# Reasoning about Consistency in Model Merging

Mehrdad Sabetzadeh† Shiva Nejati† Marsha Chechik‡ Steve Easterbrook‡

†Simula Research Laboratory
Oslo, Norway
{mehrdad,shiva}@simula.no

‡ University of Toronto
Toronto, Canada
{chechik,sme}@cs.toronto.edu

## ABSTRACT

Models undergo a variety of transformations throughout development. One of the key transformations is merge, used when developers need to combine a set of models with respect to the overlaps between them. A major question about model transformations in general, and merge in particular, is what consistency properties are preserved across the transformations and what consistency properties may need to be re-checked (and if necessary, re-established) over the result. In previous work [18], we developed a technique based on category-theoretic colimits for merging sets of inter-related models. The use of category theory leads to the preservation of the algebraic structure of the source models in the merge; however, this does not directly provide a characterization of the (in)consistency properties that carry over from the source models to the result, because consistency properties are predominantly expressed as logical formulas. Hence, an investigation of the connections between the "algebraic" and "logical" properties of model merging became necessary.

In this paper, we undertake such an investigation and use techniques from finite model theory [9] to show that the use of colimits indeed leads to the preservation of certain logical properties. Our results have implications beyond the merge framework in [18] and are potentially useful for the broad range of techniques in the graph transformation and algebraic specification literature that use colimits as the basis for model manipulations.

## Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

## General Terms

Design, Verification

## Keywords

Inconsistency, Merge, Logical Preservation



## 1. INTRODUCTION

In the past several years, we have been studying the problem of model integration, particularly in situations where the models are originating from distributed sources of information. Many activities in model-based development fall under the umbrella of integration. These include (1) *merging*, used to build a global view of a set of overlapping perspectives (e.g., [23, 20, 18, 10]); (2) *composition*, used to assemble a set of autonomous but interacting components that run sequentially or in parallel (e.g., [2, 5, 6]); and (3) *weaving*, used in aspect-oriented development to incorporate cross-cutting concerns into a base system (e.g., [22]).

Our position towards the integration problem has been that the integration operators (e.g., merge, compose, weave) must tolerate inconsistency. That is, the operators must work for *any* given set of models, even when the models are inconsistent. This position is motivated by two well-known observations: First, immediate resolution of inconsistency can be disruptive in projects where ambiguities and conflicts tend to occur frequently [11]. Second, maintaining consistency at all times can be counter-productive because it may lead to premature commitment to design decisions that have not yet been sufficiently analyzed [12].

In light of our position, it is important to understand how different consistency properties are affected by the integration operations. Specifically:

- If all the source models are consistent with respect to a given consistency property, will the integrated model be consistent with respect to that property as well?
- If there is an inconsistency in the source models, will the inconsistency necessarily carry over to the integrated model?

Answering the first question is interesting to enable compositional reasoning about consistency. Answering the second question is useful for understanding the nature of an inconsistency. In particular, inconsistencies that are due to incomplete information in the individual source models can be automatically resolved in the integrated models when the source models are complementary and address each other's areas of incompleteness. For example, an abstract class with no descendants in a UML class diagram might be seen as inconsistent. But this class might be inherited from in other models and hence the overall view might still be consistent. In contrast, cyclic inheritance in a UML class diagram cannot be resolved in the integrated model (unless the integrated model omits information from the source models).

Since consistency rules are often described in logical languages (e.g., first order logic), we are interested in studying how different integration operators preserve the logical properties of models. In general, property preservation is a powerful tool for reasoning about model transformations. The main question that property preservation tackles is the following: If a property (formula) $\varphi$ in some logic holds over a model $M$, will $\varphi$ also hold over a model $M'$ derived from $M$ via some transformation?

In this paper, we discuss the logical property preservation characteristics of our merge operator in [18]. The merge operator is based on category theory which has been widely used as a theoretical basis for characterizing model merging. In a categorical setting, merge is typically performed by computing a colimit – an algebraic construct for combining a set of objects interrelated by a set of mappings. While colimits provide an effective and mathematically precise way for merge, their pure algebraic characterization is not directly applicable for reasoning about the logical properties of model merging. Specifically, given a property $\varphi$ expressed in a particular logic, one cannot readily determine from the definition of colimit whether $\varphi$ is preserved from the source models to the merged model.

