Karlsruher Institut für Technologie

Karlsruhe Reports in Informatics 2020,3
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

A Formal Approach to Prove Compatibility in Transformation Networks

Heiko Klare, Aurélien Pepin, Erik Burger, Ralf Reussner

2020

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

A Formal Approach to Prove Compatibility in Transformation Networks

Heiko Klare · Aurélien Pepin · Erik Burger · Ralf Reussner

Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology (KIT), Germany
{klare, burger, reussner}@kit.edu, {aurelien.pepin}@alumni.kit.edu

Abstract The increasing complexity of software and cyber-physical systems is handled by dividing the description of the system under construction into different models or views, each with an appropriate abstraction for the needs of specific roles. Since all such models describe the same system, they usually share an overlap of information, which can lead to inconsistencies if overlapping information is not modified uniformly in all models. A well-researched approach to make these overlaps explicit and resolve inconsistencies are incremental, bidirectional model transformations. They specify the constraints between two metamodels and the restoration of consistency between their instances. Relating more than two metamodels can be achieved by combining bidirectional transformations to a network. However, such a network may contain cycles of transformations, whose consistency constraints can be contradictory if they are not aligned with each other and thus cannot be fulfilled at the same time. Such transformations are considered *incompatible*.

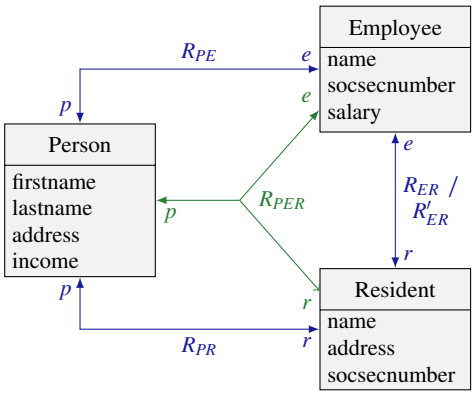
In this article, we provide a formal definition of consistency and compatibility of transformations and propose an inductive approach to prove compatibility of a given network of transformations. We prove correctness of the approach based on these formal definitions. Furthermore, we present an operationalization of the approach at the example of QVT-R. It detects contradictions between relations by transforming them into first-order logic formulae and evaluating them with an SMT solver. The approach operates conservatively, i.e., it is not able to prove compatibility in all cases, but it identifies transformations as compatible only if they actually are. We applied the approach to different evaluation networks and found that it operates conservatively and is able to properly prove compatibility in 80% of the cases, indicating its practical applicability. Its limitations especially arise from restricted capabilities of the used SMT solver, but not from conceptual shortcomings. Our approach enables multiple domain experts to define transformations independently and to check their compatibility when combining them to a network, relieving them from the necessity to align the transformations with each other a priori and to ensure compatibility manually.

Keywords model consistency · model transformation · transformation networks · transformation decomposition · transformation compatibility

1 Introduction

The scale of modern software systems and their embedding into cyber-physical systems leads to a high and even increasing complexity of systems to be built. To handle that complexity, different roles operate on appropriate extracts and abstractions of the system under construction described by different models or views. Such a fragmentation of information across different models is common at a high level, i.e., mechanical, electrical and software engineers usually use different models and associated tools to describe a system in their domain. Additionally, different models can be used on a low level by engineers from the same domain, such as software engineers using different models for architecture specification, behavior development and deployment. For example, the development of Electronic Control Units (ECUs) software in automotives comprises different tools or standards for specifying the system and software architecture, such as SysML [49] or AUTOSAR [59], for defining the behavior, such as MATLAB/Simulink [42] or ASCET [20], and for defining the deployment on multi-core hardware architectures, such as Amalthea [24, 75]. Since all these models describe the same system, they usually share an overlap of information in terms of dependencies or redundancies, which can lead to inconsistencies if overlapping information is not modified properly in all models. Recent research investigated such dependencies between ASCET and SysML [22], as well as Amalthea and how to resolve them [44, 43].

