# Analysis of Inconsistency in Graph-Based Viewpoints: A Category-Theoretic Approach

Mehrdad Sabetzadeh Steve Easterbrook
Department of Computer Science, University of Toronto
Toronto, ON M5S 3G4, Canada.
Email: {`mehrdad,sme`}`@cs.toronto.edu`

## Abstract

*Eliciting the requirements for a proposed system typically involves different stakeholders with different expertise, responsibilities, and perspectives. Viewpoints-based approaches have been proposed as a way to manage incomplete and inconsistent models gathered from multiple sources. In this paper, 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. We illustrate an application of the framework through a case-study showing how fuzzy viewpoints can serve as a requirements elicitation tool in reactive systems.*

## 1 Introduction

Requirements elicitation and analysis is significantly complicated by the incompleteness and inconsistency of the information available in early stages of software development life-cycle. Different stakeholders often talk about different aspects of a problem, use different terminologies to express their descriptions, and have conflicting goals. Viewpoints-based approaches have been proposed as a way to manage incomplete and inconsistent information gathered from multiple sources [8]. We may use viewpoints 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 [6]. By separating the descriptions provided by different stakeholders, viewpoints-based approaches facilitate the detection of inconsistencies between the descriptions.

In the classical viewpoint approach, incompleteness and inconsistency within a viewpoint cannot be modeled explicitly. Consistency checking is achieved by expressing a set of consistency rules in a rich meta-language, such as first order logic [8, 7]. With this approach, viewpoints can be merged only by making them consistent first, and then using classical composition operators. Alternatively, one could translate a set of viewpoints, together with their consistency rules, into the same logic. However, this approach discards both the structure of the original viewpoints, and the syntactic aspects of their original notations. Hence, such merges cannot be based on structural mappings between viewpoints. Neither approach is satisfactory, and the question of how to compose viewpoints that are incomplete and inconsistent has remained an open problem for the past decade.

This paper introduces a category-theoretic formalism for the syntactic 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 fuzzy viewpoints can be done by constructing 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 approach to 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 paper, our focus will be on a fairly simple kind of fuzzy viewpoints inspired by $\chi$views [6]. We will use simple state-machine models to show how nodes and edges in graphical notations can be decorated with the additional structures required for modeling incompleteness and inconsistency.

The mathematical machinery in this paper builds upon the category-theoretic properties of fuzzy sets noted in [10, 11]. The use of colimits as an abstract mechanism for putting structures together has been known for quite some time (cf. [12] for references). In [14, 15], colimits have been used for merging viewpoints. In that approach, viewpoints are described by open graph transformation systems and colimits are used to integrate them. Our work is, as far as we know, the first use of category theory for merging *inconsistent* viewpoints.

The paper is organized as follows: Section 2 reviews the required mathematical background. Section 3 outlines the definitions and lemmas on fuzzy set categories needed in this paper. Section 4 proposes a category-theoretic formalism for fuzzy viewpoints. Section 5 describes the merge operation for fuzzy viewpoints and provides a definition for syntactic inconsistency based on colimits. Section 6 demonstrates a modeling process using fuzzy viewpoints in the context of a case-study; and finally, Section 7 presents our conclusions and future work.

## 2 Mathematical Preliminaries

In this section, we briefly review the definitions of graphs, lattices, and the category-theoretic constructs used in the remainder of the paper.

### 2.1 Graphs and Graph Homomorphisms

**Definition 2.1** A *graph* is a quadruple $G = (N, E, \mathsf{s}_G, \mathsf{t}_G)$ where $N$ is a set of nodes, $E$ a set of edges, and $\mathsf{s}_G, \mathsf{t}_G : E \to N$ are functions respectively giving the source and the target for each edge. A *graph homomorphism* from $G = (N, E, \mathsf{s}_G, \mathsf{t}_G)$ to $G' = (N', E', \mathsf{s}_{G'}, \mathsf{t}_{G'})$ is a pair of functions $h = \langle h_{\mathbf{n}} : N \to N', h_{\mathbf{e}} : E \to E' \rangle$ such that: $h_{\mathbf{n}} \circ \mathsf{s}_G = \mathsf{s}_{G'} \circ h_{\mathbf{e}}$ and $h_{\mathbf{n}} \circ \mathsf{t}_G = \mathsf{t}_{G'} \circ h_{\mathbf{e}}$.

### 2.2 Partially Ordered Sets and Lattices

**Definition 2.2** A *partial order* is a reflexive, anti-symmetric, and transitive binary relation. A non-empty set with a partial order on it is called a *partially ordered set* or a *poset*, for short. We use Hasse diagrams (cf. e.g. [5]) to visualize finite posets.

**Definition 2.3** Let $Q$ be a poset. An element $\top \in Q$ is said to be the *top* element if $\forall a \in Q : a \leq \top$. Dually, an element $\bot \in Q$ is said to be the *bottom* element if $\forall a \in Q : \bot \leq a$.

**Definition 2.4** Let $Q$ be a poset and let $A \subseteq Q$. An element $q \in Q$ is an *upper bound* of $A$ if $\forall a \in A : a \leq q$. If $q$ is an upper bound of $A$ and $q \leq x$ for all upper bounds $x$ of $A$, then $q$ is called the *supremum* of $A$. Dually, an element $q \in Q$ is a *lower bound* of $A$ if $\forall a \in A : q \leq a$. If $q$ is a lower bound of $A$ and $x \leq q$ for all lower bounds $x$ of $A$, then $q$ is called the *infimum* of $A$. We write $\bigsqcup_Q A$ (resp. $\bigsqcap_Q A$) to denote the supremum (resp. infimum) of $A \subseteq Q$, when it exists.

