

A CATEGORY-THEORETIC APPROACH TO REPRESENTATION AND  
ANALYSIS OF INCONSISTENCY IN GRAPH-BASED VIEWPOINTS

by

Mehrdad Sabetzadeh

A thesis submitted in conformity with the requirements  
for the degree of Master of Science  
Graduate Department of Computer Science  
University of Toronto

Copyright © 2003 by Mehrdad Sabetzadeh

# Abstract

A Category-Theoretic Approach to Representation and Analysis of Inconsistency in  
Graph-Based Viewpoints

Mehrdad Sabetzadeh

Master of Science

Graduate Department of Computer Science

University of Toronto

2003

Eliciting the requirements for a proposed system typically involves different stakeholders with different expertise, responsibilities, and perspectives. This may result in inconsistencies between the descriptions provided by stakeholders. Viewpoints-based approaches have been proposed as a way to manage incomplete and inconsistent models gathered from multiple sources. In this thesis, we propose a category-theoretic framework for the analysis of fuzzy viewpoints. Informally, a fuzzy viewpoint is a graph in which the elements of a lattice are used to specify the amount of knowledge available about the details of nodes and edges. By defining an appropriate notion of morphism between fuzzy viewpoints, we construct categories of fuzzy viewpoints and prove that these categories are (finitely) cocomplete. We then show how colimits can be employed to merge the viewpoints and detect the inconsistencies that arise independent of any particular choice of viewpoint semantics. Taking advantage of the same category-theoretic techniques used in defining fuzzy viewpoints, we will also introduce a more general graph-based formalism that may find applications in other contexts.

TO MY MOTHER AND FATHER WITH LOVE AND GRATITUDE.

# Acknowledgements

First of all, I wish to thank my supervisor Steve Easterbrook for his guidance, support, and patience. I benefited from his advice throughout the entire course of this research, particularly so when exploring new ideas. His positive outlook and confidence in my work has always been a tremendous source of inspiration to me.

I wish to thank Wolfram Kahl of the Department of Computing and Software at McMaster University for his invaluable help and constructive comments at different stages of my research. I am also indebted to him for taking up the responsibility of being the second reviewer of this thesis.

My thanks go to Vassos Hadzilacos, John Mylopoulos, David Penny, and Toniann Pitassi for their encouragement and intellectual support. Special thanks must also go to all those whom I have not yet met in person but have corresponded with through email regarding my research. Most notably, I would like to express my deep gratitude to Andrzej Tarlecki of the Faculty of Mathematics, Informatics and Mechanics at Warsaw University for the time he took in giving detailed answers to my questions. His hints later proved to be crucial for the development of this thesis.

Thanks are due to all those who have made my graduate career at the University of Toronto easier and more enjoyable. Particularly, I would like to thank all my friends in Toronto for their abundant help and support. I am also grateful to the technical and administration staff members of the Department of Computer Science for their kind assistance.

Finally, I cannot thank enough my beloved wife Shiva for her understanding and affection as well as all the help she has provided me with as a fellow colleague.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Preliminaries</b>	<b>6</b>
2.1	Many-Sorted Sets and Algebras . . . . .	6
2.2	Graphs and Graph Homomorphisms . . . . .	8
2.3	Lattices . . . . .	10
<b>3</b>	<b>Category Theory</b>	<b>13</b>
3.1	Categories . . . . .	14
3.2	Functors . . . . .	15
3.3	Diagrams . . . . .	16
3.4	Some Basic Category Theoretic Definitions . . . . .	18
3.5	Colimits . . . . .	21
3.6	Comma Categories . . . . .	25
<b>4</b>	<b>Categories of Fuzzy Sets</b>	<b>29</b>
4.1	Fuzzy Sets and Fuzzy Set Morphisms . . . . .	29
4.2	Cocompleteness Results for Fuzzy Set Categories . . . . .	31
4.3	Fuzzy Powersets . . . . .	33
<b>5</b>	<b>Fuzzy Viewpoints</b>	<b>35</b>
5.1	<b>ℱView</b> Categories . . . . .	36

5.2	Viewpoint Integration and Characterization of Inconsistency . . . . .	39
5.3	Case-Study . . . . .	42
<b>6</b>	<b>⋈Graph Categories</b>	<b>51</b>
<b>7</b>	<b>Conclusions</b>	<b>55</b>
	<b>Bibliography</b>	<b>57</b>
<b>A</b>	<b>Proofs for Chapter 3</b>	<b>62</b>

# List of Figures

2.1	Examples of graphs and graph homomorphisms . . . . .	9
2.2	Examples of Hasse diagrams . . . . .	12
3.1	Pushout examples in <b>Set</b> . . . . .	21
3.2	Pushout example in <b>Graph</b> . . . . .	28
4.1	Example of fuzzy sets . . . . .	30
4.2	Pushout examples in <b>Fuzz</b> ( $\mathbf{A}_4$ ) . . . . .	32
4.3	Powerset lattice example . . . . .	34
5.1	Example of fuzzy viewpoints . . . . .	37
5.2	The lattice $\mathbf{A}_{10}$ . . . . .	40
5.3	Truth ordering lattices . . . . .	41
5.4	Camera’s early reference model . . . . .	43
5.5	Bob’s viewpoint . . . . .	45
5.6	Mary’s viewpoint . . . . .	46
5.7	State interconnections . . . . .	47
5.8	Interconnecting the viewpoints . . . . .	47
5.9	Merging the viewpoints . . . . .	48
6.1	Example of $\mathfrak{F}\mathbf{Graph}(\mathcal{L}, \mathbf{1})$ objects and morphisms . . . . .	53
6.2	Pushout computation in $\mathfrak{F}\mathbf{Graph}(\mathcal{L}, \mathbf{1})$ . . . . .	53
6.3	Pushout computation in $\mathfrak{F}\mathbf{Graph}(2^{\{p,q,r\}}, \mathbf{A}_4)$ . . . . .	54

# Chapter 1

## Introduction

Gathering the requirements for a proposed system is a critical activity that involves different stakeholders with different perspectives, expertise, responsibilities, goals, and terminologies. This may bring about various inconsistencies between the descriptions provided by the stakeholders. Viewpoints-based approaches have been proposed as a way to manage incomplete and inconsistent pieces of information gathered from multiple sources. Through separating the descriptions provided by different stakeholders, these approaches make it possible to capture the expectations and needs of all involved stakeholders and also facilitate the identification, management, and resolution of inconsistencies between the descriptions as early as possible.

A viewpoint is regarded as a projection of a system and its domain from a particular angle. More precisely, a viewpoint represents the context in which a role is performed [Eas93]. Viewpoints may be employed to specify different features of a system, describe different perspectives on a single functionality, or model individual processes that need to be composed in parallel [EC01]. The benefits of using viewpoints for gathering requirements were first made explicit in the CORE method [Mul79]. Since then, various viewpoints-based approaches [FKN<sup>+</sup>92, DvF93, KS96] have emerged.

In [DS96], a conceptual framework has been developed for comparing viewpoints-

based approaches. This conceptual framework, which is based on the analysis of the state-of-the-art research in the area, enlists six major activities as the backbone constituents of every viewpoints-based process model:

- ***Viewpoint identification***: identifying the relevant viewpoints and acquiring the contents of each them.
- ***Viewpoint delineation***: delineating viewpoints using appropriate techniques and schemes.
- ***Viewpoint analysis***: analyzing each viewpoint to identify any intra-viewpoint inconsistencies and to determine the completeness of each viewpoint.
- ***Viewpoint comparison***: comparing all viewpoints to identify any inter-viewpoint inconsistencies and conflicts.
- ***Inconsistency management***: using inconsistency handling and conflict resolution strategies to manage conflicts and inconsistencies.
- ***Viewpoint integration***: combination of all viewpoints into an integrated viewpoint if it is considered desirable and feasible to do so.

In most viewpoints-based approaches, a distinction is made between the syntax and the semantics of viewpoints [DS96]: the syntax is concerned with the rules for expressing viewpoints while the semantics are concerned with the meaning and the interpretation of viewpoints. This distinction naturally separates the concerns in the above-mentioned viewpoint development activities into two different levels: *syntactic* and *semantic*. The aim of this thesis is to provide a unified framework that supports all viewpoint development activities at a syntactic level.

The diversity of viewpoints-based approaches makes it too difficult, if not impossible, to provide a framework encompassing the syntactic aspects of all existing viewpoint representations. For this reason, we are compelled to make some assumptions about the

general structure of viewpoints. Since graph-based formalisms have proven very successful in modeling requirements, we are going to orient our discussion around *graph-based viewpoint representations*, i.e. the viewpoint representations whose underlying syntactic structure is based on graphs.

The most challenging issue to be addressed in our framework concerns finding a proper definition for inconsistency at a syntactic level. Due to the incapability of the classical viewpoint approach to model incompleteness and inconsistency explicitly, the existing inconsistency analysis approaches [FGH<sup>+</sup>94, EN96, NCEF02] require the translation of the entire structure of viewpoints into an intermediate formalism which is typically based on a rich meta-language such as first order logic. This usually complicates the exploration of structural relationships between viewpoints and blurs the distinction between the syntactic and the semantic aspects of viewpoint representations, thus making it too difficult to give a semantics-independent characterization of inconsistency based on structural mappings between viewpoints.

It turns out that augmenting the syntactic structure of viewpoints with a means for describing incompleteness and inconsistency can provide a basis for distinguishing between the inconsistencies that arise independent of any particular choice of semantics for viewpoints, and the inconsistencies that materialize only when certain semantics are in place.

This thesis introduces a category-theoretic formalism for the representation of a family of graph-based viewpoints, hereafter called *fuzzy viewpoints*, that are capable of modeling incompleteness and inconsistency explicitly. Informally, a fuzzy viewpoint is a graph in which the details of nodes and edges are annotated with the elements of a lattice to specify the *amount of knowledge* available about them. By defining an appropriate notion of morphism between fuzzy viewpoints, we construct fuzzy viewpoint categories and prove that they are (finitely) cocomplete. We then show how merging a set of interconnected viewpoints can be done by computing the colimiting viewpoint in an appropriate fuzzy

viewpoint category. Colimits will also be used as a basis for defining a notion of *syntactic inconsistency* between a set of interconnected viewpoints.

Our proposed framework for modeling incompleteness and inconsistency is very general and should apply to any of the large number of graph-based notations commonly used in Software Engineering. In this thesis, however, our focus will be on a fairly simple kind of fuzzy viewpoints inspired by  $\chi$ views [EC01]. We will use state-machine-like models to show how nodes and edges in graphical notations can be decorated with the additional structures required for modeling incompleteness and inconsistency. An application of our framework will be illustrated through a case-study. Taking advantage of the same category-theoretic techniques used in defining fuzzy viewpoints, we will also introduce a more general graph-based formalism that may find applications in other contexts.

## Related Work

The underlying ideas of this work have, to a great extent, been influenced by the inconsistency modeling techniques first explained in [EC01]. Our mathematical machinery builds upon the nice category-theoretic properties of fuzzy sets noted in [Gog68, Gog74]. The use of colimits as an abstract mechanism for putting structures together has been known for quite some time in the algebraic specification community (cf. [Gog91] for references). In [Hec98, HEET99], colimits have been used for merging *consistent* viewpoints. In that approach, viewpoints are described by open graph transformation systems and colimits are employed to integrate them. Our proposed framework is, as far as we know, the first use of category theory for merging *inconsistent* viewpoints.

Fuzzy viewpoints bear some similarity to fuzzy graphs [MN00]. What distinguishes our work from the body of work done on fuzzy graphs in other computing disciplines is our emphasis on algebraic structural relationships rather than graph-theoretic analysis techniques.

## Organization of the Thesis

The remainder of the thesis is organized as follows: Chapter 2 covers some preliminary notions including sets, algebras, graphs, and lattices. Chapter 3 explains the categorical concepts referred to throughout the thesis. Chapter 4 outlines the definitions and lemmas on fuzzy set categories needed in the thesis. Chapter 5 introduces fuzzy viewpoint categories and proposes a definition for syntactic inconsistency based on the structural mappings between viewpoints. The chapter also includes a case-study that illustrates how fuzzy viewpoints can be used as a requirements elicitation tool in reactive systems. Chapter 6 develops a graph-based formalism that generalizes fuzzy viewpoints. This chapter is independent of Chapter 5 and can be read directly after Chapter 4. Finally, Chapter 7 presents our conclusions and future work.

## Guide to the Reader

The end of each lemma, theorem, and corollary is marked by  $\diamond$ ; the end of each definition, remark, note, and introduced notation is marked by  $\square$ ; the end of each proof is marked by  $\blacksquare$ ; and the end of each example is marked by  $*$ .

# Chapter 2

## Preliminaries

In this and the next chapter, we present the mathematical background for the thesis: this chapter covers the non-categorical topics including sets, algebras, graphs, and lattices while Chapter 3 is essentially concerned with explaining the categorical notions referred to throughout the thesis.

### 2.1 Many-Sorted Sets and Algebras

This section presents some elementary definitions regarding many-sorted sets and algebras. The notation we use here is quite standard and is entirely based on [GTW87, ST99].

**Definition 2.1 (many-sorted set)** Let  $S$  be a set of sorts. An  *$S$ -sorted set* is an  $S$ -indexed family of sets  $X = \langle X_s \rangle_{s \in S}$ , which is empty if  $X_s$  is empty for all  $s \in S$ .

For  $S$ -sorted sets  $X = \langle X_s \rangle_{s \in S}$  and  $Y = \langle Y_s \rangle_{s \in S}$ :

$$1. X \cup Y = \langle X_s \cup Y_s \rangle_{s \in S} \quad (\textit{Union})$$

$$2. X \cap Y = \langle X_s \cap Y_s \rangle_{s \in S} \quad (\textit{Intersection})$$

$$3. X \times Y = \langle X_s \times Y_s \rangle_{s \in S} \quad (\textit{Cartesian Product})$$

$$4. X \uplus Y = \langle X_s \uplus Y_s \rangle_{s \in S} \quad (\textit{Disjoint Union})$$

5.  $X \subseteq Y \iff (\forall s \in S : X_s \subseteq Y_s)$  *(Inclusion)*

6.  $X = Y \iff (X \subseteq Y \wedge Y \subseteq X)$  *(Equality)*

□

**Definition 2.2 (many-sorted function)** An *S-sorted function*  $f: X \rightarrow Y$  is an  $S$ -indexed family of functions  $f = \langle f_s: X_s \rightarrow Y_s \rangle_{s \in S}$ . We respectively call  $X$  and  $Y$  the *source* and the *target* of  $f$ . □

**Definition 2.3 (function composition)** If  $f: X \rightarrow Y$  and  $g: Y \rightarrow Z$  are  $S$ -sorted functions, then their *composition*  $g \circ f: X \rightarrow Z$  is the  $S$ -sorted function defined by  $(g \circ f)_s(x) = g_s(f_s(x))$  for  $s \in S$  and  $x \in X_s$ . □

**Definition 2.4 (binary relation)** An  $S$ -sorted *binary relation* on  $X$ , written  $R \subseteq X \times X$ , is an  $S$ -indexed family of binary relations  $R = \langle R_s \subseteq X_s \times X_s \rangle_{s \in S}$ . For  $s \in S$  and  $x, y \in X_s$ ,  $xR_s y$ , also written  $xRy$ , means  $\langle x, y \rangle \in R_s$ . □

**Definition 2.5 (equivalence relation)** Let  $R$  be an  $S$ -sorted relation on  $X$ .  $R$  is an  $S$ -sorted *equivalence* on  $X$  if it is reflexive ( $xR_s x$ ), symmetric ( $xR_s y \implies yR_s x$ ), and transitive ( $xR_s y \wedge yR_s z \implies xR_s z$ ). The symbol  $\equiv$  is used for ( $S$ -sorted) equivalence relations. □

**Definition 2.6 (quotient set)** Let  $\equiv$  be an  $S$ -sorted equivalence on  $X$ . If  $s \in S$  and  $x \in X_s$ , then the *equivalence class of  $x$  modulo  $\equiv$*  is the set  $[x]_{\equiv_s} = \{y \in X_s \mid x \equiv_s y\}$ . The *quotient* of  $X$  modulo  $\equiv$ , denoted  $X/\equiv$ , is the  $S$ -sorted set  $\langle \{[x]_{\equiv_s} \mid x \in X_s\} \rangle_{s \in S}$ . □

**Definition 2.7 (many-sorted signature)** An  $S$ -sorted *signature* is a pair  $\Sigma = \langle S, \Omega \rangle$ , where  $S$  is a set of sort names and  $\Omega$  is an  $(S^* \times S)$ -sorted set of operation names. By  $S^*$  we mean the set of all finite strings from  $S$ , including the empty string  $\lambda$ . Call  $f$  an *operation symbol* of *arity*  $s_1 \dots s_n$  and of *result sort*  $s$  if  $f \in \Omega_{s_1 \dots s_n, s}$ . □

**Definition 2.8 (many-sorted algebra)** Let  $\Sigma = \langle S, \Omega \rangle$  be a signature. A  $\Sigma$ -*algebra*  $A$  consists of an  $S$ -sorted set  $|A|$  of *carrier sets*; and for each  $f \in \Omega_{s_1 \dots s_n, s}$ , a function

$$f_A: |A|_{s_1} \times \cdots \times |A|_{s_n} \rightarrow |A|_s.$$

For an operator symbol  $f \in \Omega_{\lambda, s}$ , the function  $f_A \in |A|_s$  (also written  $f_A : \rightarrow |A|_s$ ) is a *constant* of  $A$  of sort  $s$ .  $\square$

**Definition 2.9 (many-sorted homomorphism)** Let  $\Sigma = \langle S, \Omega \rangle$  be a signature and let  $A$  and  $B$  be  $\Sigma$ -algebras. A  $\Sigma$ -*homomorphism*  $h: A \rightarrow B$  is an  $S$ -sorted function  $h: |A| \rightarrow |B|$  such that if  $f \in \Omega_{s_1 \dots s_n, s}$  and if  $a_1 \in |A|_{s_1}, \dots, a_n \in |A|_{s_n}$ , then

$$h_s(f_A(a_1, \dots, a_n)) = f_B(h_{s_1}(a_1), \dots, h_{s_n}(a_n)). \quad \square$$

## 2.2 Graphs and Graph Homomorphisms

The type of graph introduced in this section is a specific version of directed graph adapted to category theory as well as many areas of the literature on algebraic graph transformation (see [ET96] for references). The importance of graphs in this thesis is two-fold: on the one hand, the definition of graph serves as a basis for defining the term “diagram” in category theory (this is quite standard in any introductory treatment of category theory). And, on the other hand, the notions of graph and graph homomorphism is core to the frameworks developed in Chapters 5 and 6.