Incremental model transformations are a common approach to resolve such inconsistencies by enabling developers to explicitly specify how inconsistencies can be resolved (semi-)automatically. Especially bidirectional model transformations [63], which specify the relations between two metamodels and routines how consistency of their instances can be restored, are well suited and well researched. Relating more than two metamodels can either be achieved by defining a multi-directional transformation between all of them or by specifying bidirectional transformations between pairs of them in a modular way and combine them to a network that is able to check and preserve consistency between several models. Figure 1 exemplifies these different possibilities at the example of relation of transformations between three



$$R_{PER} = \{\langle p,e,r\rangle \mid p.firstname + "\text{␣}" + p.lastname = e.name = r.name \wedge p.address = r.address \wedge p.income = e.salary \wedge e.socsecnumber = r.socsecnumber\}$$

$$R_{PE} = \{\langle p,e\rangle \mid p.firstname + "\text{␣}" + p.lastname = e.name \wedge p.income = e.salary\}$$

$$R_{PR} = \{\langle p,r\rangle \mid p.firstname + "\text{␣}" + p.lastname = r.name \wedge p.address = r.address\}$$

$$R_{ER} = \{\langle e,r\rangle \mid e.name = r.name \wedge e.socsecnumber = r.socsecnumber\}$$

$$R'_{ER} = \{\langle e,r\rangle \mid e.name.toLower = r.name \wedge e.socsecnumber = r.socsecnumber\}$$

Fig. 1: Three simple metamodels for persons, employees and residents, one ternary relation R_{PRE} between them and three binary relations R_{PE}, R_{PR}, R_{ER} for each pair of them, with R'_{ER} as an alternative for R_{ER}.

simple metamodels for persons, employees and residents. We use an informal notion of consistency, defined more precisely later on, which requires that if any person, employee or resident is contained in a model, there must also be the other two elements with the same names, addresses, incomes and social security numbers. This relation can either be expressed as a ternary relation, denoted as R_{PER}, or as three binary relations R_{PE}, R_{PR}, R_{ER}. In such a simple scenario a single developer may be able to define all these relations. However, in a more complex scenarios, like the relations between the previously mentioned SysML, Amalthea and ASCET metamodels, there may not be a single person having the knowledge about all these dependencies [52], but there may be different domain experts knowing about relations between subsets of the metamodels [26]. Additionally, it is difficult to think about complex multiary relations [62]. In consequence, building networks of bidirectional transformations provides several benefits over building multi-directional transformations.

Such a network of bidirectional transformations may contain cycles of transformations. Figure 1 exemplifies why it may be unavoidable to have such cycles. There is no pair of binary relations, such that it is equivalent to the ternary relation R_{PER}, because each pair of metamodels shares unique information that is not represented in the third one. An essential issue with such cycles is that they impose the possibility of defining contradictory constraints, such that the relations cannot be fulfilled at the same time. In such a case, the relations are considered *incompatible*. Consider the three binary relations R_{PE}, R_{PR}, R'_{ER} in Figure 1. These relation cannot always be fulfilled, because R'_{ER} requires the resident name to be lowercase, whereas the other relations relate the names as they are and thus allow the lowercase names. In consequence, for a resident with a non-lowercase name it is not possible to find a consistent person and employee. However, in a transformation network, compatibility of the relations defined by the transformations is a necessary requirement for their repair routines to properly restore consistency [28].

