Towards Development with Multi-Version Models: Detecting Merge Conflicts and Checking Well-Formedness

Matthias Barkowsky and Holger Giese

Hasso-Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany
{matthias.barkowsky,holger.giese}@hpi.de

Abstract. Developing complex software requires that multiple views and versions of the software can be developed in parallel and merged as supported by views and managed by version control systems. In this context, this paper considers monitoring merging and related consistency problems permanently at the level of models and abstract syntax to permit early and frequent conflict detection while developing in parallel. The presented approach introduces multi-version models based on typed graphs that permit to store changes and multiple versions in one graph in a compact form and allow (1) to study well-formedness for all versions without the need to extract each version individually, (2) to report all possible merge conflicts without the need to merge all pairs of versions, and (3) to report all violations of well-formedness conditions that will result for merges of any two versions independent of any merge decisions without the need to merge all pairs of versions. The paper defines the related concepts and algorithms operating on multi-version models, proves their correctness w.r.t. the usually employed three-way-merge, and reports on preliminary experiments concerning the scalability.

1 Introduction

Developing complex software nowadays requires that multiple views and versions of the software can be developed in parallel and merged as supported by views and managed by version control systems [10]. For complex software, living with inconsistencies at least temporarily is inevitable, as enforcing consistency may lead to loss of important information [9] and is hence neither always possible nor desirable. However, working with multiple versions in parallel and changing each version on its own for longer periods of time can introduce substantial conflicts

* This work was developed mainly in the course of the project modular and incremental Global Model Management (project number 336677879) funded by the DFG. This preprint has not undergone peer review (when applicable) or any post-submission improvements or corrections. The Version of Record of this contribution is published in Graph Transformation, 15th International Conference, Proceedings, and is available online at https://doi.org/10.1007/978-3-031-09843-7
that are difficult and expensive to resolve. Therefore, it is necessary to manage consistency when combining views and versions using merge approaches \[10,16\].

In this context, this paper considers monitoring merging and related consistency problems permanently at the level of models and abstract syntax. This aims to permit early and frequent conflict detection while developing in parallel, as suggested in approaches to detect conflicts early and to enable collaboration to manage conflicts and their risks \[5\].

The presented approach therefore introduces multi-version models based on typed graphs, which permit to store changes and multiple versions in one graph in a compact form and allow to study the different versions and their merge combinations. The following capabilities are considered: (1) Study well-formedness for all versions at once without the need to extract and explicitly consider each version individually. (2) Report all possible merge conflicts that may result for merges of any two versions without the need to extract and explicitly merge all pairs of versions. (3) Report all violations of well-formedness conditions that will result for merges of any two versions independent of any merge decisions without the need to extract and explicitly merge all pairs of versions.

The approach thus promises to support early conflict detection and collaboration for managing conflicts and their risks, while not having to decide how to later merge conflicting versions. The technique also aims for a better scalability in case there are many versions that are considered in parallel.

Furthermore, the developed multi-version models permit to study the phenomena of versions, merging, and well-formedness conditions in the unifying framework of typed graphs. This enables us to (a) formulate algorithms that can obtain several analysis results without the need to consider a specific version, merge of a pair of versions, or strategy for conflict resolution and (b) prove that the algorithms compute the same results as if we would explicitly consider all specific versions, merges of pairs of versions, or strategies for conflict resolution.

The paper defines the related concepts and algorithms operating on multi-version models, proves their correctness w.r.t. the usually employed three-way-merge, and reports on first experiments concerning the scalability. In Section 2, we summarize the preliminaries of the presented approach, including basic definitions for typed graphs, well-formedness conditions, and graph modifications. Then, as a baseline, single-version models in the form of typed graphs with well-formedness conditions are defined in Section 3, before multi-version models are introduced in Section 4. Determining all merge conflicts and checking well-formedness for all merge results based on multi-version models is then considered in Section 5. Results of first experiments for our prototypical implementation of the algorithms are presented in Section 6. Finally, the conclusions of the paper and an outlook of planned future work are presented in Section 7. For the submitted review version, more detailed proofs and the conditions used in the evaluation are in addition presented in an appendix.
2 Preliminaries

We briefly reiterate the basic concepts of graphs, graph modifications, and well-formedness conditions used in the remainder of the paper.

A graph $G = (V^G, E^G, s^G, t^G)$ consists of a set of nodes $V^G$, a set of edges $E^G$ and two functions $s^G : E^G \rightarrow V^G$ and $t^G : E^G \rightarrow V^G$ assigning each edge its source and target, respectively. We assume that graph elements have identities and source and target of an edge are invariant if an edge is part of multiple graphs, that is, for two graphs $G$ and $H$ and an edge $e \in E^G \cap E^H$, it holds that $s^G(e) = s^H(e)$ and $t^G(e) = t^H(e)$. This also implies that, in the context of this paper, $(V^G = V^H \land E^G = E^H) \rightarrow (G = H)$.

A graph morphism $m : G \rightarrow H$ is given by a pair of functions $m^V : V^G \rightarrow V^H$ and $m^E : E^G \rightarrow E^H$ that map elements from $G$ to elements from $H$ such that $s^H \circ m^E = m^E \circ s^G$ and $t^H \circ m^E = m^E \circ t^G$.

A graph $G$ can be typed over a type graph $TG$ via a typing morphism $type : G \rightarrow TG$, forming the typed graph $G^T = (G, type^G)$. A typed graph morphism between two typed graphs $G^T = (G, type^G)$ and $H^T = (H, type^H)$ with the same type graph then denotes a graph morphism $m^T : G \rightarrow H$ such that $type^G = type^H \circ m^T$. A (typed) graph morphism is a monomorphism iff its functions are both surjective and injective.

Figure 1 shows an example typed graph $M_1$ and associated type graph $TM$ from the software development domain. $M_1$ represents an abstract syntax graph for a program written in an object-oriented language that contains four classes represented by nodes, with edges representing superclass relationships.

The structure of a typed graph $G$ can be restricted by a well-formedness condition $\phi$, which in the context of this paper is characterized by a typed graph $Q$ typed over the same type graph. $G$ then satisfies the condition $\phi$, denoted $G \models \phi$, iff there exists no monomorphism $m : Q \rightarrow G$. We also call such monomorphisms matches and $Q$ the violation pattern of $\phi$.

Figure 1 shows a violation pattern $Q$ for an example well-formedness constraint that forbids a class having two outgoing superclass relationships.

A graph modification as defined by Taentzer et al. [20] formalizes the difference between two graphs $G$ and $H$ and is characterized by an intermediate graph $K$ and a span of monomorphisms $(G \leftarrow K \rightarrow H)$. In this paper, we assume that the two morphisms are always partial identities. The graph $K$ then characterizes
the subgraph that is preserved through the modification, whereas all elements in \( G \) that are not in \( K \) are deleted and elements in \( H \) but not in \( K \) are created.

Figure 2 shows an example graph modification from the graph \( M_1 \) from Figure 1 to a new graph \( M_2 \), where a superclass edge from class \( c_1 \) to class \( c_3 \) is created and the class \( c_4 \) is deleted. The morphisms are implied by node labels.

Graphs and graph modifications correspond to versions and differences in conventional, line-based version control systems like Git [2], where versions of a development artifact and intermediate differences form a directed acyclic graph.

3 Single-Version Models

In this paper, we consider models in the form of typed graphs that are required to adhere to a set of well-formedness conditions. Effectively, the combination of type graph and well-formedness conditions then acts as a metamodel with potential further constraints. Note that attributes, as usually employed in real-world models, can in this context be modeled as dedicated nodes [13].

For \( \Phi \) the set of well-formedness conditions, a model \( M_i \) is well-formed iff \( \forall \phi \in \Phi : M_i \models \phi \). We assume \( \text{pcheck}(M_i, \phi) \) to report all violations to property \( \phi \) with violation pattern \( Q \) for model \( M_i \) in the form of matches for \( Q \), essentially realizing \( \models \) as
\[
\text{pcheck}(M_i, \phi) = \emptyset \iff M_i \models \phi.
\]
If violations exist, the model \( M_i \) is also called ill-formed.

For the notion of models as typed graphs, model modifications correspond to graph modifications as presented in Section 2. We say a model modification \((M_i \leftarrow K \rightarrow M_j)\) with identity morphisms is maximally preserving iff it does not delete and recreate identical elements. Formally, \( K = (V_{M_i} \cap V_{M_j}, E_{M_i} \cap E_{M_j}, s^K, t^K) \), where \( s^K \) and \( t^K \) are uniquely defined assuming invariant edge sources and targets. Consequently, for two models \( M_i \) and \( M_j \), the maximally preserving model modification \((M_i \leftarrow K \rightarrow M_j)\) is uniquely defined.