**Definition 2.10 (graph)** A *graph* is a quadruple  $G = (N, E, \text{source}_G, \text{target}_G)$  where  $N$  is a set of nodes,  $E$  is a set of edges, and  $\text{source}_G, \text{target}_G: E \rightarrow N$  are functions respectively giving the source and the target for each edge. A *graph homomorphism* from a graph  $G = (N, E, \text{source}_G, \text{target}_G)$  to a graph  $G' = (N', E', \text{source}_{G'}, \text{target}_{G'})$  is a pair of functions  $h = \langle h_{\text{node}}: N \rightarrow N', h_{\text{edge}}: E \rightarrow E' \rangle$  such that:

$$\boxed{h_{\text{node}} \circ \text{source}_G = \text{source}_{G'} \circ h_{\text{edge}}} \quad \text{and} \quad \boxed{h_{\text{node}} \circ \text{target}_G = \text{target}_{G'} \circ h_{\text{edge}}} \quad \square$$

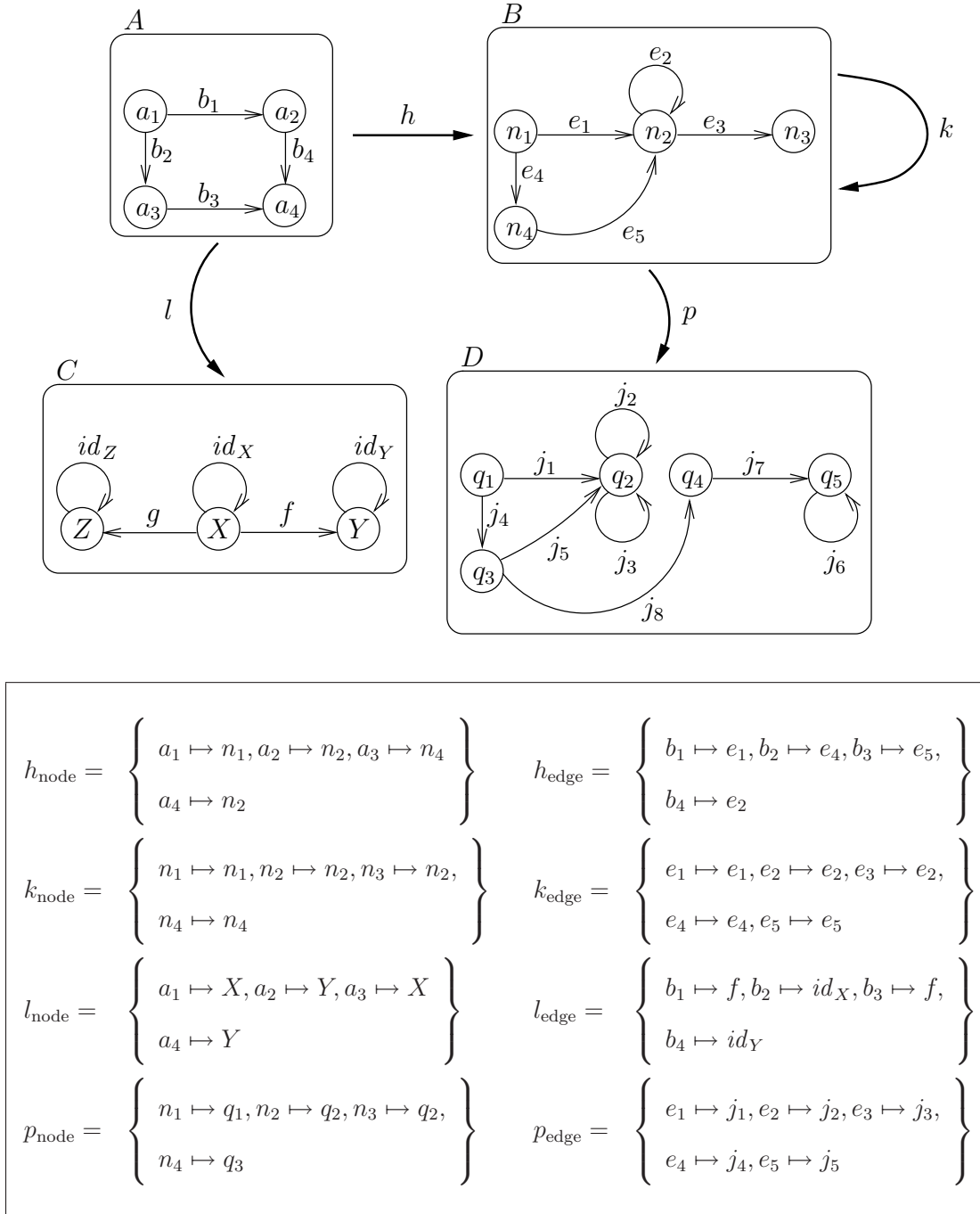


Figure 2.1: Examples of graphs and graph homomorphisms

**Example 2.11** Figure 2.1 illustrates four graphs  $A$ ,  $B$ ,  $C$ , and  $D$  along with four possible homomorphisms  $h : A \rightarrow B$ ,  $k : B \rightarrow B$ ,  $l : A \rightarrow C$ , and  $p : B \rightarrow D$ . \*

An equivalent definition for graph and graph homomorphism can be given by noticing that a graph is a (two-sorted) algebra and a graph homomorphism is a (two-sorted) homomorphism:

**Definition 2.12** A *graph* is a  $\Sigma_G$ -algebra and a *graph homomorphism* is a  $\Sigma_G$ -homomorphism where  $\Sigma_G = \langle S_G, \Omega_G \rangle$  is defined as followed:

$$S_G = \{\text{node}, \text{edge}\};$$

$$\Omega_{\text{edge}, \text{node}} = \{\text{source}, \text{target}\};$$

$$\Omega_{w,s} = \emptyset \text{ for all other } w \in S_G^* \text{ and } s \in S_G. \quad \square$$

NOTATION We usually drop the sort subscripts of the functions that comprise a graph homomorphism and use the name of the homomorphism as an overloaded operator that acts on both nodes and edges. For example, for a node  $a_1$ , we write  $h(a_1) = n_1$  instead of  $h_{\text{node}}(a_1) = n_1$ ; and for an edge  $b_1$ , we write  $h(b_1) = e_1$  instead of  $h_{\text{edge}}(b_1) = e_1$ .  $\square$

## 2.3 Lattices

This section reviews some basic definitions and results on lattices. Lattice theory [Bir79, DP02] provides a unified framework for the study of ordered sets. It finds one of its wide applications in fuzzy set theory [Gog74, HR99] which will be the focus of Chapter 4.

**Definition 2.13 (partial order relation)** A *partial order*  $\leq$  on a set  $A$  is a binary relation on  $A$  such that the following conditions hold:

1.  $\forall a \in A : a \leq a$  *(reflexivity)*
2.  $\forall a, b \in A : a \leq b \wedge b \leq a \implies a = b$  *(anti-symmetry)*
3.  $\forall a, b, c \in A : a \leq b \wedge b \leq c \implies a \leq c$  *(transitivity)*

We call  $\leq$  a **total order** on  $A$  if the following condition holds, as well:

$$4. \forall a, b \in A : a \leq b \vee b \leq a \quad (\text{totality condition}) \quad \square$$

**Definition 2.14 (partially ordered set)** A non-empty set with a partial order on it is called a **partially ordered set** or a **poset** for short. If the relation is a total order then the set is called a **totally ordered set** or more conveniently a **chain**. In a poset  $A$ , we use the expression  $a < b$  to indicate that  $a \leq b$  but  $a \neq b$ .  $\square$

**Definition 2.15 (bottom and top)** Let  $P$  be a poset.  $P$  has a **bottom** element if there exists  $\perp \in P$  such that  $\perp \leq x$  for all  $x \in P$ . Dually,  $P$  has a **top** element if there exists  $\top \in P$  such that  $x \leq \top$  for all  $x \in P$ .  $\square$

**Definition 2.16 (covering relation)** Let  $P$  be a poset and let  $x, y \in P$ . We say  $x$  **is covered by**  $y$  (or  $y$  **covers**  $x$ ), and write  $x \prec y$  or  $y \succ x$ , if  $x < y$  and  $x \leq z < y$  implies  $z = x$ .  $\square$

It simply follows that for elements  $x, y$  in a finite poset  $P$ :  $x < y$  if and only if there exists a finite sequence of covering relations  $x = x_0 \prec x_1 \prec \dots \prec x_n = y$ . This is the underlying observation for visualizing finite posets by **Hasse diagrams**. A Hasse diagram is a graphical rendering of a poset displayed via the covering relation of the poset with an implied upward orientation. In a Hasse diagram, the elements of a finite poset  $P$  are displayed in such a way that for every  $a, b \in P$ : if  $a \prec b$ , then  $b$  is located above  $a$  and the two elements are connected with a line segment. Figure 2.2 illustrates Hasse diagrams. In Figure 2.2(a), for example, the covering relation is:  $\{a \prec b, a \prec c, b \prec d, c \prec d\}$ . It can be verified that for a finite poset  $P$ , the relation  $\leq$  is equal to  $\prec^*$  (the closure of  $\prec$ ). Therefore,  $\leq$  can be reconstructed from the Hasse diagram corresponding to  $P$ .

**Definition 2.17 (upper bound and lower bound)** Let  $P$  be a poset and  $A \subseteq P$ . An element  $p \in P$  is an **upper bound** (resp. **lower bound**) of  $A$  if

$$\forall a \in A : a \leq p \quad (\text{resp. } \forall a \in A : p \leq a) \quad \square$$

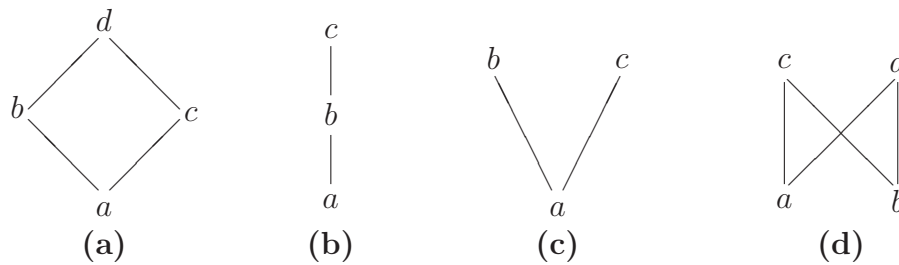


Figure 2.2: Examples of Hasse diagrams

**Definition 2.18 (supremum and infimum)** Let  $P$  be a poset and  $A \subseteq P$ . An element  $p \in P$  is the *least upper bound* or the *supremum* of  $A$ , written  $\mathbf{sup} A$ , if  $p$  is an upper bound of  $A$ , and for all upper bounds  $x$  of  $A$ ,  $p \leq x$ . Dually, an element  $p \in P$  is the *greatest lower bound* or the *infimum* of  $A$ , written  $\mathbf{inf} A$ , if  $p$  is a lower bound of  $A$ , and for all lower bounds  $x$  of  $A$ ,  $x \leq p$ .  $\square$

NOTATION We write  $x \sqcup y$ , read as “ $x$  *join*  $y$ ”, in place of  $\mathbf{sup}\{x, y\}$  when it exists; and  $x \sqcap y$ , read as “ $x$  *meet*  $y$ ”, in place of  $\mathbf{inf}\{x, y\}$  when it exists. Similarly, we write  $\sqcup A$  (“the *join of*  $A$ ”), and  $\sqcap A$  (“the *meet of*  $A$ ”) in place of  $\mathbf{sup} A$  and  $\mathbf{inf} A$ , respectively, when they exist.  $\square$

**Definition 2.19 (lattice)** Let  $P$  be a poset. If  $x \sqcup y$  and  $x \sqcap y$  exist for all  $x, y \in P$ , then  $P$  is called a *lattice*. If  $\sqcup A$  and  $\sqcap A$  exist for all  $A \subseteq P$ , then  $P$  is called a *complete lattice*.  $\square$

**Lemma 2.20** (cf. e.g. [DP02]) *Every finite lattice is complete.*  $\diamond$

**Lemma 2.21** (cf. e.g. [DP02]) *Every complete lattice has a bottom ( $\perp$ ) and a top ( $\top$ ) element.*  $\diamond$

**Example 2.22** In Figure 2.2, (a) and (b) are lattices, but (c) and (d) are not: in (c),  $\mathbf{sup}\{b, c\}$  does not exist; and in (d), the set  $\{a, b\}$  fails to have a least upper bound.  $\ast$

# Chapter 3

## Category Theory

In this chapter, we outline the categorical machinery needed in the thesis. We assume no prior familiarity with category theory and introduce all the category-theoretic notions that will be used in the next chapters. It has not been our aim in general to supply the proof for every single result mentioned in this chapter; however, in cases where the proof of some result is *computationally* [RB88] important due to its constructive nature, we have provided the proof in Appendix A.

The canonical introduction to category theory is Mac Lane's book [Mac71] although the book is fairly difficult to read for the uninitiated. An excellent introduction to category theory from a computer science perspective is Barr and Wells' book [BW99]. Unfortunately, this book does not cover the topic of comma categories which is crucial in this thesis. The reader should consult [GB84, RB88] where detailed treatments of this topic can be found. Another excellent introduction to category theory is Chapter 3 of Sannella and Tarlecki's upcoming book [ST]. The authors of this book have given a comprehensive and yet concise account of those topics in category theory that are often taken for granted in computer science<sup>1</sup>.

---

<sup>1</sup>I am grateful to the authors of [ST] for granting me access to a draft of their book.

## 3.1 Categories

**Definition 3.1 (category)** A category  $\mathcal{C}$  consists of:

1. a collection of objects denoted  $|\mathcal{C}|$ ;
2. for every  $A, B \in |\mathcal{C}|$ , a collection  $Hom_{\mathcal{C}}(A, B)$  of **morphisms** (also called **arrows**) from  $A$  to  $B$ . We write  $f : A \rightarrow B$  when  $f \in Hom_{\mathcal{C}}(A, B)$  and call  $A$  the **source** and  $B$  the **target** of  $f$ ;
3. for every  $A, B, C \in |\mathcal{C}|$ , a **composition** operation

$$\circ : Hom_{\mathcal{C}}(A, B) \times Hom_{\mathcal{C}}(B, C) \rightarrow Hom_{\mathcal{C}}(A, C)$$

such that:

- I. (*composition associativity*) any morphisms  $f \in Hom_{\mathcal{C}}(A, B)$ ,  $g \in Hom_{\mathcal{C}}(B, C)$ , and  $h \in Hom_{\mathcal{C}}(C, D)$  satisfy:  $h \circ (g \circ f) = (h \circ g) \circ f$ ;
- II. (*existence of identity*) for every  $A \in |\mathcal{C}|$ , there exists a morphism  $id_A \in Hom_{\mathcal{C}}(A, A)$ , called the **identity** of  $A$ , such that  $f \circ id_A = f$  for any morphism  $f \in Hom_{\mathcal{C}}(A, B)$ ; and  $id_A \circ g = g$  for any morphism  $g \in Hom_{\mathcal{C}}(B, A)$ . □

### Example 3.2

- A single object together with a single morphism (which must be the identity morphism) constitutes a category, denoted  $\mathbf{1}$ .
- The category of sets, denoted  $\mathbf{Set}$ , has sets as objects and functions as morphisms. Notice that the source and the target of every function is explicitly given.
- For a given signature  $\Sigma$ , the category of  $\Sigma$ -algebras, denoted  $\mathbf{Alg}(\Sigma)$ , has  $\Sigma$ -algebras as objects and  $\Sigma$ -homomorphisms as morphisms.

- For the signature  $\Sigma_G$  given in Definition 2.10,  $\mathbf{Alg}(\Sigma_G)$  is the category of graphs which will hereafter be denoted **Graph**.
- A poset  $P$  can be viewed as category whose objects are the elements of  $P$ ; and for any  $x, y \in P$  satisfying  $x \leq y$ , there is a unique morphism that has  $x$  as source and  $y$  as target.
- A set can be viewed as a category whose objects are the elements of the set and whose only morphisms are the identity morphisms.

\*

**Definition 3.3 (small, locally small, and large categories)** A category  $\mathcal{C}$  is *small* if  $|\mathcal{C}|$  is a set and for any  $A, B \in |\mathcal{C}|$ , the collection  $Hom_{\mathcal{C}}(A, B)$  is a set, as well. A category is *locally small* if  $Hom_{\mathcal{C}}(A, B)$  is a set for any  $A, B \in |\mathcal{C}|$ . If a category is not small, then it is said to be *large*.  $\square$

**Remark 3.4** All the categories referred to in this thesis are locally small.  $\square$

## 3.2 Functors

**Definition 3.5 (functor)** A functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  consists of:

- A function  $F_{Obj} : |\mathcal{C}| \rightarrow |\mathcal{D}|$ ;
- For every  $A, B \in |\mathcal{C}|$ , a function  $F_{A,B} : Hom_{\mathcal{C}}(A, B) \rightarrow Hom_{\mathcal{D}}(F_{Obj}(A), F_{Obj}(B))$ .

such that:

1. Identities are preserved:  $F_{A,A}(id_A) = id_{F_{Obj}(A)}$  for every  $A \in |\mathcal{C}|$ .
2. Composition is preserved:  $F_{A,C}(g \circ f) = F_{B,C}(g) \circ F_{A,B}(f)$  for all morphisms  $f : A \rightarrow B$  and  $g : B \rightarrow C$  in  $\mathcal{C}$ .  $\square$

**Example 3.6 (identity functor)** For a category  $\mathcal{C}$ , there exists a functor  $I_{\mathcal{C}} : \mathcal{C} \rightarrow \mathcal{C}$ , known as the *identity functor* on  $\mathcal{C}$ , that maps every object and morphism in  $\mathcal{C}$  to itself. \*

**Example 3.7 (Cartesian product functor)** There is functor  $T : \mathbf{Set} \rightarrow \mathbf{Set}$ , known as the *Cartesian product functor*, that maps every set  $N$  to  $N \times N$  and every function  $f : N \rightarrow N'$  to  $f \times f : N \times N \rightarrow N' \times N'$  where  $f \times f$  is the function such that  $(x, y) \xrightarrow{f \times f} (f(x), f(y))$  for all  $x, y \in N$ . \*

**Definition 3.8 (category of locally small categories)** The category of locally small categories, denoted  $\mathbf{Cat}$ , has locally small categories as objects and functors as morphisms. If  $F : \mathcal{A} \rightarrow \mathcal{B}$  and  $G : \mathcal{B} \rightarrow \mathcal{C}$  are  $\mathbf{Cat}$ -morphisms (i.e. functors),  $G \circ F : \mathcal{A} \rightarrow \mathcal{C}$  is defined as follows:

- $(G \circ F)_{Obj} = G_{Obj} \circ F_{Obj}$ ;
- $(G \circ F)_{A,B} = G_{F(A),F(B)} \circ F_{A,B}$  for all  $A, B \in |\mathcal{A}|$ .

The identity morphism for every  $\mathcal{A} \in |\mathbf{Cat}|$  is the identity functor  $I_{\mathcal{A}} : \mathcal{A} \rightarrow \mathcal{A}$ . □

**Example 3.9 (underlying graph functor)** Let  $\mathcal{C}$  be a category. By forgetting how morphisms in  $\mathcal{C}$  are composed and forgetting which morphisms are the identities, we obtain the *underlying graph* of  $\mathcal{C}$ . This yields a functor  $U : \mathbf{Cat} \rightarrow \mathbf{Graph}$  that maps every  $\mathcal{C} \in |\mathbf{Cat}|$  to the underlying graph of  $\mathcal{C}$  and every  $\mathbf{Cat}$ -morphism  $F$  to the graph homomorphism induced by  $F$ . Notice that, by definition, every functor is a graph homomorphism but the converse is not true. \*

### 3.3 Diagrams

**Definition 3.10 (diagram)** Let  $\mathcal{C}$  be a category and let  $G$  be a graph. A *diagram* of *shape*  $G$  in  $\mathcal{C}$  is a graph homomorphism  $D : G \rightarrow U(\mathcal{C})$ , where  $U : \mathbf{Cat} \rightarrow \mathbf{Graph}$  is the underlying graph functor (cf. Example 3.9). □

**Example 3.11** In Figure 2.1, graph  $C$  can be thought of as the underlying graph of a category  $\mathcal{C}$  with objects  $X, Y, Z$  and identities  $id_X, id_Y, id_Z$ . The non-identity morphisms are  $f : X \rightarrow Y$  and  $g : X \rightarrow Z$ . The graph homomorphism  $l : A \rightarrow C$  in Figure 2.1 identifies the following diagram in  $\mathcal{C}$ :

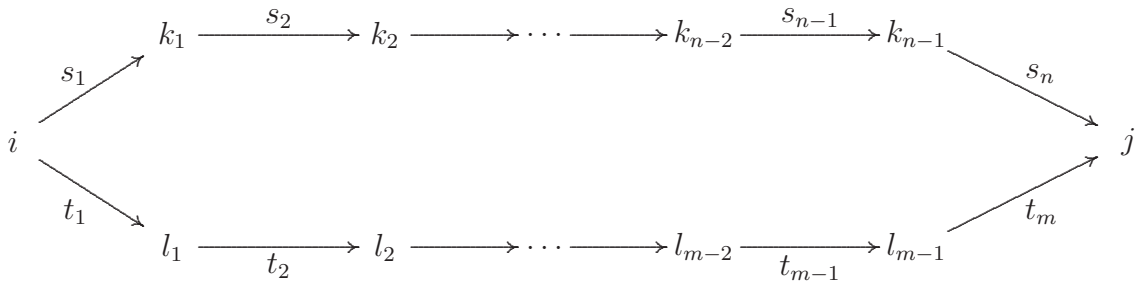
$$\begin{array}{ccc}
 X & \xrightarrow{f} & Y \\
 id_X \downarrow & & \downarrow id_Y \\
 X & \xrightarrow{f} & Y
 \end{array}$$

The shape graph for the above diagram is graph  $A$  as shown in Figure 2.1. \*

**Definition 3.12 (finite diagram)** A diagram is said to be *finite* if its shape graph is finite, that is, if its shape graph has a finite number of nodes and edges. □

**Definition 3.13 (discrete diagram)** A diagram is said to be *discrete* if its shape graph has no edges. □

**Definition 3.14 (commutative diagram)** A diagram  $D : G \rightarrow U(\mathcal{C})$  is said to *commute* (or be *commutative*) if for each pair of nodes  $i, j$  in  $G$  and any two paths:



from  $i$  to  $j$  in  $G$ , we have:

$$D(s_n) \circ D(s_{n-1}) \cdots \circ D(s_2) \circ D(s_1) = D(t_m) \circ D(t_{m-1}) \circ \cdots \circ D(t_2) \circ D(t_1). \quad \square$$

### 3.4 Some Basic Category Theoretic Definitions

Throughout this section, let  $\mathcal{C}$  be an arbitrary category.