We use techniques from finite model theory [9] to show that colimits indeed preserve a certain class of logical properties. The logical language we use as the basis for our work is first order logic extended with least fixpoints. Extension with fixpoints is important, because standard first order logic cannot express properties that require the computation of reachability. For example, acyclic inheritance for UML class diagrams is not expressible in standard first order logic. Our results have implications beyond our merge framework in [18] and are potentially useful for the broad range of techniques in the graph transformation [16] and algebraic specification [1] communities that use colimits as the basis for model manipulations.

The remainder of this paper is structured as follows: In Section 2, we provide background information on our merge algorithm and present the logical preliminaries for our work. In Section 3, we describe our general logical preservation results for colimits; and in Section 4, we use these general results to reason about the preservation of some logical expressions that are frequently used in consistency rules. We conclude in Section 5 with a summary and directions for future work.

# 2. BACKGROUND

## 2.1 Structural Model Merging

We first briefly review our merge operator. For more information, see [18]. The operator hinges on three abstractions: *models*, *mappings*, and *interconnection diagrams*. Each model is described as a graph, and each mapping – as a binary relation over two models equating their corresponding elements. Mappings preserve type information, i.e., they do not equate elements that have different types. Further, they preserve structure, i.e., if a mapping $R$ maps an edge $e$ to an edge $e'$, it must also map the source and target of $e$ to the source and target of $e'$, respectively.

The third abstraction, the interconnection diagram, captures a set of models and a set of known or hypothesized mappings between them. An example interconnection diagram is shown in Figure 1. In this example, $M_1, \ldots, M_4$ are simple UML class diagrams with their overlapping parts specified through four mappings, $R_1, \ldots, R_4$ (depicted using directed dashed lines). A simpler example with just two models, $M_1, M_2$, and one mapping, $R$, is shown in Figure 3(a). A third example is given in Figure 4(a), where the shared parts of two models, $M_2, M_3$, are captured by a third model $M_1$ and two mappings $R_1$, $R_2$.

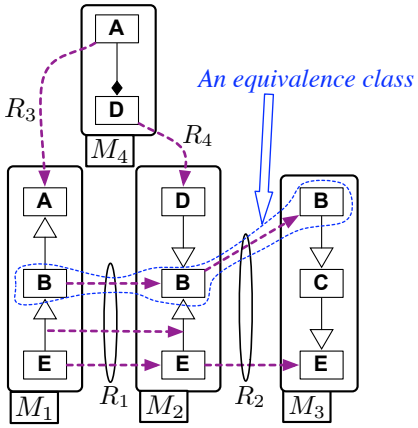


**Figure 1: Example interconnection diagram**

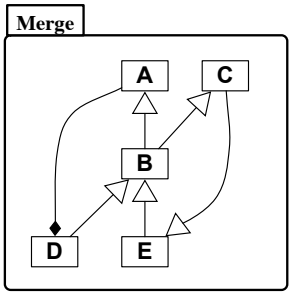


**Figure 2: Merge of the diagram in Figure 1**

The input to the merge algorithm is an interconnection diagram $D = \langle M_1, \ldots, M_i, R_1, \ldots, R_j \rangle$. The algorithm works by unifying elements in $M_1, \ldots, M_i$ that fall into the same equivalence class induced by $R_1, \ldots, R_j$. As an example, we have delineated by thin dashed lines one of the several equivalence classes in Figure 1. Note that each unmapped element in the input models falls into a distinct equivalence class of its own.

For convenience, in the example shown in Figure 1, we used a consistent vocabulary for naming the elements of $M_1, \ldots, M_4$, hence defining $R_1, \ldots, R_4$ based on name equalities. In general, models may not have a common vocabulary, and mappings are not necessarily based on vocabulary similarities (e.g., see the examples in Figures 3(a) and 4(a)).

The merged model has exactly one element corresponding to each equivalence class. Since mappings denote equality of mapped element pairs and hence are symmetric, the directionality of mappings is ignored in the computation of equivalence classes.

Figure 2 shows the resulting merge for the interconnection diagram of Figure 1. The merge provides interesting insights about how consistency properties can be broken across merge. For example, we may have consistency rules that check for multiple or cyclic inheritance in UML class diagrams. Obviously, these rules are satisfied over the in-

dividual source models in Figure 1, but the global view of the system (i.e., the merge) is inconsistent. In particular, in Figure 2: B has two parents; and B, C, E form a cycle. In Section 4, we provide a systematic explanation of what properties of the source models carry over to the merge and what properties do not.