For a set of model modifications \( \Delta^{M_{\{1,\ldots,n\}}} \) between models \( M_{\{1,\ldots,n\}} = \{M_1, \ldots, M_n\} \), with \( \forall (G \leftarrow K \rightarrow H) \in \Delta^{M_{\{1,\ldots,n\}}} : G \in M_{\{1,\ldots,n\}} \land H \in M_{\{1,\ldots,n\}} \), we can define the set of predecessors \( \text{pre}(i) \subset M_{\{1,\ldots,n\}} \) of a version \( M_i \) as the set of versions \( M_j \) such that there exists a sequence of model modifications \((M_{x_1} \leftarrow K_{x_1} \rightarrow M_{x_2}), (M_{x_2} \leftarrow K_{x_2} \rightarrow M_{x_3}), \ldots, (M_{x_{n-1}} \leftarrow K_{x_{n-1}} \rightarrow M_{x_n})\) where \( x_1 = j, x_n = i, \) and \((M_{x_k} \leftarrow K_{x_k} \rightarrow M_{x_{k+1}}) \in \Delta^{M_{\{1,\ldots,n\}}} \) for \( 1 \leq k < n \).
\(\Delta^{M_{1,\ldots,n}}\) describes a correct model versioning if all morphisms in the individual model modifications are partial identities, all model modifications are maximally preserving, the \(\text{pre}\) relation is acyclic and there exists a model \(M_\alpha\) such that \(M_\alpha \in \text{pre}(i)\) for all models \(M_i \neq M_\alpha\). Effectively, a correct model versioning describes a directed acyclic graph of model versions \(M_{1,\ldots,n}\) that are derived from an original model \(M_\alpha\) via the model modifications in \(\Delta^{M_{1,\ldots,n}}\), and therefore closely corresponds to the versioning of some development artifact in a conventional version control system.

Taentzer et al. [20] define a merge operation for model modifications \(m_1 = (M_c \leftarrow K_i \rightarrow M_i)\) and \(m_2 = (M_c \leftarrow K_j \rightarrow M_j)\) with common source \(M_c\), which unifies \(m_1\) and \(m_2\) into a merged model modification \(m_m = \text{merge}(m_1, m_2) = (M_c \leftarrow K_m \rightarrow M_m).\) We denote the merged model by \(M_m = \text{merge}_G(m_1, m_2)\). This merge operation is similar to a three-way-merge in conventional version control systems [16], since \(m_m\) in the default case (i) preserves an element \(x \in M_c\) iff it is preserved by both \(m_1\) and \(m_2\) (ii) deletes an element \(x \in M_c\) iff it is deleted by \(m_1\) or \(m_2\) (iii) creates an element \(x \in M_m\) iff it is created by \(m_1\) or \(m_2\).

However, according to [20], model modifications can be in conflict in two cases: (i) insert-delete conflict and (ii) delete-delete conflict. Taentzer et al. state that only (i), where one modification creates an edge connected to a node deleted by the other modification, is an actual conflict, which has to be resolved to create a correct merge result. In this case, the merge result may deviate from the default case. Such conflicts will be reported by \(\text{mcheck}((M_c \leftarrow K_i \rightarrow M_i), (M_c \leftarrow K_j \rightarrow M_j))\) in the form \((e, v)\), where \(e\) is an edge created by one of the modifications and \(v\) is a node deleted by the other modification.

For a correct model versioning \(\Delta^{M_{1,\ldots,n}}\), we say that two sequences of model modifications \(M_c \Rightarrow^* M_i\) and \(M_c \Rightarrow^* M_j\) are in conflict iff their corresponding maximally preserving model modifications \((M_c \leftarrow K_{c,i} \rightarrow M_i)\) and \((M_c \leftarrow K_{c,j} \rightarrow M_j)\) are in conflict. In this case, we also say that \(M_i\) and \(M_j\) are in conflict for the common predecessor \(M_c\).

Insert-delete conflicts can be resolved by equipping the merge operation with a manual or automatic strategy for conflict resolution. We consider such a strategy valid if it decides for each conflict whether to either revert the edge creation or the node deletion and always produces a proper merged graph. The approach in [20] effectively proposes an automatic strategy that favors insertion over deletion in order to preserve as many model elements as possible. Therefore, it reverts any deletions of nodes that would lead to insert-delete conflicts.

In contrast, a strategy for conflict resolution may favor deletion over insertion by reverting any creations of edges that would lead to insert-delete conflicts. Specifically, for model modifications \(m_1 = (M_c \leftarrow K_i \rightarrow M_i)\) and \(m_2 = (M_c \leftarrow K_j \rightarrow M_j)\), the model modification \(m_{\text{min}} = \text{merge}^{\text{min}}(m_1, m_2)\), with \(\text{merge}^{\text{min}}\) a merge operation equipped with this strategy, only creates an edge created by \(m_1\) or \(m_2\) if neither its source nor target is deleted by the other modification.

If all well-formedness conditions are specified by simple violation patterns, \(m_{\text{min}}\) also yields a model where all well-formedness violations are also present in the merge result for any other conflict resolution strategy:
Theorem 1. For two model modifications $m_1 = (M_c \leftarrow K_i \rightarrow M_i)$ and $m_2 = (M_c \leftarrow K_j \rightarrow M_j)$ and a well-formedness constraint $\phi$ with violation pattern $Q$, it holds that

$$pcheck(merge^{min}_G(m_1, m_2), \phi) = \bigcap_{str \in S} pcheck(merge^{str}_G(m_1, m_2), \phi),$$

with $S$ the set of all valid conflict resolution strategies.

Proof. (Sketch) Follows directly from the fact that $merge^{min}_G(m_1, m_2)$ is the smallest common subgraph of all graphs produced by the operation $merge$ for any valid conflict resolution strategy. \hfill \Box

If there are no conflicts in the merged model operations, the $merge$ operation produces the same result regardless of the chosen strategy for conflict resolution.

For a correct model versioning, two model versions $M_i$ and $M_j$, and the set of versions $P = pre(i) \cap pre(j)$, we define the function

$$pre^C(i, j) = \begin{cases} \emptyset & M_i \in pre(j) \land M_j \in pre(i) \\ \{M_c \in P \mid \forall M_x \in P : M_c \notin pre(x)\} & \text{otherwise} \end{cases},$$

which returns the set of latest common predecessors of $M_i$ and $M_j$. Note that our definition of $pre^C$ corresponds to the definition of a best common ancestor in conventional version control systems such as Git [2], which is used to compute the base for three-way merges in these systems.

Figure 3 shows an exemplary model versioning based on the graph $M_1$ from Figure 1. The initial graph $M_2 = M_1$ contains four classes. The modification $m_1$ (not to be confused with a morphism) to $M_2$ creates a superclass edge from $c_1$ to $c_3$ and deletes the node $c_4$. The modification $m_2$ to graph $M_3$ creates superclass edges from $c_1$ to $c_2$ and from $c_4$ to $c_2$. There is an insert-delete conflict between the two modifications, since the modification to $M_2$ deletes a node that is needed as the source of an edge created by the modification to $M_3$. Furthermore, the result of the merge of the two modifications would violate the well-formedness constraint with the violation pattern $Q$ from Figure 1, since without additional modifications, the node $c_1$ would have two outgoing superclass edges.

**Fig. 3.** Example model versioning
4 Multi-Version Models as Typed Graphs

A correct model versioning $\Delta^{M(1,...,n)}$ with model versions $M_{\{1,...,n\}}$ conforming
to a type graph $TM$ can be represented by a multi-version model in the form of
a single graph that is typed over an adapted type graph.

The adapted type graph $TM_{mv}$ contains a node for each node and edge in
$TM$. It also contains edges connecting each node in $TM_{mv}$ that represents an
edge in $TM$ to the nodes representing the edge’s source and target in $TM$.
This yields a bijective function $corr_{mv} : V^{TM} \cup E^{TM} \to V^{TM_{mv}}$, which maps
elements from $TM$ to the corresponding node in $TM_{mv}$, and two bijective func-
tions $corr_{mv}^s, corr_{mv}^t : E^{TM} \to E^{TM_{mv}}$ mapping edges from $TM$ to the edges
in $TM_{mv}$ encoding the source and target relation in $TM$. In addition, $TM_{mv}$
contains a node $version$, an edge $suc$ with source and target $version$, and two
edges $cv_v$ and $dv_v$ from each other node $v \in V^{TM_{mv}}$ to the $version$ node.

A multi-version model $MVM$ for $\Delta^{M(1,...,n)}$ is then constructed by an op-
eration $comb$ as follows: A subgraph $P^{MV}_{mv}$ encodes structural information about
all model versions and is constructed by translating $P^M = \bigcup_{M_i \in M_{\{1,...,n\}}} M_i$
to conform to $TM_{mv}$ using an operation $trans_{mv}$. Since source and target functions
are invariant in a correct model versioning, $P^M$ is well-defined.

For each $v \in V^{MV_{mv}}$, $trans_{mv}$ creates a node of type $corr_{mv}(v)$ in $V^{P_{mv}}$. For
each $e \in E^{P_{mv}}$, a node of type $corr_{mv}(e)$ is created. This yields a bijection $origin : \ P^{MV}_{mv} \to P^M$ mapping translated elements to their original representation.

In addition, for each edge $e \in E^{P_{mv}}$, an edge of type $corr_{mv}^s(e)$ with source
$origin^{-1}(e)$ and target $origin^{-1}(s^{P_{mv}}(e))$ and an edge of type $corr_{mv}^t(e)$ with
source $origin^{-1}(e)$ and target $origin^{-1}(t^{P_{mv}}(e))$ are created in $E^{P_{mv}}$. Since edge
sources and targets are invariant, the corresponding node $v_e = origin^{-1}(e)$ in the end has exactly one edge of type $corr_{mv}^s(e)$ and one of type $corr_{mv}^t(e)$. We thus have two functions $s_{mv} : origin^{-1}(E^{P_{mv}}) \to E^{P_{mv}}$ respectively $t_{mv} : origin^{-1}(E^{P_{mv}}) \to E^{P_{mv}}$ encoding these mappings.