**Definition 3.15 (isomorphism)** A  $\mathcal{C}$ -morphism  $f : A \rightarrow B$  is an *isomorphism* if there exists a morphism  $f^{-1} : B \rightarrow A$  such that  $f^{-1} \circ f = id_A$  and  $f \circ f^{-1} = id_B$ . The morphism  $f^{-1}$  is called the *inverse* of  $f$ ; and objects  $A$  and  $B$  are called *isomorphic*.  $\square$

**Remark 3.16 (isomorphic categories)** Categories  $\mathcal{A}$  and  $\mathcal{B}$  are said to be isomorphic if there exists an isomorphism  $F : \mathcal{A} \rightarrow \mathcal{B}$  in **Cat**.  $\square$

**Definition 3.17 (initial object)** An object  $\mathbf{0} \in |\mathcal{C}|$  is *initial* if for every  $A \in |\mathcal{C}|$ , there exists a unique morphism  $\langle \rangle : \mathbf{0} \rightarrow A$ .  $\square$

**Example 3.18 (initial objects in Set and Graph)** The initial object in **Set** (resp. **Graph**) is the empty set (resp. the empty graph).  $*$

NOTATION Throughout the rest of this chapter, a dashed morphism in a diagram indicates the uniqueness of that morphism.  $\square$

**Definition 3.19 (binary coproduct)** A *binary coproduct* of  $A, B \in |\mathcal{C}|$  is an object  $A+B \in |\mathcal{C}|$  together with a pair of morphisms  $\iota_A : A \rightarrow A+B$  and  $\iota_B : B \rightarrow A+B$ , called the *injection* morphisms, such that for any  $C \in |\mathcal{C}|$  and pair of morphisms  $f : A \rightarrow C$  and  $g : B \rightarrow C$  there is a unique morphism  $\langle f|g \rangle : A+B \rightarrow C$  making the following diagram commute:

$$\begin{array}{ccccc}
 & & C & & \\
 & & \uparrow & & \\
 & f & \nearrow & & \nwarrow g \\
 A & \xrightarrow{\iota_A} & A+B & \xleftarrow{\iota_B} & B
 \end{array}$$

$\square$

**Example 3.20 (canonical binary coproducts in Set)** The canonical binary coproduct of two sets  $A$  and  $B$  is the set  $A \uplus B$  together with the obvious injections  $\iota_A : A \rightarrow A \uplus B$  and  $\iota_B : B \rightarrow A \uplus B$ . \*

**Definition 3.21 (coequalizer)** A *coequalizer* of a pair of parallel  $\mathcal{C}$ -morphisms  $f : A \rightarrow B$  and  $g : A \rightarrow B$  is an object  $C \in |\mathcal{C}|$  together with a morphism  $q : B \rightarrow C$  such that  $q \circ f = q \circ g$  and for any morphism  $k : B \rightarrow D$  satisfying  $k \circ f = k \circ g$ , there is a unique morphism  $h : C \rightarrow D$  such that  $h \circ q = k$ .

$$\begin{array}{ccccc}
 A & \begin{array}{c} \xrightarrow{f} \\ \xrightarrow{g} \end{array} & B & \xrightarrow{q} & C \\
 & & & \searrow k & \vdots h \\
 & & & & D
 \end{array}$$

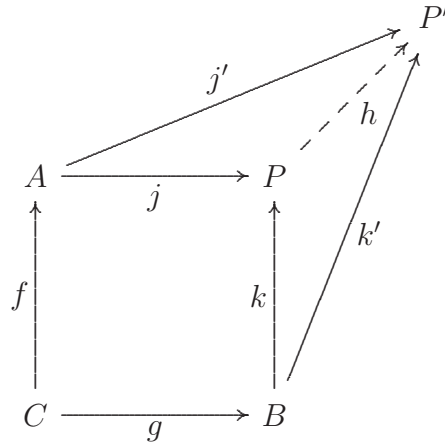
□

**Example 3.22 (canonical coequalizers in Set)** Let  $f : A \rightarrow B$  and  $g : A \rightarrow B$  be a pair of **Set**-morphisms (i.e. functions) and let  $R = \{(f(a), g(a)) \mid a \in A\}$ . Assuming  $\equiv_R$  is the smallest equivalence relation that includes  $R$ , the canonical coequalizer of  $f$  and  $g$  is  $B/\equiv_R$  (i.e. the quotient of  $B$  modulo  $\equiv_R$ ) together with the function  $q : B \rightarrow B/\equiv_R$  such that  $q(b) = [b]_{\equiv_R}$  for all  $b \in B$ . \*

**Remark 3.23** It is easy to verify that for a binary relation  $R$  on a (single-sorted) set  $S$ , the smallest equivalence relation that includes  $R$ , denoted  $\equiv_R$ , can be constructed as follows: consider an *undirected graph*  $G$  in which the set of nodes is  $S$ , and for any nodes  $x, y$  in  $G$ , there is an undirected edge between  $x$  and  $y$  if and only if  $(x, y) \in R$ . Then,  $B/\equiv_R$  will be the set of  $G$ 's connected components. Thus, for any  $x, y \in S$ :  $x \equiv_R y$  if and only if  $x$  and  $y$  belong to the same connected component of  $G$ . □

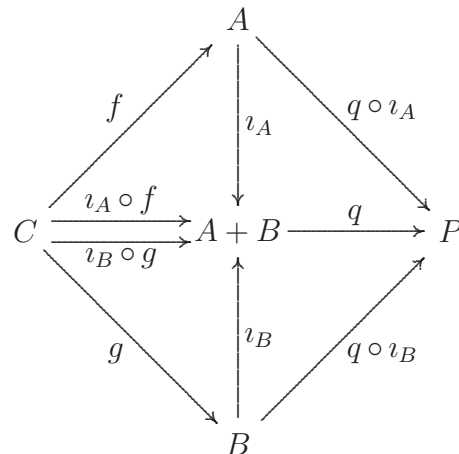
**Definition 3.24 (pushout)** A *pushout* of a pair of  $\mathcal{C}$ -morphisms  $f : C \rightarrow A$  and  $g : C \rightarrow B$  is an object  $P \in |\mathcal{C}|$  together with a pair of morphisms  $j : A \rightarrow P$  and  $k : B \rightarrow P$  such that:

- $j \circ f = k \circ g$
- for any  $P' \in |\mathcal{C}|$  and pair of morphisms  $j' : A \rightarrow P'$  and  $k' : B \rightarrow P'$  satisfying  $j' \circ f = k' \circ g$ , there is a unique morphism  $h : P \rightarrow P'$  such that the following diagram commutes:



□

**Remark 3.25 (construction of pushouts)** A pushout of any pair of  $\mathcal{C}$ -morphisms with common source can be constructed if every pair of  $\mathcal{C}$ -objects has a coproduct and every pair of parallel  $\mathcal{C}$ -morphisms has a coequalizer: let  $f : C \rightarrow A$  and  $f : C \rightarrow B$  be a pair of morphisms and let  $A + B \in |\mathcal{C}|$  together with the injections  $\iota_A : A \rightarrow A + B$  and  $\iota_B : B \rightarrow A + B$  be a binary coproduct of  $A$  and  $B$ . Let  $P \in |\mathcal{C}|$  together with a morphism  $q : A + B \rightarrow P$  be a coequalizer of  $\iota_A \circ f$  and  $\iota_B \circ g$ . It can be verified that the outer square in the following diagram is indeed a pushout square [EP72]:



□

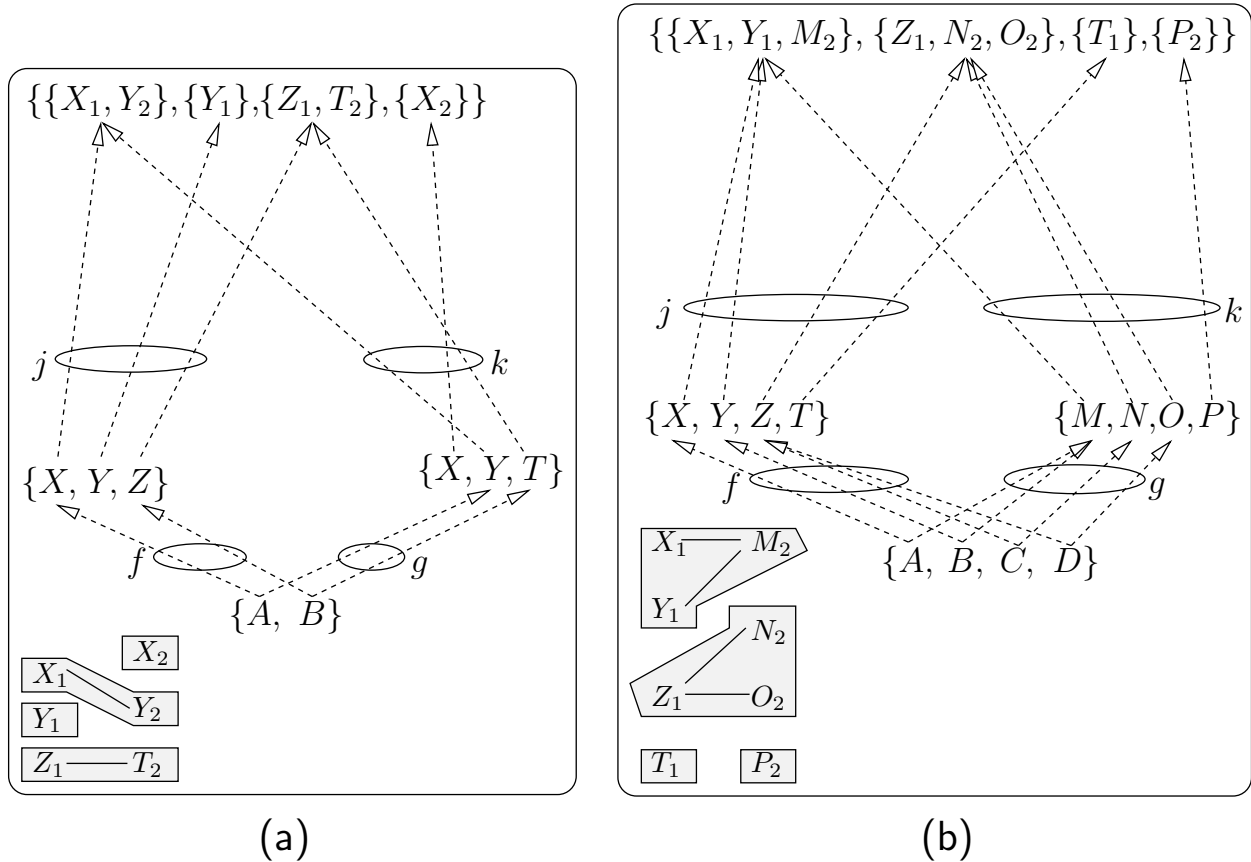


Figure 3.1: Pushout examples in **Set**

**Example 3.26 (canonical pushouts in **Set**)** Figure 3.1 shows two examples of canonical pushout computation in **Set**. The maps corresponding to the morphisms  $f$ ,  $g$ ,  $j$ , and  $k$  of the pushout square have been marked in both examples. The figure also illustrates how the required canonical coequalizer for each example has been computed (cf. Remark 3.23).

### 3.5 Colimits

**Definition 3.27 (cocone and colimit)** Let  $D$  be a diagram of shape  $G$  in a category  $\mathcal{C}$  and let  $N$  and  $E$  denote the set of  $G$ 's nodes and edges, respectively. A cocone  $\alpha$  over  $D$  is a  $\mathcal{C}$ -object  $X$  together with a family of  $\mathcal{C}$ -morphisms  $\langle \alpha_n : D(n) \rightarrow X \rangle_{n \in N}$  such

that for every edge  $e \in E$  with  $\text{source}_G(e) = i$  and  $\text{target}_G(e) = j$ , the following diagram commutes:

$$\begin{array}{ccc}
 & X & \\
 \alpha_i \nearrow & & \nwarrow \alpha_j \\
 D(i) & \xrightarrow{D(e)} & D(j)
 \end{array}$$

A **colimit** of  $D$  is a cocone  $\langle \alpha_n : D(n) \rightarrow X \rangle_{n \in N}$  such that for any cocone  $\langle \alpha'_n : D(n) \rightarrow X' \rangle_{n \in N}$ , there is a unique morphism  $h : X \rightarrow X'$  that makes the following diagram commute for all  $n \in N$ :

$$\begin{array}{ccc}
 X' & \xleftarrow{h} & X \\
 \alpha'_n \nearrow & & \nwarrow \alpha_n \\
 & D(n) &
 \end{array}$$

□

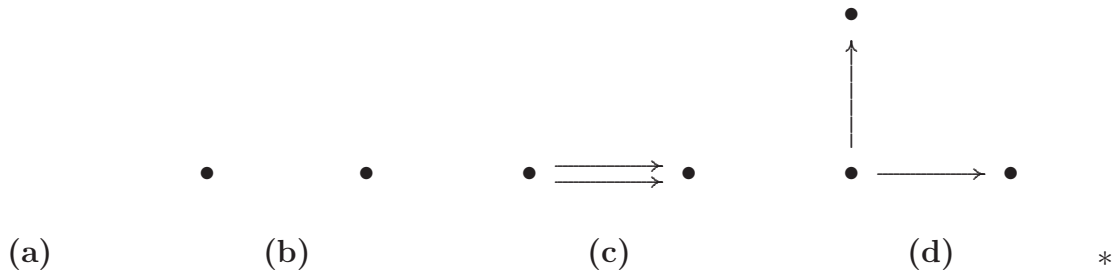
**Lemma 3.28** (cf. e.g. [BW99]) *Colimits are unique up to isomorphism.*

◇

**Definition 3.29 (cocompleteness)** A category  $\mathcal{C}$  is (finitely) cocomplete if every (finite) diagram in  $\mathcal{C}$  has a colimit.

□

**Example 3.30** Initial objects, binary coproducts, coequalizers, and pushouts are colimits over diagrams of shapes (a), (b), (c), and (d) respectively:



The shape graph **(b)** suggests the generalization of the definition of binary coproduct in the following sense:

**Definition 3.31 (coproduct)** The colimit of a diagram  $D$  is called a *coproduct* if  $D$  is discrete. □

**Lemma 3.32** *A category with an initial object and binary coproducts has all finite coproducts.* ◇

**Proof** See Appendix A. ■

Among the numerous lemmas on (finite) cocompleteness of categories, the following lemma and its corollary are of particular importance due to the constructive nature of their proofs:

**Lemma 3.33** *A category  $\mathcal{C}$  is finitely cocomplete if it has an initial object, binary coproducts of all object pairs, and coequalizers of all parallel morphism pairs.* ◇

**Proof** See Appendix A. ■

**Corollary 3.34** *A category  $\mathcal{C}$  is finitely cocomplete if it has an initial object and pushouts of all pairs of morphisms with common source.* ◇

**Proof** See Appendix A. ■

**Example 3.35** **Set** is finitely cocomplete (cf. Examples 3.18, 3.20, and 3.22). \*

**Remark 3.36** It can also be shown that  $\mathbf{Alg}(\Sigma)$  is finitely cocomplete for any signature  $\Sigma$ , but the proof is more difficult. The interested reader can consult standard textbooks on Categorical Algebra (such as [Bor94]) for the proof of cocompleteness in the single-sorted case. The proof for the many-sorted case is analogous. □

The intuition behind colimits is that they put things together, with nothing essentially new added, and nothing left over [Gog91]. A pushout of two morphisms  $f : C \rightarrow A$  and  $g : C \rightarrow B$  in **Set**, for example, can be interpreted as the **combination** of sets  $A$  and  $B$  with respect to a shared part  $C$  in such a way that only one copy of  $C$  is included in the combination. This observation was illustrated in Example 3.26. More generally:

*“Given a species of structure, say widgets, the result of interconnecting a system of widgets to form a super-widget corresponds to taking the colimit of the diagram of widgets in which the morphisms show how they are interconnected.” [Gog91].*

The above dogma is, in fact, the main reason behind our interest in (finite) cocompleteness results. For reasons that will become more clear later in Section 3.6, we are also interested in functors that preserve (finite) colimits:

**Definition 3.37 (cocontinuity)** A functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  is said to be (finitely) **cocontinuous** if it preserves the existing colimits of all (finite) diagrams in  $\mathcal{C}$ , that is, if for any (finite) diagram  $D$  in  $\mathcal{C}$ , the functor  $F$  maps any colimiting cocone over  $D$  to a colimiting cocone over  $F(D)$ . □

**Lemma 3.38** *If  $\mathcal{C}$  is a finitely cocomplete category and if a functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  preserves initial objects, binary coproducts of all object pairs, and coequalizers of all parallel morphism pairs, then  $F$  is finitely cocontinuous.* ◇

**Proof** Follows from Lemma 3.33. ■

**Corollary 3.39** *If  $\mathcal{C}$  is a finitely cocomplete category and if a functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  preserves initial objects and pushouts of all pairs of morphisms with common source, then  $F$  is finitely cocontinuous.* ◇

**Proof** Follows from Corollary 3.34. ■

### 3.6 Comma Categories

**Definition 3.40 (comma category)** Let  $\mathcal{A}$ ,  $\mathcal{B}$ , and  $\mathcal{C}$  be categories and  $L : \mathcal{A} \rightarrow \mathcal{C}$  and  $R : \mathcal{B} \rightarrow \mathcal{C}$  be functors. The comma category  $(L \downarrow R)$  has as objects, triples  $(A, f : L(A) \rightarrow R(B), B)$  where  $A$  is an object of  $\mathcal{A}$ , and  $B$  is an object of  $\mathcal{B}$ . A morphism from  $(A, f, B)$  to  $(A', f', B')$  is a pair  $(s : A \rightarrow A', t : B \rightarrow B')$  such that the following diagram commutes in  $\mathcal{C}$ :

$$\begin{array}{ccc}
 L(A) & \xrightarrow{f} & R(B) \\
 \downarrow L(s) & & \downarrow R(t) \\
 L(A') & \xrightarrow{f'} & R(B')
 \end{array}$$

Identities are pairs of identities and composition is defined component-wise, i.e. for  $(L \downarrow R)$ -morphisms  $(s, t)$  and  $(s', t')$ , we have:  $(s, t) \circ (s', t') = (s \circ s', t \circ t')$ .  $\square$

It is easy to verify that the above definition indeed gives rise to a category.