## 2.2 Logical Background

First Order (FO) logic is one of the most commonly used logical languages for expressing consistency rules [11] and is used as the basis for our work here. FO by itself is not expressive enough to describe properties that involve reachability or cycles. To address this limitation, one can add to FO a least fixpoint operator, obtaining the least fixpoint logic (LFP) [9]. Below, we first formally define the notion of relational structure and FO. We then define the concept of least fixpoint and show how FO can be extended into LFP. Our exposition follows the standard approach in finite model theory (e.g., see [9]).

**Definition 2.1 (relational structure)** A *(relational) structure* is an object $\mathfrak{A} = (A, R_1, \ldots, R_m)$, where $A$ is a nonempty set, $m$ is a natural number, $R_1, \ldots, R_m$ are abstract relation symbols with associated arities $k_1, \ldots, k_m$ (nonnegative integers), and each $R_i$ is a $k_i$-ary relation on $A$.

The set $A$ is called the *universe* of $\mathfrak{A}$. The sequence of relation symbols $R_1, \ldots, R_m$ together with corresponding arities $k_1, \ldots, k_m$ comprise the *vocabulary* of $\mathfrak{A}$. We usually denote a vocabulary by $\sigma$. Relation $R_i^{\mathfrak{A}}$ is called the *interpretation* of a relation symbol $R_i$ in $\mathfrak{A}$.

FO formulas in a vocabulary $\sigma$ are built up from atomic formulas using negation, conjunction, disjunction, and existential and universal quantification:

$$\varphi \quad ::= \quad x = y \mid R(x_1, \ldots, x_n) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid$$

$$\exists x\varphi(x) \mid \forall x\varphi(x)$$

In the above, $x, y$ and $x_1 \ldots, x_n$ are variables, $R$ is an $n$-ary relation symbol in $\sigma$, and $\varphi_1$ and $\varphi_2$ are formulas.

Given a set $U$, let $\mathcal{P}(U)$ denote its powerset. A set $X \subseteq U$ is said to be a *fixpoint* of a mapping $F : \mathcal{P}(U) \to \mathcal{P}(U)$ if $F(X) = X$. A set $X \subseteq U$ is a *least fixpoint* of $F$ if it is a fixpoint, and for every other fixpoint $Y$ of $F$, we have $X \subseteq Y$. The least fixpoint of $F$ is denoted by $\mathbf{lfp}(F)$. Least fixpoints are guaranteed to exist only if $F$ is *monotone*. That is,

$$X \subseteq Y \text{ implies } F(X) \subseteq F(Y).$$

We now add a least fixpoint operator to FO. Suppose we have a vocabulary $\sigma$, and an additional relation symbol $R \notin \sigma$ of arity $k$. Let $\varphi(R, x_1, \ldots, x_k)$ be a formula with vocabulary $\sigma \cup \{R\}$. For a structure $\mathfrak{A}$ with vocabulary $\sigma$, the formula $\varphi(R, \vec{x})$ yields a mapping $F_\varphi : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ defined as follows:

$$F_\varphi(X) = \{\vec{a} \mid \mathfrak{A} \models \varphi(X/R, \vec{a})\}$$

The notation $\varphi(X/R, \vec{a})$ means that $X$ is substituted for $R$ in $\varphi$. More precisely, if $\mathfrak{A}'$ is a $(\sigma \cup \{R\})$-structure expanding $\mathfrak{A}$, in which $R$ is interpreted as $X$, then $\mathfrak{A}' \models \varphi(\vec{a})$.

To ensure that $F_\varphi$ is monotone, we impose certain restrictions. Given a formula $\varphi$ that may contain a relation symbol $R$, we say that an occurrence of $R$ is *negative* if it is under the scope of an odd number of negations, and *positive*, otherwise. We say that a formula is *positive in $R$* if there are no negative occurrences of $R$ in it, i.e., either all occurrences of $R$ are positive, or there are none at all.

**Lemma 2.2** *[3] If $\varphi(R, \vec{x})$ is positive in $R$, then $F_\varphi$ is monotone.*

**Definition 2.3 (least fixpoint logic)** [9] The *least fixpoint logic (LFP)* extends FO with the following formula building rule:

- if $\varphi(R, \vec{x})$ is a formula positive in $R$, where $R$ is $k$-ary, and $\vec{t}$ is a tuple of terms, where $|\vec{x}| = |\vec{t}| = k$, then

  $$[\mathbf{lfp}_{R,\vec{x}}\varphi(R, \vec{x})](\vec{t})$$

  is a formula, whose free variables are those of $\vec{t}$.

