of initiality with respect to extension.

Definition 2.57 (Initial Transformation Pair)**.** Given a pair of transformation steps $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$, the pair $(s_1,s_2) : J_1 \overset{\rho_1,n_1}{\Longleftarrow} I \overset{\rho_2,n_2}{\Longrightarrow} J_2$ is an *initial transformation pair* for (t_1,t_2) if it satisfies both of the following.

(i) The pair (s_1,s_2) can be embedded into (t_1,t_2) as in diagram (2.21).

$$\begin{tikzcd}
J_1 \arrow[d] & I \arrow[l, Rightarrow, "s_1"'] \arrow[r, Rightarrow, "s_2"] \arrow[d, dashed, "f"] & J_2 \arrow[d] \\
H_1 & G \arrow[l, Rightarrow, "t_1"'] \arrow[r, Rightarrow, "t_2"] & H_2
\end{tikzcd} \tag{2.21}$$

(ii) For every pair $(\overline{t_1},\overline{t_2}) : \overline{H_1} \overset{\rho_1,\overline{m_1}}{\Longleftarrow} \overline{G} \overset{\rho_2,\overline{m_2}}{\Longrightarrow} \overline{H_2}$ that can be embedded into (t_1,t_2) as in the lower part of diagram (2.22), then (s_1,s_2) can be embedded into $(\overline{t_1},\overline{t_2})$ as in the upper part of the diagram.

$$\begin{tikzcd}
J_1 \arrow[d] & I \arrow[l, Rightarrow, "s_1"'] \arrow[r, Rightarrow, "s_2"] \arrow[d, dashed, "h"] & J_2 \arrow[d] \\
\overline{H_1} \arrow[d] & \overline{G} \arrow[l, Rightarrow, "\overline{t_1}"'] \arrow[r, Rightarrow, "\overline{t_2}"] \arrow[d, dashed, "g"] & \overline{H_2} \arrow[d] \\
H_1 & G \arrow[l, Rightarrow, "t_1"'] \arrow[r, Rightarrow, "t_2"] & H_2
\end{tikzcd} \tag{2.22}$$

Initial transformation pairs are not guaranteed to exist in any category, but Lambers et al. (2018) proved they exist in categories of typed graphs. When they do exist, they are a subset of critical pairs that also solves the problem of conflict detection. Moreover, since they are often much less numerous, they are a more efficient solution. Indeed, a pair of relatively simple rules provided by Lambers et al. (2018) has 21,478 critical pairs, but only 7 initial conflicts. More evidence of this will be provided in Chapter 7.

Definition 2.58 (Initial Conflict)**.** A conflict $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ is *initial* when it is isomorphic to its initial transformation pair.

Example 2.59. The upper pair shown in Fig. 2.13 is an initial conflict.

2.6 Initial Pushouts

An important tool for the theoretical development of this thesis is the initial pushout. In many categories, it characterises the "differences" between the domain and codomain of a morphism. In particular, it identifies parts of the codomain that are missing from the domain, serving as a kind of "complement" for subobjects.[3]

Definition 2.60 (Initial pushout)**.** Given a morphism $f : X \to Y$, diagram (2.23) is an *initial pushout (over f)* when, for any pushout ③ with monic x and y, there are unique arrows b^* and c^* making diagram (2.24) commute and square ② a pushout. That is, pushout (2.23) is initial with respect to all pushouts ③: pushouts along mono resulting in f.

$$\begin{tikzcd}
B \arrow[r, tail, "b"] \arrow[d, "\overline{f}"'] & X \arrow[d, "f"] \\
C \arrow[r, tail, "c"] & Y
\end{tikzcd} \qquad (2.23)$$

$$\begin{tikzcd}
B \arrow[r, dotted, "b^*"] \arrow[d, "\overline{f}"'] \arrow[rr, tail, bend left, "b"] \arrow[dr, phantom, "\textcircled{2}"] & U \arrow[r, tail, "x"] \arrow[d, "g"] \arrow[dr, phantom, "\textcircled{3}"] & X \arrow[d, "f"] \\
C \arrow[r, dotted, "c^*"] \arrow[rr, tail, bend right, "c"'] & V \arrow[r, tail, "y"] & Y
\end{tikzcd} \qquad (2.24)$$

Remark 2.61. The subobject $b : B \to X$ is the *boundary* of f, while $c : C \to Y$ is the *context*.

Fact 2.62 (GABRIEL et al., 2014)**.** In an adhesive category, any commutative square ② is a pushout when ③ and (2.23) are pushouts.

Lemma 2.63. In adhesive categories, a context completely determines the initial pushout.

Proof. Since initial pushouts are along mono, they are also pullbacks. Thus, the initial pushout for a morphism f can be constructed by pulling back the context along f. □

Note that initial pushouts do not exist in every category. Nonetheless, they do have a well-known construction in categories of graphs and typed graphs.

Construction 2.64 (EHRIG et al., 2006)**.** Let $f : X \to Y$ be a graph morphism, with $Y = (V, E, s, t)$. In order to construct the initial pushout of f, by Lemma 2.63, we need only construct the context. Thus, we will construct the context graph C as a subgraph of Y, taking the inclusion $c : C \hookrightarrow Y$ as context morphism.

The context graph $C = (V', E', s', t')$ for f can be constructed as follows. We denote by $\mathrm{card}(X)$ the cardinality of set X, by $f[S] = \{f(x) \mid x \in S\}$ the image of a set and

[3] It is actually the dual notion of a pseudo-complement, but a lattice-theoretical characterisation of initial pushouts is beyond the scope of this thesis.

by $f^{-1}[S] = \{x \in X \mid f(x) \in S\}$ the preimage.

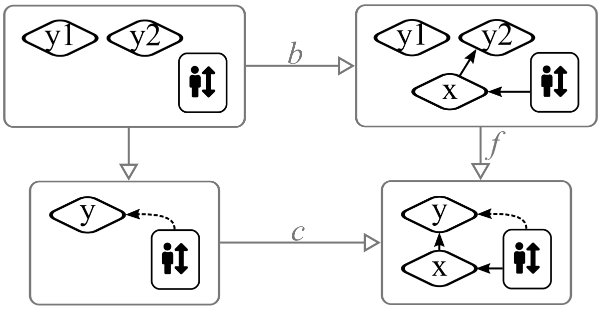
$$E' = \left\{e \in E \mid \operatorname{card}\left(f^{-1}[\{e\}]\right) \neq 1\right\} \qquad s'(e) = s(e) \quad \text{for } e \in E'$$

$$V' = \left\{v \in V \mid \operatorname{card}\left(f^{-1}[\{v\}]\right) \neq 1\right\} \cup s[E'] \cup t[E'] \qquad t'(e) = t(e) \quad \text{for } e \in E'$$

Informally, we select nodes and edges of Y that are either outside the image of f, or are the identification by f of multiple elements from X. We also add the necessary source and target nodes for the graph to be well-defined. The source and target functions are just restricted on their domain.

Example 2.65. Fig. 2.14 shows an initial pushout in the category of typed graphs for the elevator control system, that is, with the type graph from Example 2.4. Note that the context graph contains floor y, since it has two preimages under f, and the `req-stop` edge from the elevator to floor y, since it is outside the image of f. It also contains the elevator node, even though it has a unique preimage, because it is the source of the `req-stop` edge.

Figure 2.14: Initial pushout of typed graphs



Moreover, Gabriel et al. (2014) have shown the existence of initial pushouts in all finitary adhesive categories, which are adhesive categories where each object has a finite set of subobjects. These are appropriate for most applications, since rules are almost always composed of finite graphs, and system states are also usually finite.

Definition 2.66 (Finite Object, Finitary Category)**.** An object X is *finite* if it has a finite number of subobjects, that is, if the poset $\mathbf{Sub}(X)$ is finite. A category is *finitary* if all its objects are finite. The *finitary restriction* of a category is its full subcategory containing exactly the finite objects.

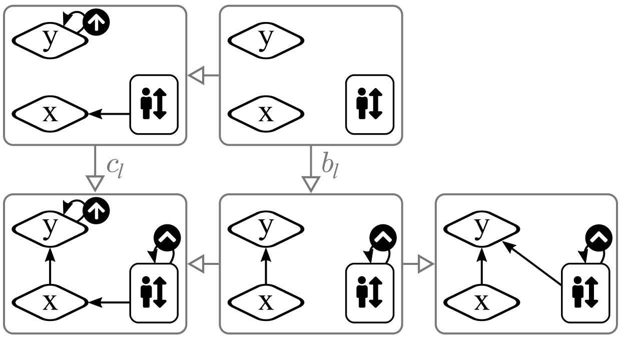
Lemma 2.67 (GABRIEL et al., 2014)**.** The finitary restriction of an adhesive category is also adhesive. In a finitary adhesive category, all morphisms have initial pushouts.

Given a transformation rule, initial pushouts can be used to determine which parts of the left-hand side are deleted.

Definition 2.68. Given rule $\rho = L \overset{l}{\leftarrowtail} K \overset{r}{\rightarrowtail} R$, its *deletion object* $c_l \in \mathbf{Sub}(L)$ is the context of the initial pushout over l.

Example 2.69. The deletion object for rule `move-up-NU` is shown in Fig. 2.15.

Figure 2.15: Deletion object for rule `move-up-NU`.



Moreover, initial pushouts can be used to determine if a match is applicable, that is, if the necessary pushout complement exists. This is the so-called *gluing condition.*

Lemma 2.70 (Gluing Condition)**.** In any adhesive category, let $l : K \rightarrowtail L$ and $m : L \rightarrowtail G$ be monomorphisms and square ② of diagram (2.25) be an initial pushout over m. Then there exists a pushout square ① if and only if there is a morphism b^* with $b_m = l \circ b^*$.

$$\begin{tikzcd}
K \arrow[r, tail, "l"] \arrow[d, dashed, "k"'] \arrow[rd, phantom, "\textcircled{1}"] & L \arrow[d, "m"] \arrow[rd, phantom, "\textcircled{2}"] & B_m \arrow[l, "b_m"'] \arrow[d, "\overline{m}"] \arrow[ll, bend right, dotted, "b^*"'] \\
D \arrow[r, dashed, "g"'] & G & C_m \arrow[l, "b_m"]
\end{tikzcd} \tag{2.25}$$

Proof. Shown, for example, by Ehrig et al. (2006). □

Since the context of an initial pushout should contain the parts of the codomain that are "different" from the domain, the context of an isomorphism should be empty in some sense. It is easy to see that this is the case for categories of graphs: each element has a unique preimage under a bijection, thus no elements are in the context. Indeed, this holds for the lattice-theoretical generalisation of "emptiness" as the bottom subobject, as described in Remark 2.26. Then in all adhesive categories, the context $c \in \mathbf{Sub}(Y)$ of (the initial pushout over) an isomorphism is the bottom subobject.

Lemma 2.71. In any adhesive category, let $f : X \rightarrowtail Y$ be a mono and $c \cong \bot \in \mathbf{Sub}(Y)$. Then the square of diagram (2.26) is an initial pushout over f if and only if it commutes and f is an isomorphism.

$$\begin{tikzcd}
B \arrow[r, tail, "b"] \arrow[d, "\overline{f}"'] & X \arrow[d, tail, "f"] \\
C \arrow[r, tail, "c"'] & Y
\end{tikzcd} \tag{2.26}$$

Proof. First note that $\overline{f}$ is an isomorphism. In fact, we have the subobject $f \circ b \in \mathbf{Sub}(Y)$ with $f \circ b \subseteq_Y c \cong \bot$, as witnessed by $\overline{f}$. This implies that $f \circ b \cong \bot$, that is, the witness $\overline{f}$ must be an isomorphism.

[Only if part] If the square is an initial pushout and $\overline{f}$ is an isomorphism, then so is f, since in adhesive categories pushouts preserve isomorphisms (Theorem 2.20).

[If part] Assume the square commutes and f is an isomorphism. Since $\overline{f}$ is also an isomorphism, the square is a trivial pushout. It remains to show it is initial.

$$\begin{tikzcd}
B \arrow[r, dotted, "u"] \arrow[rr, tail, bend left, "b"] \arrow[d, tail, "\overline{f}"] & U \arrow[r, tail, "x"] \arrow[d, "h"] & X \arrow[d, tail, "f"] \\
C \arrow[r, dotted, "v"] \arrow[rr, tail, bend right, "c"'] & V \arrow[r, tail, "y"] & Y
\end{tikzcd} \tag{2.27}$$

Assume the right square of diagram (2.27) is a pushout. We obtain a mono v with $c = y \cdot v$ because $y \subseteq_Y \bot \cong c$. Analogously, we obtain u with $f \circ b = f \circ x \circ u$ and, since f is monic, $b = x \circ u$. Finally, the left square of diagram (2.27) commutes because y is monic and $y \circ v \circ \overline{f} = c \circ \overline{f} = f \circ b = f \circ x \circ u = y \circ h \circ u$. □

Finally, we present a few important properties of initial pushouts. Interestingly, they are idempotent. Moreover, they are both preserved and reflected by pushouts along monos, as shown by Ehrig et al. (2006). We present a slightly modified statement of these results, emphasising their relation to pushout squares instead of double-pushout diagrams.

Lemma 2.72 (IPO Reflected by PO Along Mono)**.** In any category, every unique pushout induced by an initial pushout is also initial. That is, if rectangle ① + ② in diagram (2.28) is an initial pushout, square ② is a pushout with monic x and y, and square ① is the unique pushout making the diagram commute, then square ① is also an initial pushout.

$$\begin{tikzcd}
B \arrow[r, dotted, "b^*"] \arrow[rr, tail, bend left, "b"] \arrow[d, "\overline{f}"'] \arrow[dr, phantom, "①"] & U \arrow[r, tail, "x"] \arrow[d, "g"'] \arrow[dr, phantom, "②"] & X \arrow[d, "f"] \\
C \arrow[r, dotted, "c^*"] \arrow[rr, tail, bend right, "c"'] & V \arrow[r, tail, "y"] & Y
\end{tikzcd} \tag{2.28}$$

Proof. Assume diagram (2.29) is a pushout with monic u and v. We need to show the existence of unique b^{**} and c^{**} making diagram (2.30) commute, and its front-left face a pushout. This follows from the fact that ③ + ② is a pushout, providing the unique morphisms, and that x and y are monic, making the whole diagram commute. □

$$\begin{tikzcd}
U' \arrow[r, tail, "u"] \arrow[d, "h"'] \arrow[dr, phantom, "\textcircled{3}"] & U \arrow[d, "g"] \\
V' \arrow[r, tail, "v"] & V
\end{tikzcd} \qquad (2.29)$$

$$\begin{tikzcd}
B \arrow[rr, tail, "b^*"] \arrow[dd, "\overline{f}"'] \arrow[dr, dotted, tail, "b^{**}"'] & & U \arrow[r, tail, "x"] \arrow[dd, "g"] \arrow[ddr, phantom, "\textcircled{2}"] & X \arrow[dd, "f"] \\
& U' \arrow[ur, tail, "u"'] \arrow[dd, "h"] & & \\
C \arrow[rr, tail, "c^*"] \arrow[dr, dotted, tail, "c^{**}"'] & & V \arrow[r, tail, "y"] & Y \\
& V' \arrow[ur, tail, "v"'] & &
\end{tikzcd} \qquad (2.30)$$

Corollary 2.73 (Initial Pushouts are Idempotent)**.** If square ② below is an initial pushout over f, then ① is also an initial pushout. Equivalently, square ③ below is an initial pushout over $\overline{f}$ if and only if $\overline{b}$ and $\overline{c}$ are isomorphisms.

$$\begin{tikzcd}
B \arrow[r, tail, "\mathrm{id}_B"] \arrow[d, "\overline{f}"] \arrow[dr, phantom, "\textcircled{1}"] & B \arrow[r, tail, "b"] \arrow[d, "\overline{f}"] \arrow[dr, phantom, "\textcircled{2}"] & X \arrow[d, "f"] \\
C \arrow[r, tail, "\mathrm{id}_C"'] & C \arrow[r, tail, "c"'] & Y
\end{tikzcd} \qquad (2.31) \qquad
\begin{tikzcd}
\overline{B} \arrow[r, tail, "\overline{b}"] \arrow[d, "\overline{\overline{f}}"] \arrow[dr, phantom, "\textcircled{3}"] & B \arrow[r, tail, "b"] \arrow[d, "\overline{f}"] \arrow[dr, phantom, "\textcircled{2}"] & X \arrow[d, "f"] \\
\overline{C} \arrow[r, tail, "\overline{c}"'] & C \arrow[r, tail, "c"'] & Y
\end{tikzcd} \qquad (2.32)$$

Proof. Follows from Lemma 2.72, since id_B and id_C are the unique morphisms making the diagram above left commute. □

Lemma 2.74 (IPO Preserved by PO Along Mono)**.** In any category, if squares ① and ③ in diagram (2.33) are initial pushouts over f and h, while square ② is a pushout with monic g and k, then rectangle ① + ② is an initial pushout. Moreover, the unique arrows b^* and c^* making diagram (2.33) commute are isomorphisms.

$$\begin{tikzcd}
B \arrow[r, tail, "b"] \arrow[d, "\overline{f}"] \arrow[dr, phantom, "\textcircled{1}"] & W \arrow[r, tail, "g"] \arrow[d, "f"] \arrow[dr, phantom, "\textcircled{2}"] & X \arrow[d, "h"] \arrow[dr, phantom, "\textcircled{3}"] & B' \arrow[l, tail, "b'"'] \arrow[d, "\overline{h}"] \arrow[lll, dotted, bend right, "b^*"'] \\
C \arrow[r, tail, "c"] & Y \arrow[r, tail, "k"] & Z & C' \arrow[l, tail, "c'"'] \arrow[lll, dotted, bend left, "c^*"]
\end{tikzcd} \qquad (2.33)$$

Proof. Since ① + ② is a pushout, there are unique morphisms b^* and c^* making the diagram above commute, and diagram (2.34) a pushout. If we show these are isomorphisms, it follows immediately that ① + ② is an initial pushout.

$$\begin{tikzcd}
B' \arrow[r, tail, "b^*"] \arrow[d, "\overline{h}"] \arrow[dr, phantom, "\textcircled{4}"] & B \arrow[d, "\overline{f}"] \\
C' \arrow[r, tail, "c^*"] & C
\end{tikzcd} \qquad (2.34) \qquad
\begin{tikzcd}
B \arrow[r, dotted, "b'"] \arrow[d, "\overline{f}"] \arrow[dr, phantom, "\textcircled{5}"] \arrow[rr, bend left, "b"] & B' \arrow[r, tail, "b \circ b^*"] \arrow[d, "\overline{h}"] \arrow[dr, phantom, "\textcircled{4}+\textcircled{1}"] & W \arrow[d, "f"] \\
C \arrow[r, dotted, "c'"] \arrow[rr, bend right, "c"'] & C' \arrow[r, tail, "c \circ c^*"] & Y
\end{tikzcd} \qquad (2.35)$$

Now consider diagram (2.35). Since $\textcircled{4} + \textcircled{1}$ is a pushout along mono and the outer rectangle is an initial pushout, there are unique morphisms b' and c' making the diagram commute and square $\textcircled{5}$ a pushout. By Lemma 2.72, $\textcircled{5}$ is also an initial pushout, and since initial pushouts are idempotent, b' and c' are isomorphisms. It then follows from diagram chasing that $b \circ b^* = b \circ b'$ and, since b is monic, $b^* = b'$, which is an isomorphism. Analogously, $c^* = c'$ is also an isomorphism. □

3 FUNCTOR CATEGORIES AS GENERALIZED GRAPHS

As explained in the previous chapter, the algebraic approach to graph transformation is largely defined in terms of category theory. This allows its application in many different contexts, using suitable graph-like structures such as labelled, typed or attributed graphs, Petri nets, algebraic specifications or transformation rules themselves.

Nevertheless, some of the latest results regarding the characterisation of conflicts have not been proven categorically. In fact, the three main advances in this field have some results proven specifically for categories of (typed) graphs: the conflict reasons and essential critical pairs of Lambers, Ehrig and Orejas (2008); the conflict atoms, parts and reasons of Born et al. (2017); and the initial conflicts of Lambers et al. (2018). In all of these cases, some of the proofs required a notion of "element" which was not available in the categorial setting.

In this chapter, we propose categories of set-valued functors[1] as a novel context for graph transformation. We will show how this generalises many important graph-like structures, having the necessary properties for graph transformation: adhesivity, epi-mono pair factorisation and initial pushouts. Moreover, it still provides an appropriate notion of "element", allowing proofs to be carried out at a set-theoretical level. The reader is assumed familiar with the notions of functor and natural transformation.

Definition 3.1 (Set-Valued Functor Category)**.** Given any category $\mathbb{S}$, the category $\mathbb{S}\mathrm{et}^{\mathbb{S}}$ has functors $\mathbb{S} \to \mathbb{S}\mathrm{et}$ as objects and natural transformations as arrows.

Remark 3.2. If $t : F \dashrightarrow G$ is a natural transformation between functors $F, G : \mathbb{S} \to \mathbb{S}\mathrm{et}$ and S is an object of $\mathbb{S}$, we denote by $t_S : F(S) \to G(S)$ the component of t on S.

It is easy to see that the category of graphs is isomorphic to $\mathbb{S}\mathrm{et}^{\mathbb{G}}$, where $\mathbb{G}$ is depicted in diagram (3.1). A functor $G : \mathbb{G} \to \mathbb{S}\mathrm{et}$ selects two sets $G(V)$ and $G(E)$ as well as two functions $G(s), G(t) : G(E) \to G(V)$. A natural transformation $f : G \dashrightarrow G'$ has two components $f_V : G(V) \to G'(V)$ and $f_E : G(E) \to G'(E)$, then naturality corresponds to preservation of incidence.

$$\begin{tikzcd} \mathbb{G} = V \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & E \end{tikzcd} \tag{3.1}$$

In fact, set-valued functors generalise *graph structures*, which in turn generalise graphs and many variations (e.g. labelled graphs, hypergraphs, E-graphs), as described by

[1] By duality, these are precisely the categories of *presheaves*, that is, of *contravariant* set-valued functors. Indeed, we use many results from the theory of presheaves, but avoid explicitly considering contravariant functors $\mathbb{S}\mathrm{et}^{\mathbb{S}^{\mathrm{op}}}$ due to notational overhead.

Ehrig et al. (1997).

Definition 3.3 (Graph Structure)**.** A *graph structure signature* $\Sigma = (S, \Omega)$ contains a set S of sorts and a family $\Omega = (\Omega_{s,t} \mid s,t \in S)$ determining operation names. A *graph structure* G is a Σ-algebra of a graph structure signature Σ. That is, $G = (A, O)$ where $A = (A_s \mid s \in S)$ is a family of carrier sets for each sort, along with an assignment of functions to each operation name $O = (O_n : A_s \rightarrow A_t \mid s,t \in S, n \in \Omega_{s,t})$.

Definition 3.4 (Category of Graph Structures)**.** Given a graph structure signature'Σ as well as graph structures A and A', a *graph structure morphism* $f : A \rightarrow A'$ is a family of functions $(f_s \mid s \in S)$ that is compatible with the operations. That is, given an operation O_n for the name $n \in \Omega_{s,t}$, it holds that $O'_n \circ f_s = f_t \circ O_n$. Graph structures for a signature Σ along with their morphisms form the *category of graph structures* $\mathbb{A}\mathrm{lg}_\Sigma$.

Fact 3.5. The category of graphs is the category of graph structures for a signature with the sorts $\{V, E\}$ and the operations $\{s,t\} = \Omega_{E,V}$.

Lemma 3.6. Every category of graph structures is isomorphic to $\mathbb{S}\mathrm{et}^{\mathbb{S}}$ for some small and free category $\mathbb{S}$.

Proof. A graph structure signature Σ defines a graph by taking sorts as nodes and operation symbols as edges. Let $\mathbb{S}$ be the free category generated by this graph. It is easy to see that the category of Σ-algebras is isomorphic to $\mathbb{S}\mathrm{et}^{\mathbb{S}}$. □

Interestingly, the construction of functor and slice categories preserve the nature of set-valued functors.

Lemma 3.7. Let $\mathbb{S}$ and $\mathbb{T}$ be small categories and $C : \mathbb{S} \rightarrow \mathbb{S}\mathrm{et}$ a functor.

(i) The functor category $(\mathbb{S}\mathrm{et}^{\mathbb{S}})^{\mathbb{T}}$ is isomorphic to $\mathbb{S}\mathrm{et}^{\mathbb{S}\times\mathbb{T}}$.

(ii) The slice category $\mathbb{S}\mathrm{et}^{\mathbb{S}} \downarrow C$ is equivalent to $\mathbb{S}\mathrm{et}^{\mathbb{C}}$, for some small category $\mathbb{C}$.

Proof. Item (i) follows from a standard property of exponentials, proven e.g. by MacLane and Moerdijk (2012). Item (ii) was shown in the same book, and in this case $\mathbb{C}$ is the category of elements for the functor C. □

It follows trivially that typed graphs are set-valued functors. Moreover, categories of spans over some $\mathbb{S}\mathrm{et}^{\mathbb{S}}$ are also categories of set-valued functors, since they are just $(\mathbb{S}\mathrm{et}^{\mathbb{S}})^{\mathbb{P}}$ for $\mathbb{P}$ depicted in diagram (3.2). These categories are the basis for higher-order graph transformation, as proposed by Machado, Ribeiro and Heckel (2015), which is useful to express modifications of software in the maintenance or evolution phase of development. Note that such categories are not always graph structures, since the product category with $\mathbb{P}$ is often not free, for example $\mathbb{P} \times \mathbb{G}$.

$$\mathbb{P} = \; L \xleftarrow{\;l\;} K \xrightarrow{\;r\;} R \tag{3.2}$$

Set-valued functor categories $\mathbb{Set}^{\mathbb{S}}$ are particularly well-behaved. It is well known[2] that they inherit a lot of structure from $\mathbb{Set}$, and that many categorical concepts can be considered pointwise for each object of $\mathbb{S}$. When dealing with such concepts, we will apply set-theoretical reasoning to $\mathbb{Set}^{\mathbb{S}}$.

Remark 3.8. Henceforth, let $\mathbb{S}$ be an arbitrary small category.

Lemma 3.9. In $\mathbb{Set}^{\mathbb{S}}$, limits and colimits are constructed pointwise for each object of $\mathbb{S}$. In particular, there is a strict initial object $\mathbf{0}$ composed only of empty sets.

Proof. Proven by MacLane and Moerdijk (2012). □

Lemma 3.10. In $\mathbb{Set}^{\mathbb{S}}$, a morphism $f : X \to Z$ is monic (epic) if and only if each component f_S is injective (surjective). A pair of morphisms $X \overset{f}{\to} Z \overset{g}{\leftarrow} Y$ is jointly epic if and only if each pair of components (f_S, g_S) is jointly surjective.

Proof. Recall that f is monic if and only if its pullback along itself results in the identity, which by the previous lemma is equivalent to having each component f_S be monic. The case for epimorphisms follows by duality. Moreover, the case for jointly epic pairs follows from the fact that a pair (f, g) is jointly epic if and only if the morphism $f + g : X + Y \to Z$, induced by the coproduct and defined componentwise, is an epimorphism. □

Remark 3.11. Statements in $\mathbb{Set}^{\mathbb{S}}$ involving pullbacks, pushouts, monomorphisms, epimorphisms and jointly epic pairs can be proved with set-theoretical reasoning. Then, given $\mathrm{X}, \mathrm{Y} \in \mathbb{Set}^{\mathbb{S}}$ and $f : \mathrm{X} \to \mathrm{Y}$, we will write X, Y and f instead of $\mathrm{X}(S)$, $\mathrm{Y}(S)$ and f_S for an implicit, universally quantified S.

The suitability of set-valued functors as a basis for DPO transformation follows from its adhesivity. It also has unique epi-mono factorisations, necessary for the completeness of critical pairs.

Lemma 3.12. Given a small category $\mathbb{S}$, the category of functors $\mathbb{Set}^{\mathbb{S}}$ is a topos. Then it is adhesive and has unique epi-mono factorisations.

[2] These results are usually stated for presheaves, but they hold trivially for set-valued functors by duality.

Proof. The fact that $\mathbb{S}\mathrm{et}^{\mathbb{S}}$ is a topos was shown, for example, by Johnstone (2002), who has also shown that toposes have unique epi-mono factorisations. Lack and Sobocinski (2005) have shown that toposes are adhesive. □

In the context of graph transformation, we often have commutative squares that are both a pullback and a pushout, including all pushouts along monos. A set-theoretic characterisation will be useful in the following.

Lemma 3.13. In any category of functors $\mathbb{S}\mathrm{et}^{\mathbb{S}}$, let square (3.3) be a pullback. It is also a pushout if and only if both of the following hold for any element $z \in Z$.

(i) If $z \notin f(X)$, there is a unique $y \in Y$ with $z = g(y)$.

(ii) if $z \notin g(Y)$, there is a unique $x \in X$ with $z = f(x)$.

$$\begin{tikzcd}
W \arrow[r, "g'"] \arrow[d, "f'"'] & X \arrow[d, "f"] \\
Y \arrow[r, "g"'] & Z
\end{tikzcd} \tag{3.3}$$

Proof. We will use set-theoretical reasoning as explained in Remark 3.11.

[Only if] Assume square (3.3) is a pushout and take any element $z \in Z$. We will prove item (i) leaving the analogous proof of (ii) to the reader.

Assuming $z \notin f(X)$, since (f,g) is jointly epic it must hold that $z \in g(Y)$. This implies the existence of some $y \in Y$ with $z = g(y)$. To prove its uniqueness, assume $y' \in Y$ with $z = g(y')$.

Recall that the pushout is equivalent to the coequalizer over a coproduct. That is, if j_X and j_Y are the coproduct injections in diagram (3.4), and e is the coequalizer of $(j_X \circ g', j_Y \circ f')$, then $f = e \circ j_X$ and $g = e \circ j_Y$. Since $g(y) = g(y')$ and j_X is monic, it must also hold that $e(y) = e(y')$. Moreover, since e is a coequalizer, there must be $w, w' \in W$ with $f'(w) = y$, $f'(w') = y'$ and $g'(w) = g'(w')$. But then $z = f(g'(w)) \in f(X)$, contradicting our assumption.

$$\begin{tikzcd}
 & X \arrow[d, "j_X"] \arrow[dr, bend left, "f"] & \\
W \arrow[ur, bend left, "g'"] \arrow[dr, bend right, "f'"'] & X+Y \arrow[r, "e"] & Z \\
 & Y \arrow[u, "j_Y"'] \arrow[ur, bend right, "g"'] &
\end{tikzcd} \tag{3.4}$$

[If] Assume that (i) and (ii) hold. We will show that square (3.3) is a pushout by showing it is isomorphic to any pushout of (f', g'), as in diagram (3.5).

$$\begin{tikzcd}
 & X \arrow[dr, "f''"'] \arrow[rrd, bend left, "f"] & & \\
W \arrow[ur, "g'"] \arrow[dr, "f'"'] & & U \arrow[r, dotted, "h"] & Z \\
 & Y \arrow[ur, "g''"] \arrow[rru, bend right, "g"'] & &
\end{tikzcd} \tag{3.5}$$

Assuming that the inner square of diagram (3.5) is a pushout, we have a unique morphism h making the diagram commute. Note that (f, g) is jointly surjective: for every $z \in Z$, having $z \notin f(X)$ and $z \notin g(Y)$ contradicts assumptions (i) and (ii). If we show that h is monic, we will have two epi-mono pair factorisations for (f, g), namely h after (f'', g'') and id_Z after (f, g). Then, by uniqueness of epi-mono pair factorisations, h must be an isomorphism.

In order to prove h is monic, by Theorem 3.10 we may show that all components of h are injective: taking any elements $u_1, u_2 \in U$ such that $h(u_1) = h(u_2)$, we will show $u_1 = u_2$. Since (f'', g'') is a pushout and thus jointly epic, $u_1, u_2 \in f''(X) \cup g''(Y)$. We have the following cases.

- Assume u_1 and u_2 have preimages on either object, which without loss of generality we assume to be $u_1 \in f''(X)$ and $u_2 \in g''(Y)$. There is $x \in X$ with $u_1 = f''(x)$ and $y \in Y$ with $u_2 = g''(y)$. We will show that x and y are identified by (f, g), so they have a common preimage in the pullback W and they must be identified in the pushout U.

  First note that $f(x) = h(f''(x)) = h(u_1) = h(u_2) = h(g''(y)) = g(y)$. Since (f', g') is the pullback of (f, g), there is some $w \in W$ with $x = g'(w)$ and $y = f'(w)$. Then $u_1 = f''(x) = f''(g'(w)) = g''(f'(w)) = g''(y) = u_2$.

- Assume that u_1 and u_2 only have preimages in one of the objects, which without loss of generality we assume to be X. There are $x_1, x_2 \in X$ with $u_1 = f''(x_1)$ and $u_2 = f''(x_2)$, but there is no $y \in Y$ with $u_1 = g''(y)$ or $u_2 = g''(y)$. We will prove by contradiction that $x_1 = x_2$, which implies $u_1 = u_2$. Thus assume $x_1 \neq x_2$.

  Note that, since u_1 and u_2 are identified by h, x_1 and x_2 must be identified by f: we have $f(x_1) = h(f''(x_1)) = h(u_1) = h(u_2) = h(f''(x_2)) = f(x_2)$. Then there is no unique $x \in X$ with $f(x_1) = f(x)$, since this holds for x_1 and x_2. By the contrapositive of assumption (ii), this implies the existence of some $y \in Y$ with $f(x_1) = g(y)$.

But the identification of x_1 and y by (f,g) implies they are also identified by (f'',g''). In fact, since (f',g') is the pullback of (f,g), when $f(x_1) = g(y)$ there exists some $w \in W$ with $x_1 = g'(w)$ and $y = f'(w)$. Then $f''(x_1) = f''(g'(w)) = g''(f'(w)) = g''(y)$.