**Definition 2.5** Let $Q$ be a poset. If both $\bigsqcup_Q \{a, b\}$ and $\bigsqcap_Q \{a, b\}$ exist for any $a, b \in Q$, then $Q$ is called a *lattice*. If both $\bigsqcup_Q A$ and $\bigsqcap_Q A$ exist for any $A \subseteq Q$, then $Q$ is called a *complete lattice*.

**Lemma 2.6** *(cf. e.g. [5]) Every finite lattice is complete.*

**Lemma 2.7** *(cf. e.g. [5]) Every complete lattice has a bottom ($\bot$) and a top ($\top$) element.*

### 2.3 Category Theory

We assume acquaintance with basic concepts of category theory and only review diagrams, colimits, and comma categories in order to provide context and establish notation. An excellent introduction to category theory from a computer science perspective is [1]. Detailed discussion about comma categories can be found in [13, 16].

#### 2.3.1 Diagrams and Colimits

**Definition 2.8** A *(finite) diagram* $D$ in a category $\mathscr{C}$ consists of: a (finite) graph $G(D)$; a $\mathscr{C}$-object $D_n$ for each node $n$ in $G(D)$; and a $\mathscr{C}$-morphism $D_e : D_{\mathsf{s}_{G(D)}(e)} \to D_{\mathsf{t}_{G(D)}(e)}$ for each edge $e$ in $G(D)$. A diagram $D$ is said to *commute* if for any nodes $i$ and $j$ in $G(D)$ and any two paths $i \xrightarrow{s_1} k_1 \to \cdots \to k_{n-1} \xrightarrow{s_n} j$ and $i \xrightarrow{t_1} l_1 \to \cdots \to l_{m-1} \xrightarrow{t_m} j$ from $i$ to $j$ in $G(D)$: $D_{s_n} \circ \cdots \circ D_{s_1} = D_{t_m} \circ \cdots \circ D_{t_1}$.

**Definition 2.9** A *pushout* of a pair of morphisms $f : C \to A$ and $g : C \to B$ in a category $\mathscr{C}$ is a $\mathscr{C}$-object $P$ together with a pair of morphisms $j : A \to P$ and $k : B \to P$ such that: $j \circ f = k \circ g$; and for any $\mathscr{C}$-object $P'$ and pair of morphisms $j' : A \to P'$ and $k' : B \to P'$ satisfying $j' \circ f = k' \circ g$, there is a unique morphism $h : P \to P'$ such that the following diagram commutes:

$$\begin{tikzcd}
 & & P' \\
A \arrow[urr, "j'"] \arrow[r, "j"'] & P \arrow[ur, dashed, "h"'] & \\
C \arrow[u, "f"] \arrow[r, "g"'] & B \arrow[u, "k"] \arrow[uur, "k'"'] &
\end{tikzcd}$$

Pushout is a special type of a more general construct known as colimit (see Definition 2.10). In order to help a reader without working knowledge of category theory follow the paper, we spell out the construction of pushouts in

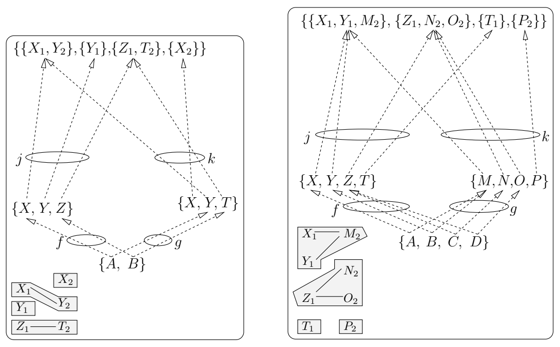


**Figure 1. Pushout examples in $\mathbf{Set}$**

$\mathbf{Set}$ (category of sets and functions) without directly appealing to the underlying category-theoretic constructs (i.e. binary coproducts and coequalizers).

The canonical pushout of a pair of $\mathbf{Set}$-morphisms (i.e. functions) $f : C \to A$ and $g : C \to B$ is constructed as follows: take the disjoint union of sets $A$ and $B$, denoted $A \uplus B$. Let $\imath_A : A \to A \uplus B$, $\imath_B : B \to A \uplus B$ be the injection functions respectively taking $A$ and $B$ to their images in $A \uplus B$. The functions $\imath_A \circ f$ and $\imath_B \circ g$ now yield a binary relation $R = \big\{\big(\imath_A \circ f(c), \imath_B \circ g(c)\big) \mid c \in C\big\}$ on $A \uplus B$. Consider the *undirected* graph $G$ in which the set of nodes is $A \uplus B$ and for any nodes $x, y$ in $G$, there is an (undirected) edge between $x$ and $y$ if and only if $(x, y) \in R$. Let $P$ be the set of $G$'s connected components and let $q : A \uplus B \to P$ be the function taking every $x \in A \uplus B$ to the connected component of $G$ which $x$ belongs to. The following diagram will be a pushout square:

$$\begin{tikzcd}
 & A \arrow[d, "\imath_A"] \arrow[dr, "j = q \circ \imath_A"] & \\
C \arrow[ur, "f"] \arrow[r, shift left, "\imath_A \circ f"] \arrow[r, shift right, "\imath_B \circ g"'] \arrow[dr, "g"'] & A \uplus B \arrow[r, "q"] & P \\
 & B \arrow[u, "\imath_B"'] \arrow[ur, "k = q \circ \imath_B"'] &
\end{tikzcd}$$

Figure 1 shows two examples of pushout computation in $\mathbf{Set}$. The maps corresponding to the morphisms $f$, $g$, $j$, and $k$ of the pushout square have been marked in both examples.