Another, distinct subgraph $P^{V}_{mv}$ contains versioning information and is con-
structed as follows: For each $M_i \in M_{\{1,...,n\}}$, $P^{V}_{mv}$ contains a corresponding node
of type $version$. For each $(M_i \leftarrow K \to M_j) \in \Delta^{M(1,...,n)}$, $P^{V}_{mv}$ contains an edge
of type $suc$ from the node representing $M_i$ to the node representing $M_j$.

For each modification $(M_i \leftarrow K \to M_j)$, a $cv$-edge with the node correspond-
ing to $M_i$ as its target is added to all nodes corresponding to elements created by
the modification. A $dv$-edge with the node corresponding to $M_j$ as its target
is added to all nodes corresponding to elements deleted by the modification.
Additionally, a $cv$ edge with the node corresponding to the initial version $M_\alpha$ as
its target is added to all nodes corresponding to elements in $M_\alpha$.

Since attributes can be encoded by dedicated nodes and assignment edges
[13], the construction can be performed analogously for attributed graphs.

For $v \in P^{MV}_{mv}$ and $M_i \in M_{\{1,...,n\}}$, we say that $v$ is $mv$-present in $M_i$, iff for
a node $m_{cv}$ connected to $v$ via a $cv$ edge, there exists a path from $m_{cv}$ to the
node representing $M_i$ via $suc$ edges that does not go through a node connected
to $v$ via a $dv$ edge. We denote the set of versions where $v$ is $mv$-present by $p(v)$. 

A model version $M_i$ can then be derived from $MVM$ via an operation $proj$ as follows: Collect all nodes $V_p = \{ v_p \in V^{P_{M_i}} | M_i \in p(v_p) \}$, that is, all nodes that are mv-present in $M_i$, and translate the induced subgraph into the single-version model $M_i$ with $V^{M_i} = \{ \text{origin}(v) | v \in V^{MVM} \land \text{corr}^{-1}(\text{type}^{MVM}(v)) \in V^{TM_i}, E^{M_i} = \{ \text{origin}(v) | v \in V^{MVM} \land \text{corr}^{-1}(\text{type}^{MVM}(v)) \in E^{TM_i} \}$. $s^{M_i} = \text{origin} \circ t^{MVM} \circ s_{mv} \circ \text{origin}^{-1}$, and $t^{M_i} = \text{origin} \circ t^{MVM} \circ t_{mv} \circ \text{origin}^{-1}$.

**Correctness**

**Theorem 2.** For a correct model versioning $\Delta^{M(1,...,n)}$ holds concerning comb and $proj$:

$$\forall i \in \{1,\ldots,n\} : M_i = proj(\text{comb}(\Delta^{M(1,...,n)}), i).$$

**Proof.** (Sketch) Any element in a version $M_i$ has a corresponding node $v$ in $\text{comb}(\Delta^{M(1,...,n)})$. By construction, $v$ is connected to a node corresponding to some version $M_j$ via a cv edge, for which there exists a path of suc edges to the node corresponding to $M_i$. That path does not go through a node connected to $v$ by a dv edge. $v$ is thus mv-present in $M_i$ and hence contained in the projection.

Inclusion of elements in the opposite direction can be shown analogously. Because edge sources and targets are invariant over all graphs, the edges in $\text{comb}(M_1,\ldots,M_n)$ correctly encode the source and target functions by construction. Thus, $\forall i \in \{1,\ldots,n\} : M_i = proj(\text{comb}(M_1,\ldots,M_n), i)$.

A maximally preserving model modification $(M_i \leftarrow K \rightarrow M_j)$ with $M_i, M_j \in M_{\{1,\ldots,n\}}$ (and thus any model modification in $\Delta^{M(1,...,n)}$) can be derived from $MVM$ via $proj^A$ as follows: $M_i$ and $M_j$ can be derived via the operation $proj$. $K$ is then the graph containing all elements from $M_i \cap M_j$, with $s^K$ and $t^K$ uniquely defined by the corresponding functions from $M_i$ and $M_j$ and partial identities as morphisms into $M_i$ and $M_j$.

**Theorem 3.** For a correct model versioning $\Delta^{M}_{\{1,...,n\}}$, concerning comb and $proj^A$, it holds that:

$$\forall M_i, M_j \in M_{\{1,...,n\}} : m_{i,j} = proj^A(\text{comb}(\Delta^{M_{\{1,...,n\}}}), i, j),$$

with $m_{i,j}$ the maximally preserving model modification from $M_i$ to $M_j$.

**Proof.** Follows trivially from **Theorem 2** and the definition of the maximally preserving model modification $(M_i \leftarrow K_{i,j} \rightarrow M_j)$.

Figure 3 visualizes the multi-version model $MVM$ constructed for the example versioning in Figure 2 and the associated adapted type graph $TM_{mv}$. $MVM$ contains a node for each node and edge in the models of the example versioning, one node of type version for each of the graphs $M_1$, $M_2$, and $M_3$, and appropriate edges as created by $\text{comb}$.
4.1 Directly Checking Well-Formedness for Multi-Version Models

We can use a multi-version model to directly find all well-formedness violations in all individual versions via an operation $pcheck_{mv}$. For a multi-version model $MVM$ with a bijective mapping into a union of original model versions $\text{origin}_M$ and a well-formedness constraint $\phi$ with associated violation pattern $Q$, $pcheck_{mv}(MVM, \phi)$ works as follows:

First, the graph $Q$ typed over the original type graph is translated into a corresponding graph $Q_{mv}$ typed over the adapted type graph using $\text{trans}_{mv}$. This yields a bijective mapping $\text{origin}_Q : Q_{mv} \rightarrow Q$.

Then, all matches for $Q_{mv}$ in $MVM$ are found. For each such match $m_{mv}$, $pcheck_{mv}$ computes all versions for which all vertices in the image of the match are $mv$-present by $P = \bigcap_{v \in V Q_{mv}} p(m_{mv}(v))$. If $P \neq \emptyset$, the match into the original model versions $m = \text{origin}_M \circ m_{mv} \circ \text{origin}_Q^{-1}$ is constructed and reported as a violation in all versions in $P$.

Correctness

**Theorem 4.** For a well-formedness constraint $\phi$ with violation pattern $Q$, a correct model versioning $\Delta M_{\{1,...,n\}}$, and $MVM = \text{comb}(\Delta M_{\{1,...,n\}})$ holds:

$$pcheck_{mv}(MVM, \phi) = \bigcup_{i \in \{1,...,n\}} \{(i, m) | m \in pcheck(\text{proj}(MVM, i), \phi)\}.$$  

**Proof.** (Sketch) A match $m : Q \rightarrow M_i$ for any version $M_i$ has one corresponding match $m_{mv}$ with $m = \text{origin}_M \circ m_{mv} \circ \text{origin}_Q^{-1}$, where edges created by $\text{trans}_{mv}$ ensure correct connectivity. $P = \bigcap_{v \in V Q_{mv}} p(m_{mv}(v))$ contains exactly the versions containing all elements in $m(Q)$. This yields the stated equality. $\square$

**Complexity** The effort for searching all versions $M_{\{1,...,n\}}$ of some model versioning $\Delta M_{\{1,...,n\}}$ for a pattern $Q$ using $pcheck$ is in $O(\sum_{M_i \in M_{\{1,...,n\}}} C(M_i, Q))$, with $C(M_i, Q)$ the effort for finding all matches of $Q$ into $M_i$. $P_{mv}^M = \text{trans}_{mv}(P^M)$ and $Q_{mv} = \text{trans}_{mv}(Q)$ are only different encodings of $P_{mv}^M = \bigcup_{M_i \in M_{\{1,...,n\}}} M_i$ and $Q$. Considering computation of the $mv$-present...
predicate, the effort for $pcheck_{mv}$ is hence in $O(C(\bigcup_{M_i \in M_{\{1, \ldots, n\}}} M_i, Q) + X \cdot |V_{Q_{mv}}| \cdot |\Delta^{M_1, \ldots, n}|)$, with $X$ the number of matches for $Q_{mv}$ into $P^{M_{mv}}$.

**Discussion** If many elements are shared between individual versions and modifications only perform few changes, the size of the union of all model versions will be small compared to the sum of the sizes of all individual versions. If pattern matching is efficient with respect to the size of the considered model, pattern matching over the union of all model versions will then likely require less effort than matching over each individual version. If the number of matches for violation patterns is low, the associated checks performed by $pcheck_{mv}$ will likely be more efficient than the pattern matching over the individual versions.

Overall, $pcheck_{mv}$ will thus likely be more efficient than using $pcheck$ in scenarios where pattern matching is efficient, the number of changes between versions is low, and the number of violations in the union of versions is low.