This contradicts our assumption that u_1 has no preimage in Y, since now $g''(y) = f''(x_1) = u_1$. Thus, $x_1 \neq x_2$ must be false. □

From the previous lemma, we can define a construction for initial pushouts. In fact, its contrapositive determines which elements *must* be in the context: those without a unique preimage. The following construction takes only those elements along with their boundary, which necessary to make an object well-defined. Compare it to Construction 2.64, which it generalises.

Construction 3.14. Given a morphism $f : X \to Y$ in category $\mathbb{Set}^{\mathbb{S}}$, its context object $C \subseteq Y$ can be constructed by taking all elements $y \in Y$ whose preimage $f^{-1}(y)$ is not a singleton set, along with their boundary. Then the context morphism is just the inclusion $c : C \hookrightarrow Y$.

Formally, we denote the preimage $f_S^{-1}(y) = \{x \in X(S) \mid y = f_S(x)\}$ and by $\mathrm{card}(Z)$ the cardinality of a set Z. Then we define the action of C on each object $S \in \mathbb{S}$ with equations (3.6) and (3.7). Then for each morphism $k : T \to S$ of $\mathbb{S}$, we define $C(k)$ by restricting $Y(k)$ on its domain.

$$\overline{C_S} = \left\{ y \in Y(S) \mid \mathrm{card}(f_S^{-1}(y)) \neq 1 \right\} \tag{3.6}$$

$$C(S) = \left\{ Y(k)(y) \mid T \in \mathbb{S}, y \in \overline{C_T}, k : T \to S \right\} \tag{3.7}$$

Intuitively, equation (3.6) selects elements of Y that are either outside the image of f, or are the identification by f of multiple elements from X. Then equation (3.7) adds the necessary "subelements" to ensure that the construction is well-defined.

Lemma 3.15. Every category of functors $\mathbb{Set}^{\mathbb{S}}$ has initial pushouts for all arrows, whose context corresponds to Construction 3.14.

Proof. Construct c in diagram (3.8) as in Construction 3.14, then obtain rectangle ① + ② as a pullback.

$$\begin{tikzcd}
B \arrow[r, dotted, tail, "b^*"] \arrow[rr, bend left, hook, "b"] \arrow[d, "f'"'] \arrow[dr, phantom, "\textcircled{1}"] & U \arrow[r, tail, "u"] \arrow[d, "g"'] \arrow[dr, phantom, "\textcircled{2}"] & X \arrow[d, "f"] \\
C \arrow[r, dotted, tail, "c^*"] \arrow[rr, bend right, hook, "c"'] & V \arrow[r, tail, "v"] & Y
\end{tikzcd} \tag{3.8}$$

To show that the constructed rectangle is a pushout, by Theorem 3.13 it suffices to show that every $y \in Y$ outside the image of c has a unique preimage by f. This follows directly from the construction of C: if y had no or multiple preimages by f, it would be in $\overline{C_S}$ and therefore in C.

To show the constructed pushout is initial, take any pushout ② as in diagram (3.8). For each $y \in C \subseteq Y$, again by Theorem 3.13, there is a unique $y' \in V$ with $v(y') = y$. This determines a morphism $c^* : C \to V$ making the lower triangle commute. Then by the pullback property there is a unique b^* making the whole diagram commute. □

Finally, from the construction of initial pushouts, we can provide a set-theoretical characterisation of the gluing condition from Lemma 2.70, which ensures the existence of pushout complements. This corresponds to the usual *dangling condition* for graphs, stating that no unmatched elements of the system state may be incident to a deleted element. If there were such elements, they would be left "dangling" by the transformation.

Lemma 3.16 (Dangling Condition)**.** In a category $\mathbb{S}\mathrm{et}^{\mathbb{S}}$ of set-valued functors, given monomorphisms $l : K \rightarrowtail L$ and $m : L \rightarrowtail G$ the following statements are equivalent.

(i) There exists a pushout square ① as in diagram (3.9)

(ii) For every $y \in G^T \setminus m(L^T)$ incident to some $x \in m(L^S)$, that is, with $G^i(y) = x$ for some $i : T \to S$ in category $\mathbb{S}$, it holds that $x \in m(l(K^S))$.

$$\begin{tikzcd}
K \arrow[r, "l"] \arrow[d, dashed, "k"] \arrow[rd, phantom, "①"] & L \arrow[d, "m"] \arrow[rd, phantom, "②"] & B_m \arrow[l, "b_m"'] \arrow[d, "\overline{m}"] \arrow[ll, dotted, bend right, "b^*"'] \\
D \arrow[r, dashed, "g"'] & G & C_m \arrow[l, "b_m"]
\end{tikzcd} \tag{3.9}$$

Proof. Given the initial pushout ② over m, we will show that item (ii) holds if and only if there exists some b^* with $b_m = l \circ b^*$, which is equivalent to (i) by Lemma 2.70. Assume also, without loss of generality, that the context of the initial pushout corresponds to Construction 3.14, and that morphisms l and b_m are inclusion. Then, if b^* exists, it must also be an inclusion.

[If] Assume the inclusion b^* exists, that is, $B_m \subseteq K \subseteq L$. Let $y \in G^T \setminus m(L^T)$ be incident to some some $x \in m(L^S) \subseteq G^S$. It follows from Construction 3.14 that $y \in C_m^S$ and $x \in C_m^T$. Then, since square ② is a pullback, there is a preimage $x' \in B_m$ with $x = \overline{m}(x') = m(x)$. But since $x' \in B_m \subseteq K$, we also have $x = m(l(x'))$, and we conclude that $x \in m(l(K^T))$.

[Only if] Assuming item (ii) holds, we will show that $B_m \subseteq K$ by taking any $x' \in B_m \subseteq L$ and showing that $x' \in K$. Consider the image $x = m(x') = \overline{m}(x') \in C_m \subseteq G$, it follows from Construction 3.14 that there exists some incident $y \in G^T$ and $i : T \to S$ such that $x = G^i(y)$ and $\mathrm{card}(m^{-1}(y)) \neq 1$. Moreover, since m is injective, we have

$\mathrm{card}(m^{-1}(y)) = 0$ and $y \in G^T \setminus m(L^T)$. It then follows from item (ii) that $x \in m(l(K^S))$. We conclude, since l is an inclusion, that $x' \in B_m$. □

4 GENERALIZED SUBOBJECTS

An important goal of this thesis is to provide a formal account for root causes of conflicts between transformation steps and rules. Recall from Section 2.4 that conflicts occur when a transformation step deletes parts of the system state that are necessary for another step. When dealing with a particular pair of steps, an appropriate characterisation will identify such elements of the system state, that is, an appropriate subobject.

To analyse a transformation system, however, one requires an overview of *potential* conflicts between each pair of rules. In this case, referring to particular system states should be avoided. An alternative, given a pair a rules, is to identify the parts of the left-hand sides that cause a conflict *when identified by the matches*.

In set-theoretical terms, given rules ρ_1 and ρ_2, a potential conflict could be characterised by an appropriate set $S \subseteq L_1 \times L_2$. Such a set would contain pairs (x_1, x_2) of elements from the left-hand sides such that, when a pair of matches identifies all those elements, they cause a conflict.

Example 4.1 (Set-Theoretic Characterization of Conflicts)**.** Recall from Example 2.6 the rule `move-up-NU`. It has multiple potential conflicts with itself, as shown in Fig. 4.1. The causes of these conflicts could be described by the following sets containing pairs of edges. We denote each edge by their label from Fig. 4.1a, with a subscript to indicate whether they are matched by the transformation step on the left (e_1) or right (e_2).

$$S_1 = \{(at_1, at_2)\} \tag{4.1}$$

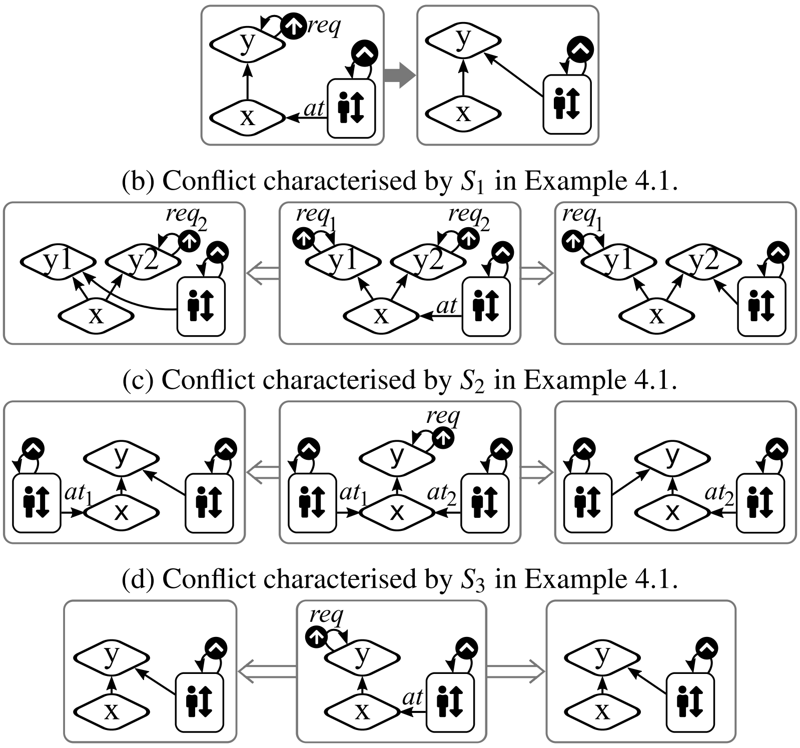
$$S_2 = \{(req_1, req_2)\} \tag{4.2}$$

$$S_3 = \{(at_1, at_2), (req_1, req_2)\} \tag{4.3}$$

Note that each of the conflicts shown in Fig. 4.1 corresponds to a different overlapping of the left-hand sides. Note also that set S_3 is just the union of S_1 and S_2, because the corresponding transformation overlaps exactly the edges from S_1 and S_2.

An advantage of a set-theoretical characterisation is that the a notion of “subcause” is naturally defined by containment of sets, and composing or decomposing the causes is also possible with set-theoretical constructions. On the other hand, its precise definition would have to be restated for each kind of transformed object. In fact, it may not be obvious how to define it for some kinds of objects, such as attributed or symbolic graphs. Therefore, a category-theoretic characterisation would be preferred.

Figure 4.1: Conflicting transformation steps of `move-up-NU` with itself



(a) Rule `move-up-NU` with edges labelled according to Example 4.1.

(b) Conflict characterised by S_1 in Example 4.1.

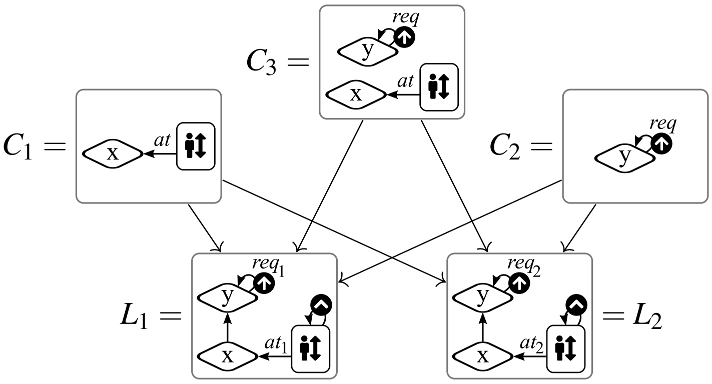
(c) Conflict characterised by S_2 in Example 4.1.

(d) Conflict characterised by S_3 in Example 4.1.

A natural generalisation for subsets of a Cartesian product are spans. For causes of conflicts, *spans of monomorphisms* are appropriate. They generalise the intuition that, since the matches are monic, each element from a rule's left-hand side can only be identified with a single element from the other rule.

Example 4.2 (Characterization with Spans)**.** Each set S_i identified in Example 4.1 corresponds to the span C_i of monomorphisms shown in Fig. 4.2.

Figure 4.2: Causes of conflicts for `move-up-NU` as spans.



The generality of this view, however, comes at a cost: notions of containment, composition and decomposition are not immediately available for such spans. The main goal of this chapter is to reconstruct these notions in a categorical setting, applying notions of lattice theory to appropriate sets of spans Indeed, lattice theory is the natural language for reasoning about containment (which is a partial order) as well as composition and decomposition (having notions similar to union and intersection).

Interestingly, the tools we desire are already available for the categorical notion of subobject (Definition 2.23). The set $\mathbf{Sub}(X)$ of subobjects for an object X is naturally equipped with a "containment" relation. Moreover, in adhesive categories, $\mathbf{Sub}(X)$ is a distributive lattice (Theorem 2.24). Then, when it is finite, Birkhoff's representation theorem implies that $\mathbf{Sub}(X)$ behaves exactly like a lattice of subsets. In fact, finite distributive lattices are built from their *irreducible elements*, each element corresponding to a set of irreducibles. The utility of lattice theory in the context of adhesive categories was recognised by Baldan et al. (2011), who provided many useful results regarding *irreducible subobjects*.

In the remainder of this chapter, we will use subobjects as a basis to provide an appropriate context for characterising causes of conflicts between transformation steps and rules. First, we will review some definitions and important results of lattice theory. Then we will define a generalised notion of subobject for arbitrary diagrams, and study in particular the partially ordered sets $\mathbf{Sub}(L_1 \overset{m_1}{\to} G \overset{m_2}{\leftarrow} L_2)$ of subobjects for a monic cospan, and $\mathbf{Sub}(L_1, L_2)$ of subobjects for a pair of objects. The former will be useful when characterising conflicts for a particular pair of steps, and the latter for potential conflicts between rules. An overview of the main results is provided in Table 4.1.

Table 4.1: Main results of Chapter 4

Required Context	Result	
Any category	Theorem 4.36	$\mathbf{Sub}(m_1, m_2) \subseteq \mathbf{Sub}(L_1, L_2)$
Adhesive category $\mathbb{C}$	Theorem 4.38	$\mathbf{Sub}(L_1, L_2) = \bigcup \left\{ \mathbf{Sub}(m_1, m_2) \mid L_1 \overset{m_1}{\rightarrowtail} G \overset{m_2}{\leftarrowtail} L_2 \right\}$
	Corollary 4.33	$\mathbf{Sub}(m_1, m_2)$ is distributive lattice
Adhesive category with equalisers	Theorem 4.48	$\mathbf{Sub}(L_1, L_2) \uplus \{\top\}$ is a lattice
	Lemma 4.46	Construction of intersections in $\mathbf{Sub}(L_1, L_2)$
	Lemma 4.47	Construction of unions in $\mathbf{Sub}(L_1, L_2)$
	Lemma 4.51	Characterisation of irreducibles in $\mathbf{Sub}(L_1, L_2)$

Remark 4.3. As we will show, the notions of containment, union and intersection coincide for $\mathbf{Sub}(m_1, m_2)$ and $\mathbf{Sub}(L_1, L_2)$. Thus, in the following chapters, we will denote them by $a \subseteq_{\mathbf{Sub}} b$, $a \cup_{\mathbf{Sub}} b$ and $a \cap_{\mathbf{Sub}} b$, whenever the pair of objects (L_1, L_2) or the cospan of monos (m_1, m_2) is clear from the context. The subscript **Sub** will also be omitted

when no ambiguity arises.

4.1 Lattice Theory

This section briefly introduces some important concepts of lattice theory. For a more thorough explanation, we point to the textbook by Davey and Priestley (2002).

Definition 4.4 (Poset)**.** A *partially ordered set (poset)* is a set P equipped with a reflexive, anti-symmetric and transitive relation, denoted $\subseteq_P$ and called a *partial order*.

Fact 4.5 (Strict Order, Cover)**.** Every poset P determines a *strict order* $\subset_P$ and a *covering relation* $\mathrel{\subset\!\!\cdot}_P$ defined as follows. Moreover, every strict order or covering relation uniquely determines a partial order.

(i) $a \subset_P b$ if and only if $a \subseteq_P b$ and $a \neq b$.

(ii) $a \mathrel{\subset\!\!\cdot}_P b$ if and only if $a \subset_P b$ and $a \subseteq_P a' \subset_P b$ implies $a' = a$, for every $a' \in P$.

Definition 4.6 (Top, Bottom)**.** A poset P may have a *top element* that is greater than all other elements, denoted $\top$. Dually, it may have a *bottom element* denoted $\bot$.

If posets don't have a top or bottom element, it is sometimes useful to add one. Then we define $P_\top = P \uplus \{\top\}$, where $a \subseteq_{P_\top} \top$ for every $a \in P_\top$, and $a \subseteq_{P_\top} b$ if and only if $a \subseteq_P b$ for every $a, b \in P$. The poset $P_\bot$ is defined analogously.

Definition 4.7 (Supremum, Infimum)**.** Let P be a poset and $S \subseteq P$. An upper bound for S is an element $a \in P$ such that $b \subseteq_P a$ for every $b \in S$. Then the *supremum* for S is its least upper bound, which may not exist. Dually, the *infimum* for S is its greatest lower bound.

Posets where every finite subset has a supremum and infimum are particularly well-behaved, and called lattices. In this case, suprema and infima determine binary operations that are usually called *join* and *meet*, and sometimes called *union* and *intersection*.

Definition 4.8 (Lattice)**.** A poset L is a *lattice* when every two elements $a, b \in L$ have a supremum and an infimum. Then the supremum of $\{a, b\}$ is called a *join* and denoted $a \cup_L b$. Dually, the infimum of $\{a, b\}$ is called a *meet* and denoted $a \cap_L b$.

Fact 4.9. Any non-empty finite subset $S \subseteq L$ of a lattice has a supremum and an infimum.

Remark 4.10. We will refer to joins as unions and to meets as intersections. These names are usually reserved for distributive lattices as defined below, but in the context of subobjects we will apply them to non-distributive lattices as well.

Lattices where all subsets have suprema and infima are called *complete*. In this case, joins and meets can be defined as operations over *subsets* of the lattice. Moreover, these operations are compatible with the unions and intersections of subsets.

Definition 4.11 (Complete Lattice)**.** A lattice L is *complete* when every subset $S \subseteq L$ has a supremum and an infimum. Then the supremum and infimum of S are also called its join (or union) and meet (or intersection), denoted $\bigcup_L S$ and $\bigcap_L S$, respectively.

Fact 4.12. Every finite lattice is complete.

Fact 4.13. Every complete lattice L has $\top = \bigcup_L L$ and $\bot = \bigcap_L L$.

Fact 4.14. For all subsets $A, B \subseteq L$ of a complete lattice L, equations (4.4) and (4.5) hold.

$$\left(\bigcup\nolimits_L A\right) \cup_L \left(\bigcup\nolimits_L B\right) = \bigcup\nolimits_L (A \cup B) \tag{4.4}$$

$$\left(\bigcap\nolimits_L A\right) \cup_L \left(\bigcap\nolimits_L B\right) = \bigcap\nolimits_L (A \cup B) \tag{4.5}$$

Lattices where unions and intersections distribute over each other are called *distributive*, and are even more well-behaved. In fact, every finite distributive lattice is isomorphic to some lattice of powersets, a result known as Birkhoff's representation theorem. For a thorough discussion of these concepts we refer to the textbook by Davey and Priestley (2002).

Definition 4.15 (Distributive Lattice)**.** A lattice L is *distributive* when equations (4.6) and (4.7) hold for every $a, b, c \in L$.

$$a \cup_L (b \cap_L c) = (a \cup_L b) \cap_L (a \cup_L c) \tag{4.6}$$

$$a \cap_L (b \cup_L c) = (a \cap_L b) \cup_L (a \cap_L c) \tag{4.7}$$

Interestingly, there is an appropriate notion of minimal "building blocks" for a lattice. These are the *irreducible elements*: non-bottom elements that cannot be decomposed as a union of strictly smaller elements. It turns out that every element of a lattice is a union of irreducible elements.

Definition 4.16 (Irreducible Element)**.** Let L be a lattice. An element $a \in L$ is *weak irreducible* when $a = b \cup_L c$ implies that $a = b$ or $a = c$, for every $b, c \in L$. Moreover, a is *irreducible* when it is weak irreducible and $a \neq \bot$.

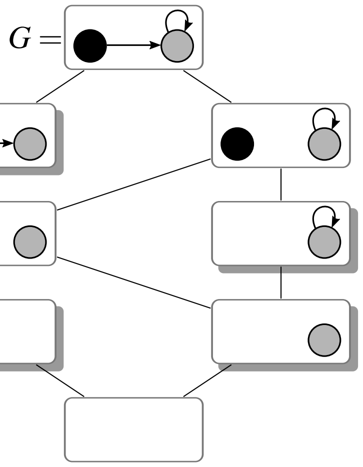
Fact 4.17. In a finite lattice L, an element $a \in L$ is irreducible if and only if it covers exactly one element, that is, when there is a unique $b \in X$ with $b \lessdot_L a$.

Fact 4.18. In a finite lattice L, every element is a union of irreducibles. In fact, equation (4.8) holds for every $a \in L$.

$$a = \bigcup_L \{b \in L \mid b \subseteq_L a,\ b \text{ irreducible}\} \tag{4.8}$$

Example 4.19 (Irreducible Subgraphs)**.** Every graph determines a lattice of subgraphs. In this lattice, the irreducible elements are those subgraphs containing exactly one node or exactly one edge along with its source and target. Figure 4.3 presents the lattice of subgraphs for a particular graph, highlighting the irreducible elements.

Figure 4.3: Lattice of subgraphs with irreducibles highlighted by shadows



Finally, we present an induction principle for elements of finite posets, including finite subsets of a lattice. It is based on the *height* of an element, which is the maximal distance between it and any minimal element.

Definition 4.20. A *chain* is a poset C with a total order, that is, where $a \subseteq_C b$ or $a \subseteq_C b$ for every $a, b \in C$. Given a poset P, we call a poset $C \subseteq P$ with total order a *chain in P*.

Definition 4.21. Let P be a poset. A subset $S \subseteq P$ is *a down-set* when $a \subseteq_P b$ and $a \in S$ imply $b \in S$, for every $a, b \in P$. Moreover, *the down-set of* $a \in P$ is $\downarrow a = \{b \in P \mid b \subseteq_P a\}$.

Definition 4.22. In a finite[1] poset P, the *height* $h(a)$ of an element $a \in P$ is the size of the longest maximal chain in $\downarrow a$. That is, denoting by $\operatorname{card}(X)$ the cardinality of set X, we define $h(a) = \max\{\operatorname{card}(C) \mid C \subseteq \downarrow a, C \text{ is a chain}\}$.

Fact 4.23. If $a \subset_P b$, then $h(a) < h(b)$.

[1] We could require a weaker property of the poset, namely that it has no infinitely descending chains (DAVEY; PRIESTLEY, 2002), that is, for every chain $c_0 \supseteq c_1 \supseteq c_2 \supseteq \ldots$ there is some i with $c_i = c_j$ for every $j \geq i$.

Remark 4.24. We can prove statements about all elements of a finite poset using mathematical induction on the height of these elements. That is, we have to show that the statement holds for all minimal elements and, if it holds for all elements of lesser height, then it holds for a particular element.

4.2 Generalised Subobjects

In order to define a generalised notion of subobject, we first review some concepts of category theory related to diagrams. The reader is assumed familiar with the definition of diagram as a functor from a small category, which is given in most textbooks of category theory such as Riehl (2017).

Definition 4.25 (Diagram)**.** Let $\mathbb{C}$ be a category and $\mathbb{I}$ a small category. A *diagram of shape* $\mathbb{I}$ *in* $\mathbb{C}$ is a functor $\mathrm{D} : \mathbb{I} \to \mathbb{C}$. We denote by $|\mathrm{D}| = (D_i \in \mathbb{C})_{i \in \mathbb{I}}$ the family of objects contained in diagram D, indexed by the objects of $\mathbb{I}$.

Remark 4.26. A pair of objects in a category $\mathbb{C}$ can be seen as a diagram of shape $\mathbb{P}$, where $\mathbb{P}$ contains only two objects and their identities. Similarly, a cospan $L_1 \overset{m_1}{\to} G \overset{m_2}{\leftarrow} L_2$ can be seen as a diagram of shape $\mathbb{S}$, where $\mathbb{S}$ contains three objects, their identities and the two morphisms with same codomain.

Sources and sinks generalise morphisms, allowing multiple domain or codomain objects. Cones for a diagram D are sinks with codomain $|\mathrm{D}|$, covering the whole diagram and commuting with all its morphisms. There is a dual notion of cocone, but it will not be relevant for this work.

Definition 4.27 (Source, Sink)**.** Given a family of objects $\mathscr{L} = (L_i)_{i \in I}$ as well as objects A and G, a *source* $a : A \to \mathscr{L}$ is a family of morphisms $(a_i : X \to L_i)_{i \in I}$ and a *sink* $m : \mathscr{L} \to G$ is a family $(m_i : L_i \to G)_{i \in I}$. A source (sink) is *fully monic* when all its components are monic.

Definition 4.28 (Cone)**.** Let D be a diagram of shape $\mathbb{I}$ in $\mathbb{C}$. A *cone* for diagram D is a source $a : A \to |\mathrm{D}|$ in $\mathbb{C}$ such that diagram (4.9) commutes for all morphisms $d : D_i \to D_j$ in the diagram D. Given cones $a : A \to |\mathrm{D}|$ and $b : B \to |\mathrm{D}|$, a *cone morphism* $f : a \to b$ is a morphism $f : A \to B$ in category $\mathbb{C}$ such that $a_i = b_i \circ f$ for every $D_i \in |\mathrm{D}|$. That is, all triangles as in diagram (4.10) commute.

$$\begin{tikzcd}
& A \arrow[dl, "a_i"'] \arrow[dr, "a_j"] & \\
D_i \arrow[rr, "d"'] & & D_j
\end{tikzcd}$$

(4.9)

$$\begin{tikzcd}
& & D_i \\
A \arrow[r, dashed, "f"] \arrow[urr, bend left, "a_i"] \arrow[drr, bend right, "a_j"'] & B \arrow[ur, dashed, "b_i"] \arrow[dr, dashed, "b_j"'] & \vdots \\
& & D_j
\end{tikzcd}$$

(4.10)

Fact 4.29. Cone morphisms are mono or isomorphisms if and only if their underlying morphisms are.

We can now define a generalised notion of subobject for arbitrary diagrams. Recall that subobjects for an object X are monomorphisms with codomain X. Similarly, the subobjects of a diagram D are its fully monic cones.

Definition 4.30 (D-Subobject)**.** Let D be a diagram on a category $\mathbb{C}$. Any cones $a : A \rightarrowtail \mathrm{D}$ and $b : B \rightarrowtail \mathrm{D}$ are *isomorphic*, denoted $a \cong b$, if and only is there exists a cone isomorphism between them. Then a D-*subobject* is an isomorphism class of fully monic cones over D. Moreover, $\mathbf{Sub}(\mathrm{D})$ is the *poset of* D-*subobjects*, with $a \subseteq_{\mathrm{D}} b$ if and only there exists a cone monomorphism $f : a \rightarrowtail b$, called a *witness*.

Remark 4.31. When diagram D consists of a single object X, the poset $\mathbf{Sub}(\mathrm{D})$ is exactly $\mathbf{Sub}(X)$ in the sense of Definition 2.23.

The properties of the poset $\mathbf{Sub}(\mathrm{D})$ vary greatly according to the diagram D and underlying category $\mathbb{C}$. In the following, we study subobjects for monic cospans and for pairs of objects, which will be useful in Chapter 5 to characterise the root causes of conflicts between transformation steps and of potential conflicts between rules, respectively.

4.2.1 Subobjects for Monic Spans

When characterising the conflict for a particular pair $H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ of transformations, we must consider the matches (m_1, m_2). Then any span $L_1 \overset{e_1}{\leftarrow} C \overset{e_2}{\rightarrow} L_2$ characterising the conflict should commute with the matches as in diagram (4.11), since only elements matched by both rules can cause a conflict.

$$\begin{tikzcd}
& C \arrow[dl, "e_1"'] \arrow[dr, "e_2"] & \\
L_1 \arrow[dr, "m_1"'] & & L_2 \arrow[dl, "m_2"] \\
& G &
\end{tikzcd}$$

(4.11)

A suitable context for characterising these conflicts is the poset $\mathbf{Sub}(m_1,m_2)$ of subobjects for the matches. Since it imposes more restrictions than the poset $\mathbf{Sub}(L_1,L_2)$ of subobjects for the left-hand sides, namely commutation with the matches, it has a richer structure. Indeed, if $L_1 \overset{p_1}{\leftarrowtail} L_1L_2 \overset{p_2}{\rightarrowtail} L_2$ is the pullback of the matches, $\mathbf{Sub}(m_1,m_2)$ is isomorphic to $\mathbf{Sub}(L_1L_2)$, which in adhesive categories is a distributive lattice. Thus, we obtain notions of union, intersection and emptiness for m-subobjects.

Theorem 4.32. Let $L_1 \overset{m_1}{\rightarrowtail} G \overset{m_1}{\leftarrowtail} L_2$ be a cospan of monos with pullback object L_1L_2. Then the poset $\mathbf{Sub}(m_1,m_2)$ is isomorphic to $\mathbf{Sub}(L_1L_2)$.

$$\begin{tikzcd}
 & & L_1 \arrow[dr, tail, "m_1"] & \\
A \arrow[urr, tail, bend left, "a_1"] \arrow[r, dashed, tail, "\hat{a}"] \arrow[drr, tail, bend right, "a_2"'] & L_1L_2 \arrow[ur, tail, "p_1"] \arrow[dr, tail, "p_2"'] & & G \\
 & & L_2 \arrow[ur, tail, "m_2"'] &
\end{tikzcd} \tag{4.12}$$

Proof. This follows directly from the universal property of pullbacks. In fact, given a cone $L_1 \overset{a_1}{\leftarrowtail} A \overset{a_2}{\rightarrowtail} L_2$ commuting with (m_1,m_2), there is a unique morphism $\hat{a} : A \rightarrow L_1L_2$ making diagram (4.12) commute. Since $a_1 = p_1 \circ \hat{a}$ is monic, $\hat{a}$ is also monic, thus we constructed a unique mono corresponding to the cone. Given a mono $\hat{a} : X \rightarrowtail L_1L_2$, we can construct the cone $a = (p_1 \circ \hat{a}, p_2 \circ \hat{a})$. These constructions establish a bijection. □

Corollary 4.33. In an adhesive category, $\mathbf{Sub}(m_1,m_2)$ is a distributive lattice, with the following notions analogous to Theorem 2.24 and Remark 2.26, given $a,b \in \mathbf{Sub}(m_1,m_2)$.

(i) The *intersection* $a \cap_m b$ is obtained from a pullback.

(ii) The *union* $a \cup_m b$ is obtained from a pushout over the intersection.

(iii) The *top* or maximum subobject $\top \in \mathbf{Sub}(m_1,m_2)$ is the pullback of (m_1,m_2).

(iv) The *bottom* or minimum subobject $\bot \in \mathbf{Sub}(m_1,m_2)$ can be seen as the empty subobject, if it exists. When the category has a strict initial object $\mathbf{0}$, then $\bot$ is determined by it.

4.2.2 Subobjects for Pairs of Objects

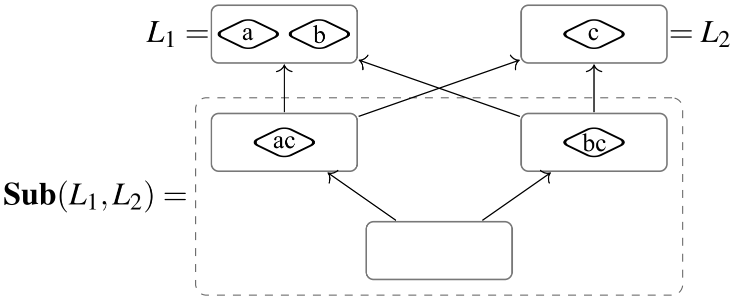
When characterising potential conflicts for a pair of rules, we cannot explicitly depend on pairs of matches: each potential cause should *implicitly* represent multiple pairs of matches. Thus, the appropriate context to investigate potential conflicts is the poset $\mathbf{Sub}(L_1,L_2)$ of subobjects for the left-hand sides.

Unfortunately, subobjects for the left-hand sides are not as well-behaved as subobjects for pairs of matches. In fact, they may fail to have upper bounds and therefore unions, so $\mathbf{Sub}(L_1, L_2)$ is often not a lattice. Nevertheless, in the remainder of this section we will provide appropriate notions of intersection, union (as a partial function) and irreducible elements for $\mathbf{Sub}(L_1, L_2)$.

Remark 4.34. Henceforth, let $\mathscr{L} = (L_1, L_2)$ be an arbitrary pair of objects.

Example 4.35. Let L_1 and L_2 be the graphs shown in Fig. 4.4. Graph L_1 contains two nodes of type `floor`, denoted a and b, while L_2 contains a single node of this type, denoted c. Their poset of subobjects $\mathbf{Sub}(L_1, L_2)$ is shown in Fig. 4.4. Note that the spans corresponding to $\{(a,c)\}$ and $\{(b,c)\}$ have no upper bound, since they are both maximal elements. Indeed, the span corresponding to $\{(a,c),(b,c)\}$ has a non-monic projection to L_2 and therefore does not determine a subobject. Since a pair of elements has no upper bound and therefore no supremum, the poset $\mathbf{Sub}(L_1, L_2)$ is not a lattice.

Figure 4.4: Example of subobjects for a pair of typed graphs.