**Definition 2.10** Let $D$ be a diagram in $\mathscr{C}$ and let $N$ and $E$ denote the set of $G(D)$'s nodes and edges, respectively. A *cocone* $\alpha$ over $D$ is a $\mathscr{C}$-object $X$ together with a family of $\mathscr{C}$-morphisms $\langle \alpha_n : D_n \to X \rangle_{n \in N}$ such that for every edge $e \in E$ with $\mathsf{s}_{G(D)}(e) = i$ and $\mathsf{t}_{G(D)}(e) = j$, we have: $\alpha_j \circ D_e = \alpha_i$. A *colimit* of $D$ is a cocone $\langle \alpha_n : D_n \to X \rangle_{n \in N}$ such that for any cocone $\langle \alpha'_n : D_n \to X' \rangle_{n \in N}$, there is a unique morphism $h : X \to X'$ such that for every $n \in N$, we have: $\alpha'_n = h \circ \alpha_n$.

**Definition 2.11** A category $\mathscr{C}$ is *(finitely) cocomplete* if every (finite) diagram in $\mathscr{C}$ has a colimit.

Among the numerous results on cocompleteness of categories, the following lemma is of interest to us:

**Lemma 2.12** *(cf. e.g. [1]) A category with an initial object, binary coproducts of object pairs, and coequalizers of parallel morphism pairs is finitely cocomplete.*

The general intuition behind colimits is that they put structures together with nothing essentially new added, and nothing left over [12]. This is the main reason behind our interest in (finite) cocompleteness results. The examples given in Figure 1 illustrate that the pushout of a pair of morphisms $f : C \to A$ and $g : C \to B$ in $\mathbf{Set}$ can be considered as the combination of $A$ and $B$ with respect to a shared part $C$.

For reasons that will become clear in Section 2.3.2, we are also interested in functors that preserve (finite) colimits:

**Definition 2.13** A functor $F : \mathscr{C} \to \mathscr{D}$ is said to be *(finitely) cocontinuous* if it preserves the existing colimits of all (finite) diagrams in $\mathscr{C}$, that is, if for any (finite) diagram $D$ in $\mathscr{C}$, the functor $F$ maps any colimiting cocone over $D$ to a colimiting cocone over $F(D)$.

**Lemma 2.14** *(cf. e.g. [1]) If $\mathscr{C}$ is a finitely cocomplete category and if a functor $F : \mathscr{C} \to \mathscr{D}$ preserves initial objects, binary coproducts of all object pairs, and coequalizers of all parallel morphism pairs, then $F$ is finitely cocontinuous.*

### 2.3.2 Comma Categories

**Definition 2.15** Let $\mathscr{A}$, $\mathscr{B}$, and $\mathscr{C}$ be categories and let $L : \mathscr{A} \to \mathscr{C}$ and $R : \mathscr{B} \to \mathscr{C}$ be functors. The *comma category* $(L \downarrow R)$ has as objects, triples $(A, f : L(A) \to R(B), B)$ where $A$ is an object of $\mathscr{A}$ and $B$ is an object of $\mathscr{B}$. A morphism from $(A, f, B)$ to $(A', f', B')$ is a pair $(s : A \to A', t : B \to B')$ such that the following diagram commutes in $\mathscr{C}$:

$$\begin{tikzcd}
L(A) \arrow[r, "f"] \arrow[d, "L(s)"'] & R(B) \arrow[d, "R(t)"] \\
L(A') \arrow[r, "f'"'] & R(B')
\end{tikzcd}$$

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

**Lemma 2.16** *[19, 16] Let $L : \mathscr{A} \to \mathscr{C}$ and $R : \mathscr{B} \to \mathscr{C}$ be functors with $L$ (finitely) cocontinuous. If $\mathscr{A}$ and $\mathscr{B}$ are (finitely) cocomplete so is the comma category $(L \downarrow R)$.*

A remarkable fact about colimit construction in $(L \downarrow R)$ is that (finite) colimits are inherited from those in $\mathscr{A}$ and $\mathscr{B}$ when $L$ is (finitely) cocontinuous [19, 16].

Using a triple $(E, \langle \mathsf{s}_G, \mathsf{t}_G\rangle : E \to N \times N, N)$ to denote a graph $G$, it can be verified [16] that the category of graphs, denoted $\mathbf{Graph}$, is isomorphic to the comma category $(I_{\mathbf{Set}} \downarrow T)$ where $I_{\mathbf{Set}} : \mathbf{Set} \to \mathbf{Set}$ is the identity functor on $\mathbf{Set}$ and $T : \mathbf{Set} \to \mathbf{Set}$, known as the Cartesian product functor, is the functor that maps every set $N$ to $N \times N$ and every function $f : N \to N'$ to $f \times f$. Here, $f \times f$ is the function that takes every $(x, y) \in N \times N$ to $\big(f(x), f(y)\big) \in N' \times N'$. Since $\mathbf{Set}$ is cocomplete and the identity functor is cocontinuous, Lemma 2.16 implies that $\mathbf{Graph}$ is cocomplete. Moreover, colimits are computed component-wise for nodes and edges. In Section 4, we give an analogous definition for fuzzy viewpoint categories by using fuzzy set categories in place of $\mathbf{Set}$ and in Section 6, we will exploit the component-wise nature of colimit construction in comma categories without further remarks.