In this article, we consider the relations defined by bidirectional transformations. We clarify a notion of *compatibility* for such relations and develop an approach to prove compatibility of relations in a given network of transformations. To achieve this, we formally define a notion of consistency, based on fine-grained consistency relations, as well as compatibility. Building on this formalism, we are able to derive an inductive, formal approach for proving compatibility of relations by identifying those that are redundant. Its essential idea is that if consistency relations have a specific kind of tree structure, we are able to show that they are inherently compatible. Furthermore, we show that adding redundant relations to such a tree preserves compatibility. In consequence, reducing an arbitrary network of relations to a tree by removing redundant relations proves compatibility. Finally, we present an operationalized approach at the example of QVT-R, which proves compatibility in a network of QVT-R relations. It transforms QVT-R relations into first-order logical formulae and finds redundant relations by applying an SMT solver. In detail, we make the following contributions:

Compatibility Formalization (C1): We formalize a notion of consistency and precisely define *compatibility* of relations in a network of transformation.

Formal Approach (C2): We define a formal, inductive approach for proving compatibility of relations based on a notion of redundancy and relation trees.

Operationalized Approach (C3): We propose an approach that applies the formalism to QVT-R and show how a translation to logical formulae and the usage of SMT solver can be used to prove compatibility.

Applicability Evaluation (C4): While correctness of the approach is given by construction and proven on the formalism, we apply the approach to case studies to show applicability of the approach.

Notations	
$\mathbb{S} = \{a, b, \ldots\}$	Notation for a set $\mathbb{S}$ of elements
$\mathfrak{T} = \langle a, b, \ldots \rangle$	Notation for a tuple $\mathfrak{T}$ of elements
Properties and Classes	
P	Property (attribute or reference)
$I_P = \{p_1, p_2, \ldots\}$	Property values of a property P
$C = \langle P_1, \ldots, P_n \rangle$	Class
$I_C = \{o = \langle p_1, \ldots, p_n \rangle \mid p_i \in I_{P_i}\}$	Instances (objects) of a class C
$o \in I_C$	Object of a class C
(Meta-)Models	
$M = \{C_1, \ldots, C_m\}$	Metamodel
$I_M = \{m \mid m \subseteq \bigcup_{C \in M} I_C\}$	Instances of a metamodel
$\mathbb{M} = \{M_1, \ldots, M_k\}$	Set of metamodels
$I_{\mathbb{M}} = \{\{m_1, \ldots, m_k\} \mid m_i \in I_{M_i}\}$	Instances of a metamodel set $\mathbb{M}$
$m \in I_M$	Model of metamodel M
$\mathfrak{m} \in I_{\mathbb{M}}$	Model set of a metamodel set $\mathbb{M}$

Table 1: Notations and elements

It is, in general, not possible to prove that transformations are incompatible if the language used to describe consistency relations has sufficient expressiveness and is thus undecidable, such as QVT-R. On the other hand, it is possible to prove that transformations are compatible. Our approach is designed to operate conservatively, thus in cases it claims compatibility, the transformations actually are compatible. However, there may be cases in which relations are compatible but the approach is not able to prove that.

The main benefit of our approach is that it enables domain experts to define transformations independently and to automatically detect their compatibility. This supports two development scenarios: First, when transformations are developed concurrently, they can be analyzed for compatibility continuously or even on the fly. Second, when transformations are developed independently, in the extreme case as reusable components to be applied in various contexts, i.e., with different sets of other transformations, compatibility can be analyzed whenever transformations are combined to a network. In both scenarios, our approach releases developers them from the necessity to align the transformations with each other a priori and ensuring compatibility manually.

2 Notation and Assumptions

In this section, we introduce basic terminology, notations and assumptions that we make throughout the article. We give an overview of the used notation and elements in Table 1.

2.1 Notation

We usually denote variables representing sets of any kinds of elements in blackboard bold font $\mathbb{S}$, those representing tuples of any kinds of elements in gothic font $\mathfrak{T}$ and write the elements of a tuple in angle brackets $\langle a, b, \ldots \rangle$.