Even though the elements of $\mathbf{Sub}(L_1, L_2)$ are not *explicitly* related to any pair of matches, we can reveal a lot of its structure by exploring their *implicit* relations. Indeed, given any object G and cospan $m : \mathscr{L} \rightarrowtail G$, every (m_1, m_2)-subobject is also a subobject of (L_1, L_2). Thus, each element of $\mathbf{Sub}(L_1, L_2)$ may be also in $\mathbf{Sub}(m_1, m_2)$ for zero or more cospans $m : \mathscr{L} \rightarrowtail G$. Moreover, the partial orders for $\mathbf{Sub}(m_1, m_2)$ and $\mathbf{Sub}(L_1, L_2)$ are compatible, and the former is a down-set of the latter.

Theorem 4.36. Let $m : \mathscr{L} \rightarrowtail G$ be a cospan of monos.

(i) $\mathbf{Sub}(m_1, m_2) \subseteq \mathbf{Sub}(L_1, L_2)$

(ii) Given $a, b \in \mathbf{Sub}(m_1, m_2)$, $a \subseteq_{\mathscr{L}} b$ if and only if $a \subseteq_m b$

(iii) Given $a, b \in \mathbf{Sub}(L_1, L_2)$, if $a \subseteq_{\mathscr{L}} b$ and $b \in \mathbf{Sub}(m)$ then $a \in \mathbf{Sub}(m)$

Proof. Note that all cones for (m_1,m_2) are also cones for (L_1,L_2), and all cone morphisms for (m_1,m_2) are also cone morphisms for (L_1,L_2). Diagram (4.13) illustrates the proof.

$$\begin{tikzcd}
 & & L_1 \arrow[dr, "m_1"] & \\
A \arrow[urr, bend left, "a_1"] \arrow[r, "a_B"] \arrow[drr, bend right, "a_2"'] & B \arrow[ur, "b_1"] \arrow[dr, "b_2"'] & & G \\
 & & L_2 \arrow[ur, "m_2"'] &
\end{tikzcd} \tag{4.13}$$

(i) Given $a \in \textbf{Sub}(m_1,m_2)$, since a is a cone for (L_1,L_2), then $a \in \textbf{Sub}(L_1,L_2)$.

(ii) Given $a,b \in \textbf{Sub}(m_1,m_2)$ with $a \subseteq_m b$, the witness $a_B : a \rightarrowtail b$ is also a cone monomorphism for (L_1,L_2), witnessing $a \subseteq_{\mathscr{L}} b$.

(iii) Since $b : B \rightarrowtail \mathscr{L}$ is a cone for m, we have $m_1 \circ b_1 = m_2 \circ b_2$. Moreover, $a \subseteq_{\mathscr{L}} b$ is witnessed by some cone monomorphism $a_B : a \rightarrowtail b$. Then $a : A \rightarrowtail \mathscr{L}$ is also a cone for m, since $m_1 \circ a_1 = m_1 \circ b_1 \circ a_B = m_2 \circ b_2 \circ a_B = m_2 \circ a_2$ (compare the diagram above). □

Remark 4.37. Henceforth, we will omit the subscripts from $a \subseteq_{\mathscr{L}} b$ and $a \subseteq_m b$ when there is no ambiguity.

In adhesive categories, we have an even stronger result: $\textbf{Sub}(L_1,L_2)$ is the union of all $\textbf{Sub}(m_1,m_2)$. That is, every (L_1,L_2)-subobject is also a subobject for some monic cospan. An important consequence of this relates the lattice-theoretical notion of upper bounds to the categorical property of commuting with a cospan.

Theorem 4.38. *Let $\mathbb{C}$ be an adhesive category with $L_1,L_2 \in \mathbb{C}$ and $a,b \in \textbf{Sub}(L_1,L_2)$.*

(i) $\textbf{Sub}(L_1,L_2) = \bigcup\{\textbf{Sub}(m_1,m_2) \mid m : \mathscr{L} \rightarrowtail G,\ G \in \mathbb{C}\}$

(ii) *$a \subseteq_{\mathscr{L}} b$ if and only if $a \subseteq_m b$ for some $m : \mathscr{L} \to G$.*

Proof. ***(i)*** Equation (4.14) follows from Theorem 4.36, so we must show equation (4.15).

$$\textbf{Sub}(L_1,L_2) \supseteq \bigcup\{\textbf{Sub}(m_1,m_2) \mid m : \mathscr{L} \rightarrowtail G,\ G \in \mathbb{C}\} \tag{4.14}$$

$$\textbf{Sub}(L_1,L_2) \subseteq \bigcup\{\textbf{Sub}(m_1,m_2) \mid m : \mathscr{L} \rightarrowtail G,\ G \in \mathbb{C}\} \tag{4.15}$$

Given $b \in \textbf{Sub}(L_1,L_2)$, consider the pushout $n = L_1 \overset{n_1}{\to} G \overset{n_2}{\leftarrow} L_2$ of (b_1,b_2). By adhesivity, n_1 and n_2 are monic. Then b is a cone for $n : \mathscr{L} \rightarrowtail G$, which implies:

$$y \in \textbf{Sub}(n_1,n_2) \subseteq \bigcup\{\textbf{Sub}(m_1,m_2) \mid m : \mathscr{L} \rightarrowtail G,\ G \in \mathbb{C}\} \tag{4.16}$$

(ii) The if direction follows from Theorem 4.36. For the only if direction, take any $a,b \in \textbf{Sub}(L_1,L_2)$ with $a \subseteq_{\mathscr{L}} b$. It follows from (i) that $b \in \textbf{Sub}(m_1,m_2)$ for some $m : \mathscr{L} \to G$, and then from Theorem 4.36 that $a \in \textbf{Sub}(m_1,m_2)$ and $a \subseteq_m b$. □

Corollary 4.39. In an adhesive category, any subobjects $a,b \in \mathbf{Sub}(L_1,L_2)$ have an upper bound $a \subseteq_{\mathscr{L}} c \supseteq_{\mathscr{L}} b$ if and only if $a,b \in \mathbf{Sub}(m_1,m_2)$ for some $m : \mathscr{L} \to G$.

Proof. If a and b have an upper bound c, then $c \in \mathbf{Sub}(m)$ for some $m : \mathscr{L} \to G$ and, since $a \subseteq_m c \supseteq_m b$, we have $a,b \in \mathbf{Sub}(m)$. On the other hand, if $a,b \in \mathbf{Sub}(m)$, then their union $a \cup_m b$ is also an upper bound in $\mathbf{Sub}(\mathscr{L})$. □

We have fully characterised the elements of $\mathbf{Sub}(L_1,L_2)$ and their partial order in adhesive categories. Next, we investigate the existence of unions (suprema) and intersections (infima). As Example 4.35 shows, suprema don't always exist in $\mathbf{Sub}(L_1,L_2)$, but we will show that they are well-behaved. Moreover, under mild categorical assumptions, all infima exist and then $\mathbf{Sub}(L_1,L_2)$ behaves nearly as a lattice.

Remark 4.40. We will refer to suprema and infima in $\mathbf{Sub}(L_1,L_2)$ as *global unions and intersections*. In contrast, unions and intersections taken in some $\mathbf{Sub}(m_1,m_2)$ will be called *local*.

Interestingly, all $\mathbf{Sub}(m_1,m_2)$ agree on local unions and intersections. Then every local union and intersection is also global. Moreover, every global union is the local union from some $\mathbf{Sub}(m_1,m_2)$.

Lemma 4.41. Let $a,b \in \mathbf{Sub}(L_1,L_2)$, and let $m : \mathscr{L} \rightarrowtail G$ and $n : \mathscr{L} \rightarrowtail H$ be cospans. If $a,b \in \mathbf{Sub}(m_1,m_2)$ and $a,b \in \mathbf{Sub}(n_1,n_1)$, then $a \cup_m b = a \cup_n b$ and $a \cap_m b = a \cap_n b$.

Proof. [Intersection] Since intersections are lower bounds we have $a \supseteq a \cap_m b \subseteq b$. Then, since intersections are *greatest* lower bounds, it follows that $a \cap_n b \subseteq a \cap_m b$. We can analogously show that $a \cap_m b \subseteq a \cap_n b$, concluding that $a \cap_m b \cong a \cap_n b$.

[Union] Recall from Corollary 4.33 that unions are obtained from the intersections, by pushing out witnesses of $a \supseteq a \cap_m b \subseteq b$. Since the intersections are the same in $\mathbf{Sub}(m_1,m_2)$ and $\mathbf{Sub}(n_1,n_2)$, the corresponding witnesses and pushouts are also the same. □

Lemma 4.42. Let $m : \mathscr{L} \to G$ be a monic cospan and $x,y \in \mathbf{Sub}(m_1,m_2)$.

(i) The local intersection $a \cap_m b$ is also the global intersection of a and b in $\mathbf{Sub}(L_1,L_2)$.

(ii) The local union $a \cup_m b$ is also the global union of a and b in $\mathbf{Sub}(L_1,L_2)$.

Proof. ***(i)*** Take any lower bound $c \in \mathbf{Sub}(L_1,L_2)$ with $a \supseteq c \subseteq b$. It follows from Theorem 4.38 that $c \in \mathbf{Sub}(m_1,m_2)$. Since c is a lower bound in $\mathbf{Sub}(m_1,m_2)$, we have

$c \subseteq a \cap_m b$. We conclude that $a \cap_m b$ is the global intersection, that is, the infimum in $\mathbf{Sub}(L_1, L_2)$.

(ii) Take any upper bound $c \in \mathbf{Sub}(L_1, L_2)$ with $a \subseteq c \supseteq b$. It follows from Theorem 4.38 that there is some $n : \mathscr{L} \to G$ with $c \in \mathbf{Sub}(n_1, n_2)$. Then $a, b \in \mathbf{Sub}(n_1, n_2)$ by Theorem 4.36. Since c is an upper bound in $\mathbf{Sub}(n_1, n_2)$, we have $a \cup_n b \subseteq c$. Finally, it follows from Lemma 4.41 that $a \cup_m b \cong a \cup_n b \subseteq c$. We conclude that $a \cup_m b$ is the global union, that is, the supremum in $\mathbf{Sub}(L_1, L_2)$. □

Lemma 4.43. Let $m : \mathscr{L} \to G$ be a monic cospan. If $a, b \in \mathbf{Sub}(m_1, m_2)$, have an upper bound $a \subseteq c \supseteq b$, then their global union exists and is $a \cup_m b$ for some $m : \mathscr{L} \to G$.

Proof. From Theorem 4.38, there is some $m : \mathscr{L} \to G$ with $c \in \mathbf{Sub}(m_1, m_2)$. It also follows from Theorem 4.36 that $a, b \in \mathbf{Sub}(m_1, m_2)$. We will show that $a \cup_m b$ is the least upper bound in $\mathbf{Sub}(L_1, L_2)$. Take any upper bound $a \subseteq d \supseteq b$ and $n : \mathscr{L} \to H$ with $a, b, d \in \mathbf{Sub}(n_1, n_2)$. Since d is an upper bound in $\mathbf{Sub}(n_1, n_2)$, we have $a \cup_n b \subseteq d$. It then follows from Lemma 4.41 that $a \cup_m b \cong a \cup_n b \subseteq d$. □

Corollary 4.44. Subobjects $a, b \in \mathbf{Sub}(L_1, L_2)$ have a global union if and only if they have an upper bound, or equivalently if $a, b \in \mathbf{Sub}(m_1, m_2)$ for some $m : \mathscr{L} \to G$.

Unlike unions, global intersections may be non-local. As the following example shows, two subobjects $a, b \in \mathbf{Sub}(L_1, L_2)$ may have a global intersection even when they share no cospan of monos $m : \mathscr{L} \to G$ with $a, b \in \mathbf{Sub}(m_1, m_2)$. In fact, all global intersections exist in adhesive categories with equalisers.

Example 4.45 (Non-Local Intersections)**.** Recall graphs L_1 and L_2 from Example 4.35. As Fig. 4.4 shows, the subobjects corresponding to $\{(a, c)\}$ and $\{(b, c)\}$ have a global intersection, namely the empty graph $\varnothing$. This is, however, not a local intersection. In fact, any cospan $m : \mathscr{L} \to G$ that commutes with both subobjects has non-monic m_1, since it must identify a and b.

Lemma 4.46 (Construction of Global Intersections)**.** In an adhesive category with equalisers, all global intersections exist and are constructed as limits. Let $a, b \in \mathbf{Sub}(L_1, L_2)$ be subobjects and $i : I \to |\mathrm{D}|$ be the limit of diagram D as in (4.17). Then we can construct the intersection $a \cap_{\mathscr{L}} b \cong (a_1 \circ i_A, a_2 \circ i_A) = (b_1 \circ i_B, b_2 \circ i_B)$. Note that this limit can be constructed by an equaliser over a pullback.

$$\mathrm{D} = \begin{tikzcd} & W \arrow[d, dotted, "h"] \arrow[ddl, bend right, "w_X"'] \arrow[ddr, bend left, "w_Y"] & \\ & I \arrow[dl, "i_X"'] \arrow[dr, "i_Y"] & \\ X \arrow[d, "x_1"'] \arrow[drr, "x_2"' near start] & & Y \arrow[d, "y_2"] \arrow[dll, "y_1" near start] \\ L_1 & & L_2 \end{tikzcd} \tag{4.17}$$

Proof. We will show that $(a_1 \circ i_A, a_2 \circ i_A)$ is the greatest lower bound for a and b in the poset $\mathbf{Sub}(L_1, L_2)$. Take any lower bound $c \in \mathbf{Sub}(L_1 L_2)$ with $a \supseteq c \subseteq b$. The witnesses c_A and c_B determine a cone for diagram D. Then there exists a unique cone morphism $h : c \to i$, which is monic because $c_A = i_A \circ h$ is monic. Finally, h witnesses $c \subseteq i$. □

Having a construction for intersections, we can use it to construct unions. In this case, however, we must check that the construction is a valid (L_1, L_2)-subobject.

Lemma 4.47 (Construction of Global Unions)**.** In an adhesive category with equalisers, global unions can be constructed as a pushout over the intersection. Let $a, b \in \mathbf{Sub}(L_1, L_2)$ have intersection $i \cong a \cap_{\mathscr{L}} b$. Let also square ③ in diagram (4.18) be the pushout of the witnesses for $a \supseteq i \subseteq b$, which induces unique morphisms u_1 and u_2 making the diagram commute. If u_1 and u_2 are both monic, they determine the union $a \cup_{\mathscr{L}} b = (u_1, u_2)$. If either of them isn't monic, there is no union.

$$\begin{tikzcd} & A \arrow[rr, "a_1"] \arrow[dr, "a_U"'] \arrow[ddrr, "a_2" near start] & & L_1 \\ I \arrow[ur, "i_A"] \arrow[dr, "i_B"'] & \textcircled{3} & U \arrow[ur, dotted, "u_1"] \arrow[dr, dotted, "u_2"'] & \\ & B \arrow[rr, "b_2"'] \arrow[ur, "b_U"] \arrow[uurr, "b_1"' near start] & & L_2 \end{tikzcd} \tag{4.18}$$

Proof. Assume the cone $u = (u_1, u_2) : U \to \mathscr{L}$, constructed as above, is fully monic. Then we have $u \in \mathbf{Sub}(L_1, L_2)$ and, by Theorem 4.36, there is some $m : \mathscr{L} \to G$ with $u \in \mathbf{Sub}(m_1, m_2)$. Since the construction above is essentially the same as the construction of local unions from Corollary 4.33, we have $u \cong a \cup_m b$. Then, since local unions are also global, u is a global union of a and b.

On the other hand, assume a global union $a \cup_{\mathscr{L}} b$ exists. Then it is also a local union $a \cup_{\mathscr{L}} b \cong a \cup_m b$, by Lemma 4.43. But then by Corollary 4.33 we must have $a \cup_m b \cong u$, which implies u_1 and u_2 are monic. Thus, if u_1 or u_2 isn't monic, they have no global union. □

We can now conclude that, in adhesive categories, $\mathbf{Sub}(L_1,L_2)$ behaves almost like a lattice. In fact, it is missing only a top element: the poset $\mathbf{Sub}(L_1,L_2)_\top$ is a lattice, constructed by adding a top element (Definition 4.6).

Theorem 4.48. In an adhesive category with equalisers, $\mathbf{Sub}(L_1,L_2)_\top$ is a lattice.

Proof. [Infima] If follows from Lemma 4.46 that any two elements of $\mathbf{Sub}(L_1,L_2)$ have an infimum, the global intersection. Moreover, for any $a \in \mathbf{Sub}(L_1,L_2)_\top$, since $a \subseteq_{\mathscr{L}_\top} \top$ the infimum of a and $\top$ exists and is a.

[Suprema] Corollary 4.44 ensures that any two elements that have an upper bound have a supremum, the global union. Any two elements of $\mathbf{Sub}(L_1,L_2)$ that have a supremum c will have the same supremum in $\mathbf{Sub}(L_1,L_2)_\top$, since $c \subseteq \top$. On the other hand, elements with no supremum in $\mathbf{Sub}(L_1,L_2)$ had no upper bound, thus their supremum in $\mathbf{Sub}(L_1,L_2)_\top$ is their only upper bound $\top$. Finally, given any $a \in \mathbf{Sub}(L_1,L_2)_\top$, since $a \subseteq_{\mathscr{L}_\top} \top$ the supremum of a and $\top$ exists and is $\top$. □

Then we can apply the notion of irreducibility to $\mathscr{L}$-subobjects, with the result that every $\mathscr{L}$-subobject is a union of irreducibles. In Chapter 5 we will exploit a similar result to choose a minimal set of "causes for conflicts", reducing redundancy without loss of information.

Definition 4.49 (Irreducible $\mathscr{L}$-subobject)**.** An *irreducible $\mathscr{L}$-subobject* is an irreducible element of $\mathbf{Sub}(L_1,L_2)_\top$ different from $\top$.

Lemma 4.50. Every $\mathscr{L}$-subobject is a union of irreducible $\mathscr{L}$-subobjects.

Proof. Follows directly from Definition 4.49 and Fact 4.18. □

The final result of this chapter is a characterisation of irreducible $\mathscr{L}$-subobjects. It turns out they are exactly the irreducible m-subobjects for each $m : \mathscr{L} \rightarrowtail G$, which in turn are just irreducible subobjects in the original sense. Baldan et al. (2011) provided many useful results about irreducible subobjects in adhesive categories.

Lemma 4.51. A $\mathscr{L}$-subobject $a \in \mathbf{Sub}(L_1,L_2)$ is irreducible if and only if it is irreducible in every $\mathbf{Sub}(m_1,m_2)$ such that $a \in \mathbf{Sub}(m_1,m_2)$, with $m : \mathscr{L} \to G$.

Proof. [If] Assume a is irreducible in all $\mathbf{Sub}(m_1,m_2)$. Take any $b,c \in \mathbf{Sub}(L_1,L_2)$ with $a \cong b \cup_{\mathscr{L}} c$. Then there is some $n : \mathscr{L} \rightarrowtail H$ with $a,b,c \in \mathbf{Sub}(n_1,n_2)$, by Theorems 4.36 and 4.38. It follows by Lemma 4.42 that $b \cup_{\mathscr{L}} c \cong b \cup_n c$. Then since a is irreducible in $\mathbf{Sub}(n_1,n_2)$, we must have $a \cong b$ or $a \cong c$. We conclude a is irreducible in $\mathbf{Sub}(L_1,L_2)$.

[Only if] Assume a is irreducible in $\mathbf{Sub}(L_1, L_2)$. Take any $m : \mathscr{L} \to G$ such that $a \in \mathbf{Sub}(m_1, m_2)$, and any $b, c \in \mathbf{Sub}(m_1, m_2)$ with $a \cong b \cup_m c$. Since a is irreducible in $\mathbf{Sub}(L_1, L_2)$ and $b \cup_m c \cong b \cup_{\mathscr{L}} c$, we must have $a \cong b$ or $a \cong c$. We conclude a is irreducible in $\mathbf{Sub}(m_1, m_2)$. □

Remark 4.52. Due to Theorem 4.32, every irreducible m-subobject corresponds to an irreducible subobject in the original sense.

5 CHARACTERIZING CONFLICTS

In this chapter, we tackle the problem of characterising root causes of conflicts. This includes the concrete conflicts in pairs of transformation steps, as well as potential conflicts between rules.

First, we will propose *conflict and disabling essences* for pairs of transformation steps, showing they have many important properties. They are trivially related to parallel independence, which we will show equivalent to having empty conflict essence. Moreover, essences are preserved and reflected by extension in many adhesive categories of interest. They are also more precise than the conflict reasons proposed by Lambers, Ehrig and Orejas (2008), since the conflict essence for a pair of steps is always contained in its conflict reason.

In the second part of this chapter, we will turn to the *potential* conflicts and disablings for pairs of rules. We will define the sets of potential conflict and disabling essences for such a pair, showing that potential conflict essences uniquely determine the initial conflicts proposed by Lambers et al. (2018) in many adhesive categories of interest. Additional redundancy is also characterised, and *irreducible essences* are proposed as an appropriate subset for manual inspection, avoiding redundancy with no loss of information.

The main concepts involved in this chapter are shown in Fig. 5.1 along with their relations. The main results and the necessary categorical assumptions are shown in Fig. 5.2.

Figure 5.1: Overview of concepts related to conflicts and their root causes, with new concepts and results written in bold.

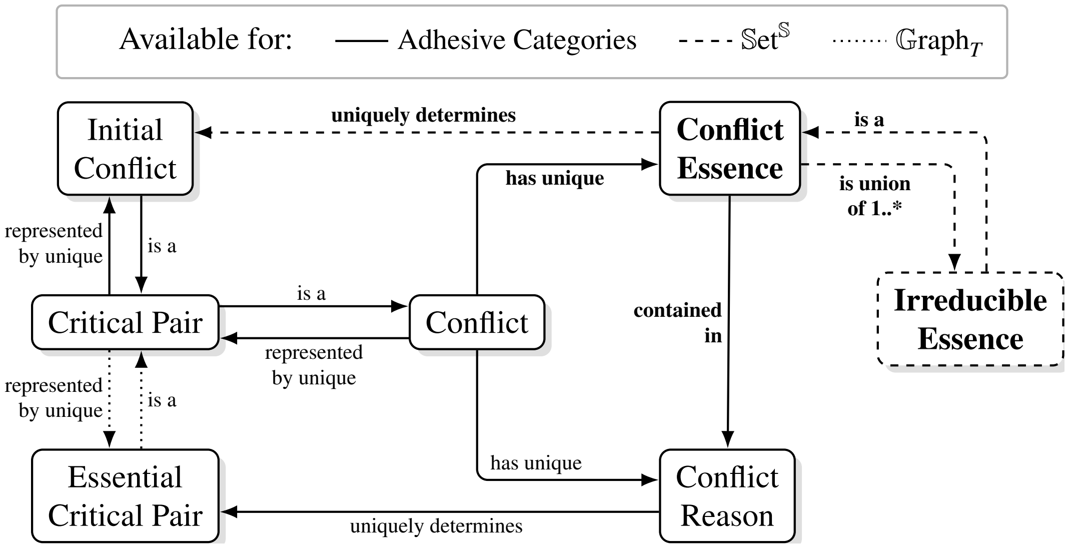
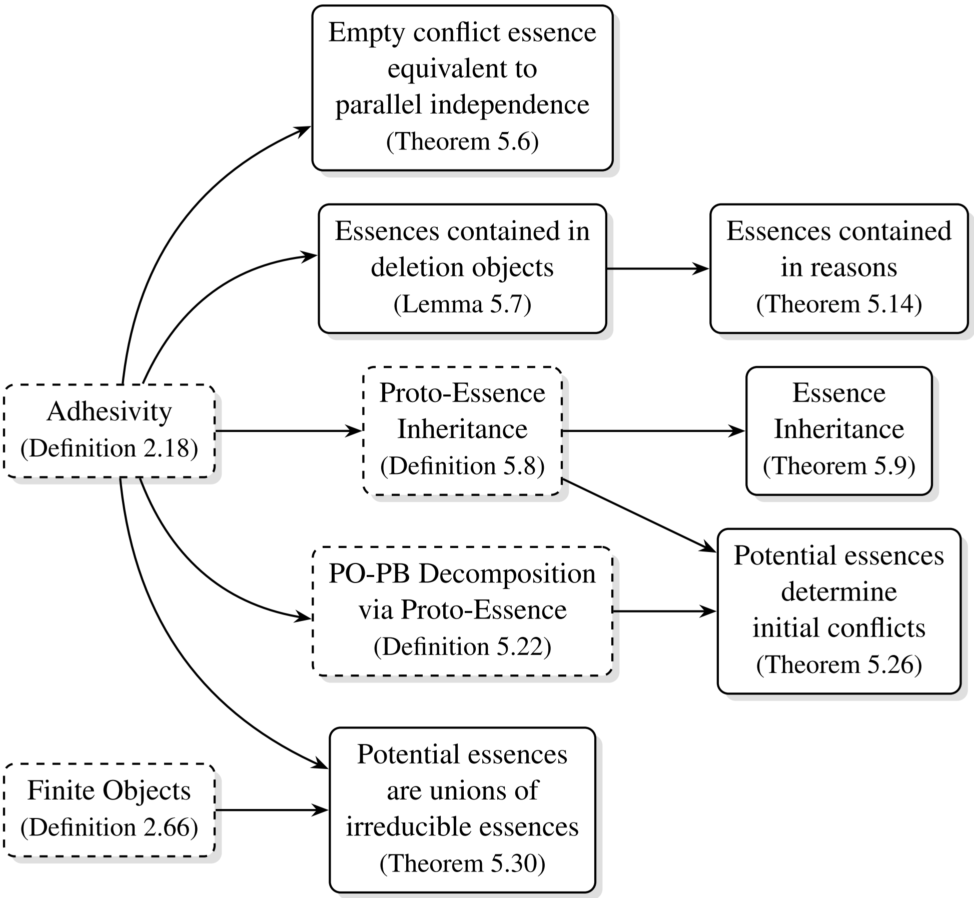


Figure 5.2: Overview of main results and categorical assumptions in Chapter 5. Results are drawn with solid lines while assumptions are drawn with dashed lines.


Empty conflict essence equivalent to parallel independence (Theorem 5.6)
Essences contained in deletion objects (Lemma 5.7)
Essences contained in reasons (Theorem 5.14)
Adhesivity (Definition 2.18)
Proto-Essence Inheritance (Definition 5.8)
Essence Inheritance (Theorem 5.9)
Potential essences determine initial conflicts (Theorem 5.26)
PO-PB Decomposition via Proto-Essence (Definition 5.22)
Finite Objects (Definition 2.66)
Potential essences are unions of irreducible essences (Theorem 5.30)

5.1 Disabling and Conflict Essences

Recall from Section 2.4 that transformation steps $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ determine a disabling when some element is matched by both steps and deleted by at least one of them, as illustrated in Example 2.46. Since matches are monic, they determine subobjects of G and the pullback L_1L_2 as their intersection, as in diagram (5.1). Pulling it back along l_1 removes exactly the elements that would be deleted by step t_1 and that are matched by t_2. This informally justifies the essential condition of parallel independence presented in Definition 2.43.

$$\begin{tikzcd}
 & K_1L_2 \arrow[rr, "\cong"] \arrow[d] \arrow[dr, phantom, "\lrcorner\ \textcircled{2}"] & & L_1L_2 \arrow[dl, "p_1"'] \arrow[dr, "p_2"] \arrow[d, phantom, "\vee"] & & L_1K_2 \arrow[ll, "\cong"'] \arrow[d] \arrow[dl, phantom, "\textcircled{3}\ \llcorner"] & \\
R_1 \arrow[d, "n_1"'] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "k_1"'] & L_1 \arrow[dr, "m_1"] & \textcircled{1} & L_2 \arrow[dl, "m_2"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "k_2"] & R_2 \arrow[d, "n_2"] \\
H_1 & D_1 \arrow[l, "h_1"'] \arrow[rr, "g_1"] & & G & & D_2 \arrow[ll, "g_2"'] \arrow[r, "h_2"] & H_2
\end{tikzcd} \tag{5.1}$$

In order to determine which elements are deleted by pullbacks ② and ③, we use an initial pushout over the arrow $K_1L_2 \to L_1L_2$ (Definition 2.60). In fact, in any adhesive category with initial pushouts, the context of a mono $f : X \rightarrowtail Y$ is the smallest subobject $c \in \mathbf{Sub}(Y)$ such that $c \cup_{\mathbf{Sub}} f \cong \top$, that is, containing the parts of Y which are not in the image of f. This brings us to the following definition.

Definition 5.1 (Conflict and Disabling Essence)**.**

(i) Let $l_1 : K_1 \rightarrowtail L_1$, $m_1 : L_1 \rightarrowtail G$ and $m_2 : L_2 \to G$ be monos. Then the *proto-essence for* (m_1, m_2) *under* l_1, denoted $ess_{l_1}(m_1, m_2)$, is defined as follows. If squares ① and ② in diagram (5.2) are pullbacks, and ④ is an initial pushout over q_2, then $ess_{l_1}(m_1, m_2) \cong (p_1 \circ c, p_2 \circ c) \in \mathbf{Sub}(m_1, m_2)$.

$$\begin{tikzcd}
B \arrow[r, "\overline{q_2}"] \arrow[d, tail, "b"] \arrow[dr, phantom, "\textcircled{4}"] & C \arrow[d, tail, "c"] & \\
K_1L_2 \arrow[r, dashed, "q_2"] \arrow[d, "q_1"] \arrow[dr, phantom, "\textcircled{2}"] & L_1L_2 \arrow[r, dashed, tail, "p_2"] \arrow[d, tail, "p_1"] \arrow[dr, phantom, "\textcircled{1}"] & L_2 \arrow[d, tail, "m_2"] \\
K_1 \arrow[r, "l_1"] & L_1 \arrow[r, tail, "m_1"] & G
\end{tikzcd} \tag{5.2}$$

(ii) If the proto-essence is taken for the matches of $(t_1, t_2) : H_1 \overset{\rho_1, m_1}{\Longleftarrow} G \overset{\rho_2, m_2}{\Longrightarrow} H_2$ under the left morphism of $\rho_1 = L_1 \overset{l_1}{\leftarrowtail} K_2 \overset{r_1}{\rightarrowtail} R_2$, we call it the *disabling essence* of (t_1, t_2), denoted $ess_{\mathrm{dbl}}(t_1, t_2) \in \mathbf{Sub}(m_1, m_2)$.

(iii) The *conflict essence* of transformation steps $(t_1, t_2) : H_1 \overset{\rho_1, m_1}{\Longleftarrow} G \overset{\rho_2, m_2}{\Longrightarrow} H_2$, denoted $ess_{\mathrm{cfl}}(t_1, t_2) \in \mathbf{Sub}(m_1, m_2)$, is the union of disabling essences in both directions. That is, $ess_{\mathrm{cfl}}(t_1, t_2) = ess_{\mathrm{dbl}}(t_1, t_2) \cup_{\mathbf{Sub}} ess_{\mathrm{dbl}}(t_2, t_1)$.

Remark 5.2. We will omit subscripts from ess_{dbl} and ess_{cfl} when clear from the context.

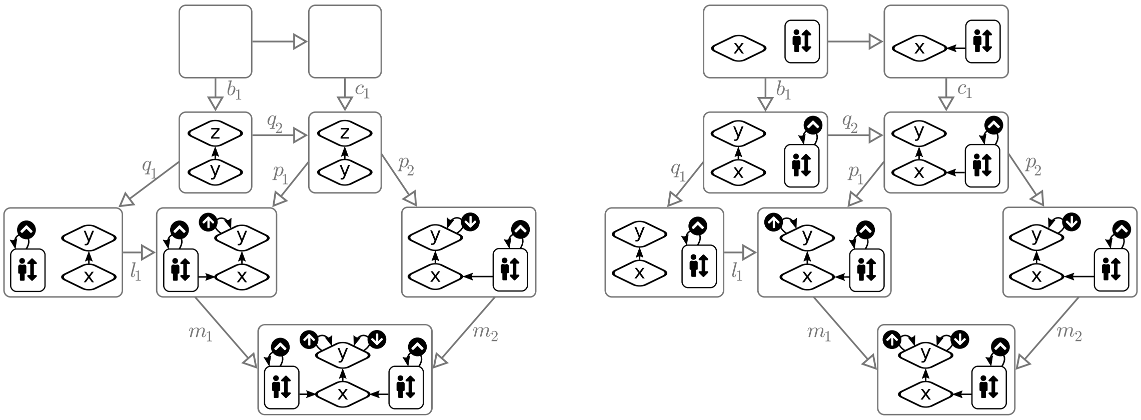
Remark 5.3. Because of Theorem 4.32, the disabling and conflict essences can be equivalently seen as subobjects of L_1L_2, as originally presented by Azzi, Corradini and Ribeiro (2018). Essences are also subobjects of the system state G, since the composite morphism $m_1 \circ p_1 \circ c = m_2 \circ p_2 \circ c : C \rightarrowtail G$ is monic.

Lemma 5.4. Since each disabling essence corresponds to a subobject of L_1L_2, conflict essences can be obtained as a pushout over the pullback of these subobjects.

Proof. Follows from Theorems 2.24 and 4.32. □

Example 5.5. Recall the conflicting transformations from Examples 2.45 and 2.46. Their construction of their disabling essences is shown in Fig. 5.3. In Fig. 5.3a, where transformations are independent, the essence is empty. In Fig. 5.3b, where a disabling exists, the essence contains an edge from elevator to floor.

Figure 5.3: Examples of disabling essences



(a) Essence for transformations of Fig. 2.12a

(b) Essence for transformations of Fig. 2.12b

From Example 5.5 we may expect that an empty disabling essence is equivalent to having no disabling. More generally, we can show that this holds in every adhesive category, interpreting “empty essence” as the bottom element of $\mathbf{Sub}(m_1, m_2)$.