**5 Directly Checking Merge Results for Multi-Version Models**

We can consider multi-version models to directly detect whether (a) merge conflicts exist for any valid pair of encoded model modifications via an operation $mcheck_{mv}$ and (b) any resulting merged model is ill-formed via an operation $pcheck_{mv}$, where a pair of model modifications $(M_c \leftarrow K_i \rightarrow M_i)$ and $(M_c \leftarrow K_j \rightarrow M_j)$ is valid iff $M_c \in pre^C(M_i, M_j)$.

**5.1 Directly Checking for Merge Conflicts**

$mcheck_{mv}$ can be realized for a multi-version model $MVM = \text{comb}(\bigDelta^{M_{\{1, \ldots, n\}}})$ as follows: First, the operation collects all nodes in $MVM$ representing edges that are created by some model modification. This means all nodes $v_e \in V^{MVM}$ where $\text{corr}^{-1}_{mv}(\text{type}^{MVM}(v)) \in E^{TM}$ connected to a node $m_x$ via a $cv$ edge, where $m_x$ does not correspond to $M_\alpha$ and with $TM$ the original type graph. Then, for each node $v_e$, we compute the set of versions $P = p(v_e)$ where it is mv-present.

If $P \neq p(v_s)$, where $v_s = s^{MVM}(s_{mv}(v_e))$, we then compute a set of versions $D$ that correspond to nodes reachable via $suc$ edges from a node connected to $v_s$ via a $dv$ edge without going through nodes connected to $v_s$ via a $cv$ edge.

Afterwards, for each pair of versions $M_i \in P$ and $M_j \in D$, we check for each latest common predecessor $M_c \in pre^C(i, j)$ whether $M_c \in p(v_s) \land M_c \notin P$. For any triplet of versions $(i, j, c)$ where this is the case, the edge $\text{origin}(v_e)$ is then in an insert-delete conflict with its source. To facilitate formalization, this conflict is reported in the normalized form $(\text{min}(i, j), \text{max}(i, j), c, (\text{origin}(v_e), \text{origin}(v_s)))$.

Insert-delete conflicts with the edge’s target are computed analogously.

**Correctness**
The efficiency of overall number of elements created in the model versioning will be more efficient. Otherwise, mcheck on the original model version and are shared between many model versions, the number of edges in the individual versions. If most edges are present in the pends on the number of edges created by some model modification compared to the complexity of \( O \). Computing and checking the required version sets is the maximum number of edges present in a single model version.

Theorem 5. For a model versioning \( \Delta^M_{1,...,n} \) and the associated multi-version model \( \text{MVM} = \text{comb}(\Delta^M_{1,...,n}) \) holds:

\[
\text{mcheck}_{mv}(\text{MVM}) = \bigcup_{(i,j,c) \in Y} \{(i, j, c, m)|m \in \text{mcheck}(c, i, m, j)\},
\]

where \( Y = \{(i, j, c)|i, j \in \{1, \ldots, n\} : i < j, c \in \{e|e \in \text{pre}_C(i, j)\}\} \) and with \( m_{c, i} = \text{proj}^A(\text{MVM}, c, i) \) and \( m_{c, j} = \text{proj}^A(\text{MVM}, c, j) \).

Proof. (Sketch) The collected nodes representing edges correspond to a superset of edges that may be involved in a conflict. The construction of the sets \( F \) and \( D \) for a collected node \( v_e \) ensures that any pair of versions where one may create \( e = \text{origin}(v_e) \) and the other may delete the source (or target) of \( e \) is considered. The condition checked for each common predecessor of a version pair then yields exactly the triplets of versions where \( e \) is part of an insert-delete conflict. Because of the normalization of the results of \( \text{mcheck}_{mv} \), we have the stated equality. \( \square \)

Complexity The function \( \text{pre}_C^e \) can be precomputed in \( O(|M_{1,...,n}|^4) \).

Since information about creation and deletion of elements is not explicitly available in a naive representation, finding all insert-delete conflicts between two model modifications via \( \text{mcheck} \) has to be done by checking for each edge in either modification’s resulting model whether it is created by that modification and its source or target is deleted by the other modification. Since there may exist up to \( O(|M_{1,...,n}|^3) \) possible merges in a model versioning, in the worst case, this implies effort in \( O(|M_{1,...,n}|^4 + |E_{M_{\text{max}}}| \cdot |M_{1,...,n}|^3) \), where \( |E_{M_{\text{max}}} | \) is the maximum number of edges present in a single model version.

Created edges can be retrieved efficiently from a multi-version model given appropriate data structures. Computing and checking the required version sets takes \( O(|M_{1,...,n}|^3) \) steps per edge. Therefore, the overall computational complexity of \( \text{mcheck}_{mv} \) is in \( O(|M_{1,...,n}|^4 + \Delta_+ \cdot |M_{1,...,n}|^3) \), where \( \Delta_+ \) is the overall number of elements created in the model versioning.

Discussion The efficiency of \( \text{mcheck}_{mv} \) compared to using \( \text{mcheck} \) mostly depends on the number of edges created by some model modification compared to the number of edges in the individual versions. If most edges are present in the original model version and are shared between many model versions, \( \text{mcheck}_{mv} \) will be more efficient. Otherwise, \( \text{mcheck}_{mv} \) will not achieve a significant improvement and might even perform worse than the operation based on \( \text{mcheck} \).

Version control systems such as Git typically select a single latest common predecessor as the base for a three way merge [2]. Using a corresponding partial function \( \text{pre}_1^C : \mathbb{N} \times \mathbb{N} \to M_{1,...,n} \) with \( \text{pre}_1^C(i, j) \) with \( \text{pre}_C^e(i, j) \neq \emptyset \) and \( \text{pre}_C^e(i, j) = \bot \) to select a single latest common predecessor of two versions \( i \) and \( j \) rather than \( \text{pre}_C^e \) in \( \text{mcheck}_{mv} \), by the same logic as used in the proof of correctness, we instead have an analogous equality for \( \text{pre}_1^C \). Disregarding the computational effort for precomputing \( \text{pre}_1^C \), replacing \( \text{pre}_C^e \) by \( \text{pre}_1^C \) reduces the remaining computational complexity of \( \text{mcheck}_{mv} \) to \( O(\Delta_+ \cdot |M_{1,...,n}|^3) \).
5.2 Directly Checking Well-Formedness for Merge Results

To find all violations of a well-formedness constraint \( \phi \) characterized by a pattern \( Q \) via \( \text{pcheck}_{\text{mv}}^m \) in merge results of a multi-version model \( MVM \), we first translate \( Q \) into \( Q_{\text{mv}} = \text{trans}_{\text{mv}} \). We then find all matches for \( Q_{\text{mv}} \) in \( MVM \).

For a match \( m_{\text{mv}} \) for \( Q_{\text{mv}} \), we determine the set of versions \( P_v = \{ v \} \) for each \( v \in m_{\text{mv}}(V_{Q_{\text{mv}}}) \). For each pair of versions \( M_i \in \arg\min_{v \in \{ v \}} |P_v| \) and \( M_j \in \bigcup_{v \in V_{Q_{\text{mv}}}} P(v) \), we check whether \( \forall v \in m_{\text{mv}}(V_{Q_{\text{mv}}}) : M_i \in P(v) \lor M_j \in P(v) \). We then check for each latest common predecessor \( M_c \in \text{pre}^C(i,j) \) if for all \( v \in V_{Q_{\text{mv}}} \), it holds that \( v \in V_{M_i} \rightarrow (v \in V_{M_i} \land v \in V_{M_j}) \), that is, \( v \) is not deleted in \( M_i \) or \( M_j \). If this is the case, the match \( m \) into \( \bigcup_{M_x \in M(1, \ldots, n)} M_x \) corresponding to \( m_{\text{mv}} \) represents a violation in \( \text{merge}^{\text{min}}(M_c \leftarrow K_i \rightarrow M_i), (M_c \leftarrow K_j \rightarrow M_j) \). We report results in the normalized form \( (\min(i,j), \max(i,j), c, m) \).

Correctness

**Theorem 6.** For a well-formedness constraint \( \phi \), a correct model versioning \( \Delta_M(1, \ldots, n) \), and the multi-version model \( MVM = \text{comb}(\Delta_M) \) holds:

\[
\text{pcheck}_{\text{mv}}^m(MVM, \phi) = \bigcup_{(i,j,c) \in Y} \{ (i,j,c,m) | m \in \text{pcheck}(M_{i,j,c}^{\text{min}}) \},
\]

where \( Y = \{ (i,j,c) | i, j \in \{1, \ldots, n\} : i < j, c \in \{ c | M_c \in \text{pre}^C(i,j) \} \} \) and \( M_{i,j,c}^{\text{min}} = \text{merge}_G^{\text{min}}(\text{proj}^\Delta(MVM, c, i), \text{proj}^\Delta(MVM, c, j)) \).