We define the following operators for concise expressions on tuples, which basically allow to treat tuples as sets wherever necessary: For a tuple $\mathfrak{t} = \langle t_1, \ldots, t_n \rangle$, we say that:

$$t' \in \mathfrak{t} :\Leftrightarrow \exists i \in \{1, \ldots, n\} : t' = t_i$$

For two tuples $\mathfrak{t}_1, \mathfrak{t}_2$, we define:

$$\mathfrak{t}_1 \cap \mathfrak{t}_2 := \{t \mid t \in \mathfrak{t}_1 \wedge t \in \mathfrak{t}_2\}$$

Note that the intersection of tuples is not a tuple but a set, because we are only interested in getting the elements contained in both tuples but do not need to match their order.

Furthermore, we use variables of uppercase letters for all elements at the metamodel level, such as M for a metamodel or C for a class, whereas we use lowercase letters for all elements at the model level, such as m for a model and o for an object. If not further specified, we use the same indices on related elements on the metamodel and the model level, such as model m_1 being an instance of metamodel M_1.

2.2 Elements

In general, we consider metamodels as a composition of meta-classes, which in turn are composed of properties representing attributes or references. Models instantiate metamodels and are composed of objects, which are instances of meta-classes and in turn consist of property values, which instantiate properties.

We denote *properties*, which are the information a meta-class consists of, such as attributes or references, as P and the *property values* as instances of a property as $I_P = \{p_1, p_2, \ldots\}$ of property P. We do not need to further differentiate different types of properties into attributes and references, like it is done in other formalizations, such as the OCL standard [47, A.1] or the thesis of Kramer [31, 2.3.2].

We denote *meta-classes*, in the following shortly called *classes*, as tuples of properties $C = \langle P_1, \ldots, P_n \rangle$. Instances of a *class* are *objects*, each being a tuple of instances of the properties of the class and we denote all instances of a class C as $I_C = \{o = \langle p_1, \ldots, p_n \rangle \mid p_i \in I_{P_i}\}$.

We denote a metamodel $M = \{C_1, \ldots, C_m\}$ as a finite set of classes. The instances of a metamodel are sets of objects $I_M = \{m \mid m \subseteq \bigcup_{C \in M} I_C\}$. Each instance, denoted as a *model*, is a finite set of objects that instantiate the classes in the metamodel. For a set of metamodels $\mathbb{M} = \{M_1, \ldots, M_k\}$, we denote the set that contains all sets of instances of that metamodels as $I_{\mathbb{M}} = \{\{m_1, \ldots, m_k\} \mid m_i \in I_{M_i}\}$.

2.3 Assumptions

We assume models to be finite, so for each model m, we assume that $|m| < \infty$. Additionally, our formalism assumes objects to be unique within a model m. This is already implicitly covered by the definition of I_M for the instances of a metamodel M. In practice, it is usually allowed to have the same object, i.e., an element with the same type, attribute and reference values, multiple times within the same model. This is, however, only a matter of identity, which we assume, without loss of generality, to be represented within the objects. In practice, identity is usually given by different objects being placed at specific places in memory or by assigning unique IDs to them, which allow the existence of and the possibility to distinguish multiple objects with the same attribute and reference values.

Finally, we assume consistency to be defined in terms of multiple bidirectional transformations, even if more than two metamodels are related. It has to be noted that not all multiary relations can be expressed by binary ones [62]. Thus, in theory, combining multiple bidirectional transformations is less expressive than using a multi-directional transformation. We have, however, already discussed in Section 1 why using multiple bidirectional transformations is still a reasonable approach in comparison to defining multi-directional transformations from a practical perspective: complexity and distributed knowledge foster the modular development of transformations, as argued in existing work [62, 26]. Finally, our assumption is especially for reasons of simplicity in understanding the concepts but can also be extended to the multiary case. Even if multi-directional transformations are to be used, there will not only be a single one, but potentially a combination of multiple such transformations. Thus, there is still need for checking compatibility.

3 Compatibility in Transformation Networks

In this section, we precisely define our notion of consistency, and motivate and formally introduce the term *compatibility*. We first discuss properties of transformation networks with an intuitive notion of compatibility, based on considerations in existing work. We then define consistency based on fine-grained consistency relations and, finally, derive a compatibility notion from the consistency formalization and its pursued perception. This serves as our contribution **C1**.