## 3 Categories of Fuzzy Sets

Since its inception in the 1960s, fuzzy set theory has received considerable attention from different computing disciplines. This section presents the definitions and results on fuzzy set categories needed in the paper. Most of the definitions and lemmas given here can be found in [10, 11] and are probably very well-known in the literature on topos theory; 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.

**Definition 3.1** Let $Q$ be a poset. A *$Q$-valued set* is a pair $(S, \sigma)$ consisting of a set $S$ and a function $\sigma : S \to Q$. 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 $(S, \sigma)$.

**Definition 3.2** Let $(S, \sigma)$ and $(T, \tau)$ be two $Q$-valued sets. A *morphism* $\mathbf{f} : (S, \sigma) \to (T, \tau)$ is a function $f : S \to T$ such that $\sigma \leq \tau \circ f$, i.e. the degree of membership of $s$ in $(S, \sigma)$ does not exceed that of $f(s)$ in $(T, \tau)$. The function $f : S \to T$ is called the *carrier function* of $\mathbf{f}$.

**Lemma 3.3** *For a fixed poset $Q$, the objects and morphisms defined above together with the obvious identities give rise to a category, denoted $\mathbf{Fuzz}(Q)$.*

Figure 2 (informally) shows two $\mathbf{Fuzz}(\mathbf{K})$ objects $(S, \sigma)$ and $(T, \tau)$ along with the carrier function $f : S \to T$ of a $\mathbf{Fuzz}(\mathbf{K})$-morphism $\mathbf{f} : (S, \sigma) \to (T, \tau)$ where $\mathbf{K}$ is the three-point poset shown in the same figure.

**Lemma 3.4** *[10, 11] $\mathbf{Fuzz}(Q)$ is finitely cocomplete when $Q$ is a complete lattice.*

**Proof (*sketch*)** We show how to construct the initial object, binary coproducts, and coequalizers. Finite cocompleteness of $\mathbf{Fuzz}(Q)$ then follows from Lemma 2.12.

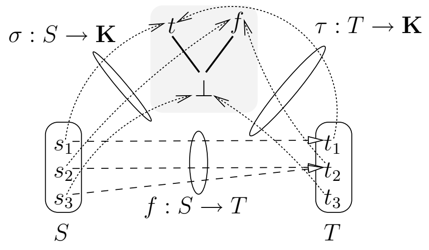


**Figure 2. Example of fuzzy sets**

***Initial object***: $\mathbf{0} = (\emptyset, \lambda)$ where $\lambda : \emptyset \to Q$ 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 $\mathbf{Set}$-coproduct (disjoint union) of $S_1$ and $S_2$ with injections $\imath_n : S_n \to S_1 + S_2$ for $n = 1, 2$; and $\kappa\big(\imath_n(s)\big) = \sigma_n(s)$ for $s \in S_n$ and $n = 1, 2$.
***Coequalizer***: given objects $X = (A, \sigma)$ and $Y = (B, \tau)$ with parallel morphisms $\mathbf{h_1} : X \to Y$ and $\mathbf{h_2} : X \to Y$, we first take the canonical $\mathbf{Set}$-coequalizer of the carrier functions $h_1 : A \to B$ and $h_2 : A \to B$ to find a set $C$ and a function $q : B \to C$. 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 $q : B \to C$ to a morphism $\mathbf{q} : Y \to Z$, which is a coequalizer of $\mathbf{h_1}$ and $\mathbf{h_2}$. ∎

Since we want to avoid using the details of the above proof in Section 6, 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 $\mathbf{f} : (C, \gamma) \to (A, \sigma)$ and $\mathbf{g} : (C, \gamma) \to (B, \tau)$, first compute the canonical $\mathbf{Set}$-pushout of the carrier functions $f : C \to A$ and $g : C \to B$ (as discussed in Section 2) to find a set $P$ along with functions $j : A \to P$ and $k : B \to P$. Then, compute a membership degree for every $p \in P$ 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 $\mathbf{Fuzz}(Q)$.

**Definition 3.5** The map that takes every $\mathbf{Fuzz}(Q)$-object $(S, \sigma)$ to its carrier set $S$ and every $\mathbf{Fuzz}(Q)$-morphism $\mathbf{f} : (S, \sigma) \to (T, \tau)$ to its carrier function $f : S \to T$ yields a functor $K_Q : \mathbf{Fuzz}(Q) \to \mathbf{Set}$, known as the *carrier functor*.

**Lemma 3.6** *The carrier functor $K_Q : \mathbf{Fuzz}(Q) \to \mathbf{Set}$ is finitely cocontinuous when $Q$ is a complete lattice*[1].

[1]The carrier functor is finitely cocontinuous even when $Q$ is an arbitrary poset, but a separate proof is required.

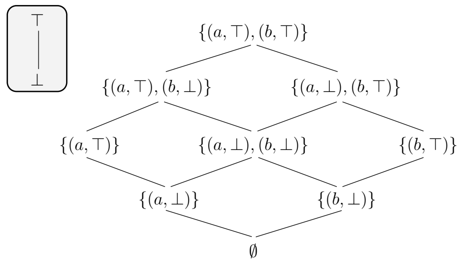


**Figure 3. Powerset lattice example**

**Proof** Based on the proof of Lemma 3.4, it is obvious that $K_Q : \mathbf{Fuzz}(Q) \to \mathbf{Set}$ preserves the initial object, binary coproducts, and coequalizers. Finite cocontinuity of $K_Q$ then follows from Lemma 2.14. ∎

**Definition 3.7** 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 $\mathbf{Fuzz}(Q)$-objects $(C, \xi)$ such that $C \subseteq S$ and for every element $c \in C$: $\xi(c) \leq \sigma(c)$.