**Remark 3.41 (projection functors)** Every comma category  $(L \downarrow R)$  is equipped with a pair of projection functors  $\pi_1 : (L \downarrow R) \rightarrow \mathcal{A}$  and  $\pi_2 : (L \downarrow R) \rightarrow \mathcal{B}$ . The former projects objects and morphisms onto their first coordinates; and the latter projects objects onto their third coordinates and morphisms onto their second.  $\square$

**NOTATION** For an arbitrary category  $\mathcal{C}$ , any  $\mathcal{C}$ -object  $C$  can be considered a functor  $1_C : \mathbf{1} \rightarrow \mathcal{C}$ . Assuming  $F : \mathcal{A} \rightarrow \mathcal{C}$  is an arbitrary functor, we usually write  $(F \downarrow C)$  in place of  $(F \downarrow 1_C)$  and  $(C \downarrow F)$  in place of  $(1_C \downarrow F)$ .  $\square$

**Example 3.42 (many-sorted sets over an index set)** Let  $S$  be a fixed set. The category  $S\text{-Set}$  whose objects are  $S$ -indexed families of disjoint sets and whose morphisms are  $S$ -indexed families of functions is isomorphic to the comma category  $(I_{\text{Set}} \downarrow S)$ , where  $I_{\text{Set}} : \text{Set} \rightarrow \text{Set}$  is the identity functor on  $\text{Set}$ . \*

**Example 3.43 (morphism category)** For an arbitrary category  $\mathcal{C}$ , the *morphism category* of  $\mathcal{C}$ , denoted  $\mathcal{C}^{\rightarrow}$ , has the morphisms of  $\mathcal{C}$  as objects. A  $\mathcal{C}^{\rightarrow}$ -morphism from  $f : A \rightarrow B$  to  $f' : A' \rightarrow B'$  is a pair of  $\mathcal{C}$ -morphisms  $(h : A \rightarrow A', k : B \rightarrow B')$  making the following diagram commute in  $\mathcal{C}$ :

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \downarrow h & & \downarrow k \\ A' & \xrightarrow{f'} & B' \end{array}$$

It is easy to verify that  $\mathcal{C}^{\rightarrow}$  is isomorphic to  $(I_{\mathcal{C}} \downarrow I_{\mathcal{C}})$ , where  $I_{\mathcal{C}}$  is the identity functor on  $\mathcal{C}$ . \*

**Example 3.44 (category of graphs, revisited)** Denote every graph  $G$  as a triple  $G = (E, f : E \rightarrow N \times N, N)$  where  $E$  is a set of edges,  $N$  is a set of nodes, and  $f$  is a function taking every  $e \in E$  to a tuple  $(\text{source}_G(e), \text{target}_G(e)) \in N \times N$ . Then, a graph homomorphism from a graph  $G = (E, f, N)$  to a graph  $G' = (E', f', N')$  consists of a pair of functions  $(h_{\text{edge}} : E \rightarrow E', h_{\text{node}} : N \rightarrow N')$  making the following diagram commute in  $\text{Set}$ :

$$\begin{array}{ccc} E & \xrightarrow{f} & N \times N \\ \downarrow h_{\text{edge}} & & \downarrow h_{\text{node}} \times h_{\text{node}} \\ E' & \xrightarrow{f'} & N' \times N' \end{array}$$

It is now easy to verify that **Graph** is isomorphic to the comma category  $(I_{\mathbf{Set}} \downarrow T)$  where  $I_{\mathbf{Set}} : \mathbf{Set} \rightarrow \mathbf{Set}$  is the identity functor on **Set** and  $T : \mathbf{Set} \rightarrow \mathbf{Set}$  is the Cartesian product functor (cf. Example 3.7). \*

**Lemma 3.45** [Tar86, RB88, Bor94]<sup>2</sup> Let  $L : \mathcal{A} \rightarrow \mathcal{C}$  and  $R : \mathcal{B} \rightarrow \mathcal{C}$  be functors with  $L$  finitely cocontinuous. If  $\mathcal{A}$  and  $\mathcal{B}$  are finitely cocomplete, so is the comma category  $(L \downarrow R)$ ; moreover, the projections  $\pi_1 : (L \downarrow R) \rightarrow \mathcal{A}$  and  $\pi_2 : (L \downarrow R) \rightarrow \mathcal{B}$  preserve finite colimits.  $\diamond$

**Proof** See Appendix A.  $\blacksquare$

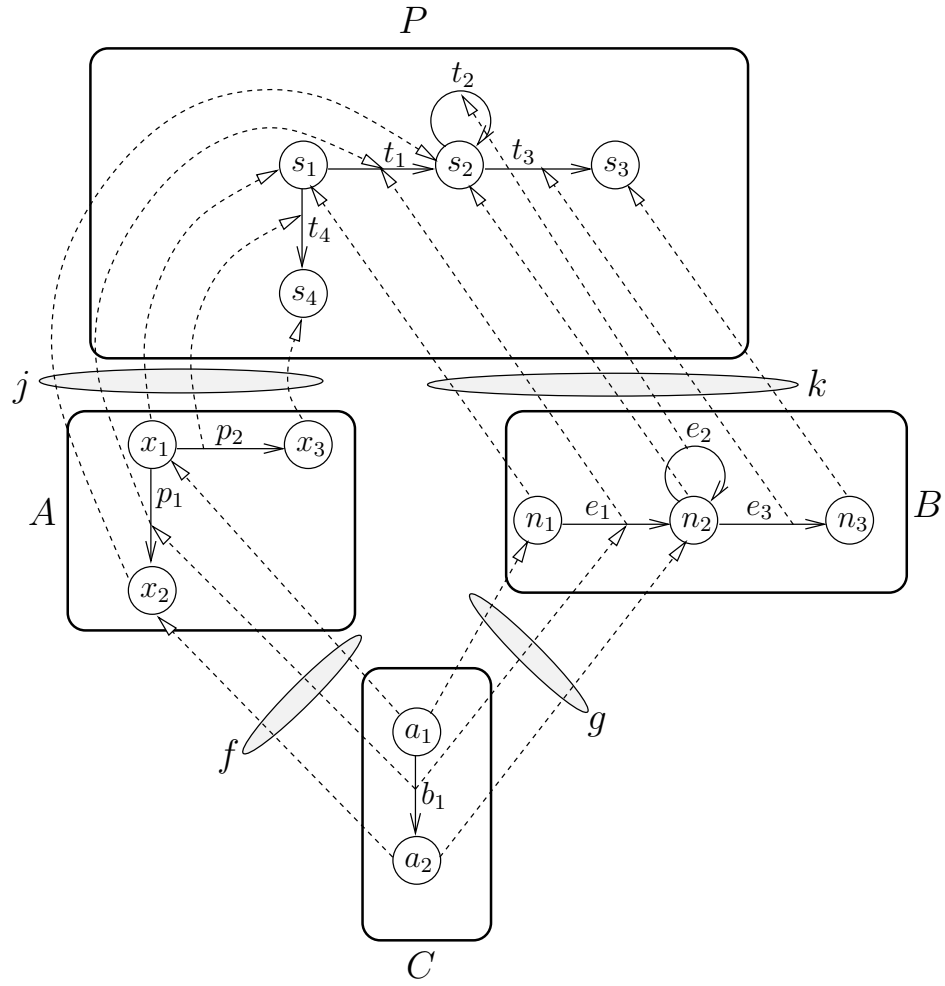
**Remark 3.46** Notice that the colimit preservation property of the projection functors (established in Lemma 3.45) implies that finite colimits in a comma category  $(L \downarrow R)$  are inherited from those in the constituent categories (i.e.  $\mathcal{A}$  and  $\mathcal{B}$ ) when  $L$  is finitely cocontinuous.  $\square$

**Example 3.47** Since **Set** is finitely cocomplete and the identity functor is finitely cocontinuous, Lemma 3.45 implies that  $(I_{\mathbf{Set}} \downarrow T)$ , i.e. the category of graphs, is finitely cocomplete as well. Moreover, *finite colimits in the category of graphs are computed component-wise for nodes and edges.* \*

**Example 3.48 (pushouts in Graph)** Figure 3.2 shows an example of canonical pushout computation in **Graph**. The naming of graphs and graph homomorphisms in this figure conforms to that of the objects and morphisms of the pushout square in Definition 3.24.\*

---

<sup>2</sup>The cited references take the non-finite case into account and prove a stronger result.



$f_{\text{node}} = \{a_1 \mapsto x_1, a_2 \mapsto x_2\}$	$f_{\text{edge}} = \{b_1 \mapsto p_1\}$
$g_{\text{node}} = \{a_1 \mapsto n_1, a_2 \mapsto n_2\}$	$g_{\text{edge}} = \{b_1 \mapsto e_1\}$
$j_{\text{node}} = \{x_1 \mapsto s_1, x_2 \mapsto s_2, x_3 \mapsto s_4\}$	$j_{\text{edge}} = \{p_1 \mapsto t_1, p_2 \mapsto t_4\}$
$k_{\text{node}} = \{n_1 \mapsto s_1, n_2 \mapsto s_2, n_3 \mapsto s_3\}$	$k_{\text{edge}} = \{e_1 \mapsto t_1, e_2 \mapsto t_2, e_3 \mapsto t_3\}$

Figure 3.2: Pushout example in **Graph**

# Chapter 4

## Categories of Fuzzy Sets

Since its inception in the 1960s, fuzzy set theory has received considerable attention from different computing disciplines. In this chapter, we briefly introduce fuzzy set categories and immediately turn our attention to a few important results regarding them. Most of the material presented in this chapter can be found in [Gog68, Gog74] and is probably well-known in the literature on topos theory (cf. e.g. [BW99, BW84]); however, we were not able to find any reference that includes all the results we need in a context close to that of our work.

### 4.1 Fuzzy Sets and Fuzzy Set Morphisms

**Definition 4.1 (fuzzy set)** Let  $Q$  be a poset. A  **$Q$ -valued set** is a pair  $(S, \sigma)$  consisting of a set  $S$  and a function  $\sigma : S \rightarrow Q$ . We call  $S$  the **carrier set** of  $(S, \sigma)$  and  $Q$  the **truth set** of  $\sigma$ . For every  $s \in S$ , the value  $\sigma(s)$  is interpreted as the **degree of membership** of  $s$  in  $(S, \sigma)$ .  $\square$

**Definition 4.2 (fuzzy set morphism)** Let  $Q$  be a poset and let  $(S, \sigma)$  and  $(T, \tau)$  be a pair of  $Q$ -valued sets. A morphism  $\mathbf{f} : (S, \sigma) \rightarrow (T, \tau)$  is a function  $f : S \rightarrow T$  such that

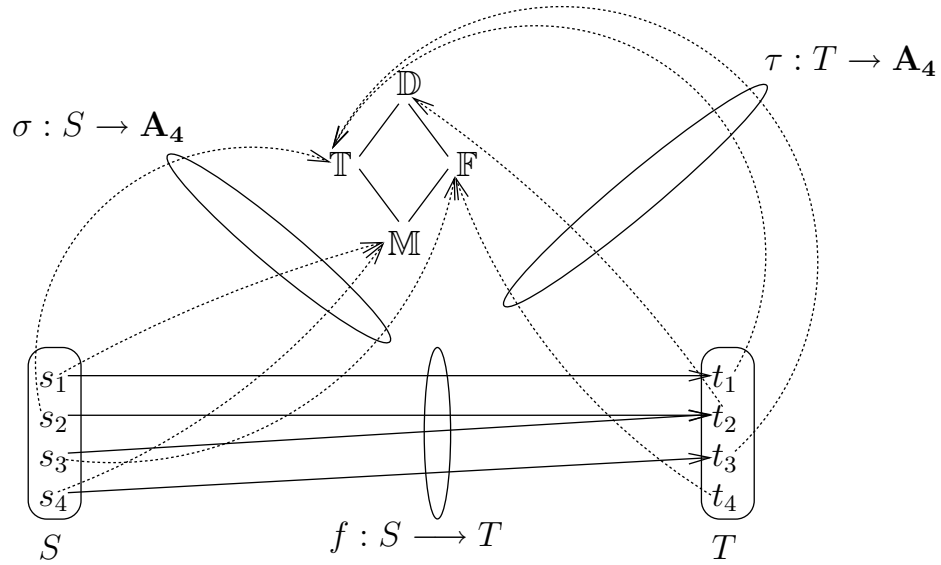


Figure 4.1: Example of fuzzy sets

$\sigma \leq \tau \circ f$ , i.e. the degree of membership of  $s$  in  $(S, \sigma)$  does not exceed that of  $f(s)$  in  $(T, \tau)$ . The function  $f: S \rightarrow T$  is called the **carrier function** of  $\mathbf{f}$ .  $\square$

**Note** In classical fuzzy set theory, it is implicitly assumed that the poset  $Q$  is the closed real interval  $[0, 1]$  with the obvious linear ordering. *The reader should be aware that no such assumption has been made in this thesis.*  $\square$

**Example 4.3** Figure 4.1 (informally) shows two  $\mathbf{Fuzz}(\mathbf{A}_4)$  objects  $(S, \sigma)$  and  $(T, \tau)$  along with the carrier function  $f: S \rightarrow T$  of a  $\mathbf{Fuzz}(\mathbf{A}_4)$ -morphism  $\mathbf{f}: (S, \sigma) \rightarrow (T, \tau)$  where  $\mathbf{A}_4$  is the lattice shown in the same figure.  $*$

**Definition/Proposition 4.4 (fuzzy set category)** For a fixed poset  $Q$ , the objects and morphisms defined above together with the obvious identities give rise to a category, denoted  $\mathbf{Fuzz}(Q)$ .  $\square$

## 4.2 Cocompleteness Results for Fuzzy Set Categories

**Lemma 4.5** [Gog68, Gog74]  $\mathbf{Fuzz}(Q)$  is finitely cocomplete when  $Q$  is a complete lattice.  $\diamond$

**Proof (sketch)**<sup>1</sup> We show how to construct the initial object, binary coproducts, and coequalizers. Finite cocompleteness of  $\mathbf{Fuzz}(Q)$  then follows from Lemma 3.33.

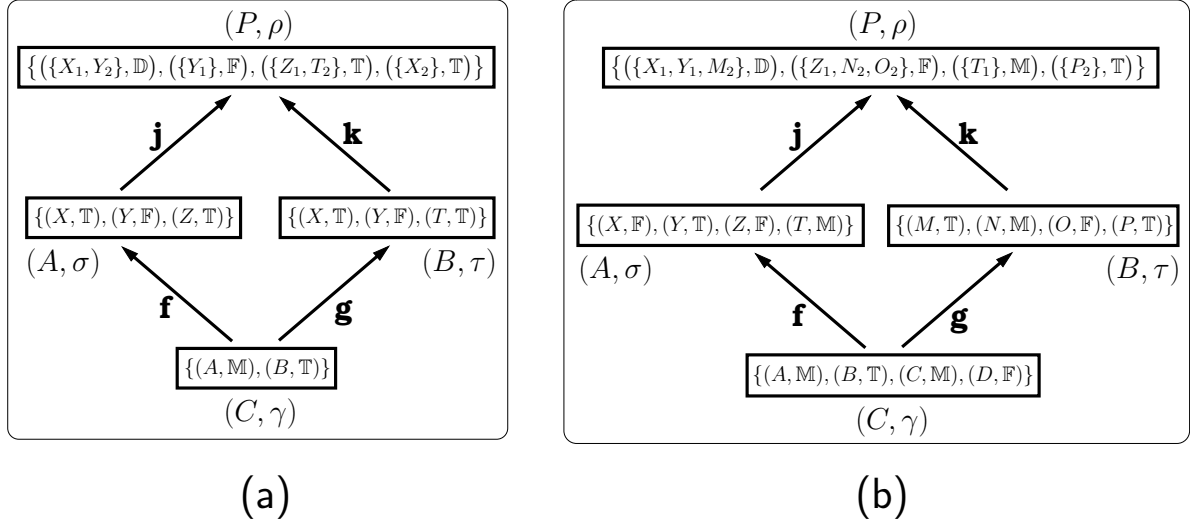
**Initial object:**  $\mathbf{0} = (\emptyset, \lambda)$  where  $\lambda : \emptyset \rightarrow Q$  is the empty function.

**Binary coproduct:** given objects  $X_1 = (S_1, \sigma_1)$  and  $X_2 = (S_2, \sigma_2)$ , a coproduct is  $X_1 + X_2 = (S_1 + S_2, \kappa)$  where  $S_1 + S_2$  is a **Set**-coproduct (disjoint union) of  $S_1$  and  $S_2$  with injections  $\iota_n : S_n \rightarrow S_1 + S_2$  for  $n = 1, 2$ ; and  $\kappa(\iota_n(s)) = \sigma_n(s)$  for  $s \in S_n$  and  $n = 1, 2$ .

**Coequalizer:** given objects  $X = (A, \sigma)$  and  $Y = (B, \tau)$  with parallel morphisms  $\mathbf{h}_1 : X \rightarrow Y$  and  $\mathbf{h}_2 : X \rightarrow Y$ , we first take the canonical **Set**-coequalizer of the carrier functions  $h_1 : A \rightarrow B$  and  $h_2 : A \rightarrow B$  to find a set  $C$  and a function  $q : B \rightarrow C$ . Thus,  $C$  is the quotient of  $B$  by the smallest equivalence relation  $\equiv$  on  $B$  such that  $h_1(a) \equiv h_2(a)$  for all  $a \in A$ ; and  $q$  is the function such that  $q(b) = [b]_{\equiv}$  for all  $b \in B$ . Then, we put  $Z = (C, \mu)$  where  $\mu([b]_{\equiv}) = \bigsqcup_Q \{\tau(b') \mid b' \equiv b\}$ . This lifts the function  $q : B \rightarrow C$  to a morphism  $\mathbf{q} : Y \rightarrow Z$ , which is a coequalizer of  $\mathbf{h}_1$  and  $\mathbf{h}_2$ .  $\blacksquare$

**Remark 4.6 (pushout construction in  $\mathbf{Fuzz}(Q)$ )** Since we want to avoid using the details of the above proof in the case-study presented in Chapter 5, we explain the procedure for computing fuzzy set pushouts separately: let  $Q$  be a complete lattice. For computing the pushout of a pair of  $\mathbf{Fuzz}(Q)$ -morphisms  $\mathbf{f} : (C, \gamma) \rightarrow (A, \sigma)$  and  $\mathbf{g} : (C, \gamma) \rightarrow (B, \tau)$ , first compute the canonical **Set**-pushout of the carrier functions  $f : C \rightarrow A$  and  $g : C \rightarrow B$  (as discussed in Example 3.26) to find a set  $P$  along with functions  $j : A \rightarrow P$  and  $k : B \rightarrow P$ . Then, compute a membership degree for every

<sup>1</sup>I gratefully acknowledge Andrzej Tarlecki for sketching the proof.


 Figure 4.2: Pushout examples in  $\mathbf{Fuzz}(\mathbf{A}_4)$ 

$p \in P$  by taking the supremum of the membership degrees of all those elements in  $(A, \sigma)$  and  $(B, \tau)$  that are mapped to  $p$ . This yields an object  $(P, \rho)$  and lifts  $j$  and  $k$  to  $\mathbf{Fuzz}(Q)$ -morphisms which together with  $(P, \rho)$ , constitute the pushout of  $\mathbf{f}$  and  $\mathbf{g}$  in  $\mathbf{Fuzz}(Q)$ .  $\square$

**Note** In all the figures appearing hereafter in this chapter, a fuzzy set  $(S, \sigma)$  is depicted as a set  $\{(s, \sigma(s)) \mid s \in S\}$ .  $\square$

**Example 4.7** Figure 4.2 illustrates two examples of pushout computation in  $\mathbf{Fuzz}(\mathbf{A}_4)$ . The carrier function for each  $\mathbf{Fuzz}(\mathbf{A}_4)$ -morphism  $\mathbf{f}$ ,  $\mathbf{g}$ ,  $\mathbf{j}$ ,  $\mathbf{k}$  in Figure 4.2(a) (resp. Figure 4.2(b)) is the same as the corresponding function in Figure 3.1(a) (resp. Figure 3.1(b)).  $\ast$

**Definition 4.8 (carrier functor)** The map that takes every  $\mathbf{Fuzz}(Q)$ -object  $(S, \sigma)$  to its carrier set  $S$  and every  $\mathbf{Fuzz}(Q)$ -morphism  $\mathbf{f} : (S, \sigma) \rightarrow (T, \tau)$  to its carrier function  $f : S \rightarrow T$  yields a functor  $K_Q : \mathbf{Fuzz}(Q) \rightarrow \mathbf{Set}$ , known as the *carrier functor*.  $\square$

**Proposition 4.9** *The carrier functor  $K_Q : \mathbf{Fuzz}(Q) \rightarrow \mathbf{Set}$  is finitely cocontinuous when  $Q$  is a complete lattice<sup>2</sup>.  $\diamond$*

**Proof** Based on the proof of Lemma 4.5, it is obvious that  $K_Q : \mathbf{Fuzz}(Q) \rightarrow \mathbf{Set}$  preserves the initial object, binary coproducts, and coequalizers. Finite cocontinuity of  $K_Q$  then follows from Lemma 3.38.  $\blacksquare$

### 4.3 Fuzzy Powersets

**Definition 4.10 (fuzzy powerset)** Let  $Q$  be a poset and let  $Z = (S, \sigma)$  be a  $\mathbf{Fuzz}(Q)$ -object. The powerset of  $Z$ , denoted  $\mathcal{P}(Z)$ , is the set of all  $(C, \xi) \in |\mathbf{Fuzz}(Q)|$  such that  $C \subseteq S$  and for every  $c \in C$ :  $\xi(c) \leq \sigma(c)$ .  $\square$

**Lemma 4.11** [Gog68, Gog74] *The powerset of any  $\mathbf{Fuzz}(Q)$ -object is a complete lattice when  $Q$  is.  $\diamond$*

**Proof (sketch)** [Gog68, Gog74] For an index set  $I$ , the supremum of an  $I$ -indexed family of  $\mathcal{P}(Z)$  elements  $\langle (S_i, \sigma_i) \rangle_{i \in I}$  is a fuzzy set  $(X, \theta)$  where  $X = \bigcup_{i \in I} S_i$  and  $\theta : X \rightarrow Q$  is a function such that for every  $x \in X$ :  $\theta(x) = \bigsqcup_Q \{ \sigma_i(x) \mid i \in I; x \in S_i \}$ . The infimum is computed dually.  $\blacksquare$

**Example 4.12** Suppose the truth-set is the lattice  $\mathbf{L}_2 = \{\perp, \top\}$  with  $\perp < \top$ . Then, the powerset of the  $\mathbf{Fuzz}(\mathbf{L}_2)$ -object  $Z = (\{a, b\}, \{a \mapsto \top, b \mapsto \top\})$  is the lattice shown in Figure 4.3.  $*$

**Remark 4.13** In Example 4.12, if we interpret  $\top$  as “fully in the fuzzy set” and interpret  $\perp$  as “not in the fuzzy set at all”, then there may exist more than one element of the powerset lattice corresponding to a particular *meaning*. In other words, the semantic

---

<sup>2</sup>The carrier functor is finitely cocontinuous even when  $Q$  is only a poset; however, a separate proof is required.