3.1 Properties of Transformation Network

Keeping pairs of models consistent by means of incremental, bidirectional transformations has been well researched in recent years [63, 33, 11]. A bidirectional transformation consists of a *relation* that specifies which pairs of models are considered consistent and a pair of directional transformations, denoted as *consistency repair routines*, that take one modified and one originally consistent model and deliver a new model that is consistent to the modified one [63]. Several well-defined properties of such transformations have been identified. The essential *correctness* property states that a consistency repair routine delivers a result such that the models are actually consistent according to the defined relation [63]. Another important property is *hippocraticness*, which states that a consistency repair routine returns the input model if it was already consistent to the modified one [63].

When we combine several transformations to a network to achieve consistency between multiple models, those properties of the single transformations are still relevant, as each transformations on its own has to be at least correct to work properly in a network of transformations. However, correctness of the single transformations does not induce correctness of the transformation network. Taking an arbitrary set of correct transformations and executing them one after another does not necessarily constitute a terminating approach that delivers a result, in which all models are consistent according to the relations of the transformations, because the result of one transformation may violate the relation of another. It is possible that the approach does either not terminate, because there is a divergence or alternation in values changed or elements created, or terminates in a state that is not consistent regarding the relations of all transformations [28]. The property of a network to always result in a state in which the models are consistent to all relations of the transformations if they are executed in a specific order, can be seen as a *correctness* property for transformation networks. In this work, we focus on that correctness property and do not discuss further quality properties of transformation networks, such as *modularity*, *evolvability* and *comprehensibility* [26].

A single transformation can only be incorrect in terms of its repair routines, because there are no restrictions regarding its relation that may prevent the repair routine from being able to produce a correct result. In previous work, however, we identified that transformation networks can be incorrect at different levels [28]. Networks can also be incorrect at the level of relations rather than repair routines, because multiple relations can be contradictory, i.e., they can relate elements in different ways such that the relations cannot be fulfilled at the same time. In such a case, the consistency repair routines cannot be result in a state that is consistent according to the relations anymore, thus they may not terminate anymore. We call the relations of such transformations *incompatible*.

In consequence, compatibility of relations is a necessary prerequisite for consistency repair routines to produce correct results in transformation networks. We also found that correctness of the consistency repair routines can already be achieved by construction, whereas compatibility of the relations cannot be achieved by construction but in the best case

be checked for a set of relations. In this work, we focus on the possibility to check compatibility of the relations of a set of transformations. In the following, we therefore precisely define the notion of compatibility of relations, which excludes contradictions in relations that can prevent consistency repair routines from fulfilling the relation.

Finally, the topology of a transformation network directly influences how prone it is to incompatibilities of its relations. Contradictions of consistency relations, as exemplified with the relations R_{PE}, R_{PR}, R'_{ER} in Figure 1, can only occur if the same classes are related to each other by different (sequences of) transformations in a different way. For example, in Figure 1, each combination of two relations puts the same classes into relation as the third one. This means that a transformation network, in which each pair of classes is only related by one sequence of transformations, cannot have contradictory relations and is thus inherently compatible.