**Lemma 3.8** *[10, 11] The powerset of any* $\mathbf{Fuzz}(Q)$*-object is a complete lattice when* $Q$ *is.*

**Proof (*sketch*)** [10, 11] For an index set $I$, the supremum of an $I$-indexed family of $\mathcal{P}(Z)$ elements $\big\langle (C_i, \xi_i) \big\rangle_{i \in I}$ is a fuzzy set $(X, \theta)$ where $X = \bigcup_{i \in I} C_i$ and $\theta : X \to Q$ is a function such that for every element $x \in X$: $\theta(x) = \bigsqcup_Q \{\xi_i(x) \mid x \in C_i; i \in I\}$. The infimum is computed, dually. ∎

As an example, suppose the truth-set is the two-point lattice $\mathbf{L_2} = \{\bot, \top\}$ with $\bot < \top$. Then, the powerset of the $\mathbf{Fuzz}(\mathbf{L_2})$-object $Z = \big(\{a, b\}, \{a \mapsto \top, b \mapsto \top\}\big)$ is the lattice shown in Figure 3. For easier readability, the figure uses tuples of the form $(element, membership\ degree)$ instead of the original notation.

## 4 $\mathfrak{F}\mathrm{View}$ Categories

In this section, we define the terms "fuzzy viewpoint" and "fuzzy viewpoint morphism" and prove that the category of fuzzy viewpoints over a complete lattice $\mathcal{L}$ and an arbitrary but fixed set $U$ of atomic propositions is finitely cocomplete.

**Definition 4.1** Let $\mathcal{L}$ be a complete lattice of truth values and let $U$ (called the universe of atomic propositions) be an arbitrary but fixed set. 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 $\zeta_n : U_n \to \mathcal{L}$ 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 $\mathcal{V}$.

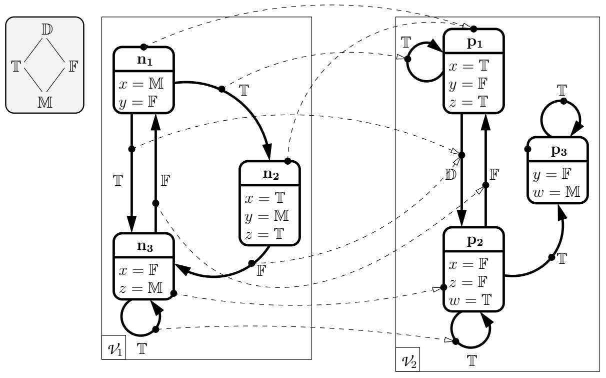


**Figure 4. Example of fuzzy viewpoints**

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 $\epsilon : U \to \mathcal{L}$ be the constant map $\{x \mapsto \top \mid x \in U\}$ (notice that $\top$ is known to exist by Lemma 2.7). Thus, $(U, \epsilon)$ is a $\mathbf{Fuzz}(\mathcal{L})$-object. Now, by Lemma 3.8, 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 $\mathbf{Fuzz}(\mathcal{X})$.

**Definition 4.2** Let $\mathcal{V}$ and $\mathcal{V}'$ be $(\mathcal{L}, U)$-fuzzy viewpoints. A *viewpoint morphism* $\hbar : \mathcal{V} \to \mathcal{V}'$ is a pair $\langle \hbar_{\mathbf{n}}, \hbar_{\mathbf{e}} \rangle$ where $\hbar_{\mathbf{n}}$ is a $\mathbf{Fuzz}(\mathcal{X})$-morphism and $\hbar_{\mathbf{e}}$ is a $\mathbf{Fuzz}(\mathcal{L})$-morphism such that $\langle K_{\mathcal{X}}(\hbar_{\mathbf{n}}), K_{\mathcal{L}}(\hbar_{\mathbf{e}}) \rangle$ is a graph homomorphism from the carrier graph of $\mathcal{V}$ to the carrier graph of $\mathcal{V}'$. Here, $K_{\mathcal{X}} : \mathbf{Fuzz}(\mathcal{X}) \to \mathbf{Set}$ and $K_{\mathcal{L}} : \mathbf{Fuzz}(\mathcal{L}) \to \mathbf{Set}$ are the carrier functors.

It can be verified that the above choice of objects and morphisms (with the obvious identities) constitutes a category of $(\mathcal{L}, U)$-fuzzy viewpoints, which we will hereafter denote $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$.

As an example, suppose $\mathcal{L} = \mathbf{A_4}$ (the lattice shown on the top-left corner of Figure 4), and $U = \{x, y, z, w\}$. Figure 4 shows two $\mathfrak{F}\mathbf{View}(\mathbf{A_4}, U)$-objects along with a $\mathfrak{F}\mathbf{View}(\mathbf{A_4}, U)$-morphism. In this and all the figures in Section 6, the viewpoint edges have been left anonymous and only the truth values labeling them have been shown. Notice that 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 4 as $\mathfrak{F}\mathbf{View}(\mathbf{A_4}, U')$ objects and morphisms.