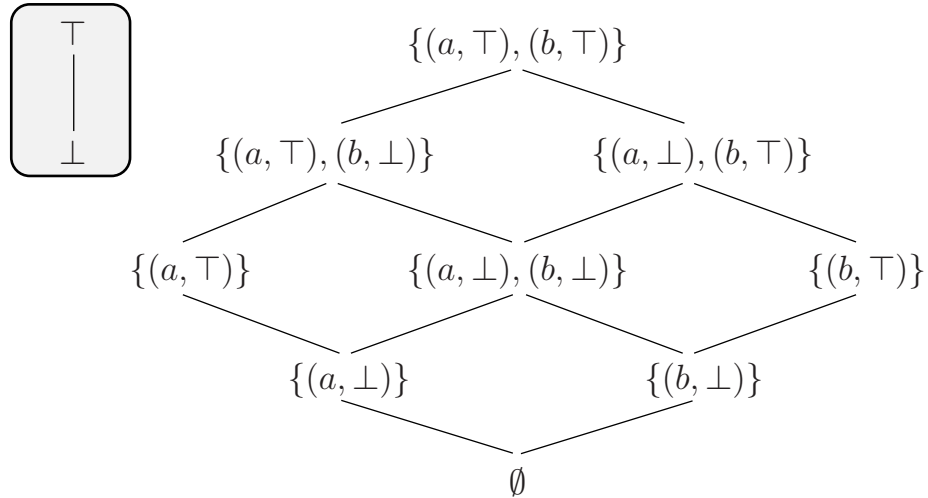


Figure 4.3: Powerset lattice example

interpretations of the powerset lattice elements are not mutually distinct when  $\top$  and  $\perp$  are interpreted in the way mentioned. For example, the elements  $(\emptyset, \lambda)$ ,  $(\{a\}, \{a \mapsto \perp\})$ ,  $(\{b\}, \{b \mapsto \perp\})$ , and  $(\{a, b\}, \{a \mapsto \perp, b \mapsto \perp\})$  in the powerset lattice convey the same meaning. *Notice that, in general, these fuzzy sets are not considered to be equal.*  $\square$

# Chapter 5

## Fuzzy Viewpoints

This chapter, which embodies the main contributions of the thesis, introduces a category-theoretic formalism for the representation of a family of graph-based viewpoints, called *fuzzy viewpoints*, that are capable of modeling incompleteness and inconsistency explicitly. Intuitively, a fuzzy viewpoint is a graph in which the details of nodes and edges are annotated with the elements of a lattice to specify the amount of knowledge available about them.

We show how an appropriate notion of morphism between fuzzy viewpoints gives rise to categories of fuzzy viewpoints. We then prove that these categories are (finitely) cocomplete and describe how merging a set of interconnected viewpoints can be done by constructing the colimiting viewpoint in an appropriate fuzzy viewpoint category. Colimits also provide a basis for defining a notion of *syntactic inconsistency* between a set of interconnected viewpoints. Finally, we illustrate an application of our proposed formalism through a case-study showing how fuzzy viewpoints can serve as a requirements elicitation tool in reactive systems.

## 5.1 $\mathfrak{F}$ View Categories

Let  $\mathcal{L}$  be a complete lattice of truth values and let  $U$  be an arbitrary but fixed set, hereafter called the *universe of atomic propositions*.

**Definition 5.1 (fuzzy viewpoint)** A  $(\mathcal{L}, U)$ -*fuzzy viewpoint*  $\mathcal{V}$  is a (directed) graph in which every edge  $e$  is labeled with a value from  $\mathcal{L}$  and every node  $n$  is labeled with an  $\mathcal{L}$ -valued set  $(U_n, \zeta_n)$  where  $U_n \subseteq U$  is the set of atomic propositions *visible* in node  $n$  and  $\zeta_n : U_n \rightarrow \mathcal{L}$  is a function assigning a value from  $\mathcal{L}$  to each element in  $U_n$ . By forgetting the labels of the nodes and edges in  $\mathcal{V}$ , we obtain a graph which is called the *carrier graph* of  $\mathcal{V}$ .  $\square$

It is clear from the above definition that the edges in a  $(\mathcal{L}, U)$ -fuzzy viewpoint form an  $\mathcal{L}$ -valued set. The question that remains is constructing the appropriate category that captures the structure of nodes along with their labels. This can be done in the following way: let  $\epsilon : U \rightarrow \mathcal{L}$  be the constant map  $\{x \mapsto \top \mid x \in U\}$  (notice that  $\top$  is known to exist by Lemma 2.21). Thus,  $(U, \epsilon)$  is a  $\mathbf{Fuzz}(\mathcal{L})$ -object. Now, by Lemma 4.11, we infer that “the powerset lattice of  $(U, \epsilon)$ ” is a complete lattice  $\mathcal{X}$ . The node-set of a fuzzy viewpoint  $\mathcal{V}$  along with the node labels can be described by an object of  $\mathbf{Fuzz}(\mathcal{X})$ . We can now define the notion of viewpoint morphism as followed:

**Definition 5.2 (fuzzy viewpoint morphism)** Let  $\mathcal{V}$  and  $\mathcal{V}'$  be  $(\mathcal{L}, U)$ -fuzzy viewpoints, and let  $K_{\mathcal{X}} : \mathbf{Fuzz}(\mathcal{X}) \rightarrow \mathbf{Set}$  and  $K_{\mathcal{L}} : \mathbf{Fuzz}(\mathcal{L}) \rightarrow \mathbf{Set}$  be the appropriate carrier functors. A *viewpoint morphism*  $h : \mathcal{V} \rightarrow \mathcal{V}'$  is a pair  $\langle h_{\mathbf{n}}, h_{\mathbf{e}} \rangle$  where  $h_{\mathbf{n}}$  is a  $\mathbf{Fuzz}(\mathcal{X})$ -morphism and  $h_{\mathbf{e}}$  is a  $\mathbf{Fuzz}(\mathcal{L})$ -morphism such that  $\langle K_{\mathcal{X}}(h_{\mathbf{n}}), K_{\mathcal{L}}(h_{\mathbf{e}}) \rangle$  is a graph homomorphism from the carrier graph of  $\mathcal{V}$  to the carrier graph of  $\mathcal{V}'$ .  $\square$

**Definition/Proposition 5.3 (fuzzy viewpoint category)** The above choice of objects and morphisms along with the obvious identities constitute a category of  $(\mathcal{L}, U)$ -fuzzy viewpoints, which we will hereafter denote  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$ .  $\square$

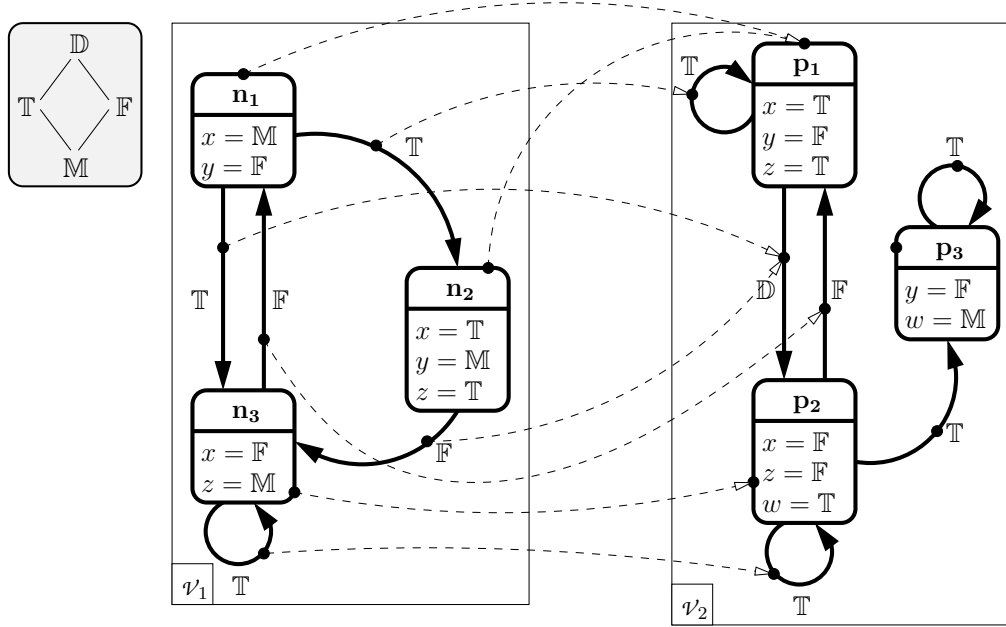


Figure 5.1: Example of fuzzy viewpoints

**Example 5.4** Let  $\mathcal{L} = \mathbf{A}_4$  and let  $U = \{x, y, z, w\}$ . Figure 5.1 shows two  $\mathfrak{F}\mathbf{View}(\mathbf{A}_4, U)$ -objects and a possible  $\mathfrak{F}\mathbf{View}(\mathbf{A}_4, U)$ -morphism. \*

**Note** In Figure 5.1 and all the figures in Section 5.3, the viewpoint edges have been left anonymous and only the truth values labeling them have been shown.  $\square$

**Remark 5.5** Notice that in Example 5.4, we can replace  $U$  in  $\mathfrak{F}\mathbf{View}(\mathbf{A}_4, U)$  with any finite or infinite  $U'$  such that  $\{x, y, z, w\} \subseteq U'$  and yet characterize the viewpoints and the viewpoint morphism in Figure 5.1 as  $\mathfrak{F}\mathbf{View}(\mathbf{A}_4, U')$  objects and morphisms.  $\square$

**Lemma 5.6** *The category  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$  is isomorphic to the comma category  $(K_{\mathcal{L}} \downarrow T \circ K_{\mathcal{X}})$  where  $K_{\mathcal{L}} : \mathbf{Fuzz}(\mathcal{L}) \rightarrow \mathbf{Set}$  and  $K_{\mathcal{X}} : \mathbf{Fuzz}(\mathcal{X}) \rightarrow \mathbf{Set}$  are the appropriate carrier functors and  $T : \mathbf{Set} \rightarrow \mathbf{Set}$  is the Cartesian product functor defined in Example 3.7.  $\diamond$*

**Proof** Obvious from Definitions 5.1 and 5.2.  $\blacksquare$

**Theorem 5.7**  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$  is finitely cocomplete for any complete lattice  $\mathcal{L}$  and any set  $U$ .  $\diamond$

**Proof** Since  $\mathcal{L}$  and  $\mathcal{X}$  are complete lattices, both  $\mathbf{Fuzz}(\mathcal{L})$  and  $\mathbf{Fuzz}(\mathcal{X})$  are finitely cocomplete by Lemma 4.5. Moreover, by Lemma 4.9, we know that the carrier functor  $K_{\mathcal{L}} : \mathbf{Fuzz}(\mathcal{L}) \rightarrow \mathbf{Set}$  is finitely cocontinuous when  $\mathcal{L}$  is a complete lattice. The theorem now follows from Lemma 3.45.  $\blacksquare$

**Remark 5.8** By Lemma 2.20, Theorem 5.7 implies that  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$  is finitely cocomplete for any *finite* lattice  $\mathcal{L}$  and any set  $U$ .  $\square$

**Remark 5.9** Although not elaborated here, Lemma 4.9 makes it possible to enrich fuzzy viewpoints with types and additional labels through well-known comma categorical techniques without violating the cocompleteness result achieved in Theorem 5.7. Another possible extension which we skip is using atomic propositions for labeling edges.  $\square$

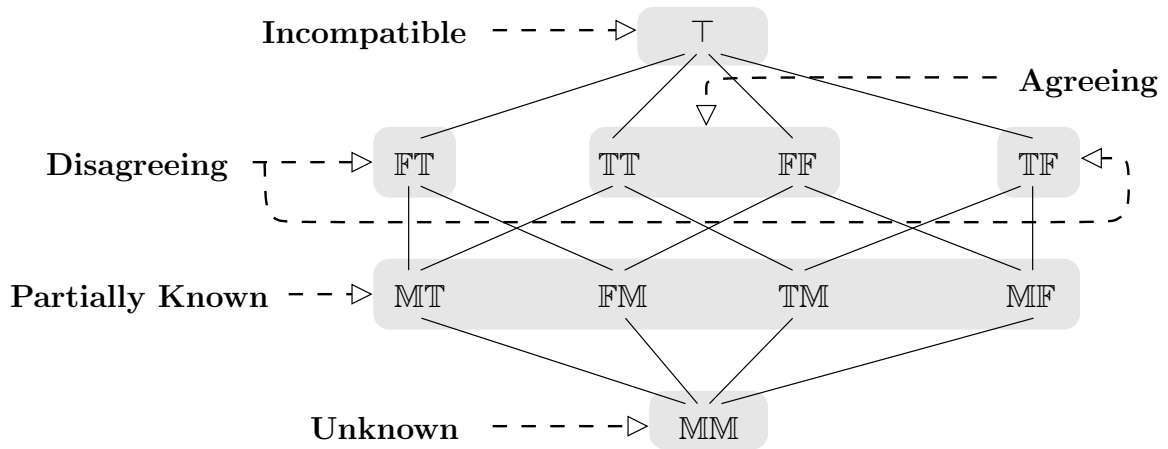
A limitation to  $\mathfrak{F}\mathbf{View}$  categories which is implicit in the notion of viewpoint morphism is that we assume the existence of a *unified* universe of atomic propositions (denoted  $U$ ). This implies that the name of a proposition suffices for uniquely identifying the concept represented by that proposition regardless of where the proposition appears; moreover, no two distinctly named propositions can represent a same concept. These restrictions pose no problems when the reference model made available at early stages of system development life-cycle explicitly specifies and explains the set of atomic propositions to be used by the stakeholders. When  $U$  is not given beforehand, the practical choice of  $U$  is unimportant as long as any finite set is isomorphic to some subset of  $U$ . In such cases, we assume  $U$  is the set of natural numbers, denoted  $\mathbb{N}$ . This would allow using as many propositions as needed; however, it is still up to the analysts to develop a shared vocabulary of atomic propositions during the elicitation phase and specify how the set of atomic propositions used in each viewpoint binds to this shared vocabulary. We will explain this further in Section 5.3.

## 5.2 Viewpoint Integration and Characterization of Inconsistency

According to the dogma discussed in Section 3.5, colimits can be employed for putting structures together. In our problem, the species of structure is  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$  for some fixed complete lattice  $\mathcal{L}$  and some fixed set  $U$ . A diagram in  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$  can be regarded as a “system” in which viewpoints are represented by  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$ -objects and viewpoint interconnections are represented by  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$ -morphisms. The cocompleteness result given in Theorem 5.7 states that the colimit exists for any finite diagram in  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$ ; therefore, we can integrate any finite set of viewpoints with known interconnections by constructing the colimit. This category-theoretic approach formalizes the *ad hoc* merge operation sketched in [EC01].

Viewpoint integration via colimits is abstract from how viewpoint interconnections are identified. In Section 5.3, we will illustrate a very simple case in which the interconnections between two viewpoints are identified by introducing a third viewpoint. The reader should also refer to [HEET99] where some useful patterns for viewpoint interconnection have been mentioned.

In the rest of this section, we will try to clarify how an appropriate choice of  $\mathcal{L}$  in  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$  enables us to model and detect inconsistencies. Our argument will also lead to a definition for syntactic inconsistency based on colimits. The simplest and maybe the most widely used lattice capable of modeling uncertainty and disagreement is Belnap’s four-valued lattice [Bel77] which was earlier referred to as  $\mathbf{A}_4$ . In  $\mathbf{A}_4$ , every value shows a possible “amount of knowledge” available about a concept. The value  $\mathbb{M}$  (i.e. MAYBE) denotes a lack of information,  $\mathbb{T}$  (i.e. TRUE) and  $\mathbb{F}$  (i.e. FALSE) denote the desired levels of knowledge, and  $\mathbb{D}$  (i.e. DISAGREEMENT) denotes a disagreement (or over-specification). Another interesting lattice is the ten-valued lattice  $\mathbf{A}_{10}$  shown in Figure 5.2. This lattice arises naturally in modeling a system with two stakeholders and

Figure 5.2: The lattice  $\mathbf{A}_{10}$ 

will be used for the case-study presented in Section 5.3.

In  $\mathbf{A}_{10}$ , the value  $\text{MM}$  indicates that no information is available. The values  $\text{TM}$  and  $\text{FM}$  (resp.  $\text{MT}$  and  $\text{MF}$ ) indicate that the *first* (resp. *second*) stakeholder has given a decisive **TRUE** or **FALSE** answer but no information has yet been provided by the other stakeholder. We also use these values when stakeholders are interviewed separately: for example, if we are interviewing the first (resp. second) stakeholder and (s)he says something is **TRUE**, the answer is recorded as  $\text{TM}$  (resp.  $\text{MT}$ ). The values  $\text{TT}$ ,  $\text{FF}$  indicate that both stakeholders agree on whether something is **TRUE** or **FALSE** while  $\text{TF}$  and  $\text{FT}$  indicate a disagreement between the stakeholders. The  $\top$  value arises when an incompatibility occurs. This value is not directly assigned during elicitation and only arises in colimit construction. We will explain this further in Section 5.3.

A nice property of both  $\mathbf{A}_4$  and  $\mathbf{A}_{10}$  is that once we remove the top element from either lattice, the rest of logical values can be reordered based on their “level of truth”. The “truth ordering” lattices corresponding to  $\mathbf{A}_4$  and  $\mathbf{A}_{10}$  have been shown in Figures 5.3(a) and 5.3(b), respectively. The existence of the truth orders is not by mere chance. In fact, the lower semi-lattice that results from removing the top element from  $\mathbf{A}_4$  (resp.  $\mathbf{A}_{10}$ ) together with the associated truth ordering in Figure 5.3(a) (resp. 5.3(b)) is an instance of a family of multivalued logics known as Kleene-like [Fit92] logics. In this thesis, we

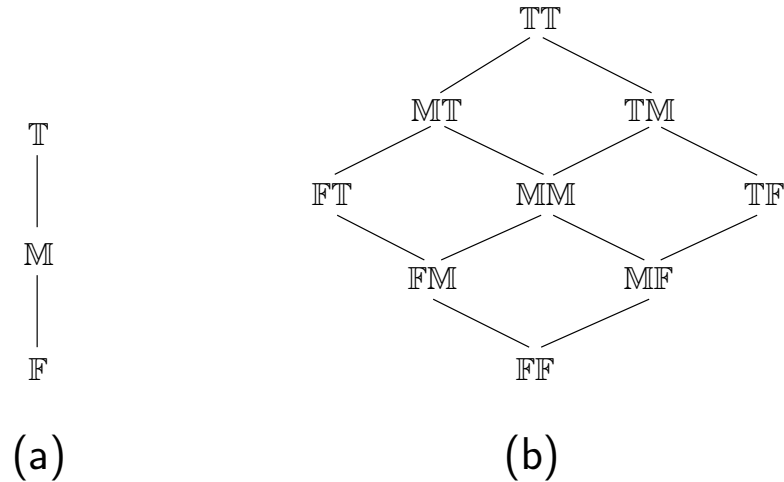


Figure 5.3: Truth ordering lattices

shall only appeal to the intuitive nature of such logics and therefore, omit the formal procedure for constructing them. The interested reader should refer to [Fit92] for further details. The procedure explained there yields  $\mathbf{A}_4$  (without  $\mathbb{D}$ ) and its corresponding truth order when the input to the procedure is the lattice  $\mathbf{B}_2 = \{\mathbb{F}, \mathbb{T}\}$  with  $\mathbb{F} < \mathbb{T}$ . The lattice  $\mathbf{A}_{10}$  (without  $\top$ ) and its corresponding truth order arise when the input is  $\mathbf{B}_2 \times \mathbf{B}_2$  (cf. e.g. [DP02] for the definition of product lattice). A suitable logic for a system with three stakeholders will arise when the input is  $\mathbf{B}_2 \times \mathbf{B}_2 \times \mathbf{B}_2$ , and so on.

The existence of such truth ordering lattices is an advantage when we want to interpret viewpoints according to certain semantics. For example, we may want to treat a viewpoint as a  $\lambda$ Kripke [CED01] structure and verify certain temporal properties in it through multivalued model-checking [CED01]. For such an application, we are naturally interested in measuring the “amount of truth” for the desired temporal properties that should hold.

In the framework we have proposed in this thesis, we are not concerned with any truth ordering lattices and the only reason for mentioning them here was to give a general idea of the links between syntax and semantics. All we take for granted here is the existence of a complete knowledge ordering lattice which we denote by  $\mathcal{L}$ . If we assume that the context of the system at hand can specify which elements of  $\mathcal{L}$  represent *consistent* amounts of

knowledge and which elements represent *inconsistent* amounts of knowledge, we can define syntactic inconsistency between a set of interconnected viewpoints as follows:

**Definition 5.10 (syntactic inconsistency)** A system of interconnected  $(\mathcal{L}, U)$ -fuzzy viewpoints is *syntactically inconsistent* if the colimit of the diagram corresponding to the system has some edge or proposition with an inconsistent truth value.  $\square$

In  $\mathbf{A}_{10}$ , for example, we may choose to designate  $\mathbb{T}\mathbb{T}$  and  $\mathbb{F}\mathbb{F}$  as consistent and the rest of the values as inconsistent. This is a reasonable choice when the system we are modeling mandates total agreement of both stakeholders on every aspect. If we are only interested in explicit conflicts and incompatibilities, we can relax this constraint and only designate  $\mathbb{T}\mathbb{F}$ ,  $\mathbb{F}\mathbb{T}$  and  $\top$  as inconsistent. Once we have a measure for how much inconsistency we want to *tolerate*, the colimit construction can also serve as a mechanism for determining when an inconsistency *amelioration phase* [EN96] is required. For example, if we want to model-check the result of viewpoint merge operation, we can *live with* all  $\mathbf{A}_{10}$  values except for incompatibilities ( $\top$ ).