Theorem 5.6. In any adhesive category, let $(t_1, t_2) : H_1 \overset{\rho_1, m_1}{\Longleftarrow} G \overset{\rho_2, m_2}{\Longrightarrow} H_2$ be a pair of transformations. Then $ess_{\text{dbl}}(t_1, t_2) \cong \bot$ if and only if t_1 doesn't disable t_2, and $ess_{\text{cfl}}(t_1, t_2) \cong \bot$ if and only if t_1 and t_2 are parallel independent.

Proof. Recall that morphism q_2 from diagrams (5.1) and (5.2) is monic, and note that $ess_{\text{dbl}}(t_1, t_2) \cong \bot$ if and only if $c \cong \bot \in \mathbf{Sub}(L_1 L_2)$. Then the case for disabling essences follows directly from Lemma 2.71: $c \cong \bot$ if and only if q_2 is an isomorphism, which means that t_1 doesn't disable t_2, according to Definition 2.43.

For conflict essences, recall that $ess_{\text{cfl}}(t_1, t_2) \cong ess_{\text{dbl}}(t_1, t_2) \cup ess_{\text{dbl}}(t_2, t_1)$. Then $ess_{\text{cfl}}(t_1, t_2) \cong \bot$ if and only if $ess_{\text{dbl}}(t_1, t_2) \cong \bot$ and $ess_{\text{dbl}}(t_2, t_1) \cong \bot$, which is equivalent to having no disablings and thus parallel independence. □

Another important property of disabling essences caused by a rule ρ_1 is that they are contained in the deletion object of ρ_1 (Definition 2.68). This means that every essence contains only deleted elements, or elements of their boundary.

The next result is exploited in Section 5.1.2 to relate our notion to the *disabling reasons* proposed by Lambers, Ehrig and Orejas (2008). It may also be the basis for a precise comparison of conflict essences with the *basic conflict conditions* introduced by Born et al. (2017), which is left for future work.

Lemma 5.7. In any adhesive category, let $(t_1, t_2) : H_1 \overset{\rho_1, m_1}{\Longleftarrow} G \overset{\rho_2, m_2}{\Longrightarrow} H_2$ be a pair of transformations with disabling essence $ess_{\text{dbl}}(t_1, t_2) = (e_1, e_2) \in \mathbf{Sub}(m_1, m_2)$, and let the deletion object of ρ_1 be $c_{l1} \in \mathbf{Sub}(L_1)$. Then $e_1 \subseteq_{L_1} c_{l1}$. Equivalently, the disabling essence factors uniquely through the deletion object, that is, there is a unique mono $\overline{e_1} : C \rightarrowtail C_{l1}$ with $e_1 = c_{l1} \circ \overline{e_1}$. Moreover, $ess_{\overline{l_1}}(m_1 \circ c_{l1}, m_2) \cong (\overline{e_1}, e_2)$.

$$\begin{tikzcd}
B_{l1} \arrow[r, tail, "\overline{l_1}"] \arrow[d, tail, "b_{l1}"'] \arrow[dr, phantom, "\textcircled{1}"] & C_{l1} \arrow[d, tail, "c_{l1}"] & C \arrow[l, tail, dotted, "\overline{e_1}"'] \arrow[dl, tail, "e_1"'] \arrow[d, tail, "c"] \arrow[dr, tail, "e_2"] & \\
K_1 \arrow[r, tail, "l_1"'] & L_1 \arrow[dr, tail, "m_1"'] & L_1L_2 \arrow[l, tail] \arrow[r, tail] & L_2 \arrow[dl, tail, "m_2"] \\
& & G &
\end{tikzcd}$$

(5.3)

Proof. Recall that the deletion object is obtained by the initial pushout ① over l_1, as in diagram (5.3). Note that any morphism $\overline{e_1} : C \to C_{l1}$ making diagram (5.3) commute is monic, since $c_{l1} \circ \overline{e_1} = e_1$ is monic, and unique, since c_{l1} is monic. Thus, we need only show that such a morphism exists.

First, construct square ③ and rectangle ② + ③ of diagram (5.4) as pullbacks. By pullback decomposition, there is a unique h_L making square ② a pullback.

$$\begin{tikzcd}
C_{l1}L_2 \arrow[r, tail, dotted, "h_L"] \arrow[rr, tail, bend left, "\overline{p_2}"] \arrow[d, tail, "\overline{p_1}"] \arrow[dr, phantom, "\textcircled{2}"] & L_1L_2 \arrow[r, tail, "p_2"] \arrow[d, tail, "p_1"] \arrow[dr, phantom, "\textcircled{3}"] & L_2 \arrow[d, tail, "m_2"] \\
C_{l1} \arrow[r, tail, "c_{l1}"] & L_1 \arrow[r, tail, "m_1"'] & G
\end{tikzcd}$$

(5.4)

$$\begin{tikzcd}
B \arrow[r, tail, "b"] \arrow[rr, tail, dotted, bend left, "b^*"] \arrow[d] \arrow[dr, phantom, "\textcircled{4}"] & K_1L_2 \arrow[d, tail, "q_2"] \arrow[dr, phantom, "\textcircled{5}"] & B_{l1}L_2 \arrow[l, tail, "h_K"'] \arrow[d, tail, "\overline{q_2}"] \\
C \arrow[r, tail, "c"] \arrow[rr, tail, dotted, bend right, "c^*"'] & L_1L_2 & C_{l1}L_2 \arrow[l, tail, "h_L"']
\end{tikzcd}$$

(5.5)

Next, consider diagram (5.7), where the front right face and the back left face ⑥ are constructed as pullbacks. Note that the front left face is the pullback square ②. Then there is a unique h_K making the cube commute and the back right face a pullback, by pullback composition and decomposition. Since the bottom face is a pushout by construction, and the vertical faces are all pullbacks, by adhesivity the top face is a pushout.

$$\begin{tikzcd}
B \arrow[r] \arrow[d, tail, "b^*"'] \arrow[dr, phantom, "\textcircled{4}+\textcircled{5}"] & C \arrow[d, tail, "c^*"] & \\
B_{l1}L_2 \arrow[r, dashed, "\overline{q_2}"] \arrow[d, tail, "\overline{q_1}"'] \arrow[dr, phantom, "\textcircled{6}"] & C_{l1}L_2 \arrow[r, tail, "\overline{p_2}"] \arrow[d, tail, "\overline{p_1}"'] \arrow[dr, phantom, "\textcircled{2}+\textcircled{3}"] & L_2 \arrow[d, tail, "m_2"] \\
B_{l1} \arrow[r, "\overline{l_1}"'] & C_{l1} \arrow[r, tail, "m_1 \circ c_{l1}"'] & G
\end{tikzcd}$$

(5.6)

$$\begin{tikzcd}
& & B_{l1}L_2 \arrow[dll, tail, "\overline{q_2}"'] \arrow[dr, tail, dotted, "h_K"] \arrow[dd, tail, "\overline{q_1}"] & \\
C_{l1}L_2 \arrow[dr, tail, "h_L"] \arrow[dd, tail, "\overline{p_1}"'] & \textcircled{6} & & K_1L_2 \arrow[dll, tail, "q_2"'] \arrow[dd, tail, "q_1"] \\
& L_1L_2 \arrow[dd, tail, "p_1"] & B_{l1} \arrow[dll, tail, "\overline{l_1}"'] \arrow[dr, tail, "b_{l1}"] & \\
C_{l1} \arrow[dr, tail, "c_{l1}"'] & & & K_1 \arrow[dll, tail, "l_1"] \\
& L_1 & &
\end{tikzcd}$$

(5.7)

Now consider diagram (5.5), where square $\textcircled{4}$ is an initial pushout over q_2 so that $e_1 = p_1 \circ c$ and $e_2 = p_2 \circ c$. Square $\textcircled{5}$ corresponds to the top face in diagram (5.7), which is a pushout. Then there exist unique b^* and c^* making the diagram commute.

Finally, if we define $\overline{e_1} = \overline{p_1} \circ c^*$, it follows from commutativity of diagrams (5.5) and (5.7) that $c_{l1} \circ \overline{e_1} = p_1 \circ c = e_1$. Moreover, we obtain from the initial pushout over $\overline{q_2}$, as in diagram (5.6), the proto-essence $ess_{\overline{l_1}}(m_1 \circ c_{l_1}, m_2) \cong (\overline{p_1} \circ c^*, \overline{p_2} \circ c^*) = (\overline{e_1}, e_2)$. □

5.1.1 Conflict Essence and Extension

Recall that the *extension* of a transformation step into a larger context (Definition 2.47) underlies the concept of *completeness* of critical pairs and initial conflicts. In fact, any pair of conflicting steps is the extension of a critical pair (Theorem 2.56) and of an initial conflict. Thus, it is important to understand how the the root causes for conflicts behave with respect to extension.

Lambers et al. (2018) has already shown that conflicts are *reflected* by extension, i.e. when the extension of a transformation pair is in conflict, the original pair of steps is in conflict as well. It turns out they are also *preserved* by extension in categories of set-valued functors, and in particular of graphs and typed graphs. Furthermore, conflict *essences* are also preserved, which means the root causes of a conflict don't change with extension.

The central notion for these results is *proto-essence inheritance*, from which we can prove inheritance of conflict and disabling essences.

Definition 5.8 (Proto-Essence Inheritance)**.** A category $\mathbb{C}$ has *proto-essence inheritance* when the following statement holds for all morphisms $l_1, \overline{m_1}, \overline{m_2}$ and f as in diagram (5.8).

If the composites $f \circ \overline{m_1}$ and $f \circ \overline{m_2}$ are monic and squares $\textcircled{1}$ and $\textcircled{2}$ in diagram (5.8) are pushouts, then the proto-essences over l_1 for $(f \circ \overline{m_1}, f \circ \overline{m_2})$ and $(\overline{m_1}, \overline{m_2})$ are the same. That is, $ess_{l_1}(f \circ \overline{m_1}, f \circ \overline{m_2}) \cong ess_{l_1}(\overline{m_1}, \overline{m_2}) \cong (e_1, e_2) \in \mathbf{Sub}(L_1, L_2)$, which is witnessed by an isomorphism h as in diagram (5.9).

$$\begin{tikzcd}
 & & C \arrow[dl, "e_1"'] \arrow[dr, "e_2"] & \\
K_1 \arrow[r, "l_1"] \arrow[d, "\overline{k_1}"'] \arrow[dr, phantom, "\textcircled{1}"] & L_1 \arrow[dr, "\overline{m_1}"'] & & L_2 \arrow[dl, "\overline{m_2}"] \\
\overline{D_1} \arrow[rr, "\overline{g_1}"] \arrow[d, "d_1"'] \arrow[drr, phantom, "\textcircled{2}"] & & \overline{G} \arrow[d, "f"] & \\
D_1 \arrow[rr, "g_1"] & & G &
\end{tikzcd} \tag{5.8}$$

$$\begin{tikzcd}
 & \overline{C} \arrow[dl, bend right, "\overline{e_1}"'] \arrow[dr, bend left, "\overline{e_2}"] \arrow[d, <->, dotted, "h"] & \\
L_1 & C \arrow[l, "e_1"'] \arrow[r, "e_2"] & L_2
\end{tikzcd} \tag{5.9}$$

Theorem 5.9 (Essence Inheritance)**.** In an adhesive category with proto-essence inheritance, let $(\overline{t_1},\overline{t_2}) : \overline{H_1} \overset{\rho_1,\overline{m_1}}{\Longleftarrow} \overline{G} \overset{\rho_2,\overline{m_2}}{\Longrightarrow} \overline{H_2}$ and $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ be pairs of transformation steps and $f : \overline{G} \to G$ be any morphism.

(i) If the left extension diagram of (5.10) exists, then the transformation pairs share a disabling essence, that is, $ess_{\mathrm{dbl}}(t_1,t_2) \cong ess_{\mathrm{dbl}}(\overline{t_1},\overline{t_2}) \in \mathbf{Sub}(L_1,L_2)$.

(ii) If both extension diagrams of (5.10) exist, then the transformation pairs share a conflict essence, that is, $ess_{\mathrm{cfl}}(t_1,t_2) \cong ess_{\mathrm{cfl}}(\overline{t_1},\overline{t_2}) \in \mathbf{Sub}(L_1,L_2)$.

$$\begin{tikzcd}
\overline{H_1} \arrow[d] & \overline{G} \arrow[l, Rightarrow, "\overline{t_1}"'] \arrow[r, Rightarrow, "\overline{t_2}"] \arrow[d, dashed, "f"] & \overline{H_2} \arrow[d] \\
H_1 & G \arrow[l, Rightarrow, "t_1"'] \arrow[r, Rightarrow, "t_2"] & H_2
\end{tikzcd} \tag{5.10}$$

Proof. Inheritance of disabling essences follows directly from from proto-essence inheritance. In fact, if the left extension diagram in (5.10) exists, then $f \circ \overline{m_1} = m_1$ and $f \circ \overline{m_2} = m_2$ are monomorphisms, and we have the necessary pushouts of diagram (5.8). Then we have $ess_{\mathrm{dbl}}(t_1,t_2) \cong ess_{l_1}(m_1,m_2) \cong ess_{l_1}(\overline{m_1},\overline{m_2}) \cong ess_{\mathrm{dbl}}(t_1,t_2)$.

Inheritance of conflict essences, in turn, follows directly from the previous point:

$$\begin{aligned} ess_{\mathrm{cfl}}(t_1,t_2) &\cong ess_{\mathrm{dbl}}(t_1,t_2) \cup ess_{\mathrm{dbl}}(t_2,t_1) \\ &\cong ess_{\mathrm{dbl}}(\overline{t_1},\overline{t_2}) \cup ess_{\mathrm{dbl}}(\overline{t_2},\overline{t_1}) \cong ess_{\mathrm{cfl}}(\overline{t_1},\overline{t_2}) \end{aligned}$$ □

Corollary 5.10. In an adhesive category with proto-essence inheritance, assume the extension diagrams of (5.10) exist. Then t_1 disables t_2 if and only if $\overline{t_1}$ disables $\overline{t_2}$. Furthermore, t_1 and t_2 are in conflict if and only if $\overline{t_1}$ and $\overline{t_2}$ are in conflict.

Importantly, all categories of set-valued functors have proto-essence inheritance. Then the previous results hold in many categories of interest, including those of graphs and typed graphs.

Lemma 5.11. Every category $\mathbb{S}\mathrm{et}^{\mathbb{S}}$ of set-valued functors has proto-essence inheritance.

Proof. Given any l_1, $\overline{m_1}$, $\overline{m_2}$ and f as in diagram (5.8), let $m_1 = f \circ \overline{m_1}$ and $m_2 = f \circ \overline{m_2}$. Moreover, assume squares ① and ② are pushouts in diagram (5.8).

Now consider diagram (5.11) where $L_1 \xleftarrow{\overline{p_1}} \overline{L_1L_2} \xrightarrow{\overline{p_2}} L_2$ and $L_1 \xleftarrow{p_1} L_1L_2 \xrightarrow{p_2} L_2$ are pullbacks for $(\overline{m_1},\overline{m_2})$ and (m_1,m_2), respectively, and $h_L : \overline{L_1L_2} \to L_1L_2$ their unique mediating morphism. Constructing the squares ④ and ③ + ④ as pullbacks, by decomposition there is a unique monomorphism h_K making ③ a pullback. Recall also that $ess_{l_1}(m_1,m_2) = (p_1 \circ c, p_2 \circ c)$ and $ess_{l_1}(\overline{m_1},\overline{m_2}) = (\overline{p_1} \circ \overline{c}, \overline{p_2} \circ \overline{c})$, where c_1 and $\overline{c_1}$ are the contexts of the initial pushout over q_2 and $\overline{q_2}$, respectively.

We will show that square ③ is also a pushout. Then, since initial pushouts are reflected by pushouts along monomorphisms (Lemmas 2.72 and 2.74), there is an isomorphism $h_C : \overline{C} \to C$ with $c \circ h_C = h_L \circ \overline{c}$. This witnesses $ess_{l_1}(m_1, m_2) \cong ess_{l_1}(\overline{m_1}, \overline{m_2}) \in \mathbf{Sub}(L_1, L_2)$, since $(p_1 \circ c) \circ h_C = \overline{p_1} \circ h_L \circ \overline{c} = \overline{p_1} \circ \overline{c}$ and analogously $(p_2 \circ c) \circ h_C = \overline{p_1} \circ \overline{c}$.

$$\begin{tikzcd}
\overline{K_1L_2} \arrow[r, tail, dashed, "h_K"] \arrow[rr, tail, bend left, "\overline{q_1}"] \arrow[d, tail, "\overline{q_2}"] \arrow[dr, phantom, "③"] & K_1L_2 \arrow[r, tail, "q_1"] \arrow[d, tail, "q_2"] \arrow[dr, phantom, "④"] & K_1 \arrow[r, "\bar{k}"] \arrow[d, tail, "l_1"] \arrow[dr, phantom, "①"] & \overline{D_1} \arrow[r, "d_1"] \arrow[d, tail, "\overline{g_1}"] \arrow[dr, phantom, "②"] & D_1 \arrow[d, tail, "g_1"] \\
\overline{L_1L_2} \arrow[r, tail, "h_L"] \arrow[rr, tail, bend right, "\overline{p_1}"'] & L_1L_2 \arrow[r, tail, "p_1"] & L_1 \arrow[r, tail, "\overline{m_1}"] \arrow[rr, tail, bend right, "m_1"'] & \overline{G} \arrow[r, "f"] & G
\end{tikzcd} \tag{5.11}$$

To show that square ③ is a pushout, assume without loss of generality that all vertical morphisms of (5.11) are inclusions, as well as h_L and h_K. Then by Lemma 3.13 it suffices to show that (h_L, q_2) is jointly epic, which can be done by set-theoretical reasoning according to Remark 3.11. We will show that every element $x \in L_1L_2$ that is not in $\overline{L_1L_2}$ must be in K_1L_2.

So assume such an $x \in L_1L_2 \setminus \overline{L_1L_2}$ and consider the elements $y_1 = \overline{m_1}(p_1(x)) \in \overline{G}$ and $y_2 = \overline{m_2}(p_2(x)) \in \overline{G}$. These elements of $\overline{G}$ are distinct, but identified by f. In fact, $x \notin \overline{L_1L_2}$ implies that $\overline{m_1}(p_1(x)) \neq \overline{m_2}(p_2(x))$, that is, $y_1 \neq y_2$. On the other hand, since (p_1, p_2) is a pullback of (m_1, m_2), we have $m_1(p_1(x)) = m_2(p_2(x))$, which is equivalent to $f(y_1) = f(y_2)$.

Since ② is a pushout and pullback, and $f(y_1) \in G$ has two distinct preimages by f, it follows from Lemma 3.13 that $f(y_1)$ has a unique preimage by g. That is, $f(y_1) \in D_1 \subseteq G$. Then, since ④ + ① + ② is a pullback and $f(\overline{m_1}(p_1(x))) = f(y_1) = g_1(d_1(y_1))$, x must have a preimage by q_2. That is, $x \in K_1L_2 \subseteq L_1L_2$.

In conclusion, every $x \in L_1L_2 \setminus \overline{L_1L_2}$ is such that $x \in K_1L_2$ and therefore ③ is a pushout. This implies that q_2 and $\overline{q_2}$ have isomorphic initial pushouts, witnessing $ess_{l_1}(m_1, m_2) \cong ess_{l_1}(\overline{m_1}, \overline{m_2}) \in \mathbf{Sub}(L_1, L_2)$. □

5.1.2 Comparing with Previous Work

Conflict essences are not the first attempt to formally characterise the root causes of conflicts. In fact, Lambers, Ehrig and Orejas (2008) proposed a similar notion of *conflict reasons*. Nonetheless, we claim that essences have several advantages, as we will show in this section. We begin by recalling the definition of conflict reason.

Definition 5.12 (Disabling and Conflict Reason)**.** In an adhesive category, let (t_1,t_2) : $H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ be a pair of transformation steps.

(i) The *disabling reason* $s_1 \in \mathbf{Sub}(m_1,m_2)$ for (t_1,t_2) is defined as follows[1]. Let the deletion object for ρ_1 be $c_{l1} \in \mathbf{Sub}(L_1)$, obtained by the initial pushout ① over l_1 in diagram (5.12). Let also (o_1,s_{12}) be the pullback of $(m_1 \circ c_{l1}, m_2)$. Then we define $s_1 \cong (s_{11},s_{12}) \in \mathbf{Sub}(m_1,m_2)$, where $s_{11} = c_{l1} \circ o_1$.

$$\begin{tikzcd}
 & & S_1 \arrow[dll, dotted, bend right, "b^*"'] \arrow[dl, "o_1"'] \arrow[ddr, "s_{12}"] & \\
B_{l1} \arrow[r, "l'_1"] \arrow[d, "b_{l1}"'] \arrow[dr, phantom, "\textcircled{1}"] & C_{l1} \arrow[d, "c_{l1}"] & & \\
K_1 \arrow[r, "l_1"'] & L_1 \arrow[dr, "m_1"'] & & L_2 \arrow[dl, "m_2"] \\
 & & G &
\end{tikzcd} \tag{5.12}$$

(ii) A disabling reason satisfies the *conflict condition* if and only if there is no morphism $b^* : S_1 \to B_{l1}$ making diagram (5.12) commute.

(iii) The *conflict reason* $s \in \mathbf{Sub}(m_1,m_2)$ for (t_1,t_2) is defined as follows. Let the disabling reasons for (t_1,t_2) and (t_2,t_1) be $s_1, s_2 \in \mathbf{Sub}(m_1,m_2)$, respectively. If both s_1 and s_2 satisfy the conflict condition, then $s \cong s_1 \cup s_2 \in \mathbf{Sub}(L_1L_2)$. If only s_1 satisfies the conflict condition, then $s \cong s_1$; analogously for s_2.

Note that the relation between disabling reasons and any condition of parallel independence is not very direct. Lambers, Ehrig and Orejas (2008) have shown that a disabling exists (in the sense of Definition 2.43) if and only if the corresponding disabling reason satisfies the conflict condition, but the proof is much more involved than that of Theorem 5.6.

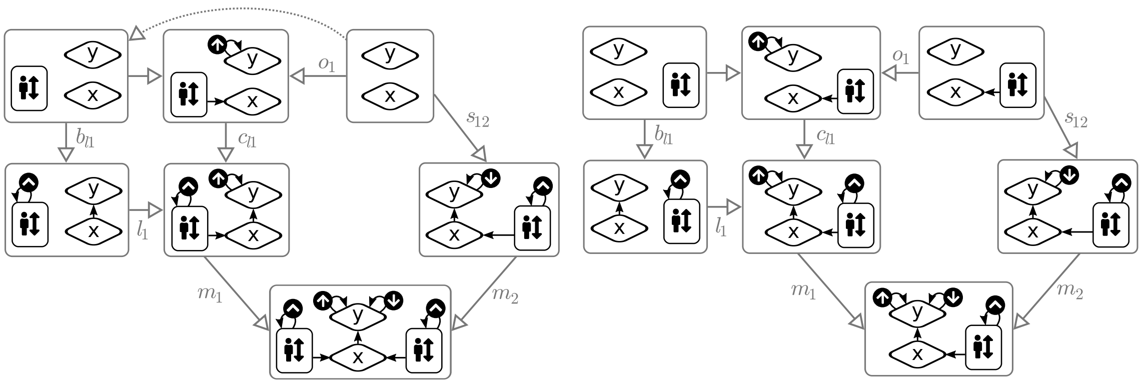
Interestingly, both Definitions 5.1 and 5.12 use the same operations, but in reversed orders. More explicitly, the disabling reason is obtained by first taking the context of (the initial pushout over) l_1, containing all elements deleted by ρ_1 (and the boundary), and then the intersection with the image of m_2. The disabling essence, on the other hand, first restricts on the elements which are matched by both transformation steps, and then takes the context, thus filtering out boundary elements of l_1 that are not relevant for the conflict.

[1] The span we call a disabling reason was actually unnamed in the original definition.

This suggests that disabling essences are in general smaller than disabling reasons, as illustrated by the following example.

Example 5.13. Recall the pairs of transformations from Examples 2.45 and 2.46, involving the rules `move-up-NU` and `move-up-ND`. Conflict reasons for them are constructed in Fig. 5.4. In Fig. 5.4a, even though the transformations were independent, the reason contains both floors. In Fig. 5.4b, the floor *y* is part of the reason despite not being involved in the conflict.

Figure 5.4: Examples of disabling reason.


(a) Disabling reason for Example 2.45.

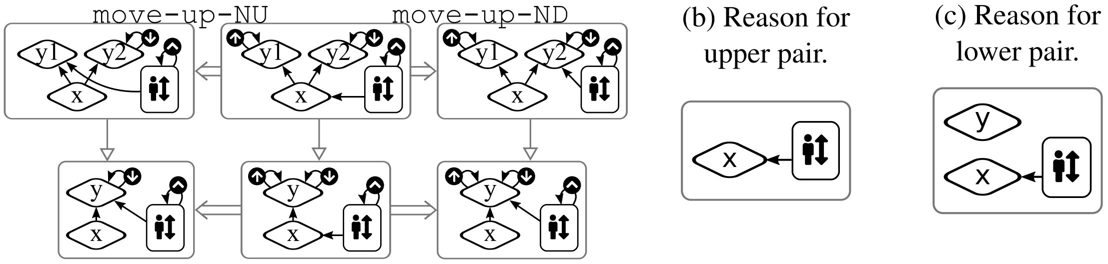
(b) Disabling reason for Example 2.46.

As Example 5.13 shows, the disabling and conflict reasons may contain elements that aren't directly related to the conflict. In the case of graphs, these are *isolated boundary nodes*, as noted by Lambers et al. (2018). That is, they are nodes adjacent to a deleted edge, where this deletion does not cause a disabling. Comparing with Example 5.5 indicates that this is not the case for the essences.

The presence of isolated boundary nodes provides another disadvantage: extending a transformation pair may modify the disabling reason by introducing new isolated boundary nodes, as shown in Fig. 5.5. This cannot happen for conflict essences, as proved in Theorem 5.9.

Figure 5.5: Extended transformation steps with distinct disabling reasons.

(a) Extension diagrams from Example 2.48


(b) Reason for upper pair.

(c) Reason for lower pair.

We conclude this section with a formal proof that every conflict essence is more precise than the corresponding reason, that is, essences are contained in reasons.

Theorem 5.14 (Precision of Essences)**.** In any adhesive category, let the pairs of transformations $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ have conflict (or disabling) essence $ess(t_1,t_2)$ and reason $s \in \mathbf{Sub}(m_1,m_2)$. Then $ess(t_1,t_2) \subseteq s \in \mathbf{Sub}(m_1,m_2)$, witnessed by a unique monomorphism $h : C \rightarrowtail S$ that commutes with both spans.

Proof. [Disabling] Let C_{l1} be the context of (the initial pushout over) l_1. By Lemma 5.7, there is a unique monomorphism $\overline{e_1}$ with $e_1 = c_{l1} \circ \overline{e_1}$, which makes the outer square of diagram (5.13) commute. Now recall from Definition 5.12 that the inner square is a pullback. Then there is a unique h making diagram (5.13) commute.

$$\begin{tikzcd}
C \arrow[r, dotted, "h"'] \arrow[rr, bend left, "e_2"] \arrow[dr, bend right, "\overline{e_1}"'] & S_1 \arrow[r, "s_{12}"'] \arrow[d, "o_1"] & L_2 \arrow[d, "m_2"] \\
& C_{l1} \arrow[r, "m_1 \circ c_{l1}"'] & G
\end{tikzcd} \tag{5.13}$$

[Conflict] If only s_1 satisfies the conflict condition, then $s = s_1$ by Definition 5.12. In this case t_2 does not disable t_1 and, by Theorem 5.6, the disabling essence c_2 is the bottom of $\mathbf{Sub}(m_1,m_2)$, which implies $c \cong c_1 \cup \bot \cong c_1$, and thus $c \cong c_1 \subseteq s_1 \cong s$.

The case when only s_2 satisfies the conflict condition is symmetrical. If both s_1 and s_2 satisfy it, then $s = s_1 \cup s_2$. Then since $c_1 \subseteq s_1$ and $c_2 \subseteq s_2$, it follows from distributivity of $\mathbf{Sub}(m_1,m_2)$ that $c \cong c_1 \cup c_2 \subseteq s_1 \cup s_2 \cong s$. □

5.2 Potential Essences

In the previous section, we have defined conflict and disabling essences as a characterisation for the root causes of a conflict between two *transformation steps*. In order to analyse a transformation system, however, one requires an overview of all *potential* causes for conflicts for a pair of *rules*. Ideally, given a pair of rules (ρ_1,ρ_2), there should be a set of *potential conflict essences* containing the conflict essences for every pair of steps $H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$. In this section, we will define this set and relate it to initial conflicts. Moreover, we will show that the set of potential essences is often redundant and provide a suitable subset to avoid this redundancy.

We begin with the straightforward definition of *potential essences* for a pair of rules, which is the set of essences for all pairs of transformation steps.

Definition 5.15 (Potential Essences)**.** Let ρ_1 and ρ_2 be transformation rules. Then the sets of *potential conflict and disabling essences* for (ρ_1,ρ_2) are defined as follows.

$$\mathbf{Ess}_{\text{dbl}}(\rho_1,\rho_2) = \left\{ess_{\text{dbl}}(t_1,t_2) \mid (t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2, \text{ object } G\right\}$$
$$\mathbf{Ess}_{\text{cfl}}(\rho_1,\rho_2) = \left\{ess_{\text{cfl}}(t_1,t_2) \mid (t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2, \text{ object } G\right\}$$

Remark 5.16. We will write $\mathbf{Ess}(\rho_1,\rho_2)$ when the context makes it unambiguous.

Remark 5.17. Note that a pair of rules (ρ_1,ρ_2) may determine an infinite set of transformation pairs $\left\{(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2 \mid \text{object } G\right\}$. Nevertheless, since multiple pairs of steps may share the same essence, $\mathbf{Ess}(\rho_1,\rho_2)$ may be finite. In fact, since $\mathbf{Ess}(\rho_1,\rho_2) \subseteq \mathbf{Sub}(L_1) \times \mathbf{Sub}(L_2)$, rules with finite left-hand sides will always have a finite set of essences.

Remark 5.18. Henceforth, we will often refer to *concrete essences* for a pair of steps, as opposed to potential essences for a pair of rules.

Fact 5.19. Potential essences are subobjects for (L_1,L_2), i.e. $\mathbf{Ess}(\rho_1,\rho_2) \subseteq \mathbf{Sub}(L_1,L_2)$.

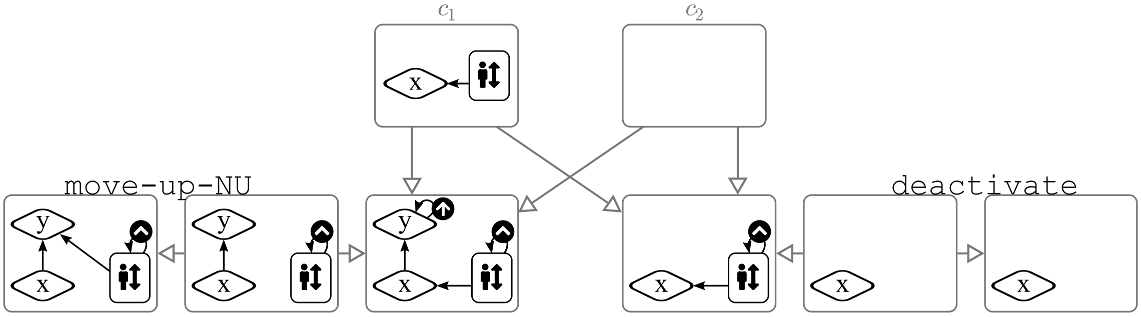
Clearly, every potential conflict essence is a union of potential disabling essences. However, the converse does not hold: some unions of potential disabling essences are *not* conflict essences, as the following example shows.

Fact 5.20. Potential conflict essences are always unions of potential disabling essences. That is, every $e \in \mathbf{Ess}_{\text{cfl}}(\rho_1,\rho_2)$ is such that $e = e_1 \cup e_2$ for some $e_1 \in \mathbf{Ess}_{\text{dbl}}(\rho_1,\rho_2)$ and $e_2 \in \mathbf{Ess}_{\text{dbl}}(\rho_2,\rho_1)$.

Proof. Follows directly from Definitions 5.1 and 5.15. □

Example 5.21. Recall the rules `move-up-NU` and `deactivate`, introduced in Examples 2.6 and 2.38. A potential disabling essence caused by `move-up-NU`, as shown in Fig. 5.6, contains only the `at` edge. Its union with the empty disabling essence caused by `deactivate`, however, is *not* a conflict essence. In fact, any match that commutes with this union must also identify the `going-up` edges. Otherwise the match of `deactivate` will not be applicable since it leaves the unmatched `going-up` edge dangling. Then the conflict essence will include the `going-up` edge, which is not present in the aforementioned union.

Figure 5.6: Potential disabling essences for $\rho_1 = \texttt{move-up-NU}$ and $\rho_2 = \texttt{deactivate}$, where $c_1 \in \mathbf{Ess}_{\mathrm{dbl}}(\rho_1, \rho_2)$ and $c_2 \in \mathbf{Ess}_{\mathrm{dbl}}(\rho_2, \rho_1)$.



5.2.1 Potential Essences and Initial Conflicts

In previous work, potential conflicts were characterised with critical pairs or with initial conflicts. It turns out that potential conflict essences are closely related to initial conflicts. In fact, in categories with proto-essence inheritance and another property we will now introduce, these concepts are in one-to-one correspondence.

Recall that initial conflicts are *initial transformation pairs* (Definition 2.57), where initiality is postulated with respect to embedding into larger contexts. Their existence, however, is not guaranteed in every category. In fact, Lambers et al. (2018) have only proved their existence in categories of graphs and typed graphs. It is an open problem whether they exist in all adhesive categories.