**Proof.** (Sketch) For two versions \( M_i, M_j \) with latest common predecessor \( M_c, \) a match \( m : Q \rightarrow \text{merge}_G^{\text{min}}(\text{proj}^\Delta(MVM, c, i), \text{proj}^\Delta(MVM, c, j)) \) has one corresponding match \( m_{\text{mv}} : \text{trans}_{\text{mv}}(Q) \rightarrow MVM \) by construction, where the edges created by \( \text{trans}_{\text{mv}} \) ensure the correct connectivity. The set of version pairs considered by \( \text{pcheck}_{\text{mv}}^m \) contains all version pairs such that each matched element is contained in at least one of the versions. The condition checked for every latest common predecessor ensures that only version triplets are reported where the merge result also contains all matched elements if there are no merge conflicts. Since \( \text{merge}^{\text{min}} \) resolves conflicts by prioritizing deletion and, as ensured by the check, no matched node is deleted by the merge, conflict resolution cannot invalidate the match or create new matches. We thus have the stated equality. \( \square \)

By Theorem 6 and Theorem 5 we also have that \( \text{pcheck}_{\text{mv}}^m \) yields the set of violations that cannot be avoided by any conflict resolution strategy:

**Corollary 1.** For a well-formedness constraint \( \phi \), a correct model versioning \( \Delta_M(1, \ldots, n) \), and the multi-version model \( MVM = \text{comb}(\Delta_M(1, \ldots, n)) \) holds:

\[
\text{pcheck}_{\text{mv}}^m(MVM, \phi) = \bigcup_{(i,j,c) \in Y} \bigcap_{S \in S} \{ (i,j,c,m) | m \in \text{mcheck}(M_{i,j,c}^{\text{str}}) \},
\]

where \( Y = \{ (i,j,c) | i, j \in \{1, \ldots, n\} : i < j, c \in \{ c | M_c \in \text{pre}^C(i,j) \} \} \) and \( M_{i,j,c}^{\text{str}} = \text{merge}_G^{\text{str}}(\text{proj}^\Delta(MVM, c, i), \text{proj}^\Delta(MVM, c, j)) \), and with \( S \) the set of all valid conflict resolution strategies.
Complexity The function \( pre_C^{mv} \) can be precomputed in \( O(|M|_{1,...,n}|^4) \).

With \( C(M_i, Q) \) the effort for finding all matches of \( Q \) into \( M_i \), finding violations characterized by a pattern \( Q \) in all results of a set of possible merges \( Y \) using \( pcheck \) takes effort in \( O(|M|_{1,...,n}|^4 + \sum_{(m_1, m_2) \in Y} C(merge_{G}^{\min}(m_1, m_2), Q)) \).

The computation and checking of version triplets for a match in \( pcheck^{mv} \) takes effort in \( O(|M|_{1,...,n}|^4 + \sum_{(m_1, m_2) \in Y} C(merge_{G}^{\min}(m_1, m_2), Q)) \).

Discussion By the same argumentation as for \( pcheck^{mv} \), \( pcheck^{mv}_{m} \) will likely be more efficient than the corresponding operation using \( pcheck \) in scenarios where pattern matching is efficient, the number of changes between versions is low, and the number of violations in the union of model versions is low.

Using some partial function \( pre_C^1 : \mathbb{N} \times \mathbb{N} \rightarrow M_{1,...,n} \) to select a single latest common predecessor rather than \( pre_C^{C} \) in \( pcheck^{mv}_{m} \), by the same logic as in the proof of correctness, we have an analogous equality for \( pre_C^1 \). Disregarding the effort for precomputing \( pre_C^1 \), replacing \( pre_C^{C} \) by \( pre_C^1 \) reduces the remaining complexity of \( pcheck^{mv}_{m} \) to \( O(C(\bigcup_{M_i \in M_{1,...,n}} M_i, Q) + X \cdot |V^{Q_{mv}}| \cdot |M|_{1,...,n}|^2) \).

6 Evaluation

For an initial empirical evaluation of the performance and scalability of the presented operations, we experiment with an application scenario from the software development domain. Therefore, we extract abstract syntax graphs from a small previous research project (rete) and a larger open source project (henshin) written in Java using the EMF-based MoDisco tool. We store the extracted models in a graph format and fold each of the projects into a multi-version model, using a mapping strategy based on hierarchy and element names.

We then run implementations of the presented operations for conflict detection and well-formedness checking based on multi-version models (MVM) and baseline implementations using corresponding single-version models (SVM). We consider three well-formedness constraints: uniqueness of a class’s superclass, uniqueness of a method’s return type, and consistency of an overridden method’s return type. We employ our own EMF-based tool for pattern matching.

Figure 5 shows the measured execution times for the operations \( pcheck^{mv}_{m} \), \( mcheck^{mv}_{m} \), and \( pcheck^{mv}_{m} \) and related single-version-model-based operations over the example models. The execution times for \( pcheck^{mv}_{m} \) and \( pcheck^{mv}_{m} \) correspond to the combined pattern matching time for all considered well-formedness constraints. All reported times exclude the time for computing any merge results required by SVM and the time required to precompute the \( pre_C^{C} \) function, since it

---

\(^1\) All experiments were executed on a Linux SMP Debian 4.19.67-2 machine with Intel Xeon E5-2630 CPU (2.3 GHz clock rate) and 386 GB system memory running OpenJDK version 1.8.0_242. Reported execution times correspond to the arithmetic mean of at least five runs of the respective experiment. Our implementation and datasets are available under [https://github.com/hpi-sam/multi-version-models](https://github.com/hpi-sam/multi-version-models).
is required by both the MVM and the SVM implementation. Precomputing $\text{pre}^{C}$ took about 5 ms for the smaller project and about 3.5 s for the larger project.

For the tasks related to well-formedness checking, the MVM variant performs better (up to factor 50) than SVM. Since there are only few to no matches for the violation patterns of the considered constraints, the MVM implementation only performs few of the potentially expensive checks over the version graph, while avoiding most of the redundancy in the pattern matching of SVM.

For conflict detection, MVM performs better than SVM for the smaller project (factor 5), but has a substantially higher execution time for the larger project (factor 10). The reason for the bad performance is that most edges are not present in the initial model version. In fact, the number of edges created throughout the model versioning is much higher than the number of edges in any individual version. Furthermore, in contrast to the solution using $\text{mcheck}_{mv}$, the operation $\text{mcheck}_{mv}$ considers versions where the source or target of an edge is not present. Due to the high number of versions in the project and because many elements are only present in few versions, this leads to the processing of large version sets, which deteriorates the performance of MVM in this scenario.

**Threats to Validity** Unexpected JVM behavior poses a threat to internal validity, which we tried to mitigate by performing multiple runs of each experiment and profiling time spent on garbage collection. To address threats to external validity, we used real-world data and well-formedness constraints in our experiments. While we used our own tool for pattern matching, said tool has already been used in our previous works and has shown adequate performance [11].

However, the example constraints are not representative and the folding of individual model versions extracted from source code may yield a larger-than-necessary multi-version model. Our results are thus not necessarily generalizable, but instead constitute an early conceptual evaluation of the presented approach.

### 7 Related Work

While most practical version control systems operate on text documents [16], versioning and merging of models has also been subject to extensive research.
There already exist several formal and semi-formal approaches to model merging, which compute the result of a three-way-merge of model modifications \cite{20,21}. Notably, the approach by Taentzer et al. \cite{20} represents a formally defined solution that works on the level of graphs, which is why for our approach, we build on their notion of model merging. In their work, Taentzer et al. also consider checking of well-formedness constraints by constructing a tentative merge result over which the check is executed. While this allows their approach to handle arbitrary constraints rather than just simple graph patterns, the check has to be executed for each individual merge.

Some approaches consider detection of merge conflicts \cite{15} or model inconsistencies \cite{4} based on the analysis of sequences of primitive changes. However, these approaches do not consider the case of multiple versions and pairwise merges and naturally do not employ a graph-based definition of inconsistencies.

For the more general problem of model versioning, both formal solutions \cite{7,18} and tool implementations \cite{14,17} have been introduced. Similar to our approach, some of these techniques are based on a joint representation of multiple model versions \cite{17,18}. However, to the best of our knowledge, joint conflict detection or well-formedness checking for all merges at once is not considered.

Westfechtel and Greiner \cite{22} present a solution to configuration management for product lines, where several configurations are represented in a unified feature model. Information about the presence of elements in certain configurations is encoded in logic formulas and propagated along model transformations. While this approach bears some similarity to the encoding of different versions and collective well-formedness checking in our solution, the technique in \cite{22} focuses on the definition of product lines and hence does not consider merging.

Folding and joint querying of the temporal evolution of graphs has been studied in previous work of our group \cite{12,19}. However, the aim of these solutions is the development of a temporal logic for graphs. The presented concepts are hence specific to sequences of graph modifications and do not consider merging.

8 Conclusion

In this paper, we have presented an approach for encoding a model versioning as a single typed graph. Based on this representation, we have introduced operations for finding merge conflicts and violations of well-formedness conditions in the form of graph patterns in the entire versioning and related merge results. We have conducted an initial empirical evaluation, which demonstrates potential benefits of the approach, but also highlights shortcomings in unfavorable scenarios.