### 5.3 Case-Study

In this section, we investigate a simple Requirements Engineering problem with the aim of showing how our proposed framework can be used in practice. Suppose Bob and Mary want to engineer a camera<sup>1</sup> from scratch with the help of a requirements analyst named Sam. Based on the early interviews, Sam has created a reference model for the operational behavior of the camera. This early reference model, shown in Figure 5.4, serves as a basis for elaborating Bob’s and Mary’s requirements using a (fictional) CASE tool called  $\mathfrak{F}$ Draw.

The primary role of Sam in this case-study is eliciting the requirements and identifying the relationships between Bob’s and Mary’s perspectives. This implies that the

---

<sup>1</sup>I am grateful to Colin Potts for suggesting the case-study.

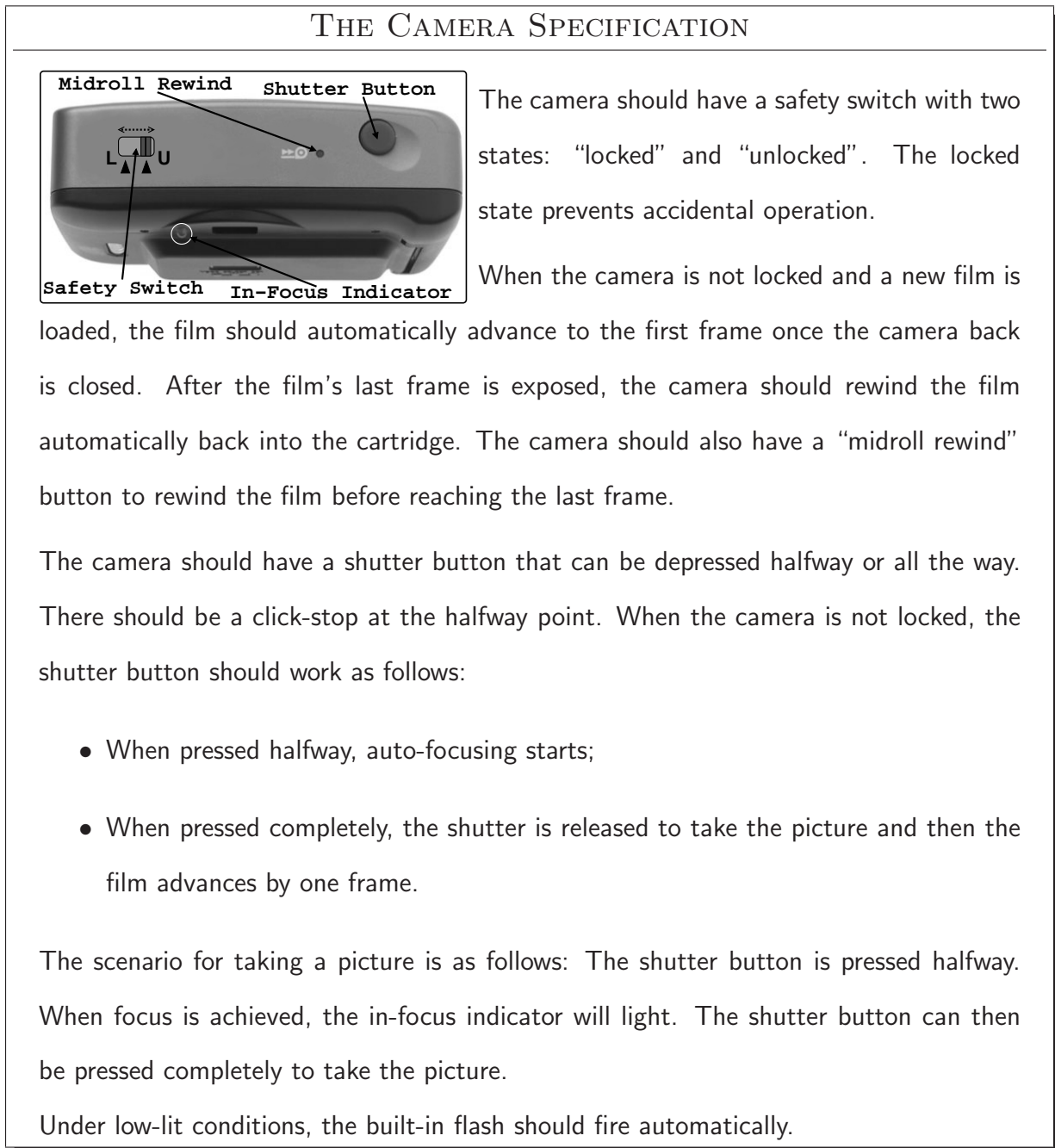


Figure 5.4: Camera’s early reference model

camera project has only two stakeholders, namely Bob and Mary; thus, the lattice  $\mathbf{A}_{10}$  (Figure 5.2) with Bob as the first and Mary as the second stakeholder is a suitable choice of knowledge ordering for this project. Since a vocabulary of atomic propositions has not yet been developed, the project is configured to use  $\mathfrak{F}\mathbf{View}(\mathbf{A}_{10}, \mathbb{N})$  where  $\mathbb{N}$  is the set of natural numbers. In practice, we do not use natural numbers as proposition names; rather, we assume that every proposition has a unique natural number assigned to it (we will not use any numbers throughout the case-study).

The set of propositions used by Bob and Mary must have no name clash and no two distinctly named propositions should represent the same thing. In order to enforce these restrictions, whenever either Bob or Mary needs a new proposition named  $p$  for some purpose, (s)he has Sam check the project's *data dictionary* to ensure that adding  $p$  causes no name clash and that no proposition (probably with a name different from  $p$ ) has already been defined for that particular purpose.

The viewpoints in the camera project are similar to  $\lambda$ Kripke structures [EC01, CED01]. In a viewpoint  $\mathcal{V}$ , each node denotes a *state* (*world*) and each edge denotes a *transition* labeled with the degree of certainty about the possibility of going from the source to the target state of the transition. Furthermore, there exists a unique transition  $t : i \rightarrow j$  from any  $\mathcal{V}$ -state  $i$  to any state  $\mathcal{V}$ -state  $j$ . This constraint is not automatically enforced by the structure of  $\mathfrak{F}\mathbf{View}(\mathbf{A}_{10}, \mathbb{N})$ , so  $\mathfrak{F}\mathbf{Draw}$  should explicitly be configured to do so.  $\mathfrak{F}\mathbf{Draw}$  disallows parallel transitions between states; but, for convenience, it allows transitions to be omitted and internally interprets the absent transitions as MM transitions. Notice that  $\mathfrak{F}\mathbf{Draw}$  could as well be configured to interpret absent transitions in Bob's (resp. Mary's) viewpoint as FM (resp. MF) transitions. This would be closer to how absent transitions are normally interpreted in classical models.

In this case-study, all propositions are treated as *global*. When a proposition  $x$  does not appear in a state  $s$  of a viewpoint  $\mathcal{V}$ , we assume that the owner of  $\mathcal{V}$  either is unaware of the existence of  $x$ , or (s)he does not care about the value of  $x$  in state  $s$ , at least from

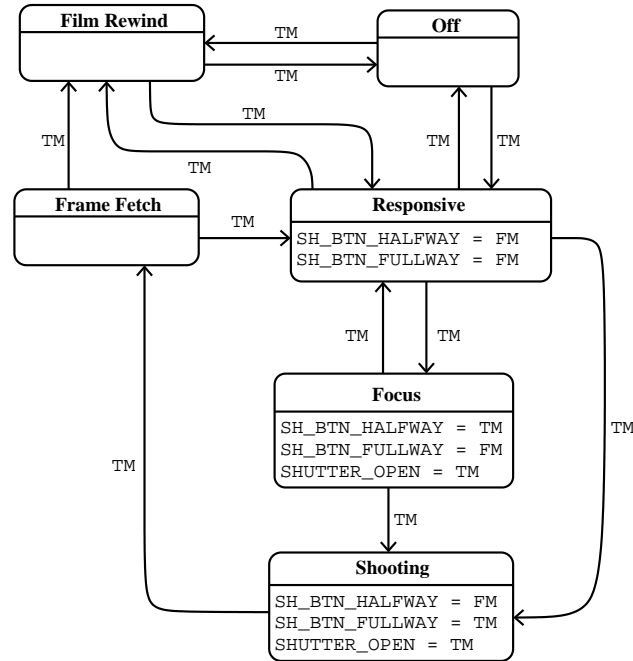


Figure 5.5: Bob's viewpoint

the particular perspective that  $\mathcal{V}$  reflects. In either case, an unspecified proposition in a state is interpreted as  $\text{MM}$ . This is analogous to the interpretation of absent transitions.

Figures 5.5 and 5.6 respectively show Bob's and Mary's viewpoints<sup>2</sup>. The important facts about each of Bob's and Mary's perspectives on the camera can be summarized as follows:

- **Bob**: he does not differentiate between different shooting modes; he believes pressing the shutter button fullway is allowed even before achieving focus; he (mistakenly) believes the shutter is open during focusing; he believes midroll film rewind can occur even when the camera is locked; he does not model the cartridge loading procedure.
- **Mary**: she distinguishes between different shooting modes; she believes pressing the shutter button fullway is inhibited until focus is achieved; she does not see the scenario in which the shutter button is pressed halfway but is released without

<sup>2</sup>In both figures,  $SH\_BTN$  is an abbreviation for  $SHUTTER\_BUTTON$ .

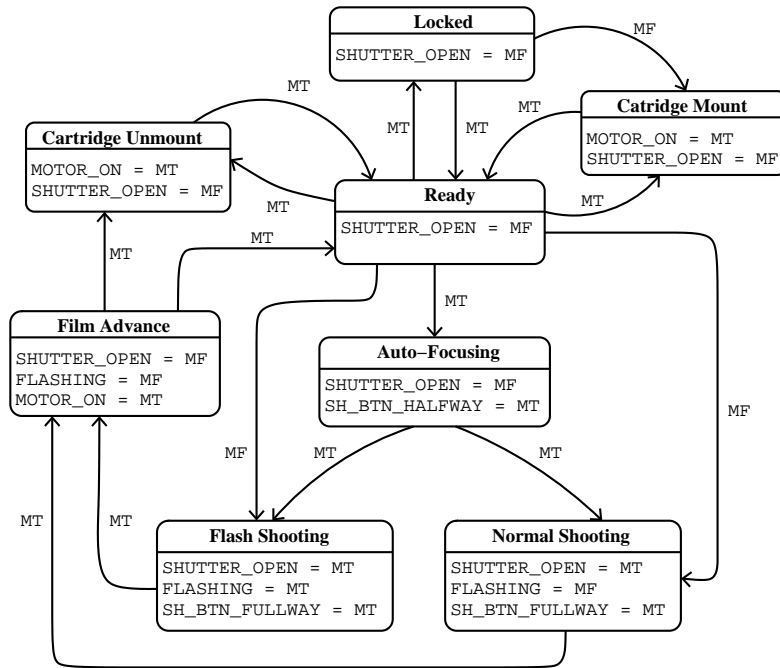


Figure 5.6: Mary's viewpoint

taking a picture; she knows that the camera has a motor that rolls the film; she models the cartridge loading procedure and also emphasizes that mounting a new cartridge can not take place when the camera is locked.

Sam now identifies the interconnection between the two viewpoints. He first identifies the state interconnections. This has been shown in Figure 5.7. Notice that Sam uses his own set of names for the states. Once the state interconnections are specified, the transition interconnections can be identified automatically. This is because there is a unique transition from any state  $i$  to any state  $j$  of every viewpoint  $\mathcal{V}$ . Thus, if a viewpoint morphism  $h : \mathcal{V} \rightarrow \mathcal{V}'$  maps states  $i$  and  $j$  in  $\mathcal{V}$  to states  $i'$  and  $j'$  in  $\mathcal{V}'$  respectively, then  $h$  must map the unique transition from  $i$  to  $j$  in  $\mathcal{V}$  to the unique transition from  $i'$  to  $j'$  in  $\mathcal{V}'$ .

Figure 5.8 shows Sam's viewpoint. This viewpoint has been computed by  $\mathfrak{F}$ Draw directly from the state interconnections. For clarity, we have chosen to show those transitions in Sam's viewpoint that are mapped to non-MIM transitions in Bob's and Mary's

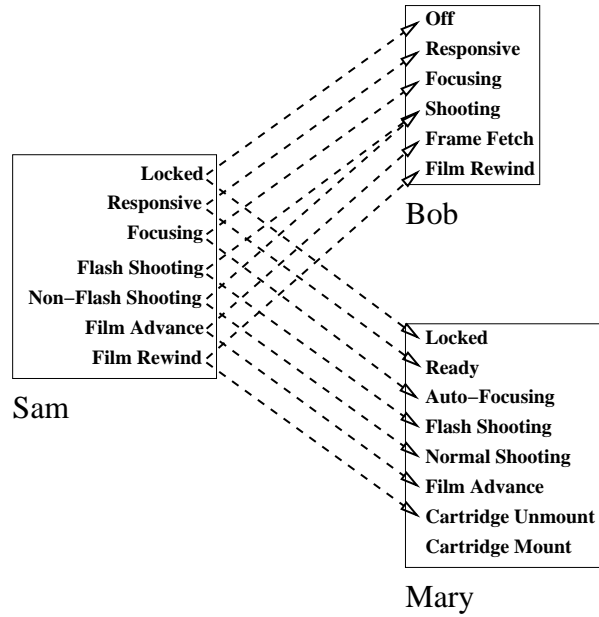


Figure 5.7: State interconnections

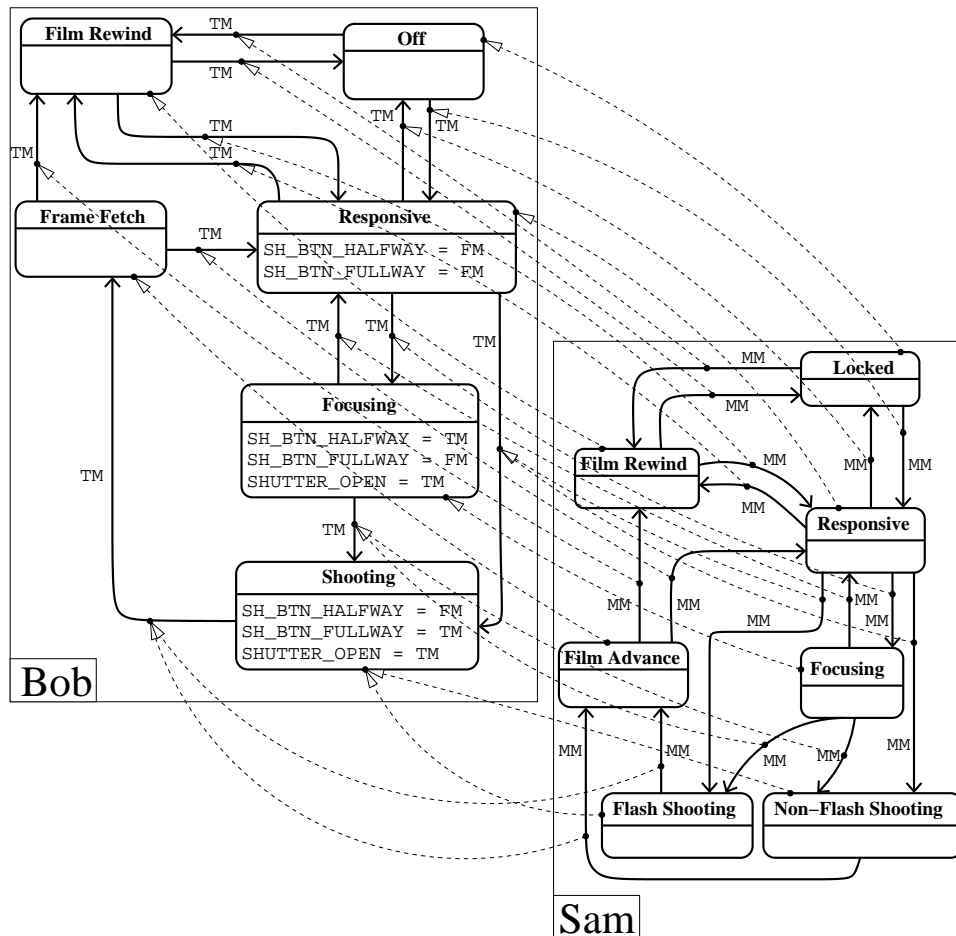


Figure 5.8: Interconnecting the viewpoints

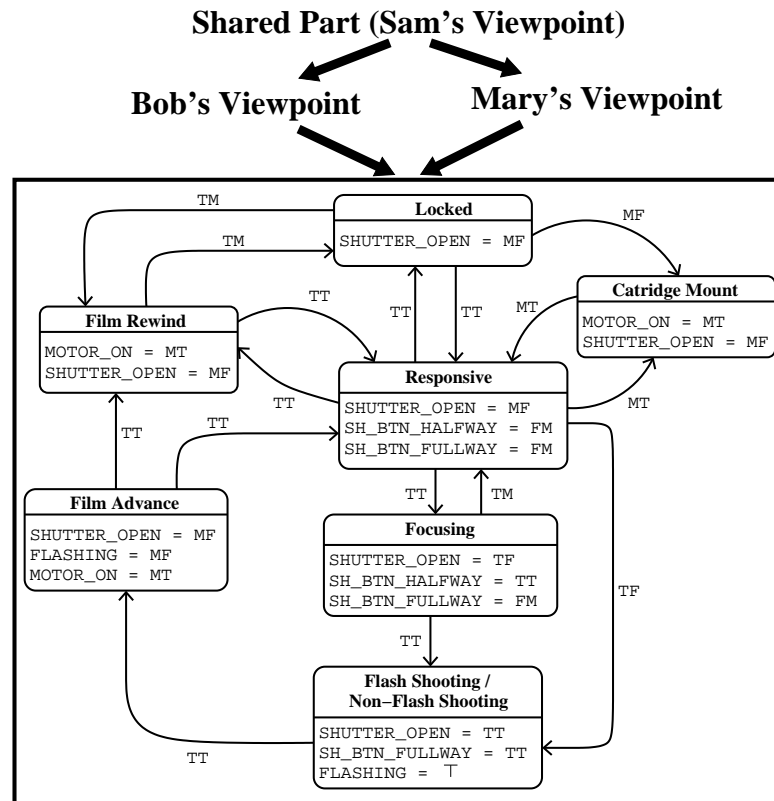


Figure 5.9: Merging the viewpoints

viewpoints. The figure also sketches the viewpoint morphism from Sam's to Bob's viewpoint. The morphism from Sam's to Mary's viewpoint, which has been omitted to avoid undue complexity in the figure, is analogous.

Notice that Sam does not engage in determining the truth or falsehood of transitions and propositions; therefore, his viewpoint solely reflects the relationships between Bob's and Mary's viewpoints. As a result, all transitions in Sam's viewpoint are labeled by  $\text{MM}$ . Sam can also forget about propositions all together when identifying the interconnections and use  $\emptyset$  as the set of visible propositions in all states of his viewpoint.

$\mathfrak{F}$ Draw now computes the pushout of Bob's and Mary's viewpoints with respect to the shared part identified by Sam. The result of merge operation (excluding  $\text{MM}$  transitions) is shown in Figure 5.9. For assigning names to the states in the pushout, we have assumed that Sam's choice of state names overrides those of Bob and Mary wherever possible.

Here, the only state not mentioned by Sam is **Cartridge Mount** in Mary’s viewpoint, so Sam’s state names override all state names except for **Cartridge Mount**. This assumption is of no theoretical importance; however, it can be used to devise a built-in solution to the name mapping problem.

The pushout in Figure 5.9 clearly reflects the result of merging Bob’s and Mary’s viewpoints. Assuming  $\text{TF}$ ,  $\text{FT}$ , and  $\top$  are the only inconsistent values, there are three cases of syntactic inconsistency in the pushout. The  $\text{TF}$  values for `SHUTTER_OPEN` in **Focusing** and the transition from **Responsive** to **Flash Shooting/Non-Flash Shooting** respectively show disagreeing perspectives on the shutter’s behavior during focusing operation and on picture taking without achieving focus. The other inconsistency is the  $\top$  value for `FLASHING` in **Flash Shooting/Non-Flash Shooting** state.