It turns out that, in categories of set-valued functors, concrete conflict essences can be used to construct initial transformation pairs. Indeed, this result holds in any adhesive category with two additional assumptions: proto-essence inheritance (Definition 5.8) and the PO-PB decomposition via proto-essence that we now introduce.

The following theorem shows that initial transformation pairs exist in any adhesive category with proto-essence inheritance and PO-PB decomposition via proto-essence. These properties should, in general, be much simpler to prove than the complete theorem.

Definition 5.22 (PO-PB Decomposition via Proto-Essence)**.** An adhesive category has *PO-PB decomposition via proto-essence* when the following statement holds for every diagram as in (5.14). Given $ess_{l_1}(f \circ n_1, f \circ n_2) \cong (e_1, e_2)$ such that conditions (i)–(iv) hold, then both squares ① and ② are pushouts.

(i) Rectangle ① + ② is a pushout and square ② is a pullback.

(ii) Diagram (5.14) commutes.

(iii) Morphisms l_1, $f \circ n_1$ and $f \circ n_2$ are monic.

(iv) The pair (n_1, n_2) is jointly epic.

$$\begin{tikzcd}
K_1 \arrow[r, "k_1'"] \arrow[d, tail, "l_1"'] \arrow[dr, phantom, "\textcircled{1}"] & D_1' \arrow[r, "f'"] \arrow[d, "g'"] \arrow[dr, phantom, "\textcircled{2}"] & D_1 \arrow[d, "g"] \\
L_1 \arrow[r, "n_1"] & I \arrow[r, "f"] & G \\
C \arrow[u, "e_1"] \arrow[r, "e_2"] & L_2 \arrow[u, "n_2"] &
\end{tikzcd} \tag{5.14}$$

Theorem 5.23 (Construction of Initial Transformation Pairs)**.** In any adhesive category with proto-essence inheritance and PO-PB decomposition via proto-essences, let $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ be a pair of transformation steps. Then the pushout $L_1 \overset{n_1}{\rightarrow} I \overset{n_2}{\leftarrow} L_2$ of the conflict essence $ess_{\text{cfl}}(t_1,t_2) \cong L_1 \overset{e_1}{\leftarrow} C \overset{e_2}{\rightarrow} L_2$ determines an initial transformation pair $(s_1,s_2) : J_1 \overset{\rho_1,n_1}{\Longleftarrow} I \overset{\rho_2,n_2}{\Longrightarrow} J_2$ for (t_1,t_2).

Proof. According to Definition 2.57, we have to show that the pushout $L_1 \overset{n_1}{\rightarrow} I \overset{n_2}{\leftarrow} L_2$ *(a)* determines a transformation pair (s_1,s_2) which *(b)* can be embedded into (t_1,t_2) via some morphism f and *(c)* can be embedded into any other pair of transformation steps $(\overline{t_1},\overline{t_2})$ that is embedded into (t_1,t_2).

Note that *(c)* follows from *(b)*. In fact, by essence inheritance (Theorem 5.9), we have $ess(\overline{t_1},\overline{t_2}) \cong ess(t_1,t_2)$. Then (n_1,n_2) is also the pushout of $ess_{\text{cfl}}(\overline{t_1},\overline{t_2})$, and it follows from *(b)* that the corresponding pair of steps can be embedded into $(\overline{t_1},\overline{t_2})$.

To prove *(a)* and *(b)*, we will construct diagram (5.15), where rectangle $\textcircled{1}+\textcircled{2}$ is the pushout determining step t_1. First, we obtain f as the unique mediating morphism from the pushout (n_1,n_2) constructing square $\textcircled{2}$ as a pullback. Then we also obtain a mediating morphism k_1' making the diagram commute.

$$\begin{tikzcd}
K_1 \arrow[r, dotted, "k_1'"] \arrow[rr, tail, bend left, "k_1"] \arrow[d, tail, "l_1"'] \arrow[dr, phantom, "\textcircled{1}"] & D_1' \arrow[r, dashed, "f'"] \arrow[d, dashed, "g'"] \arrow[dr, phantom, "\textcircled{2}"] & D_1 \arrow[d, "g"] \\
L_1 \arrow[r, "n_1"] \arrow[rr, tail, bend left, "m_1"] & I \arrow[r, dashed, "f"] & G \\
C \arrow[u, "e_1"] \arrow[r, "e_2"] & L_2 \arrow[u, "n_2"] \arrow[ur, tail, bend right, "m_2"'] &
\end{tikzcd} \tag{5.15}$$

Now it follows from PO-PB decomposition via proto-essence that squares $\textcircled{1}$ and $\textcircled{2}$ are pushouts. The former ensures the existence of step s_1, as required by *(a)*, while the latter ensures that s_1 can be embedded into t_1, as required by *(b)*. We omit the analogous proof that a step s_2 exists and can be embedded into t_2. □

Corollary 5.24. In any adhesive category with proto-essence inheritance and PO-PB decomposition via proto-essences, sets of initial conflicts exist and solve the problem of conflict detection (Problem 2.49).

Having a construction for initial transformation pairs, we can now provide a simple characterisation for initial conflicts. In fact, we can show they correspond bijectively to potential conflict essences.

Corollary 5.25. In an adhesive category with proto-essence inheritance and PO-PB decomposition via proto-essences, a conflict $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ is initial if and only if its essence is the pullback of its matches, i.e. $ess_{\mathrm{cfl}}(t_1,t_2) \cong \top \in \mathbf{Sub}(m_1,m_2)$.

Theorem 5.26. In an adhesive category with proto-essence inheritance and PO-PB decomposition via proto-essence, the set of initial conflicts for rules ρ_1 and ρ_2 corresponds bijectively to $\mathbf{Ess}_{\mathrm{cfl}}(\rho_1,\rho_2) \setminus \{\bot\}$, up to isomorphism.

Proof. Given a potential essence $e \in \mathbf{Ess}_{\mathrm{cfl}}(\rho_1,\rho_2) \setminus \{\bot\}$, it follows from Theorem 5.23 that its pushout determines an initial transformation pair $(s_1,s_2) : J_1 \overset{\rho_1,n_1}{\Longleftarrow} I \overset{\rho_2,n_2}{\Longrightarrow} J_2$ with $ess_{\mathrm{cfl}}(s_1,s_2) \cong e$. Since $e \ncong \bot$, the pair (s_1,s_2) is in conflict by Theorem 5.6. Thus it is an initial conflict. On the other hand, each initial conflict (s_1,s_2) determines a unique conflict essence $ess_{\mathrm{cfl}}(s_1,s_2)$. Since initial conflicts are isomorphic to their initial transformation pairs, the pushout of $ess_{\mathrm{cfl}}(s_1,s_2)$ determines transformations isomorphic to (s_1,s_2). □

Finally, we show that categories of set-valued functors have PO-PB decomposition by disabling essence. Thus, the previous results hold in many adhesive categories of interest, including those of graphs and typed graphs.

Lemma 5.27. Every category of set-valued functors has PO-PB decomposition via proto-essences.

Proof. Let diagram (5.14) be taken in $\mathbb{Set}^{\mathbb{S}}$, satisfying conditions *(i)–(iii)* from Definition 5.22 and having $ess_{l_1}(f \circ n_1, f \circ n_2) \cong (e_1,e_2)$. Note that, since pushouts and pullbacks preserve monos, g and g' are also monos. Moreover, morphisms e_1, e_2, n_1 and n_2 are monos by the definition of proto-essence and because $f \circ n_1$ and $f \circ n_2$ are monos.

Let us explicitly construct the proto-essence in diagram (5.16). We take the pullback (p_1,p_2) of $(f \circ n_1, f \circ n_2)$, then the pullback ④ of (p_1,l_1) and the initial pushout ③ over q_2. Note that the unlabelled square may not commute, but its composition with f

or c does, that is: $f \circ n_1 \circ p_1 = f \circ n_2 \circ p_2$ and $n_1 \circ p_1 \circ c = n_2 \circ p_2 \circ c$.

$$\begin{tikzcd}
 & & K_1 \arrow[r, tail, "k_1'"] \arrow[d, hook, "l_1"] \arrow[dr, phantom, "\textcircled{1}"] & D_1' \arrow[r, "f'"] \arrow[d, hook, "g'"] \arrow[dr, phantom, "\textcircled{2}"] & D_1 \arrow[d, hook, "g"] \\
B \arrow[r, hook, "b"] \arrow[d, hook, "\overline{q_2}"] \arrow[dr, phantom, "\textcircled{3}"] & K_1L_2 \arrow[ur, tail, "q_1"] \arrow[r, phantom, "\textcircled{4}"] \arrow[d, hook, "q_2"] & L_1 \arrow[r, tail, "n_1"] & I \arrow[r, "f"] & G \\
C \arrow[r, hook, "c"] & L_1L_2 \arrow[r, tail, "p_2"] \arrow[ur, tail, "p_1"] & L_2 \arrow[ur, tail, "n_2"] & &
\end{tikzcd} \tag{5.16}$$

We will show that square ① is a pushout, which by decomposition implies that square ② is also a pushout. Without loss of generality, assume that all vertical morphisms of diagram (5.16) as well as b and c are inclusions, denoted in the diagram by an arrow with a hook.

This can be shown using set-theoretical reasoning, as described in Remark 3.11. Indeed, by Theorem 3.13, it suffices to show that every element $x \in I \setminus n_1(L_1)$ has a unique preimage by g', that is, $x \in D_1' \subseteq I$. Taking any such x, since (n_1, n_2) is jointly epic and $x \notin n_1(L_1)$, we must have $x \in n_2(L_2)$. Thus, there is $y_2 \in L_2$ with $x = n_1(y_2)$.

Now, consider $f(x) \in G$. Recall that $(f \circ n_1, g)$ is a pushout and thus jointly epic, then $f(x)$ must be in the image of $f \circ n_1$ or g. We will show that it is always in the image of g, that is, $f(x) \in D_1$. Then, since ② is a pullback, $x \in D_1'$ as we needed to show.

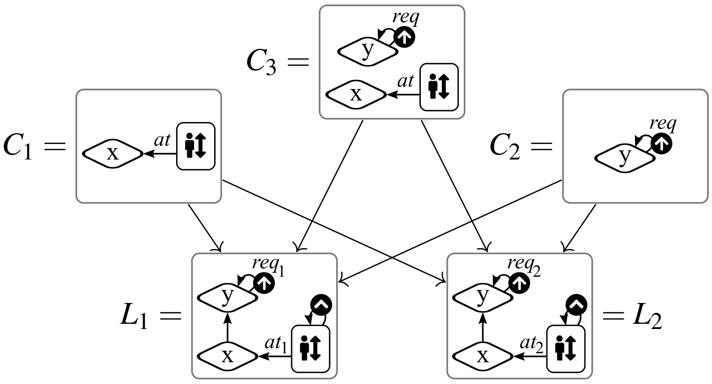
When $f(x)$ is in the image of $f \circ n_1$, there is $y_1 \in L_1$ with $(f \circ n_1)(y_1) = f(x) = (f \circ n_2)(y_2)$. Then since (p_1, p_2) is a pullback, there is a unique $z \in L_1L_2$ with $p_1(z) = y_1$ and $p_2(z) = y_2$. But z cannot be in the proto-essence C, otherwise it would hold that $x = (n_2 \circ p_2 \circ c)(z) = (n_1 \circ p_1 \circ c)(z)$, contradicting the assumption that $x \notin n_1(L_1)$. Then we must have $z \in K_1L_2$, since (c, q_2) is jointly epic. Finally, since squares ①, ② and ④ commute, we have $f(x) = (f \circ n_1 \circ p_1)(z) = (f' \circ k_1' \circ q_1)(z) \in D_1$. □

5.2.2 Irreducible Conflict Essences

An important motivation for developing initial conflicts and potential essences was avoiding redundancy that was observed on critical pairs. Although we have reached this goal, and evidence supporting this will be presented in Chapter 7, we may wonder if there is still any redundancy in potential essences. The following example confirms our suspicions, showing that some potential essences are just the union of other essences.

Example 5.28. Recall that the rule `move-up-NU` presented in Example 2.6 may conflict with itself. Its three potential conflict essences are presented in Fig. 5.7. The first essence contains only the `at` edge, representing situations when both transformations move the same elevator. The second essence contains only the `req-up` edge, representing situations when both transformations fulfil the same request. The last essence is just a union of the previous two.

Figure 5.7: Conflict essences for `move-up-NU`.



When manually inspecting the potential essences for a transformation system, the unions of other essences add little to no information, while increasing the time and effort spent. Ideally, we should select a small subset of potential essences while guaranteeing that all omitted essences are unions of reported essences.

Recall that the irreducible elements of a lattice have this property (Fact 4.18), so we define an analogous concept of irreducible essence. The proof of Fact 4.18 can be trivially adapted for irreducible essences.

Definition 5.29. An *irreducible conflict or disabling essence* $e \in \mathbf{Ess}(\rho_1, \rho_2)$ is a potential essence that is neither the bottom nor decomposable as a union of other essences. That is, $e \ncong \bot$, and $e \cong a \cup b$ implies $a \cong e$ or $b \cong e$ for any $a, b \in \mathbf{Ess}(\rho_1, \rho_2)$.

Theorem 5.30. *Let ρ_1 and ρ_2 be a pair of rules with finite $\mathbf{Sub}(L_1, L_2)$. For every conflict or disabling essence $e \in \mathbf{Ess}(\rho_1, \rho_2)$ there exists a set S of irreducible essences such that $e \cong \bigcup_{\mathbf{Sub}} S$.*

Proof. Note that $x = \bigcup_{\mathbf{Sub}} \{x\}$ holds for every $x \in \mathbf{Ess}(\rho_1, \rho_2)$. Then if x is an irreducible essence, the statement holds trivially. It only remains to show that it holds for non-irreducible elements, which we do by induction on the element's height (Remark 4.24).

[Base] All minimal elements are trivially irreducible.

[Step] Take $x \in \mathbf{Ess}(\rho_1, \rho_2)$ and assume that all elements with lesser height are unions of irreducibles. Then assume $x = y \cup z$ for $y, z \in \mathbf{Ess}(\rho_1, \rho_2)$, with $y \subset x$ and $z \subset y$. If follows from Fact 4.23 that y and z have lesser height than x, so by the induction hypothesis there are sets Y and Z of irreducible essences with $y \cong \bigcup_{\mathbf{Sub}} Y$ and $z \cong \bigcup_{\mathbf{Sub}} Z$. Then we have $x \cong (\bigcup_{\mathbf{Sub}} Y) \cup_{\mathbf{Sub}} (\bigcup_{\mathbf{Sub}} Z) \cong \bigcup_{\mathbf{Sub}} (X \cup Y)$ by Fact 4.14. □

We conclude that a manual analysis of potential essences can often be simplified by taking only irreducible essences. In this case, no information is lost, since all omitted essences are just unions of irreducibles.

6 ENUMERATING POTENTIAL CONFLICTS

In the last few chapters, we have provided a theory describing the potential causes of conflicts for a pair of rules. Now we apply this theory to the problem of conflict detection (Problem 2.49), that is, of enumerating a finite but complete set of conflicts. We restrict our scope to categories of set-valued functors, which generalise many adhesive categories of interest as described in Chapter 3. Some other important categories, such as attributed graphs, are left for future work.

In these categories, we can solve the problem of conflict detection by enumerating all potential conflict essences, since they bijectively correspond to initial conflicts.

Theorem 6.1. In a category of set-valued functors, given rules ρ_1 and ρ_2 with finite left-hand sides, the set $\mathbf{Ess}_{\mathrm{cfl}}(\rho_1, \rho_2)$ solves the problem of conflict detection (Problem 2.49) through its bijective correspondence with initial conflicts.

Proof. First note that $\mathbf{Ess}_{\mathrm{cfl}}(\rho_1, \rho_2)$ is finite. In fact, it is a subset of $\mathbf{Sub}(L_1) \times \mathbf{Sub}(L_2)$, which is finite since the left-hand sides are finite. Moreover, $\mathbf{Ess}_{\mathrm{cfl}}(\rho_1, \rho_2)$ bijectively corresponds to the set of initial conflicts for the rules (Theorem 5.26), which in categories of set-valued functors is complete (Corollary 5.24, Lemmas 5.11 and 5.27). □

In the remainder of this chapter, we discuss the problem of enumerating potential conflict and disabling essences, as well as their irreducible variants. While the enumeration of conflict essences allows us to solve the problem of conflict detection, disabling essences may also be useful for some use cases. In fact, current implementations of conflict detection often enumerate critical pairs indicating a "direction" for the conflict, which was used for example by Mens, Taentzer and Runge (2007).

The main contribution of this chapter are generic algorithms for enumerating conflict and disabling essences as well as post-processing steps for selecting irreducible essences, proven correct for categories of set-valued functors. Being stated in terms of categorical operations, the algorithms could be instantiated for other categories, although the correctness of these instantiations must still be ascertained.

The algorithms described in this chapter were implemented in the Verigraph system, which allows rapid prototyping of new ideas from the theory of algebraic graph transformation. Verigraph is an open source project implemented in Haskell, and its source code was published online by Bezerra et al. (2017). Its architecture, as described by Azzi et al. (2018), provides a series of categorical interfaces, allowing the algorithms to be implemented abstractly in terms of categorical primitives. These implementations can then be instantiated with concrete datatypes that implement different categories, for example those of typed graphs.

Remark 6.2. When implementing concepts of category theory, we face *non-determinism of representation*. For example, when computing the pullback of two given morphisms, the result may be chosen from an infinite number of isomorphic pullbacks. In the following, we always assume that implementations of categorical concepts produce *one arbitrary representative* for each isomorphism class of valid results, e.g. an arbitrary pullback. We do not impose any assumptions over *which representative* is chosen.

This also applies to operations producing a set of results, such as enumerating all jointly epic pairs with given domains. In this case, instead of producing an infinite set with many isomorphic elements, we expect the resulting set to contain exactly one representative for each isomorphism class of elements. We often emphasise this point, writing that a subroutine produces a given set *up to isomorphism*.

6.1 Enumerating Conflict Essences and Initial Conflicts

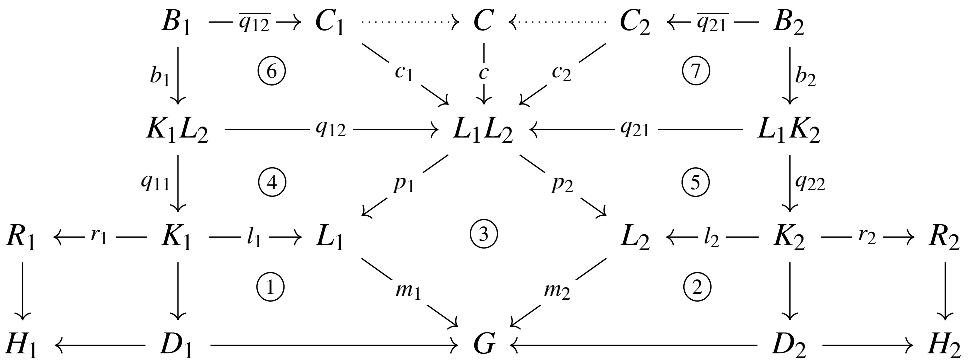
We propose Algorithm 6.1 to enumerate conflict essences, using the same strategy currently employed by Verigraph to enumerate critical pairs. The algorithm inspects all monic overlappings of the left-hand sides, that is, all jointly epic pairs $L_1 \rightarrowtail G \leftarrowtail L_2$, testing if they determine initial conflicts. If they do, they are added to a set of initial conflicts, and their conflict essence is added to the set of potential essences.

Remark 6.3. The Algorithm 6.1 is written at a high level of abstraction, assuming the implementation of certain categorical primitives that are called as subroutines. In particular, it calls subroutines for computing pullbacks and initial pushouts, as well as testing if a pair of morphisms has a pushout complement, if a monomorphism represents the bottom subobject of its codomain and if an arbitrary morphism is in fact an isomorphism. Moreover, the subroutine `jointlyEpicPairsOfMonos`, when given two objects L_1 and L_2, should produce all jointly epic $L_1 \overset{m_1}{\rightarrowtail} G \overset{m_2}{\leftarrowtail} L_2$, up to isomorphism.

Algorithm 6.1: Enumerate Conflict Essences and Initial Conflicts

```
Input  : Rules $\rho_1 = (l_1, r_1)$ and $\rho_2 = (l_2, r_2)$
Output: Set $S$ of initial conflicts and $E$ of conflict essences
— Compare diagram (6.1)

$S := \varnothing$;
$E := \varnothing$;
for $(m_1, m_2) \in$ jointlyEpicPairsOfMonos$(L_1, L_2)$ do
    if hasPushoutComplement$(l_1, m_1)$ and hasPushoutComplement$(l_2, m_2)$
     then                                              — squares ① and ② exist
        $(p_1, p_2) :=$ pullback$(m_1, m_2)$;                    — square ③
        $(q_{11}, q_{12}) :=$ pullback$(l_1, p_1)$;                — square ④
        $(q_{22}, q_{21}) :=$ pullback$(l_2, p_2)$;                — square ⑤
        $(b_1, c_1, \overline{q_{12}}) :=$ initialPushout$(q_{12})$;       — square ⑥
        $(b_2, c_2, \overline{q_{21}}) :=$ initialPushout$(q_{21})$;       — square ⑦
        $c := c_1 \cup_{\mathbf{Sub}} c_2$;
        if not isBottomSubobject$(c)$ and isIsomorphism$(c)$ then
            $S := S \cup \{(m_1, m_2)\}$;
            $E := E \cup \{(p_1 \circ c, p_2 \circ c)\}$;
        end
    end
end
```



$$\begin{tikzcd}
 & B_1 \arrow[r, "\overline{q_{12}}"] \arrow[d, "b_1"'] \arrow[dr, phantom, "\textcircled{6}"] & C_1 \arrow[r, dotted] \arrow[dr, "c_1"] & C \arrow[d, "c"] & C_2 \arrow[l, dotted] \arrow[dl, "c_2"'] & B_2 \arrow[l, "\overline{q_{21}}"'] \arrow[d, "b_2"] \arrow[dl, phantom, "\textcircled{7}"] & \\
 & K_1L_2 \arrow[rr, "q_{12}"] \arrow[d, "q_{11}"'] \arrow[dr, phantom, "\textcircled{4}"] & & L_1L_2 \arrow[dl, "p_1"'] \arrow[dr, "p_2"] & & L_1K_2 \arrow[ll, "q_{21}"'] \arrow[d, "q_{22}"] \arrow[dl, phantom, "\textcircled{5}"] & \\
R_1 \arrow[d] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d] \arrow[dr, phantom, "\textcircled{1}"] & L_1 \arrow[dr, "m_1"] \arrow[rr, phantom, "\textcircled{3}"] & & L_2 \arrow[dl, "m_2"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d] \arrow[dl, phantom, "\textcircled{2}"] & R_2 \arrow[d] \\
H_1 & D_1 \arrow[l] \arrow[rr] & & G & & D_2 \arrow[ll] \arrow[r] & H_2
\end{tikzcd} \tag{6.1}$$

Note that Algorithm 6.1 corresponds directly to the categorical definitions and results. In fact, it should be read in the context of Definitions 2.36 and 5.1 as well as Corollary 5.25, with diagram (6.1) summarising the whole construction. Then, given a correct implementation of the categorical primitives, the correctness of this algorithm is easy to ascertain.

Lemma 6.4. In a category of set-valued functors, the jointly epic pairs with domains L_1 and L_2 correspond bijectively to quotients of the disjoint union $L_1 \uplus L_2$. Thus, if L_1 and L_2 are finite, there is a finite amount of jointly epic pairs (up to isomorphism).

Proof. To establish the bijection, first note that the disjoint union $L_1 \uplus L_2$ is a coproduct of L_1 and L_2. Thus, given a jointly epic pair (m_1, m_2) there is a unique morphism h making diagram (6.2) commute, which moreover must be epic and therefore a quotient of $L_1 \uplus L_2$. On the other hand, given a quotient $h : L_1 \uplus L_2 \twoheadrightarrow G$, we can construct a jointly epic pair $(h \circ j_1, h \circ j_2)$. Clearly this establishes a bijection.

It is trivial in categories of set-valued functors that the disjoint union of finite objects is also finite, and that finite objects have a finite amount of quotients. □

$$\begin{tikzcd}
L_1 \arrow[r, "j_1"] \arrow[dr, two heads, "m_1"'] & L_1 \uplus L_2 \arrow[d, dotted, two heads, "h"] & L_2 \arrow[l, "j_2"'] \arrow[dl, two heads, "m_2"] \\
& G &
\end{tikzcd} \tag{6.2}$$

Theorem 6.5 (Correctness of Algorithm 6.1)**.** Let Algorithm 6.1 be interpreted in a category of set-valued functors, with correct implementations of the categorical primitives that it calls, as described in Remark 6.3. When the input rules ρ_1 and ρ_2 have finite left-hand sides, the algorithm terminates producing a set E that contains exactly the conflict essences for (ρ_1, ρ_2), up to isomorphism. Moreover, set S solves the problem of conflict detection, containing all initial conflicts, up to isomorphism.

Proof. It is easily seen that set S contains only initial conflicts and set E contains only conflict essences, since the algorithm directly reflects the appropriate definitions. Moreover, since each initial conflict is determined by a jointly epic pair of matches, we are guaranteed to enumerate all of them. Termination follows directly from the finite left-hand sides and Lemma 6.4. □

6.1.1 Performance Bottleneck

The strategy employed in Algorithm 6.1 has a clear performance bottleneck: the number of jointly epic pairs that must be inspected. Even in the category of sets, this number grows exponentially with the cardinality of the domains.

Lemma 6.6. Given sets L_1 and L_2 of cardinality n, the number of jointly epic pairs of monomorphisms is $\Omega(2^n)$ and $O(2^n n!)$.

Proof. We can generate the desired pairs with the following process. Choose $0 \leq i \leq n$ elements of L_1 that will be identified with L_2, leaving $n - i$ elements of L_1 disjoint from L_2. Choose also i elements of L_2 that will be identified with L_1. There are $i!$ possible pairings for the chosen subsets of L_1 and L_2. Thus, formula (6.3) expresses the number of jointly epic pairs of monomorphisms, with lower and upper bounds given by equations (6.4) and (6.5).

$$\sum_{i=0}^{n} \binom{n}{i}\binom{n}{i} i! = \sum_{i=0}^{n} \binom{n}{i} \frac{n!}{(n-i)!} \tag{6.3}$$

$$\sum_{i=0}^{n} \binom{n}{i} \frac{n!}{(n-i)!} \geq \sum_{i=0}^{n} \binom{n}{i} = 2^n \tag{6.4}$$

$$\sum_{i=0}^{n} \binom{n}{i} \frac{n!}{(n-i)!} = \sum_{i=0}^{n} \frac{n!^2}{(n-i)!^2\, i!} \leq \sum_{i=0}^{n} \frac{n!^2}{(n-i)!\, i!} = 2^n n! \tag{6.5}$$

□

For graphs, the exact number of jointly epic pairs depends on their topology. Nevertheless, we can find lower and upper bounds for this number, which also grows exponentially with the size of the graph.

Lemma 6.7. Given graphs L_1 and L_2 containing n nodes and e edges each, the number of jointly epic pairs of monomorphisms is $\Omega(2^n)$ and $O(2^{n+e}(n+e)!)$.

Proof. A lower bound can be determined by considering jointly epic pairs of injections from the sets of nodes V_1 and V_2, instead of the full graphs L_1 and L_2. In fact, given any such pair $V_1 \overset{n_1}{\rightarrowtail} V \overset{n_2}{\leftarrowtail} V_2$, we can construct $L_1 \overset{m_1}{\rightarrowtail} G \overset{m_1}{\leftarrowtail} L_2$ by taking $G = (V, E_1 + E_2, s, t)$, where $s = n_1 \circ s_1 + n_2 \circ s_2$ and $t = n_1 \circ t_1 + n_2 \circ t_2$. This determines an injective mapping from jointly epic pairs with domains V_1 and V_2 into those with domains L_1 and L_2. We conclude that the cardinality of the latter set has a lower bound of 2^n.

For the upper bound, consider the forgetful functor $\mathrm{U} : \mathbb{G}\mathrm{raph} \to \mathbb{S}\mathrm{et}$ taking each graph $G = (V, E, s, t)$ to the disjoint union $\mathrm{U}G = V + E$. Each jointly epic $L_1 \overset{m_1}{\rightarrowtail} G \overset{m_2}{\leftarrowtail} L_2$ is mapped into a jointly epic pair of functions $\mathrm{U}L_1 \overset{\mathrm{U}m_1}{\rightarrowtail} \mathrm{U}G \overset{\mathrm{U}m_2}{\leftarrowtail} \mathrm{U}L_2$. Moreover, since functor U is faithful, this mapping is injective. We conclude that the number of jointly epic pairs with domains L_1 and L_2 is bounded by the number of jointly epic pairs with domains $\mathrm{U}L_1$ and $\mathrm{U}L_2$. Since these disjoint unions have cardinality $n+e$, we obtain an upper bound of $2^{n+e}(n+e)!$. □

The lower bounds provided above are also an asymptotic lower bound for the running time of Algorithm 6.1. In fact, since it must execute *at least* a constant-time operation for each jointly epic pair of matches, its time complexity is $\Omega(2^n)$. Thus, its performance is extremely sensitive to the sizes of the left-hand sides. As illustrated by

Table 6.1, removing a single node may reduce the (lower bound for the) number of jointly epic pairs by an order of magnitude, even for small graphs.

Table 6.1: Numbers of jointly epic pairs for small sets.

Cardinality of sets L_1 and L_2	Number of jointly epic pairs of injections
0	1
1	2
2	7
3	34
4	209
5	1546
6	13327
7	130922
8	1441729
9	17572114
10	234662231

6.2 Enumerating Disabling Essences

For some applications, it might be useful to enumerate disabling essences, since they also give a sense of “directionality” to the conflict. It turns out that, in some cases, it might also be computationally cheaper.

Recall from Lemma 5.7 that conflict essences are always contained in the deletion object $c_{l1} : C_{l1} \rightarrowtail L_1$ of the rule ρ_1 that causes the disabling. In other words, any elements of L_1 that are not in the deletion object cannot be in the disabling essence, and therefore need not be identified by the matches. Thus, it should be possible to inspect overlappings of C_{l1} and L_2, instead of L_1 and L_2. Since C_{l1} is generally smaller than L_1, this could lead to a greatly reduced number of overlappings.

In order to formalise this idea, we show that every disabling essence caused by a rule ρ_1 corresponds to a conflict essence caused by its *minimal effects rule*, whose left-hand side is the deletion object of ρ_1.

Definition 6.8. Given a rule ρ, its *minimal effects rules* $\mathrm{eff}(\rho) = C_{l1} \overset{\overline{l_1}}{\leftarrowtail} B_{l1} \overset{\mathrm{id}_{B_{l1}}}{\rightarrowtail} B_{l1}$ is determined by the initial pushout ① over l_1, as in diagram (6.6)

$$\begin{tikzcd}
C_{l1} \arrow[d, "c_{l1}"'] & B_{l_1} \arrow[l, tail, "\overline{l_1}"'] \arrow[r, tail, "\mathrm{id}_{B_{l1}}"] \arrow[d, "b_{l1}"] \arrow[dl, phantom, "①"] & B_{l_1} \\
L_1 & K_1 \arrow[l, tail, "l_1"] \arrow[r, "r_1"'] & R_1
\end{tikzcd} \tag{6.6}$$

Lemma 6.9. In a category of set-valued functors, let ρ_1 and ρ_2 be rules with some disabling essence $(e_1,e_2) \in \mathbf{Ess}_{\mathrm{dbl}}(\rho_1,\rho_2)$. If $\overline{e_1} : C \to C_{l1}$ is the unique morphism such that $e_1 = c_{l1} \circ \overline{e_1}$, guaranteed to exist by Lemma 5.7, then $(\overline{e_1},e_2) \in \mathbf{Ess}_{\mathrm{cfl}}(\mathrm{eff}(\rho_1),\rho_2)$.

Proof. Since $(e_1,e_2) \in \mathbf{Ess}_{\mathrm{dbl}}(\rho_1,\rho_2)$, there is some conflict $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ with $(e_1,e_2) \cong ess_{\mathrm{dbl}}(t_1,t_2)$. From Lemma 5.7 we get a unique $\overline{e_1}$ with $e_1 = c_{l1} \circ \overline{e_1}$, and such that $ess_{\overline{l_1}}(m_1 \circ c_{l1}, m_2) \cong (\overline{e_1},e_2)$. Then, constructing the pushout $(\overline{m_1},\overline{m_2})$ of $(\overline{e_1},e_2)$, we get a unique morphism f such that diagram (6.7) commutes. We must show that (i) transformation step $\overline{t_1} : \overline{G} \overset{\mathrm{eff}(\rho_1),\overline{m_1}}{\Longrightarrow} \overline{H_1}$ exists, (ii) transformation step $\overline{t_2} : \overline{G} \overset{\rho_2,\overline{m_2}}{\Longrightarrow} \overline{H_2}$ exists and (iii) $ess_{\mathrm{cfl}}(\overline{t_1},\overline{t_2}) \cong (\overline{e_1},e_2)$.

$$\begin{tikzcd}
 & & C \arrow[dl, "\overline{e_1}"'] \arrow[dr, "e_2"] & \\
B_{l1} \arrow[r, "\overline{l_1}"] \arrow[dd, "b_{l1}"'] & C_{l1} \arrow[dd, "c_{l1}"] \arrow[dr, "\overline{m_1}"] & \textcircled{2} & L_2 \arrow[dl, "\overline{m_2}"'] \arrow[ddl, bend left, "m_2"] \\
\textcircled{1} & & \overline{G} \arrow[dd, dotted, "f"] & \\
K_1 \arrow[r, "l_1"] & L_1 \arrow[dr, "m_1"'] & & \\
 & & G &
\end{tikzcd} \qquad (6.7)$$

$$\begin{tikzcd}
B_{l1} \arrow[r, dotted] \arrow[d, tail, "\overline{l_1}"'] \arrow[rr, bend left] & \overline{D_1} \arrow[r, "f"] \arrow[d] & D_1 \arrow[d, "g"] \\
C_{l1} \arrow[r, "\overline{m_1}"] \arrow[ur, phantom, "\textcircled{3}"] & I \arrow[r, "f"] \arrow[ur, phantom, "\textcircled{4}"] & G \\
C \arrow[u, "\overline{e_1}"] \arrow[r, "e_2"] & L_2 \arrow[u, "\overline{m_2}"'] &
\end{tikzcd} \qquad (6.8)$$

(i) Since we have the proto-essence $ess_{\overline{l_1}}(m_1 \circ c_{l1}, m_2) \cong (\overline{e_1},e_2)$, we can apply PO-PB decomposition via proto-essence to diagram (6.8), where the pushout ③ + ④ exists because of step t_1 and square ④ is constructed as a pullback. This yields a pushout ③, which ensures the existence of $\overline{t_1} : \overline{G} \overset{\mathrm{eff}(\rho_1),\overline{m_1}}{\Longrightarrow} \overline{H_1}$.