In future work, we plan to address these shortcomings by studying how to compress the version graph or restrict the set of considered versions to those most relevant to users. We want to investigate how to lift our notion of well-formedness constraints to more expressive formalisms such as nested graph conditions and develop an incremental version of the approach. Finally, we will extend our empirical evaluation to better characterize our technique’s performance.
References

1. EMF. [https://www.eclipse.org/modeling/emf/] last accessed 23 February 2022
2. Git. [https://git-scm.com/] last accessed 23 February 2022
3. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: advanced concepts and tools for in-place EMF model transformations. In: International Conference on Model Driven Engineering Languages and Systems. Springer (2010)
4. Blanc, X., Mognenot, A., Mounier, I., Mens, T.: Incremental detection of model inconsistencies based on model operations. In: International Conference on Advanced Information Systems Engineering, pp. 32–46. Springer (2009)
5. Brun, Y., Holmes, R., Ernst, M.D., Notkin, D.: Early Detection of Collaboration Conflicts and Risks. IEEE Transactions on Software Engineering 39(10)
6. Bruniere, H., Cabot, J., Jouault, F., Madiot, F.: MoDisco: a generic and extensible framework for model driven reverse engineering. In: Proceedings of the IEEE/ACM international conference on Automated software engineering
7. Diskin, Z., Czarnecki, K., Antkiewicz, M.: Model-versioning-in-the-large: Algebraic foundations and the tile notation. In: 2009 ICSE Workshop on Comparison and Versioning of Software Models. pp. 7–12. IEEE (2009)
8. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of algebraic graph transformation. Springer (2006)
9. Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency Handling in Multiperspective Specifications. IEEE Transactions on Software Engineering 20(8), 569–578 (August 1994)
10. Frühaufer, K., Zeller, A.: Software configuration management: State of the art, state of the practice. In: International Symposium on Software Configuration Management. pp. 217–227. Springer (1999)
11. Giese, H., Hildebrandt, S., Seibel, A.: Improved flexibility and scalability by interpreting story diagrams. Electronic Communications of the EASST 18 (2009)
12. Giese, H., Maximova, M., Sakizloglou, L., Schneider, S.: Metric temporal graph logic over typed attributed graphs. In: International Conference on Fundamental Approaches to Software Engineering. pp. 282–298. Springer, Cham (2019)
13. Heckel, R., Küster, J.M., Taentzer, G.: Confluence of Typed Attributed Graph Transformation Systems. In: Graph Transformation (2002)
14. Koegel, M., Helming, J.: Emfstore: a model repository for emf models. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2. pp. 307–308 (2010)
15. Küster, J.M., Gerth, C., Engels, G.: Dependent and conflicting change operations of process models. In: European Conference on Model Driven Architecture-Foundations and Applications. pp. 158–173. Springer (2009)
16. Mens, T.: A State-of-the-Art Survey on Software Merging. IEEE Trans. Softw. Eng. 28(5), 449–462 (2002)
17. Murtu, L., Corrêa, C., Prudêncio, J.G., Werner, C.: Towards odyssey-vcs 2: Improvements over a uml-based version control system. In: Proceedings of the 2008 international workshop on Comparison and versioning of software models (2008)
18. Rutle, A., Rossini, A., Lamo, Y., Wolter, U.: A category-theoretical approach to the formalisation of version control in MDE. In: International Conference on Fundamental Approaches to Software Engineering. pp. 64–78. Springer (2009)
19. Sakizloglou, L., Ghahremani, S., Barkowsky, M., Giese, H.: Incremental execution of temporal graph queries over runtime models with history and its applications. Software and Systems Modeling pp. 1–41 (2021)
20. Taentzer, G., Ermel, C., Langer, P., Wimmer, M.: A fundamental approach to model versioning based on graph modifications: from theory to implementation. Software & Systems Modeling 13(1), 239–272 (2014)

21. Westfechtel, B.: A formal approach to three-way merging of emf models. In: Proceedings of the 1st International Workshop on Model Comparison in Practice

22. Westfechtel, B., Greiner, S.: Extending single-to multi-variant model transformations by trace-based propagation of variability annotations. Software and Systems Modeling 19(4), 853–888 (2020)
A Technical Details

Theorem 7. For a correct model versioning $\Delta^M(1,\ldots,n)$ holds concerning comb and proj:

$$\forall i \in \{1,\ldots,n\} : M_i = \text{proj}(\text{comb}(\Delta^M(1,\ldots,n)), i).$$

Proof. For any version $M_i$ and any element $x \in M_i$, there is a version $M_j$, such that $x \in M_j$ with $M_j = M_i$, or $\exists(M_k \leftarrow K \rightarrow M_j) \in \Delta^M(1,\ldots,n) : x \notin M_k \wedge x \in M_j$, that is, $(M_k \leftarrow K \rightarrow M_j)$ creates $x$, and there exists a (potentially empty) sequence of model modifications $M_j \Rightarrow^* M_i$ that does not delete $x$ again.

Thus, by the construction of comb, there exists an element $x_{mv} \in \text{MVM}$, where $\text{MVM} = \text{comb}(M_1,\ldots,M_n)$, which is connected to the node $m_j \in \text{MVM}$ that corresponds to $M_j$ via a mV edge.

Because there exists a sequence of model modifications $M_j \Rightarrow^* M_i$ where $x$ is not deleted, there exists a path of such edges from $m_j$ to the node representing $M_i$, that does not go through any node connected to $x_{mv}$ via a mV edge. Therefore, $x_{mv}$ is mV-present in $M_i$ for $\text{MVM}$. Hence, $x \in \text{proj}(\text{comb}(M_1,\ldots,M_n), i)$.

Since all implications used for deductions are equivalences, we have that $x \in \text{proj}(\text{comb}(M_1,\ldots,M_n), i) \Leftrightarrow x \in M_i$ for any element $x$ and model version $M_i$ and therefore the equality of vertex and edge sets of $M_i$ and the projection.

Because edge sources and targets are invariant over multiple graphs, the edges in $\text{comb}(M_1,\ldots,M_n)$ correctly encode the source and target functions by construction. Thus, $\forall i \in \{1,\ldots,n\} : M_i = \text{proj}(\text{comb}(M_1,\ldots,M_n), i).$ \hfill $\square$

Theorem 8. For a well-formedness constraint $\phi$, a correct model versioning $\Delta^M(1,\ldots,n)$, and associated multi-version model $\text{MVM} = \text{comb}(\Delta^M(1,\ldots,n))$ holds:

$$\text{pcheck}_{mv}(\text{MVM}, \phi) = \bigcup_{i \in \{1,\ldots,n\}} \{(i,m) | m \in \text{pcheck}(\text{proj}(\text{MVM}, i), \phi)\}.$$  

Proof. For a tuple $(i,m)$ with match $m : Q \rightarrow \bigcup_{M_i \in \{1,\ldots,n\}} M_i$ to be reported in the result of $\text{pcheck}_{mv}(M_{\{1,\ldots,n\}}, \phi)$, there has to exist a match $m_{mv} : Q_{mv} \rightarrow \text{MVM}$, where $Q_{mv} = \text{trans}_{mv}(Q)$, such that $m = \text{trans}_{mv}(m_{mv})$ and $\bigcap_{v \in Q_{mv}} p(m_{mv}(v)) \neq \emptyset$.

There hence exists some version $M_i \in M_{\{1,\ldots,n\}}$ such that $\forall v \in Q_{mv} : M_i \in p(m_{mv}(v))$. Therefore, $\forall v \in Q_{mv} : \text{origin}_{mv}(m_{mv}(v)) \in \text{proj}(M_{\{1,\ldots,n\}}, i)$.

Then, $m$ is a valid monomorphism into $M_i = \text{proj}(M_{\{1,\ldots,n\}}, i)$, since (i) all required elements are present and (ii) by construction of $\text{MVM}$, $Q_{mv}$, the edges in $Q_{mv}$ ensure that $\forall e \in E^Q : s^M_i(m(e)) = s^Q(e) \wedge t^M_i(m(e)) = t^Q(e)$, and the origin function is bijective. Therefore, it holds that $(i,m) \in \bigcup_{i \in \{1,\ldots,n\}} \text{pcheck}(\text{proj}(M_{\{1,\ldots,n\}}, i), \phi)$.

Thus, we have:

$$\text{pcheck}_{mv}(\text{MVM}, \phi) \subseteq \bigcup_{i \in \{1,\ldots,n\}} \{(i,m) | m \in \text{pcheck}(\text{proj}(\text{MVM}, i), \phi)\}.$$  

Any match $m : Q \rightarrow M_i$, where $M_i = \text{proj}(M_{\{1,\ldots,n\}}, i)$ is also structurally present in $\bigcup_{M_i \in M_{\{1,\ldots,n\}}} M_i$. Since $\text{trans}_{mv}$ creates isomorphic graphs for $Q$.
and the image of $Q$ under $m$, there exists a match $m_{mv} = \text{trans}_{mv}^m(m_{mv})$ for $Q_{mv} = \text{trans}_{mv}(Q)$ into $\text{MVM}$.

For $m$ to be reported by $\text{pcheck}(\text{proj}(M_{\{1,\ldots,n\}}, i), \phi)$, all elements in the image of $m$ have to be present in $M_i$. Hence, by the semantics of $\text{proj}$, we have that $\forall x \in Q : M_i \in p(\text{origin}^{-1}_M(m(x)))$ and hence, $\forall v \in V^{Q_{mv}} : M_i \in p(m_{mv}(v))$. Therefore, $\bigcap_{v \in V^{Q_{mv}}} p(m_{mv}(v)) \neq \emptyset$, so $(i, m) \in \text{pcheck}_{mv}(M_{\{1,\ldots,n\}}, \phi)$.