3.2 A Fine-grained Notion of Consistency

A common definition of consistency enumerates consistent pairs of models in a relation [63]. However, for our studies on compatibility, we need a more fine-grained notion of consistency. Considering transformations languages, such as QVT-R, first, relations are defined at the level of classes and their properties, i.e. how properties of instances of some classes are related to properties of instances of other classes. Second, they are defined in an *intensional* way, i.e., constraints specify which elements shall be considered consistent, rather than enumerating all consistent instances, known as an *extensional* specification. Both ways have equal expressiveness and especially each intensional specification can be transformed into an extensional one by enumerating all instances that fulfill the constraints. Since mathematical statements are easier to make on extensional specifications, we stick to them. However, we reuse the concept of specifying relations at the level of classes and their properties. This makes it easier to make statements about dependencies between consistency relations. For example, two fine-grained consistency relations considering completely independent sets of classes cannot interfere, and thus especially do not introduce any compatibility problems, which is not easy to express when considering relations at the level of complete models. Finally, from such a fine-grained specification, a holistic relation at level of models can always be derived by enumerating all models that fulfill all the fine-grained specifications, thus it does not restrict expressiveness in any way and can be seen as a *compositional approach* for defining consistency.

In the following, we start with introducing a fine-grained notion of consistency relations. We proceed with considerations on implicit relations, which are induced by a set of consistency relations, such as transitive relations, to finally precisely define a notion of compatibility.

3.2.1 Consistency

The first definitions on conditions and consistency relations are based on the work of Kramer [31, sec. 2.3.2, 4.1.1]. The central idea of the consistency notion is to have consistency relations, which contain pairs of objects and, broadly speaking, requires that if the objects in one side of the pair occur in a model, the others have to occur in another model as well.

Definition 1 (Condition) A condition $\mathbb{c}$ for a class tuple $\mathfrak{C}_{\mathbb{c}} = \langle C_{\mathbb{c},1}, \ldots, C_{\mathbb{c},n} \rangle$ is a set of object tuples with:

$$\forall \langle o_1, \ldots, o_n \rangle \in \mathbb{c} : \forall i \in \{1, \ldots, n\} : o_i \in I_{C_{\mathbb{c},i}}$$

An element $\mathfrak{c} \in \mathbb{c}$ is called a *condition element*. For a set of models $\mathbb{m} \in I_{\mathbb{M}}$ of a metamodel set $\mathbb{M}$ and a condition element $\mathfrak{c}$, we say that:

$$\mathbb{m} \textit{ contains } \mathfrak{c} :\Leftrightarrow \exists m \in \mathbb{m} : \mathfrak{c} \subseteq m$$

Conditions represent object tuples that instantiate the same tuple of classes. They are supposed to occur in models that fulfill a certain condition regarding consistency, i.e., they define the objects that can occur in the previously mentioned pairs of consistency relations, which we specify later. We say that a set of models contains a condition element if any of the models contains all the objects within the condition element. This implies that the metamodel of such a model has to contain all the classes in the class tuple of the condition.

Definition 2 (Consistency Relation) Let $\mathfrak{C}_{l,CR}$ and $\mathfrak{C}_{r,CR}$ be two class tuples. A consistency relation *CR* is a subset of pairs of condition elements in conditions $\mathbb{c}_{l,CR}, \mathbb{c}_{r,CR}$ with $\mathfrak{C}_{l,CR} = \mathfrak{C}_{\mathbb{c}_{l,CR}}$ and $\mathfrak{C}_{r,CR} = \mathfrak{C}_{\mathbb{c}_{r,CR}}$:

$$CR \subseteq \mathbb{c}_{l,CR} \times \mathbb{c}_{r,CR}$$

We call a pair of condition elements $\langle \mathfrak{c}_l, \mathfrak{c}_r \rangle \in CR$ a *consistency relation pair*. For a set of models $\mathbb{m}$ and a consistency relation pair $\langle \mathfrak{c}_l, \mathfrak{c}_r \rangle$, we say that:

$$\mathbb{m} \textit{ contains } \langle \mathfrak{c}_l, \mathfrak{c}_r \rangle :\Leftrightarrow \mathbb{m} \textit{ contains } \mathfrak{c}_l \wedge \mathbb{m} \textit{ contains } \mathfrak{c}_r$$

Without loss of generality, we assume that each condition element of both conditions occurs in at least one consistency relation pair, i.e.