Intuitively, the  $\top$  value in  $\mathbf{A}_{10}$  arises when a stakeholder wants a proposition  $x$  to hold in a state  $s$  of a viewpoint  $\nu$  and also wants  $x$  not to hold in a state  $s'$  of a viewpoint  $\nu'$  but the interconnections are in such a way that  $s$  and  $s'$  are mapped to the same state in the colimit. In our example, the simplest reason for getting `FLASHING` =  $\top$  is that Sam has made a mistake in finding the (state) interconnections. For example, Bob might have meant **Non-Flash Shooting** by saying **Shooting**; or maybe, at the time Sam identified the interconnections, Mary had not yet used `FLASHING` to distinguish between **Flash Shooting** and **Non-Flash Shooting** and therefore, Sam thought two distinct states for shooting would be redundant. It is also possible that the incompatible information is indeed due to the incompatible perspectives of Bob and Mary on the shooting behavior: for example, Mary may believe the flash clearly distinguishes between two modes of shooting while Bob believes the flash is a separate autonomous peripheral and deliberately refuses to differentiate between the states that result from the combination of the shooting and the flashing behaviors.

The final issue to note regarding this case-study is that although the result of merge operation in our particular scenario has a unique transition from any state  $i$  to any

state  $j$ , this is not necessarily the case in general. Based on how the edge-map component of every viewpoint morphism is identified, it can be verified that parallel transitions between states never arise in the colimit; however, the colimiting object may have some missing transitions. For example, if Bob saw a state  $\mathbf{X}$  that Mary did not see, the pushout would have had no transition from  $\mathbf{X}$  to **Cartridge Mount** and vice versa. This is natural, because  $\mathbf{X}$  and **Cartridge Mount** would have never appeared together in a same *elicitation context*. This phenomenon can be taken advantage of for specifying the activities that should be performed in the next round of elicitation to fill in the gaps. We may as well choose to interpret missing transitions in colimits as  $\mathbb{M}\mathbb{M}$  transitions.

# Chapter 6

## $\mathfrak{F}$ Graph Categories

Taking advantage of the same techniques used for defining  $\mathfrak{F}$ View categories in the previous chapter, this chapter introduces a more general family of categories called  $\mathfrak{F}$ Graph categories that may find applications in other contexts. In order to make this chapter reasonably independent from the previous, we provide some separate examples for illustrating the structure of  $\mathfrak{F}$ Graph categories. These examples are, of course, not directly related to the main theme of the thesis. The cocompleteness result presented in this chapter might also be of interest for developing graph transformation systems based on the double-pushout approach [CMR<sup>+</sup>97]; however, we have not still explored this application.

**Definition 6.1** ( *$\mathfrak{F}$ Graph category*) Let  $I$  and  $J$  be a pair of posets. The category  $\mathfrak{F}\mathbf{Graph}(I, J)$  is defined as the comma category  $(K_J \downarrow T \circ K_I)$  where  $K_I : \mathbf{Fuzz}(I) \rightarrow \mathbf{Set}$  and  $K_J : \mathbf{Fuzz}(J) \rightarrow \mathbf{Set}$  are the appropriate carrier functors and  $T : \mathbf{Set} \rightarrow \mathbf{Set}$  is the Cartesian product functor as defined in Example 3.7.  $\square$

**Theorem 6.2**  *$\mathfrak{F}\mathbf{Graph}(I, J)$  is finitely cocomplete when  $I$  and  $J$  are complete lattices.*  $\diamond$

**Proof** By Lemma 4.5, both  $\mathbf{Fuzz}(I)$  and  $\mathbf{Fuzz}(J)$  are finitely cocomplete, and by Lemma 4.9,  $K_J$  is finitely cocontinuous. Finite cocompleteness of  $\mathfrak{F}\mathbf{Graph}(I, J)$  then follows from Lemma 3.45.  $\blacksquare$

**Definition 6.3** There is a functor  $W : \mathfrak{F}\mathbf{Graph}(I, J) \rightarrow \mathbf{Graph}$ , called the *carrier graph functor*, that maps every object  $((E, E \rightarrow J), f : E \rightarrow N \times N, (N, N \rightarrow I))$  to  $(E, f : E \rightarrow N \times N, N)$  and every morphism  $(s_{\text{edge}}, t_{\text{node}})$  to  $(K_J(s_{\text{edge}}), K_I(t_{\text{node}}))$ .  $\square$

**Remark 6.4** For a complete lattice  $\mathcal{L}$  and a set  $U$ , the category  $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$ , as defined in Chapter 5, is isomorphic to  $\mathfrak{F}\mathbf{Graph}(\mathcal{P}_{\mathcal{L}}((U, \epsilon)), \mathcal{L})$  where  $\epsilon : U \rightarrow \mathcal{L}$  is the constant map  $\{x \mapsto \top \mid x \in U\}$  and  $\mathcal{P}_{\mathcal{L}}((U, \epsilon))$  is the fuzzy powerset of  $(U, \epsilon) \in |\mathbf{Fuzz}(\mathcal{L})|$ . Notice that  $\mathcal{L}$  has a top ( $\top$ ) element by Lemma 2.21.  $\square$

**Example 6.5** Let  $\mathcal{L}$  be a linear four-point lattice:  $\{\text{White}, \text{Light Grey}, \text{Dark Grey}, \text{Black}\}$  ordered by increasing intensity of the black color, and let  $\mathbf{1}$  be the one-point lattice. Figure 6.1 illustrates two  $\mathfrak{F}\mathbf{Graph}(\mathcal{L}, \mathbf{1})$ -objects along with a  $\mathfrak{F}\mathbf{Graph}(\mathcal{L}, \mathbf{1})$ -morphism.\*

**Example 6.6** Figure 6.2 illustrates an example pushout computation in  $\mathfrak{F}\mathbf{Graph}(\mathcal{L}, \mathbf{1})$ . The morphisms corresponding to  $f, g, j$ , and  $k$  in the pushout square (Definition 3.24) have been marked in the example.  $*$

**Example 6.7** Figure 6.3 illustrates an example pushout computation in  $\mathfrak{F}\mathbf{Graph}(2^S, \mathbf{A}_4)$  where  $S = \{p, q, r\}$  and  $\mathbf{A}_4$  is Belnap's four-valued lattice [Bel77]. The nodes and edges of the carrier graphs of all  $\mathfrak{F}\mathbf{Graph}(2^S, \mathbf{A}_4)$ -objects shown in the figure have been left anonymous and only the lattice values labeling them have been shown. Notice that we can replace  $S$  with any finite or infinite  $S'$  such that  $\{p, q, r\} \subseteq S'$  and yet characterize the objects and morphisms in Figure 6.3 as  $\mathfrak{F}\mathbf{Graph}(2^{S'}, \mathbf{A}_4)$  objects and morphisms.\*

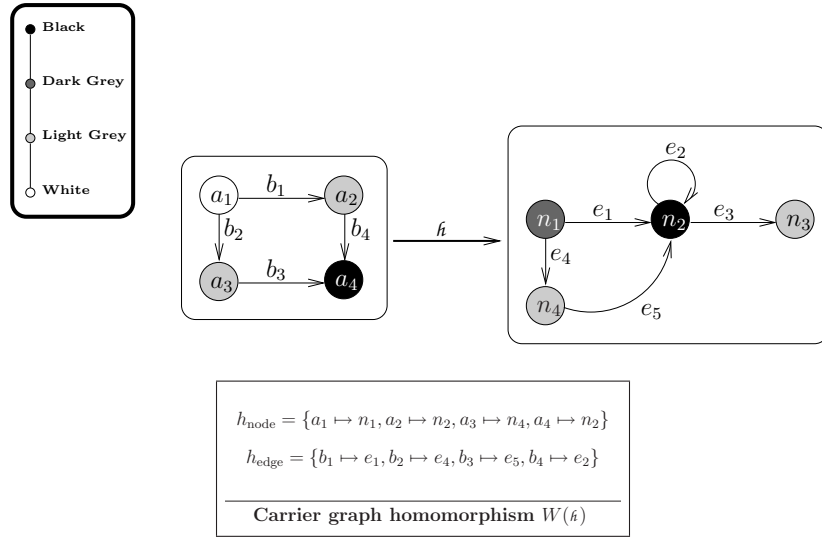


Figure 6.1: Example of  $\mathfrak{F}\mathbf{Graph}(\mathcal{L}, 1)$  objects and morphisms

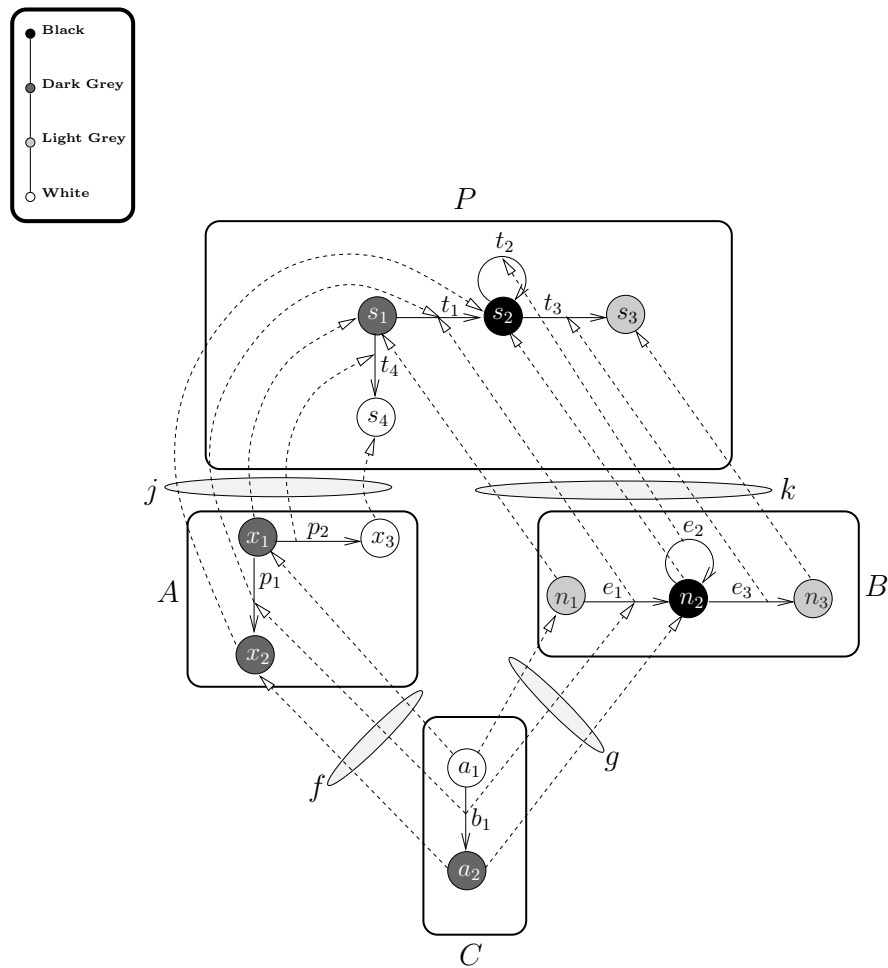


Figure 6.2: Pushout computation in  $\mathfrak{F}\mathbf{Graph}(\mathcal{L}, 1)$

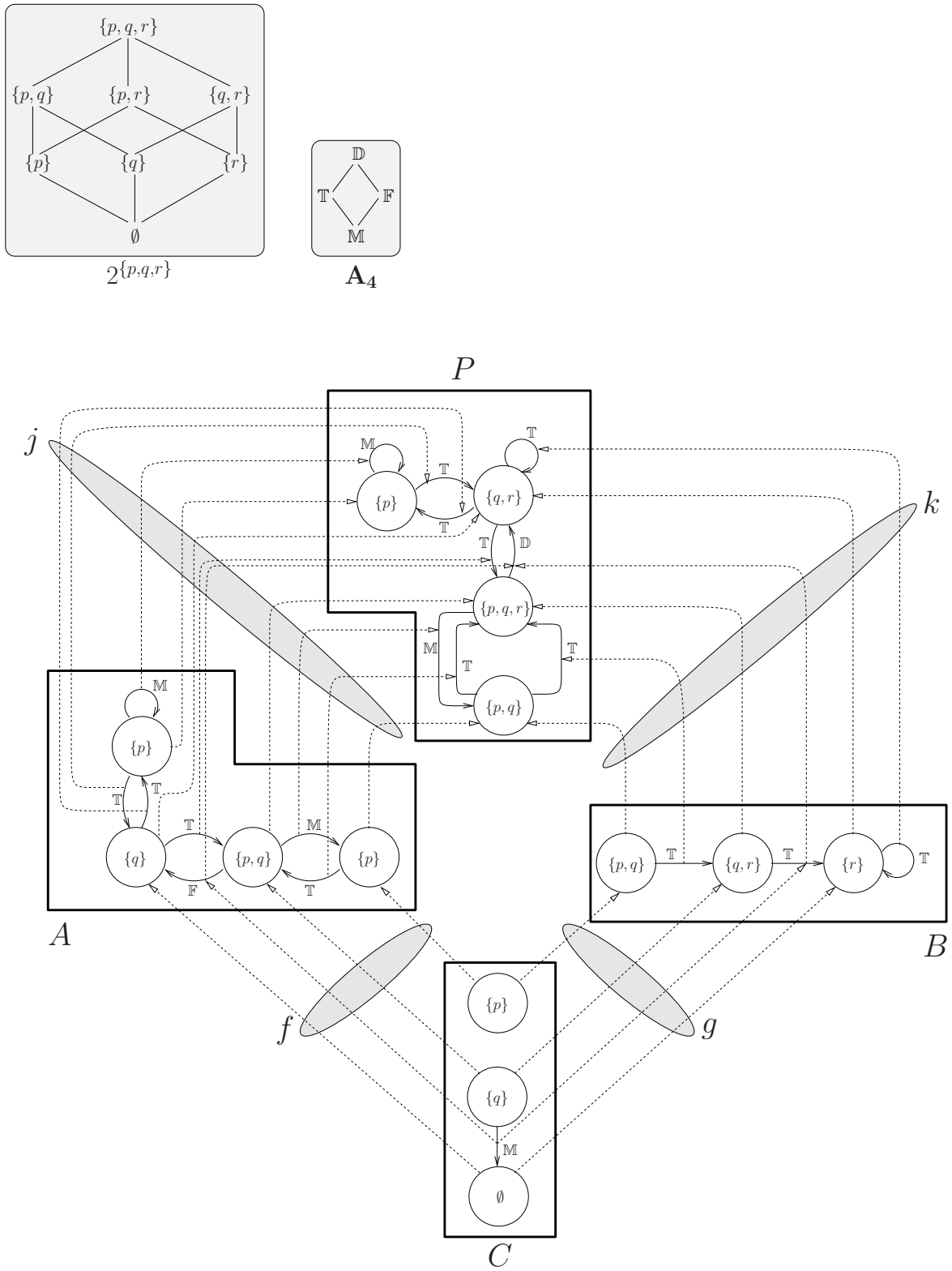


Figure 6.3: Pushout computation in  $\mathfrak{F}\text{Graph}(2^{\{p,q,r\}}, \mathbf{A}_4)$

# Chapter 7

## Conclusions

Viewpoints have proven to be effective tools for the management and analysis of incomplete and inconsistent models gathered from multiple sources. In the classical viewpoint approach, incompleteness and inconsistency within a viewpoint cannot be modeled explicitly. Hence, classical inconsistency analysis techniques typically require the translation of viewpoints into a rich intermediate formalism before being able to detect or resolve any inconsistencies. This translation usually discards structure and blurs the distinction between the syntactic and the semantic aspects of viewpoints, thus making it too difficult to devise a semantics-independent inconsistency detection mechanism based on structural mappings between viewpoints, and seriously limiting the types of analysis that can be performed on viewpoints.

Using elementary category-theoretic techniques, this thesis proposed a mechanism for explicitly representing incompleteness and inconsistency in the syntactic structure of viewpoints and described a new approach to merging and analyzing inconsistent viewpoints based on the structural interconnections between them. Our arguments also led to a definition for a notion of syntactic inconsistency between viewpoints.

The major advantages of our framework can be summarized as follows:

- The framework is independent of any particular choice of viewpoint semantics and can therefore be applied to any of the large number of graph-based notations commonly used in Software Engineering.
- Inconsistency can be parameterized through lattices. This makes the framework suitable for different types of analysis. For example, the case-study conducted in Chapter 5 showed how an appropriate choice of lattice can make a distinction between disagreements and incompatibilities. This is useful if we want to tolerate the former in our models, but not the latter.
- The use of category theory for conceptualizing the merge process naturally results in the explicit identification of interconnections between viewpoints prior to the merge operation rather than relying on naming conventions to give the desired unification.

The work reported here can be carried forward in many ways. Our future work includes adding support for hierarchical structures like Statecharts [Har87]; and capturing the change of the vocabulary of atomic propositions as well as the truth-set through viewpoint morphisms. Exploring possible ways of taking semantics-related details into account is yet another part of our future work that will involve several case-studies. We are also looking at possible ways for using the results presented in Chapter 6 for developing graph transformation systems based on the double-pushout approach [CMR<sup>+</sup>97].

# Bibliography

- [Bel77] N.D. Belnap. A useful four-valued logic. In J.M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Values Logic*, pages 5–37. Reidel Publishing, 1977.
- [Bir79] G. Birkhoff. *Lattice Theory*. American Mathematical Society, third edition, 1979.
- [Bor94] F. Borceux. *Handbook of Categorical Algebra*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994.
- [BW84] M. Barr and C. Wells. *Toposes, Triples and Theories*. Grundlehren Math. Wiss. Springer-Verlag, 1984. An updated version of the book is available at <ftp://ftp.math.mcgill.ca/pub/barr>.
- [BW99] M. Barr and C. Wells. *Category Theory for Computing Science*. Les Publications CRM, Montréal, third edition, 1999.
- [CED01] M. Chechik, S.M. Easterbrook, and B. Devereux. Model checking with multi-valued temporal logics. In *Proceedings of the International Symposium on Multivalued Logics*, pages 187–192, 2001.
- [CMR<sup>+</sup>97] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approach to graph transformation, part I: Basic concepts and dou-

- ble pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, 1997.
- [DP02] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, second edition, 2002.
- [DS96] P. Darke and Graeme Shanks. Stakeholder viewpoints in requirements definition: A framework for understanding viewpoint development approaches. *Requirements Engineering*, 1(2):88–105, 1996.
- [DvF93] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.
- [Eas93] S.M. Easterbrook. Domain modeling with hierarchies of alternative viewpoints. In *Proceedings of the 1st International Symposium on Requirements Engineering*, pages 65–72, 1993.
- [EC01] S.M. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 411–420, 2001.
- [EN96] S.M. Easterbrook and B.A. Nuseibeh. Using viewpoints for inconsistency management. *Software Engineering Journal*, 11:31–43, 1996.
- [EP72] H. Ehrig and M. Pfender. *Kategorien und Automaten*. de Gruyter Lehrbuch. Walter de Gruyter, 1972.
- [ET96] H. Ehrig and G. Taentzer. Computing by graph transformation, a survey and annotated bibliography. *Bulletin of the European Association for Theoretical Computer Science*, 59:182–226, 1996.

- [FGH<sup>+</sup>94] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. *IEEE Transactions on Software Engineering*, 20:569–578, 1994.
- [Fit92] M. Fitting. Kleene’s logic, generalized. *Journal of Logic and Computation*, 1:797–810, 1992.
- [FKN<sup>+</sup>92] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2:31–57, 1992.
- [GB84] J.A. Goguen and R.M. Burstall. Some fundamental algebraic tools for the semantics of computation, part I: Comma categories, colimits, signatures and theories. *Theoretical Computer Science*, 31:175–209, 1984.
- [Gog68] J.A. Goguen. *Categories of Fuzzy Sets: Applications of Non-Cantorian Set Theory*. PhD thesis, University of California, Berkeley, 1968.
- [Gog74] J.A. Goguen. Concept representation in natural and artificial languages: Axioms, extensions and applications for fuzzy sets. *International Journal of Man-Machine Studies*, 6:513–561, 1974.
- [Gog91] J.A. Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1:49–67, 1991.
- [GTW87] J.A. Goguen, J.W. Thatcher, and E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R.T. Yeh, editor, *Current Trends in Programming Methodology*, volume 4, chapter 5. Prentice-Hall, 1987.

- [Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [Hec98] R. Heckel. *Open Graph Transformation Systems: A New Approach to the Compositional Modeling of Concurrent and Reactive Systems*. PhD thesis, Technical University of Berlin, 1998.
- [HEET99] R. Heckel, G. Engels, H. Ehrig, and G. Taentzer. A view-based approach to system modeling based on open graph transformation systems. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2. World Scientific Publishing, 1999.
- [HR99] U. Höhle and S.E. Rodabaugh, editors. *Mathematics of Fuzzy Sets: Logic, Topology, and Measure Theory*. Kluwer Academic Publishers, 1999.
- [KS96] G. Kotonya and I. Sommerville. Requirements engineering with viewpoints. *Software Engineering Journal*, 11(1):2–18, January 1996.
- [Mac71] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, 1971.
- [MN00] J.N. Mordeson and P.S. Nair. *Fuzzy Graphs and Fuzzy Hypergraphs*. Physica-Verlag, 2000.
- [Mul79] G. Mullery. CORE: A method for controlled requirements specification. In *Proceedings of the 4th International Conference on Software Engineering*, 1979.
- [NCEF02] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a consistency checking and smart link generation service. *ACM Transactions on Internet Technology*, 2(2):151–185, 2002.