(iii) Recall from part (i) that squares ③ and ④ are pushouts in diagram (6.8). It follows from proto-essence inheritance that $ess_{\overline{l_1}}(\overline{m_1},\overline{m_2}) \cong ess_{\overline{l_1}}(f \circ \overline{m_1}, f \circ \overline{m_2}) = ess_{\overline{l_1}}(m_1 \circ c_{l1}, m_2)$. Moreover, assuming item (ii) holds, we have the disabling essence $ess_{\mathrm{dbl}}(\overline{t_1},\overline{t_2}) \cong ess_{\overline{l_1}}(\overline{m_1},\overline{m_2})$. It remains to show that this is also the conflict essence.

Since square ② of diagram (6.7) is a pullback, the top element of $\mathbf{Sub}(\overline{m_1},\overline{m_2})$ is $\top \cong (\overline{e_1},e_2)$, by Theorem 2.24. Then, since the disabling essence is the top element of $\mathbf{Sub}(\overline{m_1},\overline{m_2})$, the conflict essence is also the top element, as shown by equation (6.9).

$$ess_{\mathrm{cfl}}(\overline{t_1},\overline{t_2}) \cong \top \sqcup_{\mathbf{Sub}} ess_{\mathrm{dbl}}(\overline{t_2},\overline{t_1}) \cong \top \tag{6.9}$$

(ii) Since the steps $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$ exist, it follows from Lemma 3.16 that both (m_1,l_1) and (m_2,l_2) satisfy the dangling condition. We will show that any potentially dangling element for $(\overline{m_2},l_2)$ is actually part of the disabling essence for (t_1,t_2), and therefore it doesn't violate the dangling condition. Without loss of generality, assume that the morphisms b_{l1}, c_{l1}, l_1 and $\overline{l_1}$ are inclusions, thus $B_{l1} \subseteq K_1 \subseteq L_1$ and $B_{l1} \subseteq C_{l1} \subseteq L_1$. Assume also that $C \subseteq L_1 \times L_2$ is a subset of the Cartesian product, with pair projections c_1, $\overline{c_1}$

and c_2.

Formally, let $x \in \overline{G}^S$ be a *potentially* dangling element for $(\overline{m_2}, l_2)$, because it is incident to a deleted element $y \in \overline{m_2}(L_2^T \setminus K_2^T) \subseteq \overline{G}^T$. That is, $\overline{G}^i(x) = y$ for some $i : S \to T$. Then, since $(\overline{m_1}, \overline{m_2})$ is jointly epic, $x \in \overline{m_1}(C_{l1}^S) \cup \overline{m_2}(L_2^S)$. If $x \in \overline{m_2}(L_2^S)$, it trivially satisfies the dangling condition for $(\overline{m_2}, l_2)$. On the other hand, if $x \in \overline{m_1}(C_{l1}^S)$, it has a preimage $x_1 \in C_{l1}^S$ with $\overline{m_1}(x_1) = x$. We will show that x_1 is part of the disabling essence, that is, $(x_1, x_2) \in C^S$ for some $x_2 \in L_2^S$. Then $x = \overline{m_1}(x_1) = \overline{m_2}(x_2) \in \overline{m_2}(L_2^S)$, so it satisfies the dangling condition for $(\overline{m_2}, l_2)$.

Recall that (m_2, l_2) satisfies the dangling condition. In particular, this condition cannot be violated by the image $f(x) = m_1(x_1) \in G^S$, even though it is incident to the deleted $f(y) \in m_2(L_2^T \setminus K_2^T) \subseteq G^T$. Thus, we must have some deleted $x_2 \in L_2^S \subseteq K_2^S$ with $m_2(x_2) = f(x) = m_1(x_1)$.

Now note that, since $x_1 \in C_{l1}$ is in the context of the initial pushout over l_1, there must be some deleted element $z_1 \in L_1^U \setminus K_1^U$ incident to x_1, by Lemma 3.15. That is, we have $L_1^j(z_1) = x_1$ for some $j : U \to T$. Moreover, the image $m_1(z_1)$ is incident to the deleted element $f(y)$, since $G^{i \circ j}(m_1(z_1)) = m_1(L_1^{i \circ j}(z_1)) = m_1(L_1^i(x_1)) = m_1(y_1) = f(y)$. But then, since (m_2, l_2) satisfies the dangling condition, we must have $m_1(z_1) \in m_2(L_2^U \setminus K_2^U)$ and some preimage $z_2 \in L_2^U \setminus K_2^U$ with $m_2(z_2) = m_1(z_1)$.

Since $m_1(z_1)$ is deleted by t_1 and matched by t_2, it is part of the disabling essence for (t_1, t_2), that is, $(z_1, z_2) \in C^T$. Then we can conclude that x_1 is also part of the disabling essence, since $(x_1, x_2) = C^j(z_1, z_2) \in C^T$. □

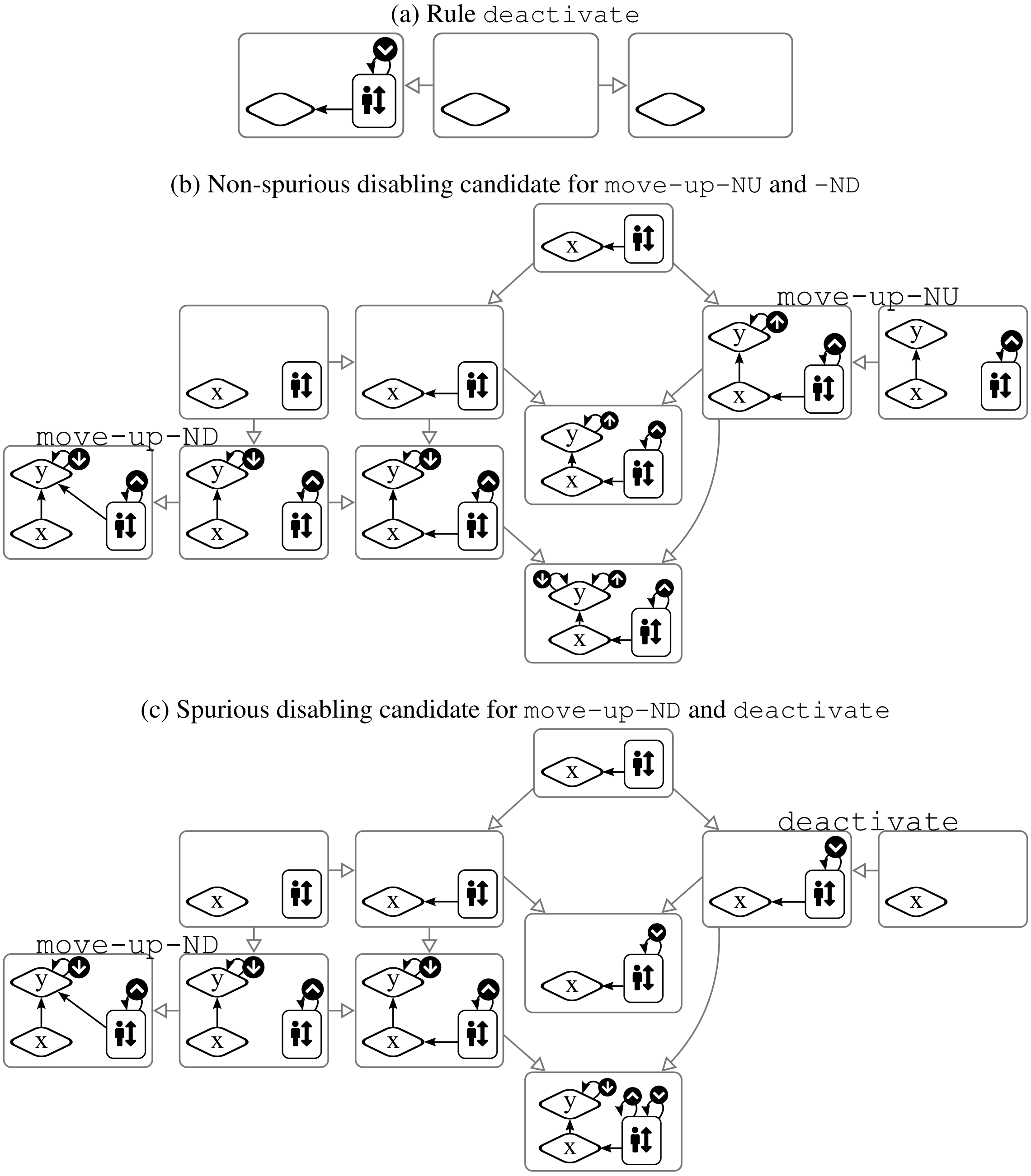
Since disabling essences for (ρ_1, ρ_2) are always conflict essences for $(\text{eff}(\rho_1), \rho_2)$, we can consider the latter *candidates* for the former. Such candidates may be *spurious*, that is, they may not actually be disabling essences for (ρ_1, ρ_2).

Definition 6.10. Given any conflict essence $(\overline{e_1}, e_2) \in \mathbf{Ess}_{\text{cfl}}(\text{eff}(\rho_1), \rho_2)$, let c_{l1} be the deletion object for ρ_1, and let $e_1 = c_{l1} \circ \overline{e_1}$. Then the pair $(e_1, e_2) \in \mathbf{Sub}(L_1, L_2)$ is a *disabling candidate* for rules (ρ_1, ρ_2). The candidate is *spurious* when $(e_1, e_2) \notin \mathit{ess}_{\text{dbl}}(\rho_1, \rho_2)$.

Example 6.11. Consider rules `move-up-NU` and `move-up-ND` from Example 2.6, as well as rule `deactivate` shown in Fig. 6.1a. Figure 6.1b shows a disabling candidate for `move-up-NU` and `-ND`, along with the matches for the corresponding initial conflict between eff(`move-up-NU`) and `move-up-ND`. This candidate is actually a disabling essence, and an appropriate pair of matches for `move-up-NU` and `-ND` is also shown.

On the other hand, Fig. 6.1c shows a spurious disabling candidate, also showing matches for the initial conflict between eff(`move-up-ND`) and `deactivate`. This candidate is spurious because any pair of matches that commutes with it will be non-applicable, since deleting the elevator would leave the `going-up` edge dangling. Again, an example of such matches is also shown.

Figure 6.1: Disabling candidates for rules `move-up-ND` and `deactivate`



(a) Rule `deactivate`

(b) Non-spurious disabling candidate for `move-up-NU` and `-ND`

(c) Spurious disabling candidate for `move-up-ND` and `deactivate`

When checking if a particular disabling candidate is spurious, we must only find a pair of applicable matches that commutes with the candidate. The candidate is then guaranteed to be the disabling essence for the induced transformation steps.

Lemma 6.12. In any adhesive category with proto-essence inheritance and PO-PB decomposition via proto-essence, let $(t_1,t_2) : H_1 \overset{\rho_1,m_1}{\Longleftarrow} G \overset{\rho_2,m_2}{\Longrightarrow} H_2$, and let (e_1,e_2) be a disabling candidate for (ρ_1,ρ_2). If $m_1 \circ e_1 = m_2 \circ e_2$, then $ess_{\mathrm{dbl}}(t_1,t_2) \cong (e_1,e_2)$.

Proof. We first construct an intermediate pair of matches (m'_1,m'_2) by constructing the pushout ① in diagram (6.10) and setting $m'_2 = f \circ \overline{m_2}$. We will (i) relate the proto-essences of $(\overline{m_1},\overline{m_2})$ and (m'_1,m'_2), then (ii) relate the proto-essences of (m'_1,m'_2) and (m_1,m_2), concluding that the disabling essence of (m_1,m_2) is indeed the candidate.

$$\begin{tikzcd}
C_{l1} \arrow[d, "c_{l1}"'] \arrow[dr, "\overline{m_1}"] & C \arrow[l, "\overline{e_1}"'] \arrow[r, "e_2"] & L_2 \arrow[dl, "\overline{m_2}"'] \arrow[ddl, dotted, "m'_2"] \arrow[dddl, bend left, "m_2"] \\
L_1 \arrow[r, phantom, "\textcircled{1}"] \arrow[dr, "m'_1"] \arrow[ddr, bend right, "m_1"'] & \overline{G} \arrow[d, "f"] & \\
& G' \arrow[d, dotted, "h"] & \\
& G &
\end{tikzcd} \tag{6.10}$$

$$\begin{tikzcd}
B_{l1} \arrow[r, "\overline{l_1}"] \arrow[d, "\overline{k_1}"'] \arrow[dr, phantom, "\textcircled{2}"] & C_{l1} \arrow[d, "\overline{m_1}"] \\
\overline{D_1} \arrow[r, "\overline{g}"] \arrow[d, "d'_1"'] \arrow[dr, phantom, "\textcircled{3}"] & \overline{G} \arrow[d, "f"] \\
D_1 \arrow[r, "g'"] & G'
\end{tikzcd} \tag{6.11}$$

(i) We want to show that $ess_{\overline{l_1}}(n_1 \circ c_{l1}, n_2) \cong (\overline{e_1}, e_2)$ by proto-essence inheritance. In order to do so, we will construct a pushout square ③ as in diagram (6.11). The other pushout ② also exists, being part of the initial conflict determined by $(\overline{m_1},\overline{m_2})$. Then, we have $ess_{\overline{l_1}}(m'_1 \circ c_{l1}, m'_2) \cong (\overline{e_1}, e_2)$ by proto-essence inheritance and, by Lemma 5.7, $essl_1(m'_1,m'_2) \cong (e_1,e_2)$.

In order to construct pushout ②, consider diagram (6.12), where the top and front-right faces of the cube are pushouts, the back-right face of the cube is an initial pushout over l_1 and square ④ is an initial pushout over $\overline{m_1}$. From the initial pushout ④, we obtain a unique morphism b^* with $\overline{l_1} \circ b^* = b_{\overline{m1}}$, which we will use to construct a pushout.

$$\begin{tikzcd}
& B_{l1} \arrow[dl, "\overline{k_1}"'] \arrow[dd, "b_{l1}"] \arrow[drr, "\overline{l_1}"] & & & & \\
\overline{D_1} \arrow[drr, "\overline{g_1}"] \arrow[dd, dotted, "\overline{d_1}"'] & & & C_{l1} \arrow[dl, "\overline{m_1}"'] \arrow[dd, "c_{l1}"] \arrow[dr, phantom, "\textcircled{4}"] & & B_{\overline{m1}} \arrow[ll, "b_{\overline{m1}}"'] \arrow[dl, "\overline{\overline{m_1}}"] \arrow[ullll, dotted, bend right, "b^*"'] \\
& K_1 \arrow[dl, dashed, "k'_1"'] \arrow[drr, "l_1"] & \overline{G} \arrow[dd, "f"] & & C_{\overline{m1}} \arrow[ll, "c_{\overline{m1}}"] & \\
D'_1 \arrow[drr, dashed, "g'_1"'] & & & L_1 \arrow[dl, "m'_1"] & & \\
& & G' & & &
\end{tikzcd} \tag{6.12}$$

Now we construct the square ⑤ of diagram (6.13) as the pushout of $(\overline{m_1}, b_{l1}\circ b^*)$. Moreover, since $l_1\circ(b_{l_1}\circ b^*) = c_{l1}\circ b_{\overline{m1}}$, the rectangle ⑤ + ⑥ is also a pushout. Then by decomposition we get a unique morphism g_1 making square ⑥ a pushout. Note that square ⑥ is also the bottom face of the cube in diagram (6.12). We can finally obtain the pushout ③, which is the front-left face of the cube, by pushout composition and decomposition.

$$\begin{tikzcd}
B_{\overline{m1}} \arrow[r, "b_{l1}\circ b^*"] \arrow[d, "\overline{m_1}"'] \arrow[rr, bend left, "c_{l1}\circ b_{\overline{m1}}"] \arrow[dr, phantom, "\textcircled{5}"] & K_1 \arrow[r, "l_1"] \arrow[d, "k_1"] \arrow[dr, phantom, "\textcircled{6}"] & L_1 \arrow[d, "m_1'"] \\
C_{l1} \arrow[r] \arrow[rr, bend right, "f\circ c_{\overline{m1}}"'] & D_1' \arrow[r, dotted, "g_1'"] & G'
\end{tikzcd} \tag{6.13}$$

$$\begin{tikzcd}
K_1 \arrow[r, "k_1'"] \arrow[d, "l_1"] \arrow[rr, bend left, "k_1"] \arrow[dr, phantom, "\textcircled{6}"] & D_1' \arrow[r, dotted, "d_1"] \arrow[d, "g_1'"] \arrow[dr, phantom, "\textcircled{7}"] & D_1 \arrow[d, "g_1"] \\
L_1 \arrow[r, "m_1'"] & G' \arrow[r, "h"] & G
\end{tikzcd} \tag{6.14}$$

(ii) Recall from part (i) that $essl_1(m_1', m_2') \cong (e_1, e_2)$. We will show by proto-essence inheritance that $essl_1(m_1, m_2) \cong essl_1(m_1', m_2')$, and therefore $ess_{\text{dbl}}(m_1, m_2) \cong (e_1, e_2)$. We first obtain a unique mediating morphism $h : G' \to G$ making diagram (6.10) commute. Then note that rectangle ⑥ + ⑦ in diagram (6.14) is a pushout, part of transformation t_1. Moreover, square ⑤ corresponds to the bottom face of diagram (6.12), which was shown to be a pushout in part (i). We obtain a unique d_1 making ⑥ a pushout by decomposition, concluding by proto-essence inheritance that $essl_1(m_1', m_2') \cong (e_1, e_2)$. □

Corollary 6.13. In any adhesive category with proto-essence inheritance and PO-PB decomposition via proto-essence, a disabling candidate (e_1, e_2) for rules (ρ_1, ρ_2) is spurious if and only if there is no pair of transformation steps $(t_1, t_2) : H_1 \overset{\rho_1, m_1}{\Longleftarrow} G \overset{\rho_2, m_2}{\Longrightarrow} H_2$ such that $m_1\circ e_1 = m_2\circ e_2$.

Proof. We will show the contrapositives of both directions. If the candidate (e_1, e_2) is not spurious, then there are steps $(t_1, t_2) : H_1 \overset{\rho_1, m_1}{\Longleftarrow} G \overset{\rho_2, m_2}{\Longrightarrow} H_2$ for which it is the disabling essence, and clearly $m_1\circ e_1 = m_2\circ e_2$. On the other hand, if there are transformations (t_1, t_2) such that $m_1\circ e_1 = m_2\circ e_2$, it follows from Lemma 6.12 that the candidate is the disabling essence of (t_1, t_2) and therefore not spurious. □

The previous considerations bring us to Algorithm 6.2, which enumerates disabling essences for a particular pair of rules.

Remark 6.14. Algorithm 6.2 also assumes the implementation of certain categorical primitives that are called as subroutines. In particular, it calls subroutines for computing initial pushouts and for testing if a pair of morphisms has a pushout complement. Moreover, the subroutine `jointlyEpicSquaresOfMonos`, when given morphisms

$L_1 \xleftarrow{e_1} C \xrightarrow{e_2} L_2$, should produce all jointly epic $L_1 \overset{m_1}{\rightarrowtail} G \overset{m_2}{\leftarrowtail} L_2$ such that $m_1 \circ e_1 = m_2 \circ e_2$, up to isomorphism.

Algorithm 6.2: Enumerate Disabling Essences

```
Input  : Rules $\rho_1 = (l_1, r_1)$ and $\rho_2 = (l_2, r_2)$
Output: Set $E$ of disabling essences
$E := \varnothing$;
$(b_{l1}, c_{l1}, \overline{l_1}) :=$ initialPushout($l_1$);
eff($\rho_1$) := $(\overline{l_1}, \mathrm{id}_{B_{l1}})$;
for $(\overline{e_1}, e_2) \in \mathbf{Ess}_{\mathrm{cfl}}(\mathrm{eff}(\rho_1), \rho_2)$ do        — computed with Algorithm 6.1
    $e_1 := c_{l1} \circ \overline{e_1}$;
    — check if candidate $(e_1, e_2)$ is spurious
    for $(m_1, m_2) \in$ jointlyEpicSquaresOfMonos($e_1, e_2$) do
        if hasPushoutComplement($l_1, m_1$) and
         hasPushoutComplement($l_2, m_2$) then
            — candidate is not spurious, stop searching
            $E := E \cup \{(e_1, e_2)\}$;
            break;
        end
    end
end
```

Once again, the correctness of this algorithm is easy to ascertain when interpreting it in a category of set-valued functors, and assuming correct implementations of the categorical primitives.

Theorem 6.15 (Correctness of Algorithm 6.2)**.** Let Algorithm 6.2 be interpreted in a category of set-valued functors, with correct implementations of the categorical primitives that it calls, as described in Remark 6.14. Let also the input consist of rules ρ_1 and ρ_2 with finite left-hand sides. Then the algorithm terminates producing a set E that contains exactly the disabling essences for (ρ_1, ρ_2), up to isomorphism.

Proof. Termination follows directly from the finite left-hand sides as well as Lemma 6.4 and Theorem 6.5. Correctness follows directly from Lemma 6.9 and Corollary 6.13. □

Regarding the performance of Algorithm 6.2, the nested loops present a possible issue. At first sight, it seems that we repeatedly check all jointly epic pairs of matches for (ρ_1, ρ_2), which grow exponentially with the size of the left-hand sides, once for each disabling candidate. This is, however, not the case: as shown in Lemma 6.12, each pair of matches commutes with only one disabling candidate, which is its disabling essence.

Thus, each jointly epic pair of matches for (ρ_1, ρ_2) will be visited *at most once*. In fact, we might expect that the algorithm never visits most of these matches: only the matches that commute with spurious candidates are guaranteed to be visited. This point will be experimentally explored in Chapter 7.

The other performance-critical part of Algorithm 6.2 is the enumeration of disabling candidates. At this point, Algorithm 6.1 is executed with $(\text{eff}(\rho_1), \rho_2)$ as input. Since the left-hand side of $\text{eff}(\rho_1)$ is generally smaller than in ρ_1, this should often be much faster than computing conflict essences for (ρ_1, ρ_2), as its running time increases exponentially with the sizes of the left-hand sides. This will also be subjected to experiments in Chapter 7.

6.3 Finding Irreducible Essences

Having enumerated the conflict or disabling essences, their irreducible counterparts can be obtained with a post-processing step that checks if each essence is irreducible. A naive approach would be to compute the unions of all pairs of essences, then check if each essence is isomorphic to any such union. However, it is easy to see that the computational cost of this process would be prohibitive.

A much more efficient approach can be devised from the following observation: if an essence is not irreducible, then it is the union of its lower cover. In fact, the converse implication is also true. Thus, we can test if a particular essence is irreducible by finding the lower cover, computing its union and checking if it is isomorphic to the essence.

Remark 6.16. Recall that the poset $\mathbf{Ess}(\rho_1, \rho_2)$ is equipped with the partial order of its superset $\mathbf{Sub}(L_1, L_2)$, restricted to the subset. We will denote the cover relation (Fact 4.5) of $\mathbf{Ess}(\rho_1, \rho_2)$ by $x \lessdot_{\mathbf{Ess}} y$, for $x, y \in \mathbf{Ess}(\rho_1, \rho_2)$. Note that the cover of $\mathbf{Ess}(\rho_1, \rho_2)$ may significantly differ from that of $\mathbf{Sub}(L_1, L_2)$.

Fact 6.17. Let P be a poset and $x, x', y \in P$. If $x \subseteq x' \subseteq x \cup_P y$, then $x \cup_P y = x' \cup_P y$.

Lemma 6.18. Given rules ρ_1 and ρ_2 in an adhesive category, a conflict or disabling essence $x \in \mathbf{Ess}(\rho_1, \rho_2)$ is irreducible if and only if it is not the union of its lower cover, that is, equation (6.15) does not hold.

$$x \cong \bigcup\nolimits_{\mathbf{Sub}} \{u \in \mathbf{Ess}(\rho_1, \rho_2) \mid u \lessdot_{\mathbf{Ess}} x\} \tag{6.15}$$

Proof. [If] We will show the contrapositive, so assume x is not irreducible. Then there are distinct $y, z \in \mathbf{Ess}(\rho_1, \rho_2)$ such that $x \cong y \sqcup_{\mathbf{Sub}} z$ with $x \ncong y$ and $x \ncong z$. From this we can obtain two distinct essences y' and z' in the lower cover of x, such that $x \cong y' \sqcup_{\mathbf{Sub}} z'$.

Take any essence $y' \in \mathbf{Ess}(\rho_1, \rho_2)$ such that $y \subseteq y' \lessdot_{\mathbf{Ess}} x$. It follows from Fact 6.17 that $x \cong y \sqcup_{\mathbf{Sub}} z \cong y' \sqcup_{\mathbf{Sub}} z$. We can analogously pick some z' with $z \subseteq z' \lessdot_{\mathbf{Ess}} x$ and $x \cong y' \sqcup_{\mathbf{Sub}} z'$. Moreover, y' and z' must be distinct. Otherwise $y' \cong z'$ would be an upper bound for y and z, which implies $x \cong y \sqcup_{\mathbf{Sub}} z \subseteq y'$, contradicting $y' \lessdot_{\mathbf{Ess}} x$.

Now let $x' \cong \{u \in \mathbf{Ess}(\rho_1, \rho_2) \mid u \lessdot_{\mathbf{Ess}} x\}$ be the union of the lower cover of x. Given that y' and z' are in the lower cover, we have $x \cong y' \sqcup_{\mathbf{Sub}} z' \subseteq x'$. On the other hand, x is an upper bound for its lower cover and, since x' is the *least* upper bound, we have $x' \subseteq x$. We can conclude that, assuming x is not irreducible, equation (6.15) holds.

[Only if] Assume x is irreducible. We will show that, if equation (6.15) could hold, we would have distinct elements y and z in the lower cover of x such that $x \cong y \sqcup_{\mathbf{Sub}} z$, contradicting the irreducibility of x.

So assuming equation (6.15), first note that the lower cover of x must contain at least two distinct elements. If it were empty we would have $x \cong \bigsqcup_{\mathbf{Sub}} \varnothing \cong \bot$, which contradicts the assumption that x is irreducible. If it were $\{y\}$, then $x \cong \bigsqcup_{\mathbf{Sub}} \{y\} \cong y$, which contradicts the assumption that $y \lessdot_{\mathbf{Ess}} x$.

Then take any distinct essences $y, z \in \mathbf{Ess}(\rho_1, \rho_2)$ in the lower cover of x, that is, with $y \lessdot_{\mathbf{Ess}} x$ and $z \lessdot_{\mathbf{Ess}} x$. We cannot have $y \cong y \sqcup_{\mathbf{Sub}} z$ since it implies $z \subset y \subset x$, contradicting $z \lessdot_{\mathbf{Ess}} x$. Moreover, it holds that $y \sqcup_{\mathbf{Sub}} z \subseteq \bigsqcup_{\mathbf{Sub}} \{u \in \mathbf{Ess}(\rho_1, \rho_2) \mid u \lessdot_{\mathbf{Ess}} x\} \cong x$, since y and z are members of the lower cover. Then we must have $y \sqcup_{\mathbf{Sub}} z \cong x$, otherwise we would have $y \subset y \sqcup_{\mathbf{Sub}} z \subset x$, contradicting the fact that $y \lessdot_{\mathbf{Ess}} x$. We conclude that equation (6.15) implies the existence of distinct $y, z \in \mathbf{Ess}(\rho_1, \rho_2)$ with $x \cong y \sqcup_{\mathbf{Sub}} z$, contradicting irreducibility of x. □

Corollary 6.19. Given rules ρ_1 and ρ_2 in an adhesive category, if a conflict or disabling essence $x \in \mathbf{Ess}(\rho_1, \rho_2)$ has less than two essences in its lower cover, then either $x \cong \bot$ or x is an irreducible essence.

Proof. If the lower cover of x is empty and we have $x \cong \bigsqcup_{\mathbf{Sub}} \varnothing \cong \bot$, then x is the bottom essence. If its lower cover is $\{y\}$, we cannot have $x \cong \bigsqcup_{\mathbf{Sub}} \{y\} \cong y$, since this contradicts $y \lessdot_{\mathbf{Ess}} x$. In this case, x is irreducible. □

Given the previous results, we can propose an algorithm that identifies irreducible essences from a precomputed set of conflict or disabling essences $\mathbf{Ess}(\rho_1, \rho_2)$. It consists of the following three steps. We discuss their worst-case time complexity with respect to the cardinality m of $\mathbf{Ess}(\rho_1, \rho_2)$ and the size n of the left-hand sides, which is an upper bound for the size of the essences.

1. Assemble a directed acyclic graph (DAG) representing the poset $\mathbf{Ess}(\rho_1, \rho_2)$, with essences as nodes and with edges pointing from containing essences to the contained ones. This could have a time complexity of $O(nm^2)$, because comparing two essences can be done in $O(n)$ time. The actual implementation in Verigraph is $O(nm^2 \log n)$ due to the use of balanced binary trees.

2. Compute the transitive reduction of this DAG, removing all edges that correspond to transitive paths and obtaining a representation of the cover relation for $\mathbf{Ess}(\rho_1, \rho_2)$. This can be done with a simple algorithm proposed by Gries et al. (1989), which has a time complexity of $O(m^3)$.

3. Check if each non-empty essence is irreducible by inspecting its lower cover, which is obtained from the outgoing edges on the DAG. If the lower cover has less than two elements, the essence is irreducible. Otherwise, we compute the union of all essences in the lower cover and check if it is isomorphic to the essence. This could have a worst-case time complexity of $O(nm^2)$, since checking isomorphism of two essences can be done in $O(n)$ time. This is much faster than the usual isomorphism check because, in this case, the desired isomorphism must commute with the monomorphisms from the essences into the left-hand sides. This greatly constrains the search space. The actual implementation in Verigraph is $O(nm^2 \log n)$ due to the use of balanced binary trees.

The complete implementation in Verigraph has a worst-case time complexity of $O(nm^2 \log n + m^3)$. Thus, this post-processing step is expected to be much faster than the enumeration of conflict or disabling essences, in most practical cases. This will be explored experimentally in Chapter 7.

7 EXPERIMENTAL EVALUATION

In the previous chapters, we have developed a theory describing the root causes of conflicts, as well as algorithms applying this theory to the problem of conflict detection. We have claimed that the proposed concepts are an improvement over critical pairs, by reporting a smaller number of potential conflicts. In this chapter, we provide experimental evidence supporting this argument. We propose the following research questions.

Q1 Is the number of initial conflicts (and thus conflict essences) in general smaller than that of critical pairs?

Q2 Is the number of irreducible essences in general smaller than all conflict or disabling essences?

We have also made various conjectures in Chapter 6 about the performance of the proposed algorithms. One such conjecture is that the post-processing step that filters irreducible essences consumes, in practice, less time than the enumeration of all conflict or disabling essences. This is formulated on the following research question.

Q3 Is the time spent enumerating conflict or disabling essences larger than than the time spent identifying irreducible essences?

Recall also that Algorithms 6.1 and 6.2 enumerate conflict and disabling essences by checking overlappings of the left-hand sides and, in the case of disablings, a deletion object. We have shown that the running time of these algorithms is at least linear with respect to the number of checked overlappings. Then we have claimed that Algorithm 6.2 checks, in general, only a fraction of all overlappings, since it is only guaranteed to check all overlappings of left-hand sides that commute with spurious disabling candidates. These considerations lead us to the following research questions.

Q4 Is the running time of Algorithms 6.1 and 6.2 determined by the number of overlappings that it checks?

Q5 How large is the number of spurious disabling candidates?

Q6 Is the total number of overlappings checked by Algorithm 6.2, including overlappings of the deletion object, larger than the total number of overlappings of left-hand sides?

Q7 Is the time required to enumerate all disabling essences for a pair of rules, in both directions, generally larger than the time required to enumerate the conflict essences?

7.1 Methods

In order to investigate these research questions, we have selected 11 graph transformation systems from the literature, as well as an extended version of the elevator system presented in Chapter 2. From each transformation system, all pairs of rules were extracted and used as input data for the tests. In practice, transformation systems contain negative application conditions (NACs), and this is the case for all of the analysed systems. Since our techniques do not support NACs, we have removed them for the experiment. It is left for future work to adapt our approach for rules with NACs, and to determine how their presence influences the numbers of critical pairs and initial conflicts.