$$\forall \mathfrak{c} \in \mathbb{c}_l : \exists \langle \mathfrak{c}_l, \mathfrak{c}_r \rangle \in CR : \mathfrak{c} = \mathfrak{c}_l$$
$$\wedge \forall \mathfrak{c} \in \mathbb{c}_r : \exists \langle \mathfrak{c}_l, \mathfrak{c}_r \rangle \in CR : \mathfrak{c} = \mathfrak{c}_r$$

A consistency relation according to Definition 2 is a set of pairs of object tuples, which are supposed to indicate the tuples of objects that are considered consistent with each other, i.e., if a model contains one of the left object tuples occurs in the relation one of the related right object tuples has to occur in a model as well. It is based on two conditions that define relevant object tuples in each of the two metamodels and defines the ones that are related to each other.

We can now define a notion of consistency for a set of models based on the definition of consistency relations.

Definition 3 (Consistency) Let CR be a consistency relation and let $\mathbb{m} \in I_{\mathbb{M}}$ be a set of models of the metamodels in $\mathbb{M}$. We say that:

$$\begin{aligned}&\mathbb{m} \textit{ consistent to } CR :\Leftrightarrow\\ &\quad \exists W \subseteq CR : \big(\forall \langle \mathfrak{c}_{l,1}, \mathfrak{c}_{r,1}\rangle, \langle \mathfrak{c}_{l,2}, \mathfrak{c}_{r,2}\rangle \in W :\\ &\qquad \langle \mathfrak{c}_{l,1}, \mathfrak{c}_{r,1}\rangle = \langle \mathfrak{c}_{l,2}, \mathfrak{c}_{r,2}\rangle \vee (\mathfrak{c}_{l,1} \neq \mathfrak{c}_{l,2} \wedge \mathfrak{c}_{r,1} \neq \mathfrak{c}_{l,2})\big)\\ &\qquad \wedge \forall \langle \mathfrak{c}_l, \mathfrak{c}_r\rangle \in W : \mathbb{m} \textit{ contains } \mathfrak{c}_l \wedge \mathbb{m} \textit{ contains } \mathfrak{c}_r\\ &\qquad \wedge \forall \mathfrak{c}'_l \in \mathbb{c}_{l,CR} : \mathbb{m} \textit{ contains } \mathfrak{c}'_l \Rightarrow \mathfrak{c}'_l \in \mathbb{c}_{l,W}\end{aligned}$$

We call such a W a *witness structure* for consistency of $\mathbb{m}$ to CR, and for all elements $\langle \mathfrak{w}_l, \mathfrak{w}_r\rangle \in W$, we call $\mathfrak{w}_l$ and $\mathfrak{w}_r$ *corresponding to* each other.

For a set of consistency relations $\mathbb{CR} = \{CR_1, CR_2, \ldots\}$, we say that:

$$\begin{aligned}&\mathbb{m} \textit{ consistent to } \mathbb{CR} :\Leftrightarrow\\ &\quad \forall CR \in \mathbb{CR} : \mathbb{m} \textit{ consistent to } CR\end{aligned}$$

A consistency relation CR relates one condition element at the left side to one or more other condition elements at the right side of the relation. The definition of consistency ensures that if one condition element $\mathfrak{c} \in \mathbb{c}_{l,CR}$ in the left side of the relation occurs in a set of models, exactly one of the condition elements related to it by a consistency relation CR occurs in another model to consider the set of models consistent. If another element that is related to $\mathfrak{c}$ occurs in the models, this one has to be, in turn, related to another condition element $\mathfrak{c}' \in \mathbb{c}_{l,CR}$ of the left side of condition elements by CR, which also occurs in the models. This is a necessary restriction, because usually a single corresponding element is expected, as we will see in examples in the following. To achieve that, the definition uses an auxiliary structure W, which serves as a witness structure for those pairs of condition elements that co-occur in the models.