Thus, we have:

$$\text{pcheck}_{mv}(\text{MVM}, \phi) \supseteq \biguplus_{i \in \{1,\ldots,n\}} \{(i, m) | m \in \text{pcheck}(\text{proj}(\text{MVM}, i), \phi)\}$$

From the two subset-relations follows the equality. \hfill \Box

**Theorem 9.** For a model versioning $\Delta^{M_{1,\ldots,n}}$ and the associated multi-version model $\text{MVM} = \text{comb}(\Delta^{M_{1,\ldots,n}})$ holds:

$$\text{mcheck}_{mv}(\text{MVM}) = \biguplus_{(i, j, c) \in Y} \{(i, j, c, m) | m \in \text{mcheck}(m_{c,i}, m_{c,j})\},$$

where $Y = \{(i, j, c) | i, j \in \{1,\ldots,n\}, i < j, c \in \{c | M_c \in \text{pre}^C(i, j)\}\}$ and with $m_{c,i} = \text{proj}^C(\text{MVM}, c, i)$ and $m_{c,j} = \text{proj}^C(\text{MVM}, c, j)$.

**Proof.** For an edge and its invariant source $s$ reported for versions $M_i$ and $M_j$ and common predecessor $M_c \in \text{pre}^C(i, j)$ by $\text{mcheck}_{mv}$, by the normalization and the fact that $\text{pre}^C \neq \emptyset \rightarrow i \neq j$, we have that $i < j$. Because $M_c \notin p(v_s)$, we also have that $e \notin E^{M_c}$. Furthermore, we either have that $M_i \in p(\text{origin}^{-1}_M(e))$ and therefore $e \notin E^{M_i}$, by Theorem 3 (case 1) or $M_j \in p(\text{origin}^{-1}_M(e))$ and therefore $e \in E^{M_j}$ (case 2). Since, $M_c \in \text{pre}(M_i)$, we therefore have a sequence of model modifications $M_c \Rightarrow^* M_i$ (case 1) or $M_j \Rightarrow^* M_j$ (case 2) that creates $e$.

We also know that, in case 1, the node in the multi-version model corresponding to $M_j$ lies on a path corresponding to a sequence of model modifications that deletes but does not recreate $s$. Hence, $M_j \notin p(\text{origin}^{-1}(s))$, since $M_j$ being in $p(\text{origin}^{-1}(s))$ would contradict the consistency of the model versioning. Furthermore, we know that $M_c \in p(\text{origin}^{-1}(s))$. This means there exists a sequence of model modifications $M_c \Rightarrow^* M_j$ that deletes $s$, but not $e$, since $e \notin E^{M_c}$. In case 2, by the same logic, there exists a sequence of model modifications $M_c \Rightarrow^* M_i$ that deletes $s$ but not $e$. Thus, there is an insert-delete conflict between $e$ and $s$ for $M_i$ and $M_j$ in either case.

The same is true for the target of an edge.

Thus, we have:

$$\text{mcheck}_{mv}(\text{MVM}) \subseteq \biguplus_{(i, j, c) \in Y} \{(i, j, c, m) | m \in \text{mcheck}(m_{c,i}, m_{c,j})\},$$

If there exists an insert-delete conflict for an edge $e$ and its source $s$ between two versions $M_i = \text{proj}(M_{\{1,\ldots,n\}}, i)$ and $M_j = \text{proj}(M_{\{1,\ldots,n\}}, j)$ for a common
By Theorem 2, it then follows that $M_c \in \text{pre}^C(i,j)$, this means that there exist a sequence of model modifications $S_i = M_c \Rightarrow^* M_i$ and at least one sequence $S_j = M_c \Rightarrow^* M_j$, where one sequence deletes $s$ and does not delete $e$ and the other sequence creates $e$. Therefore, $s \in V^{M_c}$ and $e \notin E^{M_c}$.

In the case where $S_i$ is the sequence that creates $e$, it follows that $e \in E^{M_i}$. By Theorem 2, it then follows that $M_i \in p(v_c) \land M_c \notin p(v_c)$, with $v_c = \text{origin}^{-1}(e)$. Because $S_j$ then deletes and does not recreate $s$, it follows that there exists a path from a node connected to $v_s$ via a dv edge to the node corresponding to $M_j$ that does not go through a node connected to $v_s$ via a cv edge, with $v_s = \text{origin}^{-1}(s)$.

By the same logic, we can derive mirrored statements for the case where $S_j$ is the sequence that creates $e$.

From the existence of a sequence of model modifications that creates $e$, it also follows that $v_c$ is connected to some node $m_c$ via a cv edge that does not correspond to $M_a$. From the fact that one sequence deletes $e$ but not $s$, it follows that $p(v_c) \neq p(v_s)$.

Thus, $e$ and $v$ would be reported as an insert-delete conflict for versions $M_i$ and $M_j$ with predecessor $M_c$ in its normalized format by $mcheck_{mv}$.

The same is true for the target of an edge.

Thus, we have:

$$mcheck_{mv}(MVM) \supseteq \bigcup_{(i,j,c) \in Y} \{(i, j, c, m)| m \in mcheck(m_{c,i}, m_{c,j})\},$$

By the two inclusions, we also have the equality. \hfill \Box

**Theorem 10.** For a well-formedness constraint $\phi$, a correct model versioning $\Delta^{M(1,\ldots,n)}$ and the associated multi-version model $MVM = \text{comb}(\Delta^{M}_{1,\ldots,n})$ holds:

$$pcheck_{mv}(MVM, \phi) = \bigcup_{(i,j,c) \in Y} \{(i, j, c, m)| m \in pcheck(M_{i,j,c}^{\text{min}}, \phi)\},$$

where $Y = \{(i, j, c)| i, j \in \{1, \ldots, n\}: i < j, c \in \{c|M_c \in \text{pre}^C(i,j)\}\}$ and $M_{i,j,c}^{\text{min}} = \text{merge}_{G}^{\text{min}}(\text{proj}_{A}(MVM, c, i), \text{proj}_{A}(MVM, c, j))$.

**Proof.** For a match $m$ for $Q$ into $\bigcup_{M_i \in M(1,\ldots,n)} M_i$ to be reported as a violation in the merge of versions $M_i$ and $M_j$ with latest common predecessor $M_c$ by $pcheck_{mv}$, there has to exist a corresponding match $m_{mv}: \text{trans}_{mv}(Q) \rightarrow MVM$.

We know that, because $M_c \in \text{pre}^C(M_i, M_j)$, there are sequences of model modifications $S_i = M_c \Rightarrow^* M_i$ and $S_j = M_c \Rightarrow^* M_j$. Furthermore $m_{mv}$, we know that $\forall v \in m_{mv}(Q_{mv}) : M_i \in p(v) \lor M_i \in p(v)$, and $\forall v \in m_{mv}(Q_{mv}) : M_c \in p(v) \rightarrow (M_i \in p(v) \land M_j \in p(v))$.

Hence, every element $x \in m(Q)$ is either present in $M_c$ or created by $S_i$ or $S_j$, and none of these elements are deleted by either modification sequence. The resolution of merge conflicts by $\text{merge}^{\text{min}}$ may delete an edge created by one modification sequence, if its source or target is deleted by the other modification
sequence. However, since $Q$ is a proper graph, meaning that for any edge $e \in E_Q : s^Q(e) \in V^Q \wedge t^Q(e) \in V^Q$, and no node $v \in m(V^Q)$ is deleted by either modification sequence, it follows that no edge $e \in m(E_Q)$ can be deleted in the process of resolving merge conflicts.

Hence, all elements $x \in m(Q)$ are present in $\text{merge}^\text{min}_{G}(S_i, S_j)$. By the construction of $\text{MVM}$, we also know that source and target functions of edges are preserved by $\text{trans}_{mv}(m_{mv})$. This makes $m$ a violation of $\phi$ in the merge. Therefore, $(i, j, c, m) \in p\text{check}(\text{merge}^\text{min}_{G}(\text{proj}(M_{1, \ldots, n}), i), \text{proj}(M_{1, \ldots, n}), j), \phi).

Since we know that $i < j$, we have:

$$p\text{check}_{mv}^m(\text{MVM}, \phi) \subseteq \bigcup_{(i, j, c) \in Y} \{(i, j, c, m)m \in p\text{check}(M^\text{min}_{i,j,c}, \phi)\},$$

For two versions $M_i$ and $M_j$ with common predecessor $M_c = \text{pre}^C(M_i, M_j)$, a match $m \in p\text{check}(\text{merge}^\text{min}_{G}(\text{proj}(M_{1, \ldots, n}), i), \text{proj}(M_{1, \ldots, n}), j), \phi)$ is also a valid match into $\bigcup_{M_i, M_j \in M_{1, \ldots, n}} M_i$. Hence, there exists a corresponding match $m_{mv} : Q_{mv} \rightarrow \text{MVM}$ with $m = \text{trans}_{mv}(m_{mv})$.