We have implemented the algorithms proposed in Chapter 6 in Haskell, using the categorical operations provided by the Verigraph system. Additional bookkeeping was introduced recording for each pair of rules the numbers n_{cp} of critical pairs, n_{ce} of initial conflicts/conflict essences, n_{dc} of disabling candidates, n_{de} of disabling essences n_{de}, n_{ie} of irreducible essences, n_{co} of total checked overlappings, n_{col} of checked overlappings of the left-hand sides and n_{to} of total overlappings of the left-hand sides. We also record the running time t_{e} for the enumeration algorithm and t_{f} for the filtering of irreducible essences, in terms of CPU time. The experiment was repeated ten times in an Intel Core i5-3330 processor, running at 3GHz, with 16GiB of RAM. We have checked that the counts are the same in every execution, and taken the average running times. Since Haskell uses lazy evaluation by default, we have taken care to ensure the full evaluation of each disabling and conflict essence.

The graph transformation systems selected for the experiment are described in the following, and summarised in Table 7.1. Most of them were reconstructed in the AGG tool (TAENTZER, 2003), except for fmedit, which was exported from Henshin into the GGX format. This yields 3277 ordered pairs of rules, used for calculating disabling essences, and 1708 unordered pairs of rules, used for calculating conflict essences.

automaton Models the behaviour of non-deterministic automata, and was provided by Lambers et al. (2018) as a motivating example for initial conflicts.

elev Models the behaviour of a single elevator. It was used by Lambers (2009) as a case study for static analysis techniques, including conflict detection.

multi-elev Models the behaviour of multiple elevators in a single building. This is an extended version of the transformation system presented in Chapter 1.

Table 7.1: Overview of transformation systems used as input for the experiment.

Transformation	Number	Size of LHSs	
System	of Rules	min	max
automaton	2	13	20
elev	9	1	10
elev-new	12	7	9
med1	9	4	7
med2	11	5	12
med3	12	5	12
mutex	13	1	7
pacman	6	5	7
server	4	4	6
treeToList	4	3	7
uml-refac	8	2	6
fmedit	49	1	23

med1–3 These three transformation systems model medical guidelines for a therapeutic intervention. They were created by Cota et al. (2017) to validate the guidelines with static analysis, detecting several issues in the original text.

mutex Models a distributed mutual exclusion algorithm. Provided by Varró, Schürr and Varró (2005) as a benchmark for graph transformation tools.

pacman Models the game of pacman. Provided by Ehrig et al. (1997) as an introductory example to graph transformation.

server Models the exchange of messages between clients and servers. Provided by Machado, Ribeiro and Heckel (2015) to motivate the development of second-order graph grammars.

tree-rotation Encodes tree rotations that underlie many implementations of balanced binary trees. Used by Costa et al. (2016) to compare the performance of different tools that enumerate critical pairs.

uml-refac Encodes refactorings of UML class models. Provided by Mens, Taentzer and Runge (2007), who used conflict detection to understand the interactions between these refactorings and propose guidelines for their application.

fmedit Encodes several possible transformations and refactorings of feature models that are implemented in model transformation tools. Adapted from Strüber et al. (2016), who proposed it as a benchmark for the scalability of model transformation tools.

Subtyping and boolean attributes were simulated with additional edge types, since neither is supported by our approach.

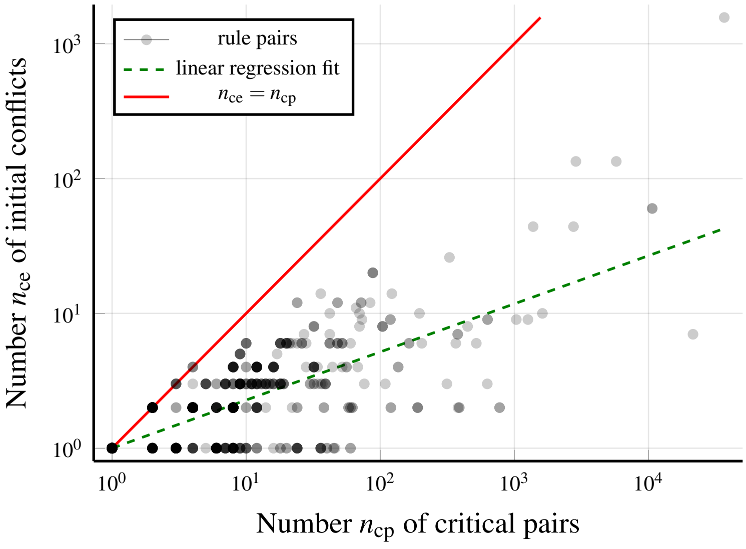
7.2 Results

Of the 1708 unordered pairs of rules, only 693 pairs have at least one potential conflict. The other 1015 were found to be parallel independent. Similarly, only 951 of the 3277 ordered pairs of rules have at least one potential disabling. Each repetition of the test took around 36min, with around 6min spent enumerating and filtering conflict essences, and around 30min spent for disabling essences. Thus, Algorithm 6.1 seems feasible for practical applications, while Algorithm 6.2 seems less feasible. Moreover, both algorithms may not scale for larger case studies.

Regarding question Q1, the experiments have presented evidence that the number n_{ce} of conflict essences, or equivalently of initial conflicts, is generally smaller than the number n_{cp} of critical pairs. In fact, a paired t-test has refuted the null hypothesis that $n_{\mathrm{ce}} \geq n_{\mathrm{cp}}$ with $p < 0.01, N = 693$. This can also be observed in Fig. 7.1, since most points are below the line $n_{\mathrm{ce}} = n_{\mathrm{cp}}$, and none are above.

Moreover, the fraction of critical pairs that are initial seems to decrease with the number of critical pairs. In fact, there is a strong correlation between $\log(n_{\mathrm{ce}})$ and $\log(n_{\mathrm{cp}})$, giving Pearson's correlation coefficient $r = 0.729 \pm 0.036$ with 99% confidence. This can also be observed on the scatter plot presented by Fig. 7.1. We can there-

Figure 7.1: Log-log scatter plot of critical pairs vs. initial conflicts, for each pair of rules, with the fit of linear regression.



fore fit a linear model $\log(n_{\text{ce}}) = \alpha + \beta \log(n_{\text{cp}})$ using ordinary least squares, obtaining $\alpha = 0.02 \pm 0.08$ and $\beta = 0.36 \pm 0.03$ with 99% confidence. From this fit we obtain the following approximation for the ratio $\frac{n_{\text{ce}}}{n_{\text{cp}}}$, which is a monotonically decreasing function of n_{cp}. This decreasing behaviour can also be observed in Fig. 7.2.

$$n_{\text{ce}} \approx e^{\alpha} \cdot n_{\text{cp}}^{\beta} \approx n_{\text{cp}}^{0.36} \tag{7.1}$$

$$\frac{n_{\text{ce}}}{n_{\text{cp}}} \approx n_{\text{cp}}^{-0.64} \tag{7.2}$$

Regarding question Q2, we have evidence that the number of irreducible essences is proportional to the *logarithm* of the total number of essences. In the case of conflict

Figure 7.2: Scatter plot of the fraction of critical pairs that is initial vs. total number of critical pairs in logarithmic scale, for each pair of rules.

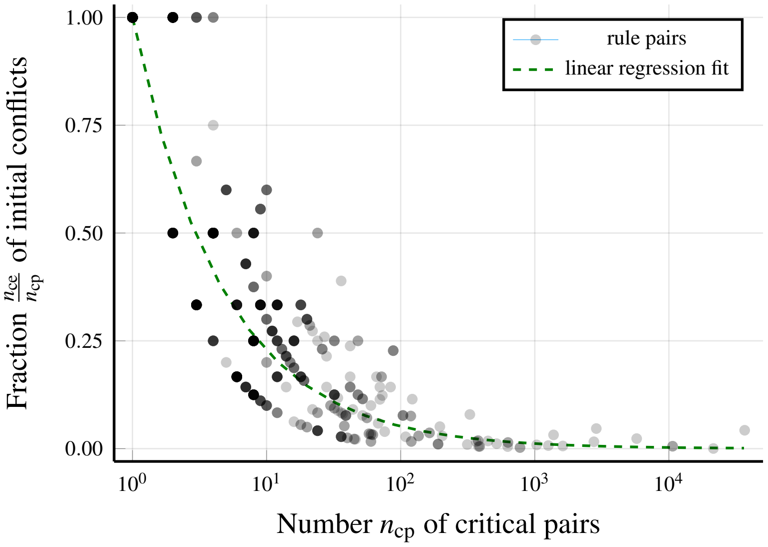


Figure 7.3: Scatter plot of conflict essences in logarithmic scale vs. irreducible conflict essences, for each pair of rules.

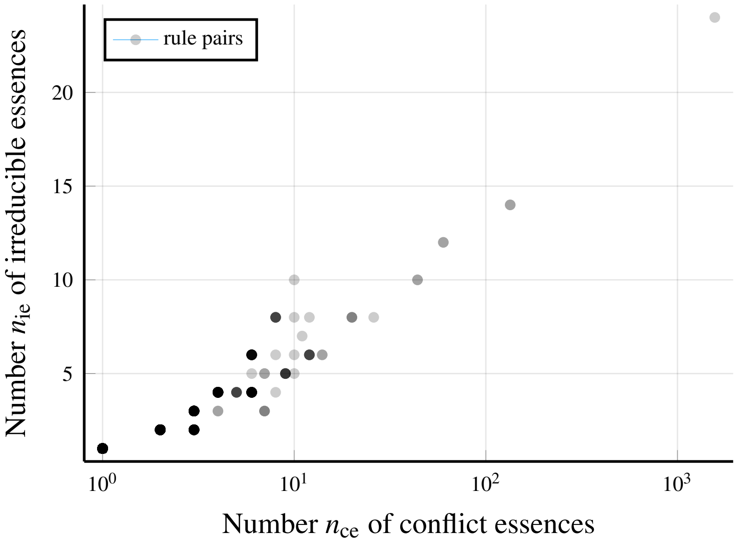
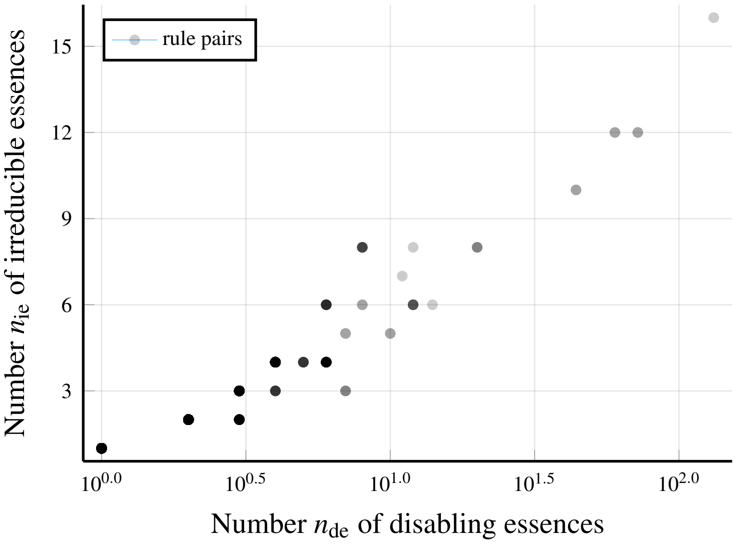


Figure 7.4: Scatter plot of disabling essences in logarithmic scale vs. irreducible disabling essences



essences, with sample size $N = 693$, we have a correlation coefficient $r = 0.94 \pm 0.009$ for the variables n_{ie} and $\log(n_{\text{ce}})$, with 99% confidence. The case of disabling essences is similar: given $N = 951$, we compare n_{ie} and $\log(n_{\text{de}})$ obtaining $r = 0.95 \pm 0.006$. This correlation is also visible on the scatter plots of Figs. 7.3 and 7.4.

An interesting related result is that *all* irreducible essences detected in the experiment had less than two essences in their lower cover. It is left for future work to determine if this holds in all cases. This would simplify the algorithm that finds irreducible essences, since computing the union of the lower cover would no longer be necessary.

We conclude that enumerating conflict or disabling essences seems strictly better than critical pairs, especially for pairs of rules with large left-hand sides, which tend to have more critical pairs. Moreover, when reporting irreducible essences is possible, it provides an advantage by cutting the number of essences logarithmically.

Filtering irreducible essences, however, could have a significant computational cost. Nevertheless, in the context of Q3, Fig. 7.5 shows that the enumeration of conflict essences demanded more time than the subsequent filtering of irreducible essences, in the vast majority of cases. The single outlier is a pair of rules with 1566 conflict essences, of which only 24 are irreducible, taking about 18s to enumerate and almost 3min to filter. Arguably, the time spent filtering all these essences is worth it, given the significant reduction. Moreover, having that many essences seems to be the exception, rather than the rule. In fact, this was the only pair of rules with more than 150 essences, and one of only five pairs with more than 50 essences. We have indeed refuted the null hypothesis that $t_{\text{f}} \geq t_{\text{e}}$ using a paired t-test ($p < 0.01$, $N = 1708$). Unfortunately, we cannot rule out the fact that both processes have the same asymptotic behaviour. In fact, when fitting the model $t_f = e^{\alpha} \cdot t_e^{\beta}$ by ordinary least squares, we obtain $\beta = 0.85 \pm 0.28$ and correlation $r = 0.74 \pm 0.03$, with 99% confidence for our sample size.

For disabling essences, on the other hand, we obtain $\beta = 0.68 \pm 0.06$ for the same model, with the scatter plot shown in Fig. 7.6. This suggests that filtering the irreducible disabling essences is asymptotically faster than enumerating all such essences. In this case, we were also able to refute the null hypothesis that $t_f \geq t_e$ using a paired t-test ($p < 0.01, N = 3277$).

The evidence suggests that the time spent filtering irreducible essences is, in general, less than the time spent enumerating them. Nevertheless, it may still incur a significant computational cost, especially for conflict essences.

When comparing the running times of Algorithms 6.1 and 6.2 to the number of overlappings they have checked, as postulated by Q3, we have evidence of a strong correlation between their logarithms. This correlation can be observed in the scatter plots of Figs. 7.7 and 7.8, along with the fitted linear models. In the case of conflict essences, we have a correlation coefficient $r = 0.98 \pm 0.003$ between $\log(t_e)$ and $\log(n_{co})$. Using ordinary least squares regression, we fit a model $\log(t_e) = \alpha + \beta \log(n_{co})$, obtaining $\alpha = -2.42 \pm 0.51$ and $\beta = 1.13 \pm 0.15$ with 99% confidence. Since we can approximate $t_e \approx e^{\alpha} \cdot n_{co}^{\beta}$, indicating that the running time of Algorithm 6.1 is approximately linear with respect to the number of checked overlappings.

For disabling essences, on the other hand, we have $r = 0.93 \pm 0.009$, as well as the linear model with $\alpha = 17.44 \pm 1.94$ and $\beta = 1.62 \pm 0.91$, for 99% confidence. This

Figure 7.5: Log-log scatter plot with running times of Algorithm 6.1 vs. filtering of irreducible essences, for each pair of rules.

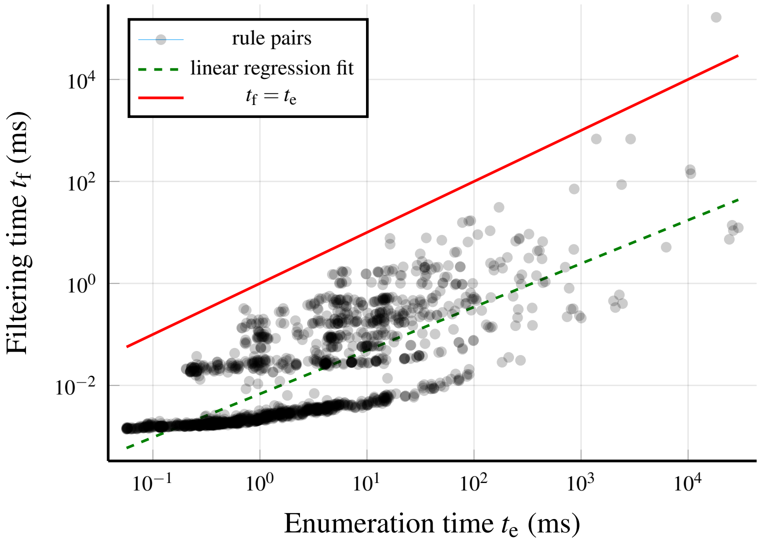


Figure 7.6: Log-log scatter plot with running times of Algorithm 6.2 vs. filtering of irreducible essences, for each pair of rules.

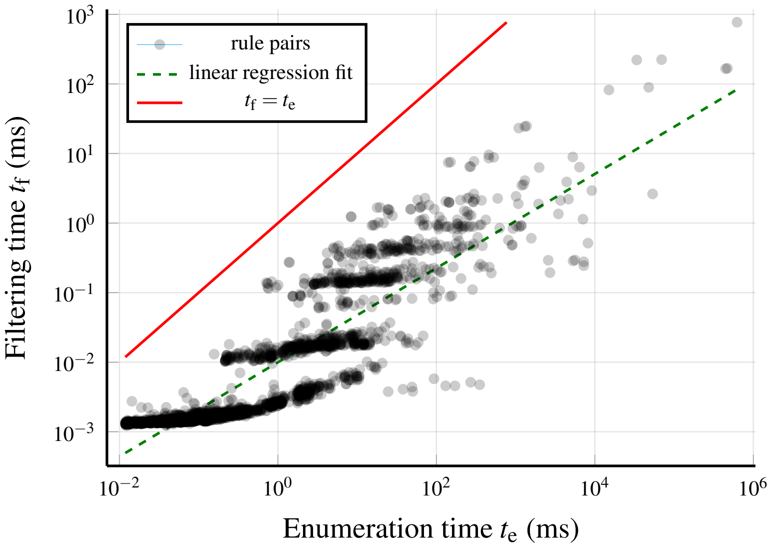
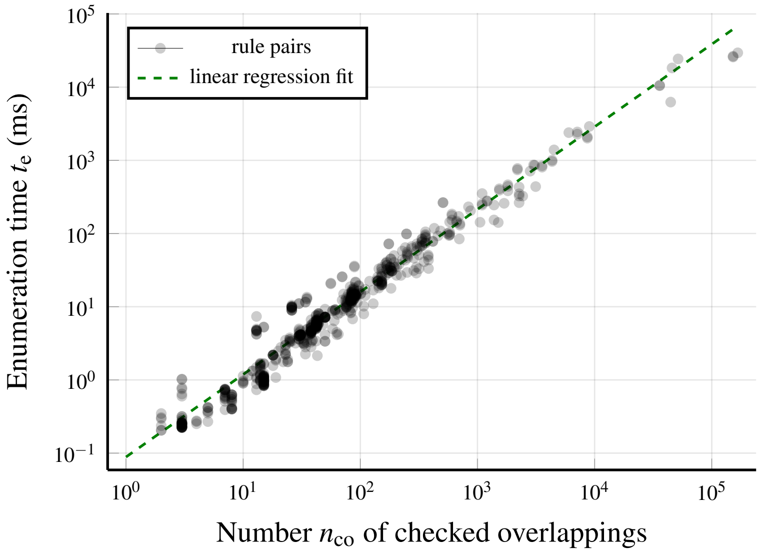


Figure 7.7: Log-log scatter plot with running time of Algorithm 6.1 vs. checked overlappings of the left-hand sides, for each pair of rules.



suggests that the running time of Algorithm 6.1 is between linear and cubic with respect to the number of checked overlappings.

Part of these checked overlappings is due to the number of spurious disabling candidates, each of which forces the algorithm to check a potentially large number of overlappings. Fortunately, evidence indicates that spurious candidates are a rare occurrence: only 1.7% of the candidates produced by the experiment were spurious, which answers question Q5. Moreover, non-spurious candidates could be detected after checking a very small number of overlappings. In fact, in almost 98% pairs of rules, each candidate required checking only one overlapping to show that it is not spurious.

Figure 7.8: Log-log scatter plot with running time of Algorithm 6.2 vs. checked overlappings of the left-hand sides and deletion object, for each pair of rules.

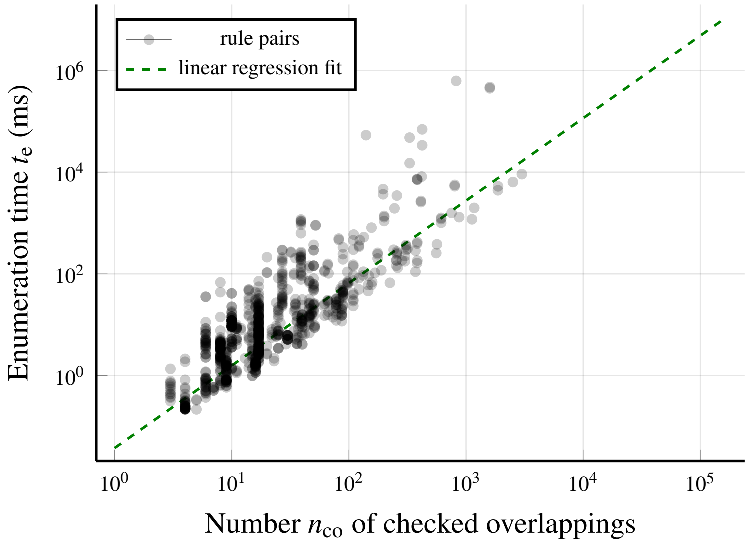
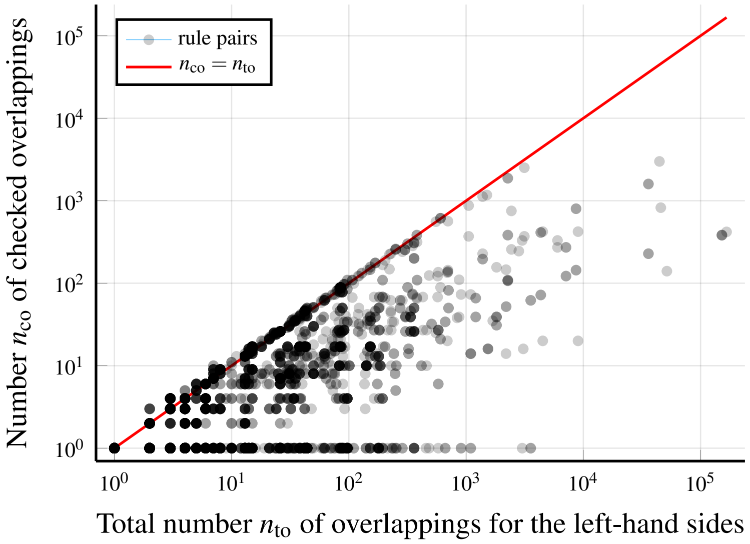


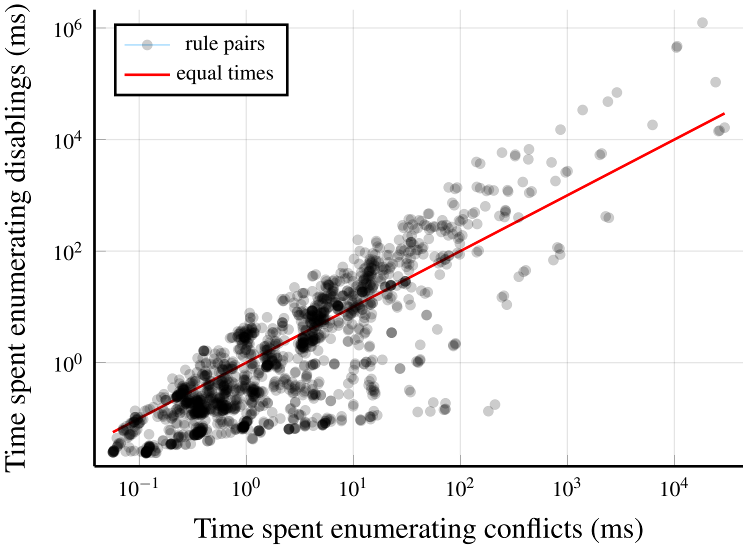
Figure 7.9: Log-log scatter plot comparing the number of overlappings checked by Algorithm 6.2 with the total number of overlappings of the left-hand sides, for each pair of rules.



When investigating Q6, our results are consistent with the previous few statements: the total number of overlappings checked by Algorithm 6.2 is usually less than the total number of overlappings of left-hand sides. This is supported by a paired t-test, which refutes the null hypothesis that $n_{co} \geq n_{to}$ with $p < 0.01$ and $N = 3277$, in spite of the algorithm checking not only overlappings of the left-hand sides, but also overlappings of the deletion object of a rule with the left-hand side of the other. As Fig. 7.9 shows, individual pairs of rules may require checking a larger number of overlappings. In fact, since the t-test compares the *population means*, its result applies to sets of rules that are “large enough”, ensuring that their mean approaches the population mean with high probability. Determining minimal sizes to ensure this is left for future work.

Finally, comparing the running time of Algorithms 6.1 and 6.2 for Q7 led to inconclusive results. Figure 7.10 compares, for each pair of rules, the time required to enumerate conflict essences with the time spent enumerating disabling essences *in both directions*. A t-test could not refute the null hypothesis that these times are equal. On the other hand, each repetition of the complete experiment took around 6min to enumerate and filter all conflict essences, and around 30min for the disabling essences. Investigating whether the enumeration of disabling essences is indeed more costly than conflict essences is left for future work.

Figure 7.10: Log-log scatter plot comparing the time spent enumerating conflict and disabling essences, for each pair of rules.



8 RELATED WORK

8.1 Algebraic Approaches to Graph Transformation

Graph transformation has been developed under various approaches. For an introduction to the most traditional ones, we refer to the "Handbook of Graph Grammars and Computing by Graph Transformation" edited by Rozenberg (1997). In this thesis, we have focused on the Double-Pushout (DPO) approach, one of the *algebraic approaches*, which are based on category theory. The DPO approach was originally developed by Ehrig, Pfender and Schneider (1973) explicitly for the category of graphs and graph morphisms. It was later generalised to arbitrary categories (that satisfy certain assumptions) by Ehrig et al. (1997), Lack and Sobocinski (2005), Ehrig et al. (2004). As discussed in the introduction, this has allowed the main definitions and results to be instantiated for several kinds of "transformed structures", beyond the usual notion of graph. Of particular interest were notions of attributed graph, further discussed in Section 8.2.2, allowing nodes and edges to have attributes of boolean, numeric, textual or even user-defined type. The book by Ehrig et al. (2006) provides a thorough exposition of the DPO approach.

There are actually four variations of the DPO approach, since both the matches and right-hand morphisms $K \to R$ may or may not be monic. Habel, Müller and Plump (1998) investigated the four variations in terms of their expressive power and properties. They have shown that allowing non-monic right-hand morphisms increases expressiveness, allowing the rules to "merge" nodes or edges. However, this comes at a cost: rules and transformations are no longer reversible, an otherwise trivial property, and the theory of parallel and sequential independence becomes more complex. On the other hand, requiring monic matches increases the expressive power of rules, while keeping reversibility and the theory of parallel and sequential independence intact. Most of the subsequent work on conflict and dependency detection requires monic right-hand morphisms, such as Ehrig et al. (2004), Lambers (2009). Much of it also requires monic matches, such as Lambers, Ehrig and Orejas (2008), Born et al. (2017), Lambers et al. (2018). We have chosen to impose both restrictions, remaining in line with previous work in conflict detection without loss of expressive power.

The DPO approach is particularly well-behaved, which has allowed a simple but powerful theory of parallelism and concurrency. However, rules are limited in their expressive power, since their application must have purely local effects. That is, a transformation step cannot modify any parts of the system state outside the match, even if they are incident to some matched element. Moreover, they cannot clone nodes or edges, which is useful for some applications such as refactorings of UML models. In order to overcome these limitations, several approaches have been proposed in the literature.

The first such approach was derived by Raoult (1984) from the insight that rules $\rho = L \leftarrow K \rightarrow R$ can also be seen as partial morphisms $\rho : L \rightarrow R$ with domain K. Thus, we can work in the category of graphs and *partial* graph morphisms, instead of the total morphisms used for DPO. In this category, given a (total) match $m : L \rightarrow G$ and a rule $\rho : L \rightarrow R$, a transformation step can be expressed as the pushout of m and ρ. This is called the Single-Pushout (SPO) approach, differing from DPO because it allows some modification to the *boundary* of the match. In fact, if there is some unmatched edge in the system state incident to a deleted node, the SPO approach will also delete it, while the DPO approach will forbid an application of the rule at this match. The SPO approach was further developed in the category of graphs by Löwe (1991), and generalised to categories of spans (that satisfy certain assumptions) by Montserrat et al. (1997).

The generalisation of SPO to categories of spans highlighted the notion of *final pullback complement* as a construction similar to pushout complements. This was exploited by Corradini et al. (2006) to define the Sesqui-Pushout (SqPO) approach, which is a strict generalisation of DPO with monic matches. It has a similar behaviour to SPO, deleting edges on the boundary of the match. Moreover, when the left-hand morphism $K \rightarrow L$ is non-monic, rules can express the *cloning* of nodes with all incident edges. In order to control which incident edges are cloned along with each node, the AGREE approach was proposed by Corradini et al. (2015) and, later, the PBPO approach was proposed by Corradini et al. (2017). It was the study of parallelism in AGREE by Corradini et al. (2016) that led to the essential condition of parallel independence, which served as the foundation for this thesis.

8.2 Categories for Algebraic Graph Transformation

8.2.1 Variations of Adhesivity

As discussed in Chapter 2, the Double-Pushout (DPO) approach to graph transformation uses pushouts as a basis for “gluing” the rules to the contexts where they are applied. Given the important role that pushouts play in this setting, any category used as a background for DPO transformation should have well-behaved pushouts.

In fact, the generalisation of algebraic graph transformation to multiple categories began by identifying several High-Level Replacement (HLR) properties necessary for graph transformation. Lack and Sobocinski (2005) have shown that these properties are satisfied in all adhesive categories (see Theorem 2.22). Nevertheless, adhesivity is often too strong a requirement, particularly for dealing with attributed graphs. This led to the definition of several relaxed versions of adhesivity that still ensure the necessary HLR properties. Adapting the results of this thesis to such variations is left for future work.

The first relaxed version, proposed already by Lack and Sobocinski (2005), are *quasi-adhesive* categories. These categories satisfy all adhesivity properties, but only with *regular* monomorphisms[1]. When applying the DPO approach to these categories, transformation rules must contain only regular monos. This is a strict generalisation of adhesive categories: all adhesive categories are quasi-adhesive, since regular monos are a particular class of monomorphisms. In fact, adhesive categories are exactly the quasi-adhesive categories where all monos are regular.

Further relaxations were surveyed by Ehrig, Golas and Hermann (2010). Notably, this includes $\mathscr{M}$-adhesive categories, where adhesivity properties hold for a distinguished class $\mathscr{M}$ of monomorphisms, which are those appropriate for transformation rules. In fact, pullbacks and pushouts need only exist along $\mathscr{M}$-morphisms, and such pushouts are only *weak VK squares*, imposing further assumptions on the vertical morphisms of the cube. This is a strict generalisation of quasi-adhesive categories: quasi-adhesive categories are $\mathscr{M}$-adhesive choosing as $\mathscr{M}$ the class of regular monos. This allows DPO transformation to be instantiated for, among others, Petri nets and algebraic specifications.

In the context of labelled and attributed graphs, however, $\mathscr{M}$-adhesivity is often too strong. Indeed, as noted by Habel and Plump (2012), categories of partially labelled graphs are necessary to allow rules that relabel nodes. In this case, nodes have different labels on the left- and right-hand sides, but are left unlabelled in the interface. These categories, however, are not $\mathscr{M}$-adhesive. In fact, pushouts along $\mathscr{M}$-morphisms are only guaranteed to exist if the $\mathscr{M}$-morphism is being pushed out along a monomorphism.

Therefore, Habel and Plump (2012) proposed $\mathscr{M},\mathscr{N}$-adhesive categories by selecting a class $\mathscr{N}$ of morphisms that are appropriate for matches, besides the class $\mathscr{M}$ of monos that are appropriate for rules. Then pushouts are only required to exist when one of the morphisms is in $\mathscr{M}$ and the other in $\mathscr{N}$, and only these are required to be

[1] Regular monos are those that equalise some pair of morphisms. They have important properties of injective functions that may not hold for all monos.

weak VK squares. Deckwerth (2017) further distinguished between classes $\mathscr{L}$ and $\mathscr{R}$ of morphisms appropriate for the left and right sides of a rule, defining $\mathscr{L}, \mathscr{R}, \mathscr{N}$-adhesive categories. These were used to provide static analysis techniques for attributed graph transformation.

8.2.2 Attributed Graphs and Structures

In many applications, the entities of a system have associated attributes, which may be boolean, numeric, textual or of some user-defined type. In this case, system states should be represented by *attributed graphs*, which contain a graphical part that will be modified during execution, an immutable universe of attribute values, and an assignment of attributes to the elements of the graphical part, which may also change with the execution of the system. There are many approaches to attributed graphs. Extending the results of this thesis to handle attributed graphs is left for future work.

The most traditional approach to representing attributed graphs is presented, for example, by Ehrig et al. (2006). In this approach, the universe of attributes is assumed to be an algebra for a particular signature. The values of this algebra are seen as special nodes of the graph, and the attribution is then given by special edges with a regular node as source, and with an attribute node as a target. This defines an underlying category of typed attributed graphs, given a particular type graph and algebraic signature, which is $\mathscr{M}$-adhesive.

A variation on this is the notion of symbolic graphs, proposed by Orejas and Lambers (2010), taking a more syntactic approach. In this case, a fixed algebra is taken for the entire category. Instead of associating values with the elements of the graph, symbolic graphs use *variables*, along with a formula in first-order logic that constrains the values these variables may take. Thus, each symbolic graph represents a (possibly empty) set of attributed graphs. Categories of symbolic graphs were shown by Deckwerth (2017) to be $\mathscr{M}, \mathscr{N}$-adhesive. Deckwerth (2017) also developed conflict detection techniques for symbolic graphs that combine the usual analysis with off-the-shelf SMT solvers.