The semantics is defined as follows:

$$\mathfrak{A} \models [\mathbf{lfp}_{R,\vec{x}}\varphi(R, \vec{x})](\vec{a}) \quad \text{iff} \quad \vec{a} \in \mathbf{lfp}(F_\varphi).$$

**Example 2.4 (reachability)** Consider graphs whose edge relation is $E$, and let

$$\varphi(R, x, y) = E(x, y) \vee \exists z \left(E(x, z) \wedge R(y, z)\right).$$

Reachability, i.e., the transitive closure of $E$, is characterized by the formula

$$\psi(x, y) = [\mathbf{lfp}_{R,x,y}\varphi(R, x, y)](x, y).$$

That is, $\psi(a, b)$ holds over a graph $G$ iff there is a path from $a$ to $b$ in $G$.

## 2.3 Homomorphisms and Preservation of Logical Properties

Our merge framework embeds each source model into the merge through a homomorphism. The existence of these homomorphisms leads to the preservation of certain consistency properties. Below, we review the theoretical results underlying our discussion of property preservation in Section 3. We begin with a definition of homomorphism:

**Definition 2.5 (homomorphism)** Let $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_m^{\mathfrak{A}})$ and $\mathfrak{B} = (B, R_1^{\mathfrak{B}}, \ldots, R_m^{\mathfrak{B}})$ be structures in the same vocabulary. A *homomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ is a function $h : A \to B$ such that $h(R_i^{\mathfrak{A}}) \subseteq R_i^{\mathfrak{B}}$, i.e., if $(a_1, \ldots, a_{k_i}) \in R_i^{\mathfrak{A}}$ then $(h(a_1), \ldots, h(a_{k_i})) \in R_i^{\mathfrak{B}}$ for every $1 \leq i \leq m$.

The first result about property preservation under homomorphisms, dating back to the 1950's, is the Los-Tarski-Lyndon Theorem:

**Theorem 2.6 (homomorphism preservation theorem)** *(e.g., see [14, 15]) A first order formula is preserved under homomorphisms on all structures (finite and infinite) if and only if it is equivalent to an existential positive formula, i.e., a formula without negation and universal quantification.*

The existential positive fragment of FO is denoted $\exists\text{FO}^+$. For our purposes, we are interested in finite structures only, and like many classical mathematical logic results that fail in the finite case (e.g., compactness), there is the danger that the above result may fail as well when restricted to finite structures. Fortunately, this is not the case.

**Theorem 2.7 (h. p. t. in the finite)** *[15] A first order formula is preserved under homomorphisms on finite structures if and only if it is equivalent to an $\exists FO^+$ formula.*

The forward direction of the if-and-only-if (i.e., sufficiency) in the above result can be extended to the existential positive fragment of LFP, denoted $\exists\text{LFP}^+$.

**Lemma 2.8** *Every $\exists LFP^+$ formula is preserved under homomorphisms on finite structures.*

A proof of the above lemma is provided in the appendix. The lemma is the basis for the results we present in Section 3.

# 3. GENERAL PRESERVATION RESULTS

Our merge framework offers three key features [18]:

*F1* Merge yields a family of mappings, in our case graph homomorphisms, one from each source model onto the merged model. This feature ensures that the merge does not loose information, i.e., it represents all the source models completely.

*F2* The merged model does not contain any unmapped elements, i.e., every element in the merged model is the image of some element in the source models. This feature ensures minimality, i.e., the merge does not introduce information that is not present in or implied by the source models.

*F3* Merge respects the mappings in the source system, i.e., the image of each element in the merged model remains the same, no matter which path through the mappings in the source system one follows. This feature ensures non-reduandancy. More precisely, if a concept appears in more than one source model, only one copy of it appears in the merged model.

From *F1* and Lemma 2.8 (in Section 2.3), it follows that the result of our merge procedure preserves the existential positive fragment of LFP.

**Theorem 3.1** *If an existential positive LFP formula $\varphi$ is satisfied by* some *source model $M$, any merge in which $M$ participates satisfies $\varphi$ as well.*

By *F2* and the above theorem, we obtain the following result regarding preservation of universal properties.