It is easy to see that $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$ is isomorphic to the comma category $(K_{\mathcal{L}} \downarrow T \circ K_{\mathcal{X}})$ where $T : \mathbf{Set} \to \mathbf{Set}$ is the Cartesian product functor defined in Section 2.3.2. Since $\mathcal{L}$ and $\mathcal{X}$ are complete lattices, both $\mathbf{Fuzz}(\mathcal{L})$ and $\mathbf{Fuzz}(\mathcal{X})$ are finitely cocomplete by Lemma 3.4. Moreover, by Lemma 3.6, we know that the carrier functor

$K_{\mathcal{L}} : \mathbf{Fuzz}(\mathcal{L}) \to \mathbf{Set}$ is finitely cocontinuous when $\mathcal{L}$ is a complete lattice. Now, by Lemma 2.16, we get:

**Theorem 4.3** *$\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$ is finitely cocomplete for any complete lattice $\mathcal{L}$ and any set $U$.*

This, by Lemma 2.6, implies that $\mathfrak{F}\mathbf{View}(\mathcal{L}, U)$ is finitely cocomplete for any *finite* lattice $\mathcal{L}$ and any set $U$.

Although not elaborated here, Lemma 3.6 makes it possible to enrich viewpoints' nodes and edges with other structures such as types and additional labels through well-known comma categorical techniques without violating the cocompleteness result achieved in Theorem 4.3. For some preliminary work in this direction, see [17].

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. The restriction is not a problem if there is a reference model made available at early stages of requirements elicitation, which describes the elements of $U$. If $U$ is not given beforehand, it is sufficient to assume $U$ is the set of natural numbers, $\mathbb{N}$. 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. Alternatively, we anticipate that the limitation could be addressed by introducing explicit signatures and signature morphisms, similar to the approach described in [18]; however, we have not yet investigated this idea.

## 5 Viewpoint Integration and Characterization of Inconsistency

It is well-known that "for a given 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" [12]. 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 4.3 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 [6].

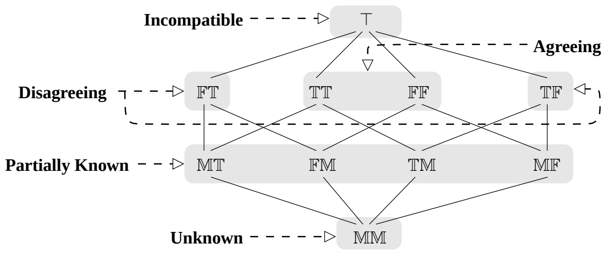


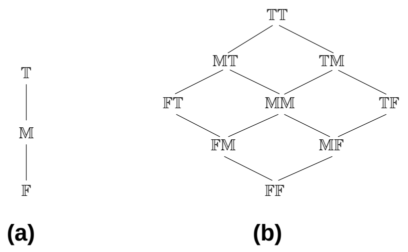
**Figure 5. The lattice $\mathbf{A_{10}}$**

Viewpoint integration via colimits is abstract from how viewpoint interconnections are identified. In Section 6, 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 [15] where some useful patterns for viewpoint interconnection have been identified.

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 [2] 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. This lattice arises naturally in modeling a system with two stakeholders and will be used for the case-study presented in Section 6.

In $\mathbf{A_{10}}$, the value $\mathbb{MM}$ indicates that no information is available. The values $\mathbb{TM}$ and $\mathbb{FM}$ (resp. $\mathbb{MT}$ and $\mathbb{MF}$) 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 $\mathbb{TM}$ (resp. $\mathbb{MT}$). The values $\mathbb{TT}$, $\mathbb{FF}$ indicate that both stakeholders agree on whether something is TRUE or FALSE while $\mathbb{TF}$ and $\mathbb{FT}$ 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 6.

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 6a and 6b, respectively. The



$\mathbb{T}$
$\mathbb{M}$
$\mathbb{F}$
(a)
$\mathbb{TT}$
$\mathbb{MT}$ $\mathbb{TM}$
$\mathbb{FT}$ $\mathbb{MM}$ $\mathbb{TF}$
$\mathbb{FM}$ $\mathbb{MF}$
$\mathbb{FF}$
(b)

**Figure 6. Truth ordering lattices**

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 6a (resp. 6b) is an instance of a family of multivalued logics known as Kleene-like [9] logics. In this paper, we 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 [9] 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. [5] 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.

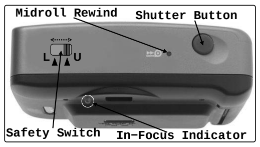
The existence of such truth ordering lattices is an advantage when we want to interpret viewpoints according to certain semantics. For example, we may want to treat a viewpoint as a $\chi$Kripke [3] structure and verify certain temporal properties in it through multivalued model-checking [3]. For such 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 paper, 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 the syntactic and the semantic aspects of viewpoints. 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 5.1** 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.

In $\mathbf{A_{10}}$, for example, we may choose to designate $\mathbb{TT}$ and $\mathbb{FF}$ 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 $\mathbb{TF}$, $\mathbb{FT}$ 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* [7] is required. When using $\mathbf{A_{10}}$, for example, we may decide to *live with* all $\mathbf{A_{10}}$ values except for incompatibilities ($\top$).

**The Camera Specification**



The camera should have a safety switch with two states: "locked" and "unlocked". The locked state prevents accidental operation.

When the camera is not locked and a new film is loaded, the film should automatically advance to the first frame once the camera back is closed. After the film's last frame is exposed, the camera should rewind the film automatically back into the cartridge. The camera should also have a "midroll rewind" button to rewind the film before reaching the last frame.

The camera should have a shutter button that can be depressed halfway or all the way. There should be a click-stop at the halfway point. When the camera is not locked, the shutter button should work as follows:

- When pressed halfway, auto-focusing starts;
- When pressed completely, the shutter is released to take the picture and then the film advances by one frame.