- [RB88] D.E. Rydeheard and R.M. Burstall. *Computational Category Theory*. Prentice Hall, 1988.
- [ST] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specifications and Formal Program Development*. Cambridge University Press. To appear.
- [ST99] D. Sannella and A. Tarlecki. Algebraic preliminaries. In E. Astesiano, H.J. Kreowski, and B. Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, chapter 2. Springer-Verlag, 1999.
- [Tar86] A. Tarlecki. Bits and pieces of the theory of institutions. In S. Abramsky, A. Poigné, and D. Rydeheard, editors, *Summer Workshop on Category Theory and Computer Programming*, pages 334–363. Springer-Verlag, 1986.

# Appendix A

## Proofs for Chapter 3

In this appendix, we give the proofs for some of the major results asserted in Chapter 3.

**Proof of Lemma 3.32** [RB88] Let  $D : G \rightarrow U(\mathcal{C})$  be a finite *discrete* diagram in a category  $\mathcal{C}$  and let  $N$  denote the set of  $G$ 's nodes. If  $N$  is empty, then the coproduct is the initial object; otherwise, there exists some  $n \in N$ . Let  $D'$  be the same diagram as  $D$  with node  $n$  removed from its shape graph. Inductively, construct the coproduct  $\langle \delta_i : D'(i) \rightarrow A \rangle_{i \in N \setminus \{n\}}$ ; and further let  $B$  together with  $\iota_A : A \rightarrow B$  and  $\iota_{D(n)} : D(n) \rightarrow B$  be a binary coproduct of  $A$  and  $D(n)$ .

Construct a cocone  $\langle \gamma_i : D(i) \rightarrow B \rangle_{i \in N}$  by letting  $\gamma_n = \iota_{D(n)}$  and  $\gamma_i = \iota_A \circ \delta_i$  for  $i \neq n$ . We claim that  $\langle \gamma_i : D(i) \rightarrow B \rangle_{i \in N}$  is a coproduct of  $D$ : suppose  $\langle \gamma'_i : D(i) \rightarrow B' \rangle_{i \in N}$  is a cocone over  $D$ . Then,  $\langle \gamma'_i \rangle_{i \in N \setminus \{n\}}$  is a cocone over  $D'$ . Since  $\langle \delta_i \rangle_{i \in N \setminus \{n\}}$  is a colimiting cocone, there is a unique morphism  $h : A \rightarrow B'$  such that the following diagram commutes for all  $i \in N \setminus \{n\}$ :

$$\begin{array}{ccc} A & \overset{h}{\dashrightarrow} & B' \\ \delta_i \swarrow & & \nearrow \gamma'_i \\ & D(i) & \end{array}$$

Now, by the definition of binary coproduct, there is a unique morphism  $v : B \rightarrow B'$  such that the following diagram commutes:

$$\begin{array}{ccccc}
 & & B' & & \\
 & \nearrow h & \uparrow v & \nwarrow \gamma'_n & \\
 A & \xrightarrow{\iota_A} & B & \xleftarrow{\iota_{D(n)}} & D(n)
 \end{array}$$

It follows from the previous two diagrams that  $v : B \rightarrow B'$  makes the following diagram commute for all  $i \in N$ :

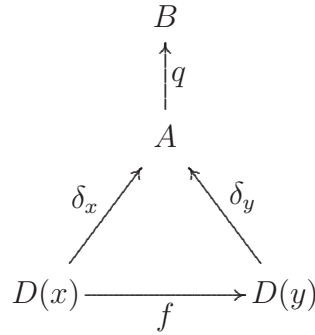
$$\begin{array}{ccc}
 B & \overset{v}{\dashrightarrow} & B' \\
 \nearrow \gamma_i & & \nwarrow \gamma'_i \\
 & D(i) &
 \end{array}$$

The uniqueness conditions on  $h$  and  $v$  ensure that  $v$  is the only such morphism. ■

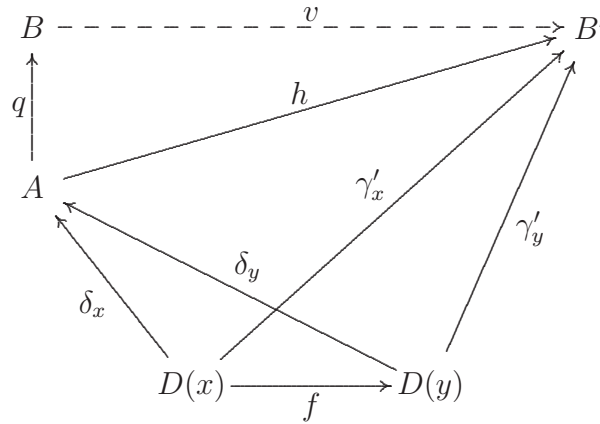
**Proof of Theorem 3.33** [RB88] Let  $D : G \rightarrow U(\mathcal{C})$  be a finite diagram in a category  $\mathcal{C}$ . If  $D$  is discrete then the colimit of  $D$  is the coproduct constructed in Lemma 3.32; otherwise, suppose  $e : x \rightarrow y$  is an edge in  $G$  with associated morphism  $f : D(x) \rightarrow D(y)$  in  $\mathcal{C}$ . Let  $D'$  be the same diagram as  $D$  with edge  $e$  removed from its shape graph. Inductively, construct the colimiting cocone  $\langle \delta_i : D'(i) \rightarrow A \rangle_{i \in N}$  on  $D'$  where  $N$  is the set of nodes in  $D$ 's shape graph (notice that the shape graphs of  $D$  and  $D'$  have the same set of nodes).

Now, consider the parallel pair of morphisms  $\delta_x : D(x) \rightarrow A$  and  $\delta_y \circ f : D(x) \rightarrow A$

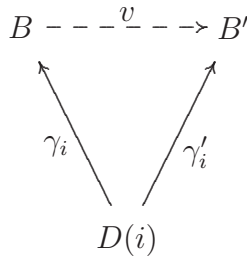
and let  $B$  together with  $q : A \rightarrow B$  be a coequalizer of  $\delta_x$  and  $\delta_y \circ f$ .



Construct a cocone  $\langle \gamma_i : D(i) \rightarrow B \rangle_{i \in N}$  over  $D$  by letting  $\gamma_i = q \circ \delta_i$ . We claim that  $\langle \gamma_i \rangle_{i \in N}$  is colimiting: suppose  $\langle \gamma'_i : D(i) \rightarrow B' \rangle_{i \in N}$  is a cocone over  $D$ . Then,  $\langle \gamma'_i \rangle_{i \in N}$  is a cocone over  $D'$ , as well. Since  $\langle \delta_i : D'(i) \rightarrow A \rangle_{i \in N}$  is colimiting, there is a unique morphism  $h : A \rightarrow B'$  such that for all  $i \in N : h \circ \delta_i = \gamma'_i$ . Hence, we have:  $h \circ \delta_x = \gamma'_x = \gamma'_y \circ f = h \circ \delta_y \circ f$ , and by the definition of coequalizer, there is a unique morphism  $v : B \rightarrow B'$  such that  $v \circ q = h$ .



Therefore, by letting  $\gamma_i = q \circ \delta_i$ , the following diagram commutes for all  $i \in N$ :



The uniqueness of  $v$  follows from the uniqueness of  $q$  and  $h$ . ■

**Proof of Corollary 3.34** It is trivial to verify that a pushout of  $A \leftarrow \mathbf{0} \rightarrow B$  is a binary coproduct of  $A$  and  $B$ ; and a coequalizer of  $f, g : A \rightarrow B$  is a pushout of  $A \xrightarrow[f]{g} B$ . The desired result then follows from Theorem 3.33. ■

**Proof of Theorem 3.45** Our proof is, in essence, very similar to that of Proposition 2 in [GB84]. We constructively prove that when the conditions stated in Theorem 3.45 are met, the category  $(L \downarrow R)$  has an initial object, binary coproducts of all object pairs, and coequalizers of all parallel morphism pairs. Colimit preservation property of the projection functors follows directly from the constructions.

**Initial object:** By assumption,  $\mathcal{A}$  and  $\mathcal{B}$  are finitely cocomplete, so both  $\mathcal{A}$  and  $\mathcal{B}$  have initial objects. Let  $\mathbf{0}_{\mathcal{A}}$  be an initial object in  $\mathcal{A}$  and  $\mathbf{0}_{\mathcal{B}}$  be an initial object in  $\mathcal{B}$ . By finite cocontinuity of  $L$ , we know that  $L(\mathbf{0}_{\mathcal{A}})$  is an initial object in  $\mathcal{C}$ ; therefore, for each  $\mathcal{C}$ -object  $C$ , there is a unique morphism from  $L(\mathbf{0}_{\mathcal{A}})$  to  $C$ . Particularly, there is a unique morphism  $u : L(\mathbf{0}_{\mathcal{A}}) \rightarrow R(\mathbf{0}_{\mathcal{B}})$ .

We claim that  $I = (\mathbf{0}_{\mathcal{A}}, u, \mathbf{0}_{\mathcal{B}})$  is an initial object in  $(L \downarrow R)$ : suppose  $C = (A, f, B)$  is a  $(L \downarrow R)$ -object. Let  $\langle \rangle : \mathbf{0}_{\mathcal{A}} \rightarrow A$  be the unique  $\mathcal{A}$ -morphism from  $\mathbf{0}_{\mathcal{A}}$  to  $A$  and let  $\langle \rangle' : \mathbf{0}_{\mathcal{B}} \rightarrow B$  be the unique  $\mathcal{B}$ -morphism from  $\mathbf{0}_{\mathcal{B}}$  to  $B$ . Since  $L(\mathbf{0}_{\mathcal{A}})$  is an initial object in  $\mathcal{C}$ , there exists a unique  $\mathcal{C}$ -morphism  $t : L(\mathbf{0}_{\mathcal{A}}) \rightarrow R(B)$ ; hence,  $t = f \circ L(\langle \rangle)$ ; and for the same reason,  $t = R(\langle \rangle') \circ u$ . Therefore,  $f \circ L(\langle \rangle) = R(\langle \rangle') \circ u$ , that is,  $(\langle \rangle, \langle \rangle') : I \rightarrow C$  is a  $(L \downarrow R)$ -morphism.

$$\begin{array}{ccc}
 L(\mathbf{0}_{\mathcal{A}}) & \xrightarrow{u} & R(\mathbf{0}_{\mathcal{B}}) \\
 \downarrow L(\langle \rangle) & \searrow t & \downarrow R(\langle \rangle') \\
 L(A) & \xrightarrow{f} & R(B)
 \end{array}$$

The uniqueness of  $(\langle \rangle, \langle \rangle') : I \rightarrow C$  follows from the uniqueness of  $\langle \rangle$  in  $\mathcal{A}$  and the uniqueness of  $\langle \rangle'$  in  $\mathcal{B}$ .

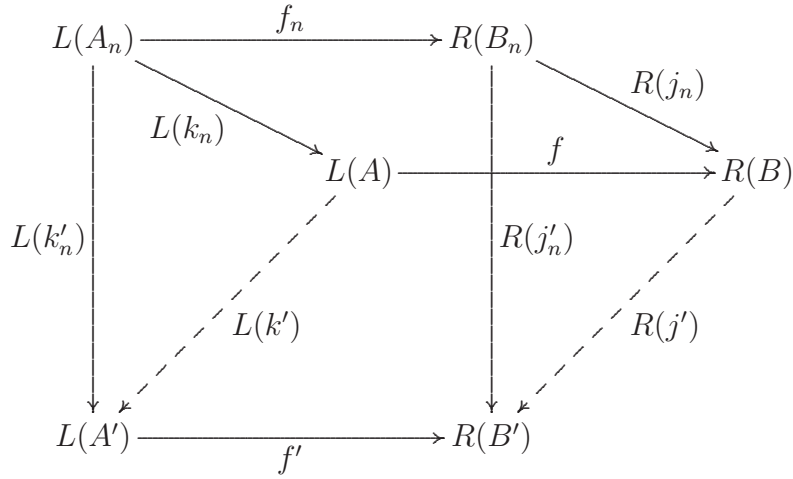
**Binary coproduct:** Suppose  $C_1 = (A_1, f_1, B_1)$ ,  $C_2 = (A_2, f_2, B_2)$  are a pair of  $(L \downarrow R)$ -objects. Let  $A = A_1 + A_2$  with injections  $k_n : A_n \rightarrow A$  and let  $B = B_1 + B_2$  with injections  $j_n : B_n \rightarrow B$  for  $n = 1, 2$ . By finite cocontinuity of  $L$ , we know that  $L(A)$  with  $\mathcal{C}$ -morphisms  $L(k_n) : L(A_n) \rightarrow L(A)$  for  $n = 1, 2$  is a binary coproduct of  $L(A_1)$  and  $L(A_2)$ . Therefore, there exists a unique morphism  $f : L(A) \rightarrow R(B)$  such that the following diagram commutes:

$$\begin{array}{ccc}
 L(A_1) & \xrightarrow{f_1} & R(B_1) \\
 \downarrow L(k_1) & \searrow R(j_1) \circ f_1 & \downarrow R(j_1) \\
 L(A) & \xrightarrow{\quad f \quad} & R(B) \\
 \uparrow L(k_2) & \nearrow R(j_2) \circ f_2 & \uparrow R(j_2) \\
 L(A_2) & \xrightarrow{f_2} & R(B_2)
 \end{array}$$

We claim that  $C = (A, f, B)$  with  $(k_n, j_n) : C_n \rightarrow C$  for  $n = 1, 2$  is a binary coproduct of  $C_1$  and  $C_2$ : suppose  $C' = (A', f', B')$  is a  $(L \downarrow R)$ -object and  $(k'_n, j'_n) : C_n \rightarrow C'$  for  $n = 1, 2$  are  $(L \downarrow R)$ -morphisms. By the properties of  $A$  and  $B$  in their respective categories, there are unique morphisms  $k' : A \rightarrow A'$  and  $j' : B \rightarrow B'$  such that  $k' \circ k_n = k'_n$  and  $j' \circ j_n = j'_n$ . Therefore, the following two diagrams commute in  $\mathcal{C}$  for  $n = 1, 2$ :

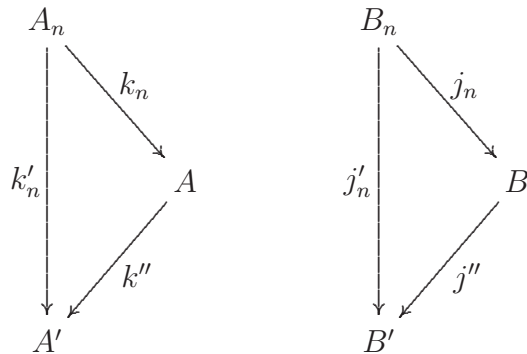
$$\begin{array}{ccc}
 L(A_n) & & R(B_n) \\
 \downarrow L(k'_n) & \searrow L(k_n) & \downarrow R(j'_n) \\
 & L(A) & R(B) \\
 & \nearrow L(k') & \nearrow R(j') \\
 L(A') & & R(B')
 \end{array}$$

We now show that the following diagram commutes in  $\mathcal{C}$  for  $n = 1, 2$ , as well:



$$\begin{aligned}
 f' \circ L(k'_n) &= R(j'_n) \circ f_n \implies \\
 f' \circ L(k') \circ L(k_n) &= R(j') \circ R(j_n) \circ f_n \implies \\
 f' \circ L(k') \circ L(k_n) &= R(j') \circ f \circ L(k_n)
 \end{aligned}$$

By the properties of  $L(A)$  in  $(L \downarrow R)$ , there is a unique morphism  $h : L(A) \rightarrow R(B')$  such that  $f' \circ L(k'_n) = h \circ L(k_n)$ . Therefore,  $f' \circ L(k') = h = R(j') \circ f$ . Thus,  $(k', j')$  is a morphism from  $C$  to  $C'$  in  $(L \downarrow R)$ . Uniqueness of  $(k', j')$  follows from the fact that any morphism  $(k'', j'')$  such that  $(k'', j'') \circ (k_n, j_n) = (k'_n, j'_n)$  will make the following two diagrams commute for  $n = 1, 2$ :



Therefore, by the properties of  $A$  and  $B$  in their respective categories,  $k''$  has to be the same as  $k'$  and  $j''$  has to be the same as  $j'$ .

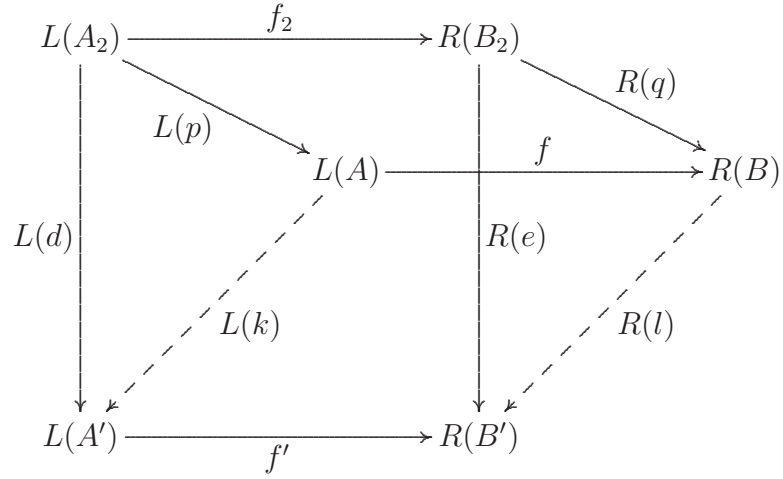
**Coequalizer:** Suppose  $C_1 = (A_1, f_1, B_1)$ ,  $C_2 = (A_2, f_2, B_2)$  are a pair of  $(L \downarrow R)$ -objects; and  $(a, b), (a', b') : C_1 \rightarrow C_2$  are a pair of parallel  $(L \downarrow R)$ -morphisms. Let  $A$  with  $p : A_2 \rightarrow A$  be a coequalizer of  $a : A_1 \rightarrow A_2$  and  $a' : A_1 \rightarrow A_2$  (in  $\mathcal{A}$ ) and let  $B$  with  $q : B_2 \rightarrow B$  be a coequalizer of  $b : B_1 \rightarrow B_2$  and  $b' : B_1 \rightarrow B_2$  (in  $\mathcal{B}$ ). By finite cocontinuity of  $L$ , coequalizers in  $\mathcal{A}$  are mapped to coequalizers in  $\mathcal{C}$ ; therefore, there exists a unique morphism  $f : L(A) \rightarrow R(B)$  such that  $f \circ L(p) = R(q) \circ f_2$ .

$$\begin{array}{ccc}
 L(A_1) & \xrightarrow{f_1} & R(B_1) \\
 \downarrow L(a) \quad \downarrow L(a') & & \downarrow L(b) \quad \downarrow L(b') \\
 L(A_2) & \xrightarrow{f_2} & R(B_2) \\
 \downarrow L(p) & \searrow R(q) \circ f_2 & \downarrow R(q) \\
 L(A) & \dashrightarrow f & R(B)
 \end{array}$$

We claim that  $C = (A, f, B)$  together with  $(p, q)$  is a coequalizer of  $(a, b)$  and  $(a', b')$ . It is clear from the above diagram that  $(p, q)$  is indeed a morphism in  $(L \downarrow R)$ ; moreover, by the properties of  $A$  and  $B$  in their respective categories, we have:  $(p, q) \circ (a, b) = (p, q) \circ (a', b')$ .

Assuming  $C' = (A', f', B')$  is a  $(L \downarrow R)$ -object and  $(d, e) : C_2 \rightarrow C'$  is a  $(L \downarrow R)$ -morphism, there exists a unique morphism  $k : A \rightarrow A'$  (in  $\mathcal{A}$ ) and a unique morphism  $l : B \rightarrow B'$  (in  $\mathcal{B}$ ) such that:  $k \circ p = d$  and  $l \circ q = e$ . We now show that the following

diagram commutes in  $\mathcal{C}$ :



$$\begin{aligned}
 f' \circ L(d) &= R(e) \circ f_2 \implies \\
 f' \circ L(k) \circ L(p) &= R(l) \circ R(q) \circ f_2 \implies \\
 f' \circ L(k) \circ L(p) &= R(l) \circ f \circ L(p)
 \end{aligned}$$

By the properties of  $L(A)$  in  $\mathcal{C}$ , there exists a unique morphism  $h : L(A) \rightarrow R(B')$  such that  $f' \circ L(d) = h \circ L(p)$ . Therefore,  $f' \circ L(k) = h = R(l) \circ f$ . Hence,  $(k, l)$  is a  $(L \downarrow R)$ -morphism from  $C$  to  $C'$ . The uniqueness of  $(k, l)$  can be proved in exactly the same way as how the uniqueness of  $(k', j')$  was proved in the construction of binary coproducts. ■