**Theorem 3.2** *Let $\varphi(x)$ be an existential positive LFP formula with a free variable $x$. If the formula $\psi = \forall x\, \varphi(x)$ is satisfied by* all *the source models, $\psi$ is satisfied by any merge of the models as well.*

Notice that Theorem 3.2 allows the introduction of *only one* universal quantifier. To gain intuition on what happens when additional universal quantifiers are introduced, consider the system in Figure 3(a) and let the relation $E(x, y)$ denote the graph edge relation. Both models in Figure 3(a) are complete graphs and hence satisfy the property $\forall x\, \forall y\ \mathsf{Node}(x) \wedge \mathsf{Node}(y) \Rightarrow E(x, y)$ [1]. However, the property is violated over the resulting merge shown in Figure 3(b), because there is no edge from node a to node d and vice versa. The general observation here is that, when there is more than one universal quantifier, universally quantified variables can be assigned values from non-shared parts of *different* source models. In such a case, property satisfaction over the individual source models may not extend to the merge.

Currently, we do not know whether *F3* leads to further property preservation results. This is an issue that we plan to investigate in future work.

[1]The property uses implication and hence has negation. But the negation can be resolved, because every element in the universe that is not a node is an edge. Therefore, the property is equivalent to $\forall x\, \forall y\ \mathsf{Edge}(x) \vee \mathsf{Edge}(y) \vee E(x, y)$.

# 4. PRESERVATION OF (IN)CONSISTENCY

In previous work [19], we identified three general types of expressions commonly used in structural consistency properties. These are:

- *Compatibility expressions*, used for ensuring compatibility of the type of an edge with the types of its endpoints.
- *Multiplicity expressions*, used for defining a minimum and a maximum number for edges of a given type incident to a node.
- *Reachability expressions*, used for checking existence of paths of edges of a given type between two nodes.

Below, we use results of Section 3 to reason about the preservation of these expressions.

*Compatibility Properties.* Preservation of compatibility properties can be established directly through algebraic means, but it is interesting to see if the same can be done through logical means. For example, in a class diagram, an edge of type "implements" must relate a class to an interface; otherwise, the diagram would not be well-formed. This property can be formalized as follows:

$$\mathbf{C1} \quad = \quad \forall e\ (\mathsf{Edge}(e) \wedge \mathsf{Type}(e, \text{“implements”}) \Rightarrow \mathbf{Compatible}_{\mathsf{class,interface}}(e))$$

where Edge is the set of edges of the graph representing a class diagram, Type is a binary relation between the set of edges and different types of relations between classes, and $\mathbf{Compatible}_{\mathsf{class,interface}}(\mathsf{e})$ is a constraint that verifies whether the source and target nodes of edge e are of type class and interface, respectively. The general form of this constraint can be formalized using an existential positive formula as follows:

$$\mathbf{Compatible}_{\alpha,\beta}(e) \quad = \quad \exists n\ (\exists m\ (\mathsf{Source}(e, n) \wedge \mathsf{Target}(e, m) \Rightarrow \mathsf{Type}(n, \alpha) \wedge \mathsf{Type}(m, \beta)))$$

where Source and Target are binary relations, respectively giving the source and target node for a given edge.

The sub-formula Type(e, "implements") of **C1** appears in negated form, but the negation can be resolved, knowing that (1) the set of types is fixed and, (2) every element has a type. More precisely, if the set of types is $\{t_1, \ldots, t_n\}$, the formula $\neg\mathsf{Type}(e, t_\ell)$ can be replaced with $\bigvee_{i \neq \ell} \mathsf{Type}(e, t_i)$. Hence, by Theorem 3.2, **C1** is preserved.

*Multiplicity Properties.* One can show through simple counterexamples that *none* of the following lift from the source models to the merge:

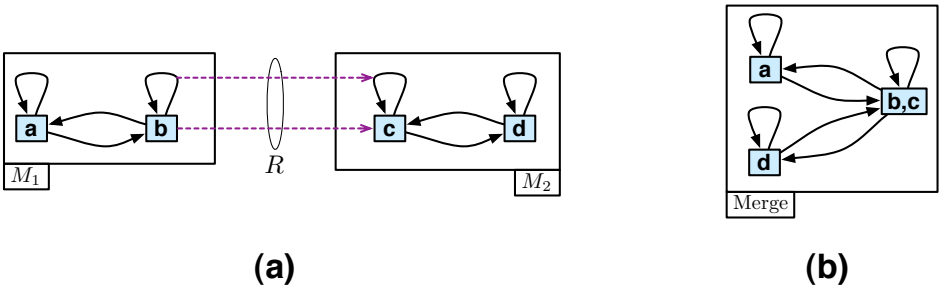


**Figure 3: Illustration for violation of universal properties**

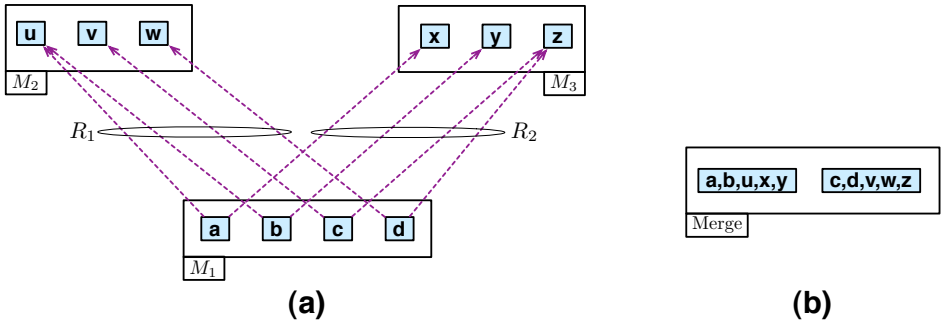


**Figure 4: Illustration for violation of multiplicity properties**

- There exists at least $c$ elements satisfying $\varphi$.
- There exists exactly $c$ elements satisfying $\varphi$.
- There exists at most $c$ elements satisfying $\varphi$.

It is easy to see why the "exactly" and "at most" cases do not get preserved, noting that merge normally has more information than any of the source models. To understand why the "at least" case is not preserved, note that homomorphisms (and functions as well) are not necessarily one-to-one, and can therefore *shrink* the number of elements satisfying a property. For example, consider the system of models in Figure 4(a) and its merge in Figure 4(b). For simplicity, the models are discrete graphs, i.e., sets, and their mappings are functions. Although $M_1, M_2, M_3$ all satisfy the property "there exists at least three (distinct) nodes", the merge has only two nodes, hence violating the property. It is important to mention that the flexibility to fuse together multiple elements of the same source model is not an undesirable feature and is indeed valuable when one needs to perform an abstraction during merge [7].

*Reachability Properties.* An interesting consequence of Theorem 3.1 is the preservation of paths in the merge. Recall that we gave a formalization for the reachability property in Example 2.4. To see how this can be used for reasoning about consistency, consider the following consistency rule over class diagrams: "Every abstract class must have a concrete implementation". This rule is formally expressed as follows:

$$\textbf{C2} \quad = \quad \forall c\ ((\mathsf{Type}(c, \text{“class”}) \wedge \mathsf{Abstract}(c)) \Rightarrow \exists c'(\mathsf{Concrete}(c') \wedge \textbf{Reachable}_{\mathsf{extends}}(c', c)))$$

where $\textbf{Reachable}_{\mathsf{extends}}(x, y)$ holds iff a path from $x$ to $y$ made up of edges of type "extends" exists. Using the argument we gave when discussing preservation of compatibility properties, we know that the negation of Type(c1, "class") can be resolved. Further, ¬Abstract(c1) can be replaced with a positive property, say, Concrete(c1). It now follows from Theorem 3.2 that **C2** is preserved.

Note that our results can be used for reasoning about preservation of inconsistency as well. For example, consider the following rule:

$$\textbf{C3} \quad = \quad \exists c\ ((\mathsf{Type}(c, \text{“class”}) \wedge \textbf{Reachable}_{\mathsf{extends}}(c, c)))$$

This rule holds over a class diagram $M$ when the inheritance hierarchy in $M$ is cyclic, i.e., $M$ is inconsistent. By Theorem 3.1, we can conclude that any merged model that has $M$ as a source model satisfies **C3** as well, and hence is also inconsistent.

## 5. CONCLUSION

In this paper, we showed that the use of algebraic colimits for model merging leads to the preservation of certain logical properties. We used our results to formally reason about the preservation of consistency properties across merge.

Based on our recent survey of existing model merging techniques [17], algebraic theories, including category theory, lattice theory, and formal concept analysis, are increasingly being used for characterizing model integration problems. What makes these theories particularly attractive is the level of abstraction to which they lead, allowing the merge process to be described in a flexible and highly generic way. At the same time, one must account for the fact that merge is often an intermediate step for activities such as behavioural synthesis [23, 21], reasoning over global behaviours of systems [4], and data integration and exchange [8, 13].

To facilitate these activities, it is crucial to be able to reason about the preservation of semantic properties (including consistency properties) of the source models in merge. Doing so requires establishing proper connections between the algebraic techniques used in model integration and the logical techniques used in the activities named above. This is a non-trivial task but is an essential step toward making model integration more effective in practice.