The scenario for taking a picture is as follows: The shutter button is pressed halfway. When focus is achieved, the in-focus indicator will light. The shutter button can then be pressed completely to take the picture. Under low-lit conditions, the built-in flash should fire automatically.

**Figure 7. Camera's early reference model**

## 6 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 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 7, 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 camera project has only two stakeholders, namely Bob and Mary; thus, the lattice $\mathbf{A_{10}}$ (Figure 5) 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 $\chi\mathrm{Kripke}$ structures [6, 3]. 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 $t : i \to j$ 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 $\mathfrak{F}\mathrm{Draw}$ should explicitly be configured to do so. $\mathfrak{F}\mathrm{Draw}$ disallows parallel transitions between states; but, for convenience, it allows transitions to be omitted and internally interprets the absent transitions as $\mathbb{MM}$ transitions. Notice that $\mathfrak{F}\mathrm{Draw}$ could as well be configured to interpret absent transitions in Bob's (resp. Mary's) viewpoint as $\mathbb{FM}$ (resp. $\mathbb{MF}$) 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 the particular perspective that $\mathcal{V}$ reflects. In either case, an unspecified proposition in a state is interpreted as $\mathbb{MM}$. This is somehow analogous to the interpretation of absent transitions.

Figures 8 and 9 respectively show Bob's and Mary's viewpoints[2]. 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 capture the situation in which the shutter button is pressed halfway but is released without 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.

[2]In both figures, SH_BTN is an abbreviation for SHUTTER_BUTTON.

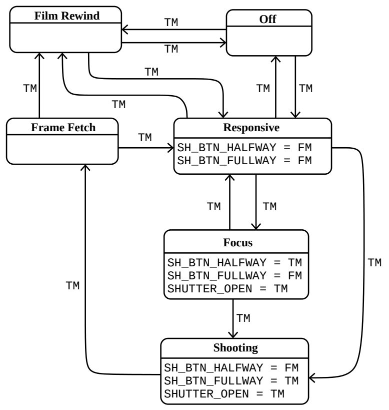


**Figure 8. Bob's viewpoint**

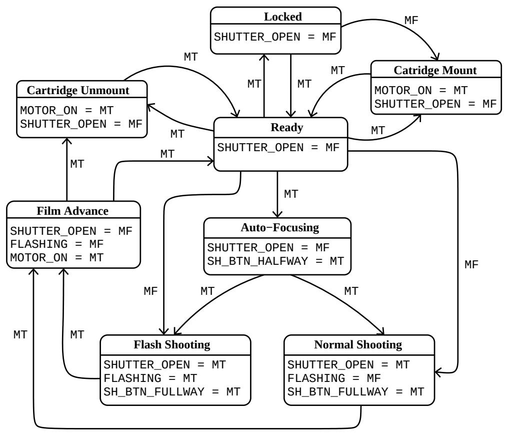


**Figure 9. Mary's viewpoint**

Sam now identifies the structural interconnection between the two viewpoints. He first identifies the state interconnections. This has been shown in Figure 10. 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 $\mathscr{h} : \mathcal{V} \to \mathcal{V}'$ maps states $i$ and $j$ in $\mathcal{V}$ to states $i'$ and $j'$ in $\mathcal{V}'$ respectively, then $\mathscr{h}$ must map the unique transition from $i$ to $j$ in $\mathcal{V}$ to the unique transition from $i'$ to $j'$ in $\mathcal{V}'$.

Figure 11 shows Sam's viewpoint. This viewpoint has been computed by $\mathfrak{F}\mathrm{Draw}$ directly from the state interconnections. For clarity, we have chosen to show those tran-

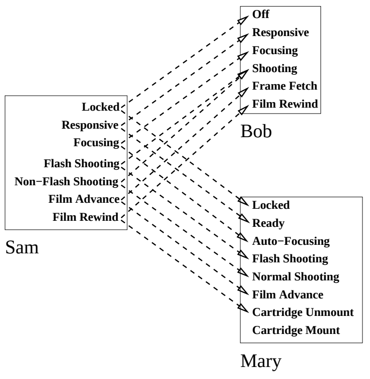


**Figure 10. State interconnections**

sitions in Sam's viewpoint that are mapped to non-$\mathbb{MM}$ transitions in Bob's and Mary's 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 for saving space) is analogous. Since Sam does not engage in determining the truth or falsehood of transitions and propositions, his viewpoint solely reflects the structural relationships between Bob's and Mary's viewpoints. As a result, all transitions in Sam's viewpoint are labeled by $\mathbb{MM}$. 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 $\mathbb{MM}$ transitions) is shown in Figure 12. 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 12 clearly reflects the result of merging Bob's and Mary's viewpoints. Assuming $\mathbb{TF}$, $\mathbb{FT}$, and $\top$ are the only inconsistent values, there are three cases of syntactic inconsistency in the pushout. The $\mathbb{TF}$ 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 $\mathcal{V}$ and also wants $x$ not to hold in a state $s'$ of a viewpoint $\mathcal{V}'$ 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