A third approach are the categories of *attributed structures* proposed by Duval et al. (2014). This approach combines two independent categories, one describing the structures (e.g. graphs) and another describing the attributes, emphasising the separation between the “mutable” graphical part and the immutable attributes. An advantage of this approach is the flexibility in the choice of attributes with a uniform theory. Determining under which conditions the categories of attributed structures are adhesive is an open problem. Thus, conflict detection techniques have not yet been developed for them, to the best of our knowledge.

8.3 Conflict Detection in the Double-Pushout Approach

The theoretical basis for conflict detection, as well as the related problem of dependency detection, is the theory of parallelism and concurrency for graph transformation. In fact, the parallel and sequential combinations of rules and transformations, as surveyed by Corradini et al. (1997), were studied in the early days of algebraic graph transformation. Conflict detection itself, in the form of critical pairs, was originally developed in the field of term rewriting by Knuth and Bendix (1970), then adapted for graph transformation by Plump (1993). These results were among the first to be brought into the framework of adhesive categories by Ehrig et al. (2004).

The theory of critical pairs was later adapted to support Negative Application Conditions by Lambers (2009), and to support Nested Application Conditions by Ehrig et al. (2012). The issue of redundant critical pairs was identified by Lambers, Ehrig and Orejas (2008), who proposed essential critical pairs as a smaller solution to the problem of conflict detection. Essential critical pairs were developed for the category of typed graphs and, to the best of our knowledge, not yet generalised to adhesive categories. Nevertheless, they were a major source of inspiration for this thesis. In fact, we have taken the same approach of finding a characterisation for the root causes of conflicts, then using it to determine an appropriate subset of critical pairs.

During the development of this thesis, Lambers et al. (2018) proposed the notion of initial conflicts as an even better solution for conflict detection than essential critical pairs. Interestingly, it turned out to coincide with our own variation of essential critical pairs. Further work in this direction was the study of root causes for conflicts by Born et al. (2017). They have provided many new concepts with varying levels of granularity, from conflict atoms which are individual elements that cause a conflict when identified, to the conflict reasons originally proposed by Lambers, Ehrig and Orejas (2008). Comparing these various notions to the generalised subobjects and conflict essences proposed in this thesis is left for future work.

8.3.1 Applications of Conflict Detection

Conflict detection and the related technique of dependency detection have found applications in a wide variety of fields, particularly related to Model-Driven approaches to software development. Dependencies are essentially the opposite of disablings: instead of hindering the application of a rule that was previously applicable, one transformation enables the application of a rule that was not previously applicable. Applying the concepts of this thesis to dependency analysis is left for future work, but this should be easily achieved since every dependency of a rule ρ_1 on a rule $\rho_2 = L \leftarrow K \rightarrow R$ corresponds to a conflict between ρ_1 and the inverse rule $\rho_2^{-1} = R \leftarrow K \rightarrow L$.

Conflict and dependency detection have been used as a basis for the certification of various properties in graph transformation systems. In fact, the Ph.D. thesis by Lambers, Ehrig and Orejas (2008) provides certification and refutation procedures based on static analysis, many of which use conflict detection. The use of initial conflicts instead of critical pairs could greatly simplify the certification of certain properties. When certifying that a model has functional behaviour, for example, every potential conflict must be inspected to determine if it is locally confluent. Since searching for local confluence is an expensive analysis, reducing the number of potential conflicts can be highly beneficial.

Dependency detection can also play a role when proving termination of a transformation system. Plump (2018) has shown that this can be done in a modular fashion, by partitioning the transformation rules of the system in way such that rules of different partitions have no potential dependencies. In this case, showing that each partition of the system terminates implies that the complete system also terminates.

There are numerous applications of conflict and dependency detection in the field of Model-Driven Software Engineering. Conflict detection was used by Mens, Taentzer and Runge (2007) to understand interactions between refactorings of UML models and propose guidelines for their application. Conflict and dependency detection was also used by Mehner, Monga and Taentzer (2009) and Whittle et al. (2009) to analyse interactions between aspects in Aspect-Oriented software development. Jayaraman et al. (2007) have proposed the modelling of features in Software Product Lines as transformation rules over UML models, and then using conflict and dependency detection to automatically detect mutually exclusive features. Mens and Straeten (2006) have developed transformations rules for detecting and resolving UML model inconsistencies, then used dependency detection to understand situations where resolving an inconsistency might introduce another, and even detect potential cycles of resolution rules.

Taentzer et al. (2010) proposed conflict detection as a theoretical basis for the versioning of models that are concurrently modified by multiple members of a team. In this setting, conflicts indicate whether these concurrent modifications can be merged, or if different members have made mutually inconsistent changes.

Self-repairing systems, which monitor their own behaviour and adapt to situations where their requirements are not met, where modelled as Dynamic Software Architectures and formalised as graph transformation systems by Bucchiarone et al. (2009). Dependency detection was then used to that the usual execution of the system, external changes and the self-repairing rules interact in desirable ways.

There are also several approaches to validate models of functional requirements with conflict and dependency analysis. Hausmann, Heckel and Taentzer (2002) proposed modelling the pre- and post-conditions of each use-case from an UML use-case model as a transformation rule, then detecting potential conflicts and dependencies between use-cases. Deckwerth (2017) has provided conflict and dependency detection techniques for attributed graphs, applying them to the requirements analysis of enterprise software. Oliveira Jr. et al. (2014) proposed a methodology for constructing graph transformation systems from textual documents and later validate them, applying it to textual use-case models. The same methodology was employed by Cota et al. (2017) to detect several issues in medical guidelines for a therapeutic intervention, producing an improved version of the document.

8.4 Verification Techniques for Graph Transformation

Static analysis techniques for graph transformation systems were thoroughly studied by Lambers (2009), including not only conflict and dependency detection, but also termination, confluence, applicability of certain sequences of rules and the preservation of invariants expressed as graphical constraints. These were further developed for attributed graphs by Deckwerth (2017), who combined traditional techniques for conflict detection with off-the-shelf SMT solvers. The former handle the graphical part of the model, while the latter perform reasoning about the attributes.

In order to ensure that invariants of the systems are preserved by all transformation rules, theorem proving can also be applied. One of the main approaches for theorem proving is based on the language of nested constraints and application conditions proposed by

Habel and Pennemann (2005). It allows the specification of invariants in a graphical way, as well as pre- and post-conditions, with the same expressive power of first-order logic. Habel and Pennemann (2005) also developed a technique for calculating weakest pre-conditions necessary for a rule to ensure a particular nested constraint. It then must be proven that this pre-condition holds in any context where the application conditions are satisfied, which can be done with many approaches. The whole problem can be translated into first-order logic, and then delegated to theorem provers and satisfiability solvers, as proposed by Habel and Pennemann (2005). Alternatively, one may directly use the proof systems for nested constraints proposed by Pennemann (2008b) and Orejas and Lambers (2010), or the satisfiability solver developed by Pennemann (2008a).

Another approach to theorem proving, proposed by Cavalheiro (2010), is based on a representation of graph grammars as relational structures. This allows their embedding in Event-B, a formal methods developed by Abrial (2010). Invariants can the be formulated in first-order logic, with some specification patterns provided by Cavalheiro, Foss and Ribeiro (2012). This technique was extended for negative application conditions by Cavalheiro, Foss and Ribeiro (2017).

Besides static analysis and theorem proving, model checking techniques have been applied to graph transformation. Csertán et al. (2002) proposed a translation of transformation rules over UML models to the Promela language, which is supported by the SPIN model checker. Moreover, the GROOVE model checker was developed by Kastenberg and Rensink (2006), specifically for graph transformation systems.

9 CONCLUSIONS

Graph transformation is a useful framework for the specification, analysis and development of software, particularly within Model-Driven methodologies. It combines an intuitive visual representation with a rich theory and several verification techniques. Static analysis techniques have been applied with particular success, but the technique of conflict detection still faces issues related to scalability. One of these issues is the running time of existing algorithms for enumerating potential conflicts. But a more fundamental problem is the amount of potential conflicts that are detected, and particularly their redundancy.

In this thesis, we have taken a step to improving the techniques of conflict detection by developing a theory that describes the *root causes* of conflicts. We have also provided algorithms for enumerating the potential causes of conflicts, as well as experimental evidence that the concepts we introduce allow a significant reduction in the number of potential conflicts detected, without loss of information.

In the following, we will summarise the main contributions of this thesis, then describe further directions of research that could improve conflict detection for a wider variety of applications.

9.1 Contributions

This thesis has provided a theoretical framework for reasoning about potential conflicts and disablings that a pair of rules may cause. The main theoretical contributions were:

Set-valued functors as generalised graphs. Categories of set-valued functors were shown in Chapter 3 to provide a good context for graph transformation, generalising graphs and graph structures while still providing a notion of element. This allows some proofs to be carried out set-theoretically, and yet apply to a wide variety of graph-like structures.

Conflict and disabling essences as a formal characterisation of root causes. In Chapter 5, we have studied both the essences for concrete pairs of transformations, as well as the sets of potential essences for a pair of rules. These concepts were shown to have important properties: they are directly related to the definition of parallel independence, that is, absence of conflicts; they are empty if and only if there is no conflict or disabling; they are preserved and reflected by extension into larger contexts, in many categories of interest. We have shown that the latter property holds in categories of set-valued functors, and provided a sufficient condition for it to hold in other adhesive categories.

A generalised notion of subobject. In order to apply lattice-theoretical techniques to conflict and disabling essences, we have proposed in Chapter 4 a generalised notion of subobject that applies to pairs of matches or left-hand side objects. We have shown that, in adhesive categories, the pairs of left-hand sides or matches have *lattices* of subobjects. This provides natural ways to compose and decompose these subobjects and, in particular, conflict and disabling essences.

Irreducible essences. We have noted that conflict or disabling essences may be unions of other essences, bringing no additional information for some applications of conflict detection. Using lattice-theoretical techniques, we have identified irreducible essences as a subset of all essences, such that every essence is a union of irreducibles. Irreducible essences are appropriate, for example, when potential conflicts are to be manually inspected.

A categorical construction for initial conflicts. We have shown in Chapter 5 that initial conflicts are closely related to conflict essences. In fact, a categorical construction of initial conflicts from the conflict essences was provided, for many adhesive categories of interest. In particular, we have shown that this construction is correct in categories of set-valued functors. Moreover, we have provided sufficient conditions for this to hold in other adhesive categories.

Besides the theoretical contributions described above, we have provided **algorithms for enumerating conflict, disabling and irreducible essences** in Chapter 6. These algorithms are generic with respect to the underlying category, and can be instantiated for different kinds of transformed objects. We have also discussed their main performance bottleneck, and provided a prototype implementation in the Verigraph system.

Finally, we have performed an **experimental evaluation** of these concepts and algorithms in Chapter 7. When comparing initial conflicts to the previous notion of critical pairs, we have evidence that the former are a significantly smaller subset of the latter, and that the difference in sizes increases with number of critical pairs. Moreover, irreducible essences seem to bring another significant reduction, since their number is highly correlated to the logarithm of the number of essences. The time spent enumerating and filtering conflict essences was appropriate for practical cases. Nevertheless, the scalability of the enumeration algorithms, particularly for disabling essences, remains a concern.

9.2 Future Work

This dissertation provides a significant step towards improving conflict detection for algebraic graph transformation. Nevertheless, the integration of certain concepts would be necessary for a wider applicability of our main results.

Another important extension of the theory is the inclusion of application conditions and graph constraints. Since they were proposed by Habel, Heckel and Taentzer (1996), the use of negative application conditions became quite widespread. They were integrated into the theory of critical pairs by Lambers, Ehrig and Orejas (2006). Moreover, Lambers (2009) has proposed using graph constraints that encode invariants of the system to reduce the number of critical pairs, invalidating those where some invariant is violated. Both of these concepts have been generalised by Habel and Pennemann (2005) as nested constraints and application conditions, which have the same expressive power as first-order logic. The theory of conflict and disabling essences must still be adapted to allow application conditions and graph constraints. In particular, it remains to be seen whether initial conflicts would still be available, since they could violate certain constraints that hold in some of their extensions.

Supporting other kinds of transformed objects would also be important for practical applications, particularly within the object-oriented approach to Model-Driven development. This includes allowing inheritance of types as formalised by Taentzer and Rensink (2005), that is, enriching type graphs with a subtype relation for nodes. Also important is adapting the theory for attributed graphs, which might require considerable effort. Both cases would involve the generalisation of this theory to the variations of adhesivity, which we have presented in Section 8.2.1.

Providing more efficient algorithms for conflict detection is also an important line of research. Born et al. (2017) proposed detecting "building blocks" for conflict reasons and checking which of their combinations are actually causes for conflicts, when studying conflicts at different levels of granularity. Comparing this study to the lattice-theoretical approach taken in this thesis could be a starting point to develop better algorithms.

An important open problem of a more theoretical nature is determining whether proto-essence inheritance and PO-PB decomposition via proto-essences hold in all adhesive categories. This would ensure that the main results of this thesis hold in a wider variety of categories.

The main definitions and results of this thesis could also be adapted for the related problem of dependency detection. This should be relatively straightforward, since dependency of a rule ρ_1 on a rule $\rho_2 = L \leftarrow K \rightarrow R$ corresponds to a conflict between ρ_1 and the inverse rule $\rho_2^{-1} = R \leftarrow K \rightarrow L$.

Finally, it should be possible to adapt this theory for other algebraic approaches of graph transformation that allow cloning. These were reviewed in Section 8.1, and most notably they include Sesqui-Pushout and AGREE.

REFERENCES

ABRIAL, J. **Modeling in Event-B - System and Software Engineering**. Cambridge University Press, 2010. ISBN 978-0-521-89556-9. Available from Internet: <http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521895569>.

ADÁMEK, J.; HERRLICH, H.; STRECKER, G. E. **Abstract and Concrete Categories - The Joy of Cats**. Dover Publications, 2009. ISBN 978-0-486-46934-8. Available from Internet: <http://store.doverpublications.com/0486469344.html>.

ARENDT, T. et al. Henshin: Advanced concepts and tools for in-place EMF model transformations. In: PETRIU, D. C.; ROUQUETTE, N.; HAUGEN, Ø. (Ed.). **Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I**. Springer, 2010. (Lecture Notes in Computer Science, v. 6394), p. 121–135. Available from Internet: <https://doi.org/10.1007/978-3-642-16145-2_9>.

AZZI, G. G. et al. The verigraph system for graph transformation. In: **Graph Transformation, Specifications, and Nets**. Springer, 2018. (LNCS, v. 10800), p. 160–178. Available from Internet: <https://doi.org/10.1007/978-3-319-75396-6_9>.

AZZI, G. G.; CORRADINI, A.; RIBEIRO, L. On the essence and initiality of conflicts. In: LAMBERS, L.; WEBER, J. H. (Ed.). **Graph Transformation - 11th International Conference, ICGT 2018, Held as Part of STAF 2018, Toulouse, France, June 25-26, 2018, Proceedings**. Springer, 2018. (Lecture Notes in Computer Science, v. 10887), p. 99–117. Available from Internet: <https://doi.org/10.1007/978-3-319-92991-0_7>.

BALDAN, P. et al. A lattice-theoretical perspective on adhesive categories. **J. Symb. Comput.**, v. 46, n. 3, p. 222–245, 2011. Available from Internet: <https://doi.org/10.1016/j.jsc.2010.09.006>.

BEZERRA, J. S. et al. **Verites/verigraph 1.1.1: Fix ‘findCospanCommuter‘**. 2017. Available from Internet: <https://doi.org/10.5281/zenodo.579644>.

BORN, K. et al. Granularity of conflicts and dependencies in graph transformation systems. In: **ICGT**. Springer, 2017. (LNCS, v. 10373), p. 125–141. Available from Internet: <https://doi.org/10.1007/978-3-319-61470-0_8>.

BUCCHIARONE, A. et al. Self-repairing systems modeling and verification using AGG. In: **WICSA/ECSA**. IEEE, 2009. p. 181–190. Available from Internet: <https://doi.org/10.1109/WICSA.2009.5290804>.

CAVALHEIRO, S. A. da C. **Relational Approach of Graph Grammars**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, Brazil, 2010.

CAVALHEIRO, S. A. da C.; FOSS, L.; RIBEIRO, L. Specification patterns for properties over reachable states of graph grammars. In: GHEYI, R.; NAUMANN, D. A. (Ed.). **Formal Methods: Foundations and Applications - 15th Brazilian Symposium, SBMF 2012, Natal, Brazil, September 23-28, 2012. Proceedings**. Springer, 2012. (Lecture Notes in Computer Science, v. 7498), p. 83–98. Available from Internet: <https://doi.org/10.1007/978-3-642-33296-8_8>.

CAVALHEIRO, S. A. da C.; FOSS, L.; RIBEIRO, L. Theorem proving graph grammars with attributes and negative application conditions. **Theor. Comput. Sci.**, v. 686, p. 25–77, 2017. Available from Internet: <https://doi.org/10.1016/j.tcs.2017.04.010>.

CORRADINI, A. et al. AGREE - algebraic graph rewriting with controlled embedding. In: **ICGT**. Springer, 2015. (LNCS, v. 9151), p. 35–51. Available from Internet: <https://doi.org/10.1007/978-3-319-21145-9_3>.

CORRADINI, A. et al. The pullback-pushout approach to algebraic graph transformation. In: **ICGT**. Springer, 2017. (LNCS, v. 10373), p. 3–19. Available from Internet: <https://doi.org/10.1007/978-3-319-61470-0_1>.

CORRADINI, A. et al. On the essence of parallel independence for the double-pushout and sesqui-pushout approaches. In: **Graph Transformation, Specifications, and Nets**. Springer, 2018. (LNCS, v. 10800), p. 1–18. Available from Internet: <https://doi.org/10.1007/978-3-319-75396-6_1>.

CORRADINI, A. et al. Parallelism in AGREE transformations. In: **ICGT**. Springer, 2016. (LNCS, v. 9761), p. 37–53. Available from Internet: <https://doi.org/10.1007/978-3-319-40530-8_3>.

CORRADINI, A. et al. Sesqui-pushout rewriting. In: **ICGT**. Springer, 2006. (LNCS, v. 4178), p. 30–45. Available from Internet: <https://doi.org/10.1007/11841883_4>.

CORRADINI, A. et al. Algebraic approaches to graph transformation - part I: basic concepts and double pushout approach. In: **Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations**. [S.l.]: World Scientific, 1997. p. 163–246.

COSTA, A. et al. Verigraph: A system for specification and analysis of graph grammars. In: RIBEIRO, L.; LECOMTE, T. (Ed.). **Formal Methods: Foundations and Applications - 19th Brazilian Symposium, SBMF 2016, Natal, Brazil, November 23-25, 2016, Proceedings**. [s.n.], 2016. (Lecture Notes in Computer Science, v. 10090), p. 78–94. Available from Internet: <https://doi.org/10.1007/978-3-319-49815-7_5>.

COTA, É. F. et al. Using formal methods for content validation of medical procedure documents. **I. J. Medical Informatics**, v. 104, p. 10–25, 2017. Available from Internet: <https://doi.org/10.1016/j.ijmedinf.2017.04.012>.

CSERTÁN, G. et al. VIATRA - visual automated transformations for formal verification and validation of UML models. In: **17th IEEE International Conference on Automated Software Engineering (ASE 2002), 23-27 September 2002, Edinburgh, Scotland, UK**. IEEE Computer Society, 2002. p. 267–270. Available from Internet: <https://doi.org/10.1109/ASE.2002.1115027>.

DAVEY, B. A.; PRIESTLEY, H. A. **Introduction to Lattices and Order (2. ed.)**. [S.l.]: Cambridge University Press, 2002. ISBN 978-0-521-78451-1.

DECKWERTH, F. **Static Verification Techniques for Attributed Graph Transformations**. Thesis (PhD) — Darmstadt University of Technology, Germany, 2017. Available from Internet: <http://tuprints.ulb.tu-darmstadt.de/6150/>.

DUVAL, D. et al. Transformation of attributed structures with cloning. In: GNESI, S.; RENSINK, A. (Ed.). **Fundamental Approaches to Software Engineering - 17th International Conference, FASE 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings**. Springer, 2014. (Lecture Notes in Computer Science, v. 8411), p. 310–324. Available from Internet: <https://doi.org/10.1007/978-3-642-54804-8_22>.

EHRIG, H. Introduction to the algebraic theory of graph grammars (A survey). In: CLAUS, V.; EHRIG, H.; ROZENBERG, G. (Ed.). **Graph-Grammars and Their Application to Computer Science and Biology, International Workshop, Bad Honnef, October 30 - November 3, 1978**. Springer, 1978. (Lecture Notes in Computer Science, v. 73), p. 1–69. Available from Internet: <https://doi.org/10.1007/BFb0025714>.

EHRIG, H. et al. Constraints and application conditions: From graphs to high-level structures. In: **ICGT**. [S.l.]: Springer, 2004. (LNCS, v. 3256), p. 287–303.

EHRIG, H. et al. **Fundamentals of Algebraic Graph Transformation**. Springer, 2006. (Monographs in Theoretical Computer Science. An EATCS Series). Available from Internet: <https://doi.org/10.1007/3-540-31188-2>.

EHRIG, H. et al. $\mathscr{M}$-adhesive transformation systems with nested application conditions. part 2: Embedding, critical pairs and local confluence. **Fundam. Inform.**, v. 118, n. 1-2, p. 35–63, 2012. Available from Internet: <https://doi.org/10.3233/FI-2012-705>.

EHRIG, H.; GOLAS, U.; HERMANN, F. Categorical frameworks for graph transformation and HLR systems based on the DPO approach. **Bulletin of the EATCS**, v. 102, p. 111–121, 2010. Available from Internet: <http://eatcs.org/beatcs/index.php/beatcs/article/view/158>.

EHRIG, H. et al. Adhesive high-level replacement categories and systems. In: **ICGT**. Springer, 2004. (LNCS, v. 3256), p. 144–160. Available from Internet: <https://doi.org/10.1007/978-3-540-30203-2_12>.

EHRIG, H. et al. Algebraic approaches to graph transformation - part II: single pushout approach and comparison with double pushout approach. In: **Handbook of Graph Grammars**. [S.l.]: World Scientific, 1997. p. 247–312.

EHRIG, H.; PFENDER, M.; SCHNEIDER, H. J. Graph-grammars: An algebraic approach. In: **14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973**. IEEE Computer Society, 1973. p. 167–180. Available from Internet: <https://doi.org/10.1109/SWAT.1973.11>.

ERMEL, C. et al. Modeling with plausibility checking: Inspecting favorable and critical signs for consistency between control flow and functional behavior. In: **FASE**. Springer, 2011. (LNCS, v. 6603), p. 156–170. Available from Internet: <https://doi.org/10.1007/978-3-642-19811-3_12>.

GABRIEL, K. et al. Finitary $\mathscr{M}$-adhesive categories. **Mathematical Structures in Computer Science**, v. 24, n. 4, 2014. Available from Internet: <https://doi.org/10.1017/S0960129512000321>.

GRIES, D. et al. An algorithm for transitive reduction of an acyclic graph. **Sci. Comput. Program.**, v. 12, n. 2, p. 151–155, 1989. Available from Internet: <https://doi.org/10.1016/0167-6423(89)90039-7>.

HABEL, A.; HECKEL, R.; TAENTZER, G. Graph grammars with negative application conditions. **Fundam. Inform.**, v. 26, n. 3/4, p. 287–313, 1996. Available from Internet: <https://doi.org/10.3233/FI-1996-263404>.

HABEL, A.; MÜLLER, J.; PLUMP, D. Double-pushout approach with injective matching. In: EHRIG, H. et al. (Ed.). **Theory and Application of Graph Transformations, 6th International Workshop, TAGT'98, Paderborn, Germany, November 16-20, 1998, Selected Papers**. Springer, 1998. (Lecture Notes in Computer Science, v. 1764), p. 103–116. Available from Internet: <https://doi.org/10.1007/978-3-540-46464-8_8>.

HABEL, A.; PENNEMANN, K. Nested constraints and application conditions for high-level structures. In: KREOWSKI, H. et al. (Ed.). **Formal Methods in Software and Systems Modeling, Essays Dedicated to Hartmut Ehrig, on the Occasion of His 60th Birthday**. Springer, 2005. (Lecture Notes in Computer Science, v. 3393), p. 293–308. Available from Internet: <https://doi.org/10.1007/978-3-540-31847-7_17>.

HABEL, A.; PLUMP, D. $\mathscr{M},\mathscr{N}$-adhesive transformation systems. In: EHRIG, H. et al. (Ed.). **Graph Transformations - 6th International Conference, ICGT 2012, Bremen, Germany, September 24-29, 2012. Proceedings**. Springer, 2012. (Lecture Notes in Computer Science, v. 7562), p. 218–233. Available from Internet: <https://doi.org/10.1007/978-3-642-33654-6_15>.

HAUSMANN, J. H.; HECKEL, R.; TAENTZER, G. Detection of conflicting functional requirements in a use case-driven approach: a static analysis technique based on graph transformation. In: **ICSE**. ACM, 2002. p. 105–115. Available from Internet: <http://doi.acm.org/10.1145/581339.581355>.

JAYARAMAN, P. K. et al. Model composition in product lines and feature interaction detection using critical pair analysis. In: **MoDELS**. Springer, 2007. (LNCS, v. 4735), p. 151–165. Available from Internet: <https://doi.org/10.1007/978-3-540-75209-7_11>.

JOHNSTONE, P. T. **Sketches of an elephant: A topos theory compendium**. [S.l.]: Oxford University Press, 2002.

KASTENBERG, H.; RENSINK, A. Model checking dynamic states in GROOVE. In: VALMARI, A. (Ed.). **Model Checking Software, 13th International SPIN Workshop, Vienna, Austria, March 30 - April 1, 2006, Proceedings**. Springer, 2006. (Lecture Notes in Computer Science, v. 3925), p. 299–305. Available from Internet: <https://doi.org/10.1007/11691617_19>.

KLAR, F. et al. Extended triple graph grammars with efficient and compatible graph translators. In: ENGELS, G. et al. (Ed.). **Graph Transformations and Model-Driven Engineering - Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday**. Springer, 2010. (Lecture Notes in Computer Science, v. 5765), p. 141–174. Available from Internet: <https://doi.org/10.1007/978-3-642-17322-6_8>.

KNUTH, D. E.; BENDIX, P. B. Simple word problems in universal algebras. In: **Computational Problems in Abstract Algebra**. Pergamon, 1970. p. 263–297. ISBN 978-0-08-012975-4. Available from Internet: <http://www.sciencedirect.com/science/article/pii/B978008012975450028X>.

LACK, S.; SOBOCINSKI, P. Adhesive and quasiadhesive categories. **ITA**, v. 39, n. 3, p. 511–545, 2005. Available from Internet: <https://doi.org/10.1051/ita:2005028>.

LAMBERS, L. **Certifying rule-based models using graph transformation**. Thesis (PhD) — Berlin Institute of Technology, 2009. Available from Internet: <http://opus.kobv.de/tuberlin/volltexte/2010/2522/>.

LAMBERS, L. et al. Initial conflicts and dependencies: Critical pairs revisited. In: **Graph Transformation, Specifications, and Nets**. Springer, 2018. (LNCS, v. 10800), p. 105–123. Available from Internet: <https://doi.org/10.1007/978-3-319-75396-6_6>.

LAMBERS, L.; EHRIG, H.; OREJAS, F. Conflict detection for graph transformation with negative application conditions. In: CORRADINI, A. et al. (Ed.). **Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17-23, 2006, Proceedings**. Springer, 2006. (Lecture Notes in Computer Science, v. 4178), p. 61–76. Available from Internet: <https://doi.org/10.1007/11841883_6>.

LAMBERS, L.; EHRIG, H.; OREJAS, F. Efficient conflict detection in graph transformation systems by essential critical pairs. **ENTCS**, v. 211, p. 17–26, 2008. Available from Internet: <https://doi.org/10.1016/j.entcs.2008.04.026>.

LÖWE, M. **Extended algebraic graph transformation**. Thesis (PhD) — Technical University of Berlin, Germany, 1991. Available from Internet: <http://d-nb.info/910935696>.

MACHADO, R.; RIBEIRO, L.; HECKEL, R. Rule-based transformation of graph rewriting rules: Towards higher-order graph grammars. **Theor. Comput. Sci.**, v. 594, p. 1–23, 2015. Available from Internet: <https://doi.org/10.1016/j.tcs.2015.01.034>.

MACLANE, S.; MOERDIJK, I. **Sheaves in geometry and logic: A first introduction to topos theory**. [S.l.]: Springer, 2012.

MEHNER, K.; MONGA, M.; TAENTZER, G. Analysis of aspect-oriented model weaving. **Trans. Aspect-Oriented Software Development**, v. 5, p. 235–263, 2009. Available from Internet: <https://doi.org/10.1007/978-3-642-02059-9_7>.

MENS, T.; STRAETEN, R. V. D. Incremental resolution of model inconsistencies. In: **WADT**. Springer, 2006. (LNCS, v. 4409), p. 111–126. Available from Internet: <https://doi.org/10.1007/978-3-540-71998-4_7>.

MENS, T.; TAENTZER, G.; RUNGE, O. Analysing refactoring dependencies using graph transformation. **Software and System Modeling**, v. 6, n. 3, p. 269–285, 2007. Available from Internet: <https://doi.org/10.1007/s10270-006-0044-6>.

MONTSERRAT, M. et al. **Single-Pushout Rewriting in Categories of Spans I: The General Setting**. [S.l.], 1997.

OBJECT MANAGEMENT GROUP. Unified modelling language version 2.5.1. 2017. Available from Internet: <http://www.omg.org/spec/UML/2.5.1>.

OLIVEIRA JR., M. et al. Use case analysis based on formal methods: An empirical study. In: **WADT**. Springer, 2014. (LNCS, v. 9463), p. 110–130. Available from Internet: <https://doi.org/10.1007/978-3-319-28114-8_7>.

OREJAS, F.; LAMBERS, L. Symbolic attributed graphs for attributed graph transformation. **ECEASST**, v. 30, 2010.

PENNEMANN, K. An algorithm for approximating the satisfiability problem of high-level conditions. **Electr. Notes Theor. Comput. Sci.**, v. 213, n. 1, p. 75–94, 2008. Available from Internet: <https://doi.org/10.1016/j.entcs.2008.04.075>.

PENNEMANN, K. Resolution-like theorem proving for high-level conditions. In: EHRIG, H. et al. (Ed.). **Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings**. Springer, 2008. (Lecture Notes in Computer Science, v. 5214), p. 289–304. Available from Internet: <https://doi.org/10.1007/978-3-540-87405-8_20>.

PLUMP, D. **Evaluation of functional expressions by hypergraph rewriting**. Thesis (PhD) — University of Bremen, Germany, 1993. Available from Internet: <http://d-nb.info/940423774>.

PLUMP, D. Modular termination of graph transformation. In: HECKEL, R.; TAENTZER, G. (Ed.). **Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig**. Springer, 2018. (Lecture Notes in Computer Science, v. 10800), p. 231–244. Available from Internet: <https://doi.org/10.1007/978-3-319-75396-6_13>.

RAOULT, J. On graph rewritings. **Theor. Comput. Sci.**, v. 32, p. 1–24, 1984. Available from Internet: <https://doi.org/10.1016/0304-3975(84)90021-5>.

RIEHL, E. **Category Theory in Context**. [S.l.]: Dover Publications, 2017. ISBN 9780486809038.

ROZENBERG, G. (Ed.). **Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations**. [S.l.]: World Scientific, 1997. ISBN 9810228848.

SCHMIDT, D. C. Guest editor's introduction: Model-driven engineering. **IEEE Computer**, v. 39, n. 2, p. 25–31, 2006. Available from Internet: <https://doi.org/10.1109/MC.2006.58>.

STRÜBER, D. et al. Scalability of model transformations: Position paper and benchmark set. In: KOLOVOS, D. S. et al. (Ed.). **Proceedings of the 4rd Workshop on Scalable Model Driven Engineering part of the Software Technologies: Applications and Foundations (STAF 2016) federation of conferences, Vienna, Austria, July 8, 2016.** CEUR-WS.org, 2016. (CEUR Workshop Proceedings, v. 1652), p. 21–30. Available from Internet: <http://ceur-ws.org/Vol-1652/paper3.pdf>.

TAENTZER, G. AGG: A graph transformation environment for modeling and validation of software. In: **AGTIVE**. Springer, 2003. (LNCS, v. 3062), p. 446–453. Available from Internet: <https://doi.org/10.1007/978-3-540-25959-6_35>.

TAENTZER, G. et al. Conflict detection for model versioning based on graph modifications. In: **ICGT**. Springer, 2010. (LNCS, v. 6372), p. 171–186. Available from Internet: <https://doi.org/10.1007/978-3-642-15928-2_12>.

TAENTZER, G.; RENSINK, A. Ensuring structural constraints in graph-based models with type inheritance. In: CERIOLI, M. (Ed.). **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**. Springer, 2005. (Lecture Notes in Computer Science, v. 3442), p. 64–79. Available from Internet: <https://doi.org/10.1007/978-3-540-31984-9_6>.

VARRÓ, G.; SCHÜRR, A.; VARRÓ, D. Benchmarking for graph transformation. In: **2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005), 21-24 September 2005, Dallas, TX, USA**. IEEE Computer Society, 2005. p. 79–88. Available from Internet: <https://doi.org/10.1109/VLHCC.2005.23>.

WHITTLE, J. et al. MATA: A unified approach for composing UML aspect models based on graph transformation. **Trans. Aspect-Oriented Software Development**, v. 6, p. 191–237, 2009. Available from Internet: <https://doi.org/10.1007/978-3-642-03764-1_6>.