Furthermore, in order to be present in the merge result, each element in the image of $m$ needs to be contained in $M_i$ or $M_j$ and, therefore, $\forall x \in m(\phi): x \in M_i \cup M_j$. We also know that none of the elements are deleted by the modification sequence $M_c \rightarrow^* M_i$ or $M_c \rightarrow^* M_j$, which means that $\forall x \in m(Q): x \in M_c \rightarrow (x \in M_i \land x \in M_j)$. We thus also have that $\forall v \in m_{mv}(V^{Q_{mv}}) : M_i \in p(v) \vee M_j \in p(v)$ and $\forall v \in m_{mv}(V^{Q_{mv}}) : M_j \in p(v) \rightarrow (M_i \in p(v) \land M_j \in p(v))$. Thereby, it must also hold that $M_i \in \text{arg} \min_{P \in \{p(v)\mid v \in m_{mv}(V^{Q_{mv}})\}} |P| \vee M_j \in \text{arg} \min_{P \in \{p(v)\mid v \in m_{mv}(V^{Q_{mv}})\}} |P|$. Because $i < j$, it follows that $(i, j, c, m) \in p\text{check}_{mv}^m(M_{1, \ldots, n}, \phi)$.

Thus, we have:

$$p\text{check}_{mv}^m(\text{MVM}, \phi) \supseteq \bigcup_{(i, j, c) \in Y} \{(i, j, c, m)m \in p\text{check}(M^\text{min}_{i,j,c}, \phi)\},$$

By the two inclusions, we also have the equality. \qed

**Complexity of $p\text{check}_{mv}$** Effectively, the graphs $P^M_{mv} = \text{trans}_{mv}(P^M)$ and $Q_{mv} = \text{trans}_{mv}(Q)$ are only a different encoding of $P^M = \bigcup_{M_i \in M_{1, \ldots, n}} M_i$ and $Q$, where edges are replaced by nodes with a single incoming and outgoing edge. Since these newly introduced nodes only cause a constant-factor overhead in pattern matching using an appropriate matching strategy and the size of the two graphs increases only by a constant factor, the effort for finding all matches for $Q_{mv}$ into $P^M_{mv}$ is in $O(C(\bigcup_{M_i \in M_{1, \ldots, n}} M_i, Q))$.

The set $p(v_x)$ for each individual element $x$ with $v_x = \text{origin}^{-1}(x)$ can be computed via breadth-first searches starting in each node connected to $v_x$ via a $cv$ edge over $P^V_{mv}$ and stopping at nodes that are either connected to $v_x$ via a $dv$ edge or that have already been visited by a previous search. Since this ensures that the breadth-first searches do not perform redundant searches, the combined
effort is in $O(|\Delta^{M_1,...,M_n}|)$. Given appropriate data structures, the effort for a single intersection of two version sets is in $O(|M_{[1,...,n]}|)$. Hence, computing $P = \cap_{v \in V_{mv}} p_{mv}(v)$ for a single match $m_{mv}$ takes effort in $O(|V_{mv}| \cdot |\Delta^{M_1,...,M_n}|)$.

For a number of matches $X$ for $Q_{mv}$ into $P_{mv}^M$, the effort for $p_{check}^{mv}$ is then in $O(C(\bigcup_{M_i \in M_{[1,...,n]}} M_i, Q) + X \cdot |V_{mv}| \cdot |\Delta^{M_1,...,M_n}|)$.

**Complexity of $mcheck^{mv}$** We can precompute the latest common predecessors for any pair of versions $M_i$ and $M_j$ according to the definition in $O(|P_{mv}^V| \cdot |M_{[1,...,n]}| + |M_{[1,...,n]}|^4)$ via breadth-first search for finding all predecessors for every version and basic set operations. Since the maximally preserving model modification between two versions along identity morphisms is uniquely defined and therefore $|E_{mv}| \leq |M_{[1,...,n]}|^2$, the complexity is in $O(|M_{[1,...,n]}|^4)$. This computation allows an efficient enumeration of the set of model modification pairs representing possible merges.

Since information about creation and deletion of elements is not explicitly available in a model-modification-based representation of a versioning, finding all insert-delete conflicts between two model modifications via $mcheck$ can be done by checking for each edge in either modification’s resulting model whether it is created by that modification and its source or target is deleted by the other modification. Given appropriate data structures for storing the individual model versions, the check for one edge can be performed in constant time thanks to the requirement of partial identity morphisms. This yields a computational complexity in $O(|M_{[1,...,n]}|^4 + \sum_{(M_s \rightarrow M_t) \in Y} |E_{mv}^M| + |E_{mv}^V|)$ for checking an entire model versioning with $Y$ the set of possible merges represented by pairs of model modifications. Note that, in the worst case, there may exist up to $O(|M_{[1,...,n]}|^3)$ possible merges in a model versioning, so in the worst case, this implies effort in $O(|M_{[1,...,n]}|^4 + |E_{mv}^{max}| \cdot |M_{[1,...,n]}|^3)$, where $|E_{mv}^{max}|$ is the maximum number of edges present in a single model version.

The computation of $P = p(v_e)$ and $D$ for every node $v_e \in V_{MV M}$, where $corr_{mv}^{-1}(type_{MV M}(v_e)) \in E_{MV M}$ and $v_e$ has a cv edge to a node representing another version than $M_a$, and the edge source $v_s = s^{M}(\text{origin}(v_e))$ can be performed in $O(\Delta_a \cdot |P_{mv}^V|)$, where $P_{mv}^V$ is the version graph and $\Delta_a$ is the overall number of elements created by the model versioning. Checking whether $p(v_e) = p(v_s)$ only takes effort in $O(|M_{[1,...,n]}|)$.

For a node $v_e$ and versions $M_i$ in $P$ and $M_j$ in $D$, we can then determine all latest predecessors for which $M_i$ and $M_j$ are in conflict in $O(|M_{[1,...,n]}|)$. Since there are at most $|M_{[1,...,n]}|^2$ pairs of versions in $P \times D$, the overall complexity of $mcheck^{mv}$ considering the effort for precomputing latest common predecessors is in $O(|M_{[1,...,n]}|^4 + \Delta_a \cdot |M_{[1,...,n]}|^3)$, which simplifies to $O(\Delta_a \cdot |M_{[1,...,n]}|^3)$ if $|M_{[1,...,n]}| \leq \Delta_a$. The computation for edge targets works analogously and thus does not change the complexity.

**Complexity of $pcheck^{mv}$** We denote the effort for finding all matches for a pattern $Q$ in a model $M_m$ by $C(M_m, Q)$. A possible merge in a model versioning is characterized by two model modifications, that is, two model ver-
sions $M_i$ and $M_j$ and a latest common predecessor $M_c$. Disregarding the effort for computing a merge itself, the effort for finding all violations of a well-formedness condition characterized by $Q$ in a model versioning with $Y$ the set of possible merges via computing all merges and searching for $Q$ in the results is then given by $O(\sum_{(m_1, m_2) \in Y} C(\text{merge}^{\text{min}}_{G}(m_1, m_2), Q))$. Potentially, there are up to $O(|M_{\{1, \ldots, n\}}|^3)$ possible merges in a model versioning with versions $M_{\{1, \ldots, n\}}$, which can be computed in $O(|M_{\{1, \ldots, n\}}|^4)$. This results in a complexity for computing and checking all these merges via $pcheck$ in $O(|M_{\{1, \ldots, n\}}|^4 + \sum_{(m_1, m_2) \in Y} C(\text{merge}^{\text{min}}_{G}(m_1, m_2), Q))$.

Finding all violations characterized by a pattern $Q$ in a model versioning by $pcheck^m$ requires a search for $Q_{mv}$ over the corresponding multi-version model. As discussed in Section 4.1, this requires effort in $O(C(\bigcup_{M_i \in M_{\{1, \ldots, n\}}} M_i, Q))$.

Then, for each structural match that is found, the set $P_v = p(v)$ has to be computed for each $v \in V^{Q_{mv}}$, which takes $O(|V^{Q_{mv}}| \cdot |P^v_{mv}|)$ computational steps. Finding all complementary versions $M_j$ regarding $Q_{mv}$ for a version $M_i$ requires $O(|V^{Q_{mv}}| \cdot |M_{\{1, \ldots, n\}}|)$ steps.

The number of candidates of complementary versions is in $O(|M_{\{1, \ldots, n\}}|^2)$. Latest common predecessors for all version pairs can be computed and cached upfront in $O(|M_{\{1, \ldots, n\}}|^4)$. For each latest common predecessor, the remaining checks can be performed in $O(|V^{Q_{mv}}|)$ given appropriate data structures. Since any pair of versions has at most $|M_{\{1, \ldots, n\}}|$ common predecessors, we thus have effort in $O(|V^{Q_{mv}}| \cdot |M_{\{1, \ldots, n\}}|^3)$ for each match.

For $X$ matches for $Q_{mv}$ into $MVM$, the overall computational effort is thus in $O(|M_{\{1, \ldots, n\}}|^4 + C(\bigcup_{M_i \in M_{\{1, \ldots, n\}}} M_i, Q) + X \cdot |V^{Q_{mv}}| \cdot |M_{\{1, \ldots, n\}}|^3)$.
B Violation Patterns of Considered Well-Formedness Constraints

Fig. 6. Violation pattern for the constraint “unique superclass”

Fig. 7. Violation pattern for the constraint “unique return type”
Fig. 8. Violation pattern for the constraint "consistent override"