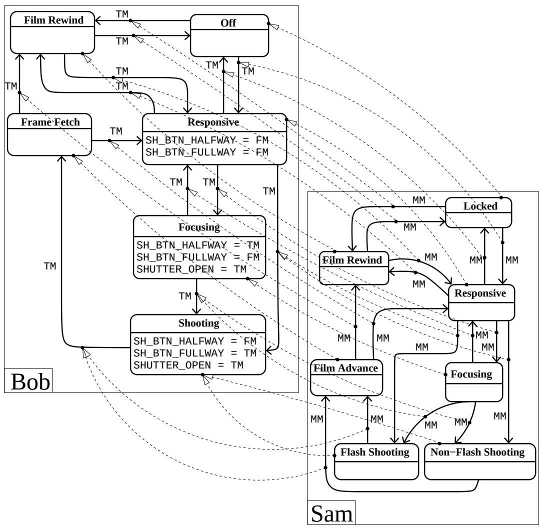


**Figure 11. Interconnecting the viewpoints**

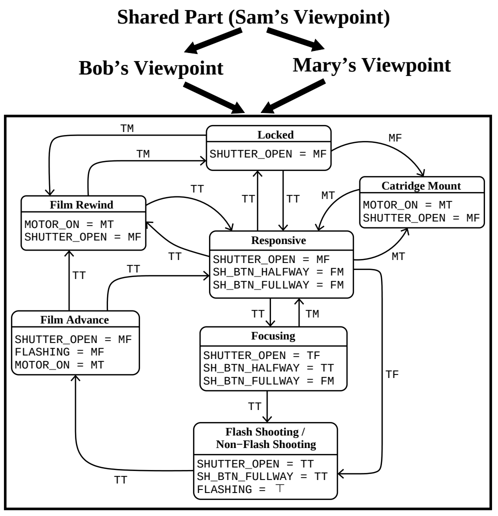


**Figure 12. Colimit computation**

simplest reason for coming up with FLASHING $= \top$ in the **Flash Shooting/Non-Flash Shooting** state of the pushout 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 colimiting object; however, the colimiting object may have some missing transitions. For example, if Bob saw a state **X** that Mary did not see, the pushout would have had no transition from **X** to **Cartridge Mount** and vice versa. This is natural, because **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 colimiting objects as $\mathbb{MM}$ transitions.

## 7 Conclusions and Future Work

We proposed a category-theoretic approach to representation and analysis of inconsistency in graph-based viewpoints. The main contribution of this work is providing an abstract mechanism for merging incomplete and inconsistent viewpoints and defining a notion of syntactic inconsistency. Our mathematically rigorous approach is a step toward a general framework for the composition of inconsistent viewpoints which has so far remained an open problem.

A major advantage of our proposed framework is its support for parameterizing inconsistency through lattices. This makes the framework suitable for different types of analysis. Another advantage that naturally results from the use of category theory for conceptualizing the merge process is 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, and studying possible ways to capture the evolution of the vocabulary of atomic propositions and the underlying knowledge ordering 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. Another interesting subject is using the framework presented in this paper for developing graph transformation systems based on the double-pushout approach [4]. In [17], some initial steps have been taken in this direction, but further work remains to be done.

**Acknowledgments.** We thank Wolfram Kahl for his extensive help in improving this paper, Andrzej Tarlecki for the proof of Lemma 3.4, Shiva Nejati for help with knowledge ordering lattices, members of the Formal Methods Group at U*of*T for their helpful remarks, Colin Potts for suggesting the case-study, and the anonymous referees for their insightful comments. Financial support was provided by NSERC.

## References

[1] M. Barr and C. Wells. *Category Theory for Computing Science*. Les Publications CRM Montréal, third edition, 1999.

[2] N. Belnap. A useful four-valued logic. In *Modern Uses of Multiple-Valued Logic*, pages 5–37. Reidel, 1977.

[3] M. Chechik, S. Easterbrook, and B. Devereux. Model checking with multi-valued temporal logics. In *Intl. Symposium on Multi-Valued Logics*, pages 187–192, 2001.

[4] A. Corradini et al. Algebraic approaches to graph transformation, part I. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, 1997.

[5] B. Davey and H. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, second edition, 2002.

[6] S. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *Intl. Conference on Software Engineering*, pages 411–420, 2001.

[7] S. Easterbrook and B. Nuseibeh. Using viewpoints for inconsistency management. *Software Engineering Journal*, 11:31–43, 1996.

[8] A. Finkelstein et al. Inconsistency handling in multi-perspective specifications. *IEEE Trans. on Software Engineering*, 20:569–578, 1994.

[9] M. Fitting. Kleene's logic, generalized. *Journal of Logic and Computation*, 1:797–810, 1992.

[10] J. Goguen. *Categories of Fuzzy Sets: Applications of Non-Cantorian Set Theory*. PhD thesis, University of California, Berkeley, 1968.

[11] J. Goguen. Concept representation in natural and artificial languages. *Intl. Journal of Man-Machine Studies*, 6:513–561, 1974.

[12] J. Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1:49–67, 1991.

[13] J. Goguen and R. Burstall. Some fundamental algebraic tools for the semantics of computation, part I. *Theoretical Computer Science*, 31:175–209, 1984.

[14] R. Heckel. *Open Graph Transformation Systems*. PhD thesis, Technical University of Berlin, 1998.

[15] R. Heckel et al. 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, 1999.

[16] D. Rydeheard and R. Burstall. *Computational Category Theory*. Prentice Hall, 1988.

[17] M. Sabetzadeh. A category-theoretic approach to representation and analysis of inconsistency in graph-based viewpoints. Master's thesis, University of Toronto, 2003.

[18] D. Smith. Constructing specification morphisms. *Journal of Symbolic Computation*, 16:571–606, 1993.

[19] A. Tarlecki. Bits and pieces of the theory of institutions. In *Summer Workshop on Category Theory and Computer Programming*, pages 334–363. Springer, 1986.