For future work, we would like to provide a full logical characterization of colimits. In particular, the logical implications of $F3$, described in Section 3, is unknown to us at the moment and need to be revisited in the future. Further, it may be possible to trade off development flexibility in favour of a broader class of preserved properties, e.g., by using more constrained mappings for relating models, or by placing restrictions on the patterns used for interconnecting the models. We leave an elaboration of these topics to future investigation. Lastly, we would like to explore the application of our results for checking the consistency of model manipulations in graph transformation and algebraic specification approaches that are based on colimits.

# 6. REFERENCES


[1] E. Astesiano, H. Kreowski, and B. Krieg-Brueckner, editors. *Algebraic Foundations of Systems Specification*. Springer-Verlag, Secaucus, NJ, USA, 1999.

[2] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[3] B. Davey and H. Priestly. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[4] S. 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.

[5] J. Hay and J. Atlee. "Composing Features and Resolving Interactions". In *SIGSOFT '00/FSE-8: Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 110–119, 2000.

[6] M. Jackson and P. Zave. "Distributed Feature Composition: a Virtual Architecture for Telecommunications Services". *IEEE Transactions on Software Engineering*, 24(10):831 –847, 1998.

[7] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: The state of the art. In Y. Kalfoglou, M. Schorlemmer, A. Sheth, S. Staab, and M. Uschold, editors, *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminar Proceedings. IBFI, 2005.

[8] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the 21st Symposium on Principles of Database Systems*, pages 233–246, 2002.

[9] L. Libkin. *Elements Of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[10] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. "Matching and Merging of Statecharts Specifications". In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 54–64, 2007.

[11] C. Nentwich, W. Emmerich, and A. Finkelstein. "Consistency Management with Repair Actions". In *ICSE '03: Proceedings of the 25 International Conference on Software Engineering*, pages 455–464, 2003.

[12] B. Nuseibeh, S. Easterbrook, and A. Russo. "Making Inconsistency Respectable in Software Development". *The Journal of Systems and Software*, 58(2):171–180, 2001.

[13] L. Popa, Y. Velegrakis, R. Miller, M. Hernández, and R. Fagin. Translating web data. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 598–609, 2002.

[14] E. Rosen. Some aspects of model theory and finite structures. *The Bulletin of Symbolic Logic*, 8(3):380–403, 2002.

[15] B. Rossman. Existential positive types and preservation under homomorphisisms. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, pages 467–476, 2005.

[16] G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation: Foundations*, volume 1. World Scientific, River Edge, NJ, USA, 1997.

[17] M. Sabetzadeh. *Merging and Consistency Checking of Distributed Models*. PhD thesis, University of Toronto, 2008.

[18] M. Sabetzadeh and S. Easterbrook. View merging in the presence of incompleteness and inconsistency. *Requirements Engineering Journal*, 11(3):174–193, 2006.

[19] M. Sabetzadeh, S. Nejati, S. Liaskos, S. Easterbrook, and M. Chechik. "Consistency Checking of Conceptual Models via Model Merging". In *RE '07: Proceedings of 15th IEEE International Requirements Engineering Conference*, pages 221–230, 2007.

[20] S. Uchitel and M. Chechik. "Merging Partial Behavioural Models". In *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 43–52, 2004.

[21] S. Uchitel and J. Kramer. A workbench for synthesising behaviour models from scenarios. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 188–197, 2001.

[22] J. Whittle, A. Moreira, J. Araújo, P. Jayaraman, A. Elkhodary, and R. Rabbi. "An Expressive Aspect Composition Language for UML State Diagrams". In *MoDELS '07: Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems*, pages 514–528, 2007.

[23] J. Whittle and J. Schumann. "Generating Statechart Designs from Scenarios". In *ICSE '00: Proceedings of 22nd International Conference on Software Engineering*, pages 314–323. ACM Press, May 2000.

# APPENDIX

## A. PROOF FOR LEMMA 2.8

Let $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_m^{\mathfrak{A}})$ and $\mathfrak{B} = (B, R_1^{\mathfrak{B}}, \ldots, R_m^{\mathfrak{B}})$ be a pair of relational structures over vocabulary $\sigma = (R_1, \ldots, R_m)$. Let $h : \mathfrak{A} \to \mathfrak{B}$ be a homomorphism. We show that for every $\varphi \in \exists LFP^+$ and for every $\vec{a} \in A^k$

$$\mathfrak{A} \models \varphi(\vec{a}) \Rightarrow \mathfrak{B} \models \varphi(h(\vec{a}))$$

where $h(\vec{a}) = (h(a_1), \ldots, h(a_k))$.

The proof for $\varphi \in \exists FO^+ \cap \exists LFP^+$ follows from Theorem 2.7. Below, we provide a proof for least fixpoint formulas.

Let $\varphi(\vec{x}) = [\mathbf{lfp}_{R,\vec{y}}\alpha(R, \vec{y})](\vec{x})$. By the definition of **lfp**, for every structure $\mathfrak{A}$, the formula $\varphi$ yields a mapping $F_{\alpha,\mathfrak{A}} : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ defined as follows:

$$F_{\alpha,\mathfrak{A}}(X) = \{\vec{a} \mid \mathfrak{A} \models \alpha(X/R, \vec{a})\}$$

By Definition 2.3 and Knaster-Tarski's fixpoint theorem, for every $\vec{a} \in A^k$ we have:

$$\vec{a} \in \bigcup_{i=0}^{\infty} F_{\alpha,\mathfrak{A}}^i(\emptyset) \Leftrightarrow \mathfrak{A} \models [\mathbf{lfp}_{R,\vec{y}}\alpha(R, \vec{y})](\vec{a})$$

We first prove by induction that $h(F_{\alpha,\mathfrak{A}}^i(\emptyset)) \subseteq F_{\alpha,\mathfrak{B}}^i(\emptyset)$.

**Base case:** Let $\vec{a} \in F_{\alpha,\mathfrak{A}}(\emptyset)$. Then, $\mathfrak{A} \models \alpha(\emptyset, \vec{a})$. Since $\mathfrak{A}$ is a substructure of $\mathfrak{B}$ by $h$ and since $h(\emptyset) = \emptyset$, we have $\mathfrak{B} \models \alpha(\emptyset, h(\vec{a}))$. Thus, $h(\vec{a}) \in F_{\alpha,\mathfrak{B}}(\emptyset)$.

**Inductive step:** Let $\vec{a} \in F_{\alpha,\mathfrak{A}}^i(\emptyset)$. Then, $\mathfrak{A} \models \alpha(F_{\alpha,\mathfrak{A}}^{i-1}(\emptyset), \vec{a})$. Since $\mathfrak{A}$ is a substructure of $\mathfrak{B}$ by $h$, we have $\mathfrak{B} \models \alpha(h(F_{\alpha,\mathfrak{A}}^{i-1}(\emptyset)), h(\vec{a}))$. Thus, $h(\vec{a}) \in F_{\alpha,\mathfrak{B}}(h(F_{\alpha,\mathfrak{A}}^{i-1}(\emptyset)))$. By the inductive hypothesis and since $F_{\alpha,\mathfrak{B}}$ is monotone, $h(\vec{a}) \in F_{\alpha,\mathfrak{B}}^i(\emptyset)$.

Thus,

$h(\bigcup_{i=0}^{\infty} F_{\alpha,\mathfrak{A}}^i(\emptyset)) \subseteq \bigcup_{i=0}^{\infty} F_{\alpha,\mathfrak{B}}^i(\emptyset)$ **(1)**

Therefore,

$\mathfrak{A} \models \varphi(\vec{a})$ $\Leftrightarrow$
(By assumption $\varphi(\vec{a}) = [\mathbf{lfp}_{R,\vec{y}}\alpha(R, \vec{y})](\vec{a})$)
$\mathfrak{A} \models [\mathbf{lfp}_{R,\vec{y}}\alpha(R, \vec{y})](\vec{a})$ $\Leftrightarrow$
(By Definition 2.3 and Knaster-Tarski's Theorem)
$\vec{a} \in \bigcup_{i=0}^{\infty} F_{\alpha,\mathfrak{A}}^i(\emptyset)$ $\Leftrightarrow$
(Since $h$ is homomorphism)
$h(\vec{a}) \in h(\bigcup_{i=0}^{\infty} F_{\alpha,\mathfrak{A}}^i(\emptyset))$ $\Rightarrow$
(By **(1)**)
$h(\vec{a}) \in \bigcup_{i=0}^{\infty} F_{\alpha,\mathfrak{B}}^i(\emptyset)$ $\Leftrightarrow$
(By Definition 2.3 and Knaster-Tarski's Theorem)
$\mathfrak{B} \models [\mathbf{lfp}_{R,\vec{y}}\alpha(R, \vec{y})](h(\vec{a}))$ $\Leftrightarrow$
(By definition of **lfp**)
$\mathfrak{B} \models \varphi(h(\vec{a}))$