Example 1 The definition of consistency is exemplified in Figure 2, which is an alternation of an extract of Figure 1 only considering employees and residents. Models with employees and residents are considered consistent if for each employee exactly one resident with the same name or the same name in lowercase exists. The model pairs 1–3 are obviously consistent according to the definition, because there is always a pair of objects that fulfills the consistency relation. In model pair 4, there is a consistent resident for each employee, but there is no appropriate employee for the resident with *name* = "*Alice*". However, our definition of consistency only requires that for each condition element of the left side of the relation that appears in the models an appropriate right element occurs, but not vice versa. Thus, a relation is interpreted unidirectionally, which we will discuss in more detail in the following. In model pair 5, there are two residents with names in different capitalizations, which would both be considered consistent to the employee according to the consistency relation. Comparably, in model pair 6, there is a resident that fulfills the consistency relations for both employees, each having a different but matching capitalization. However, the consistency definition requires that each element in a model for which consistency is defined by a consistency relation may only have one corresponding element in the model. In this case, there are two residents respectively two employees that could be considered consistent to the employee respectively resident, thus there is no appropriate witness structure with a unique mapping between the elements as required by the consistency definition.

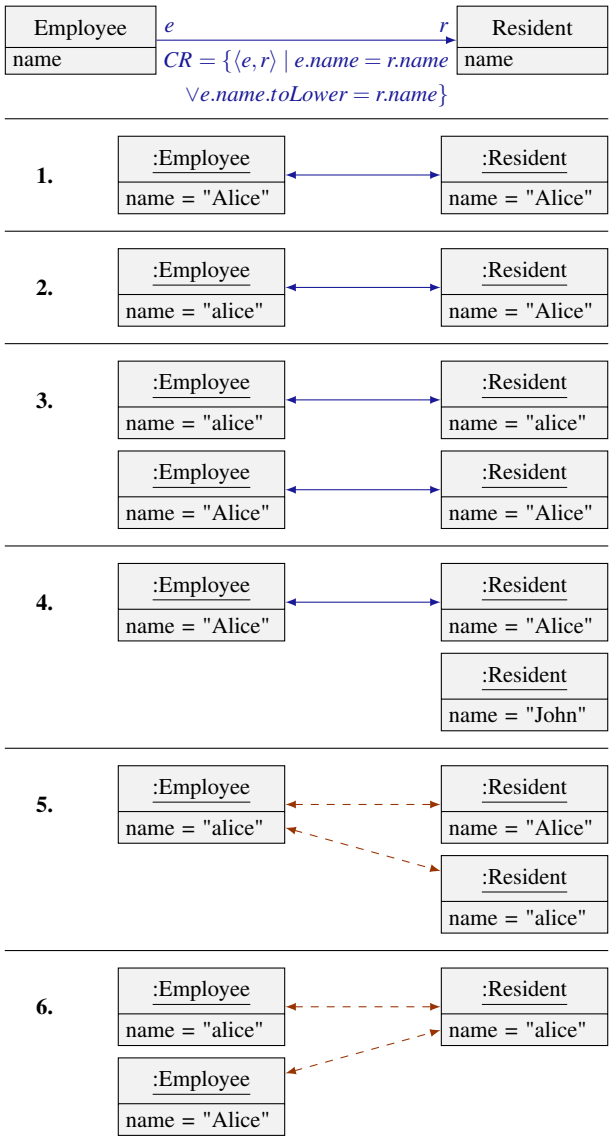


Fig. 2: A consistency relation between employee and resident and six example model pairs: model pairs 1–4 being consistent with an appropriate witness structure W shown in blue and model pair 5 and 6 being inconsistent with an inappropriate mapping structure shown in red and dashed.

As mentioned before, we define the notion of consistency in a unidirectional way, which means that a consistency relation may define that some elements $\mathfrak{c}_r$ are required to occur in a set of models if some elements $\mathfrak{c}_l$ occur, but not vice versa. Such a unidirectional notion can also be reasonable in our example, as it could make sense to require a resident for each employee, but not every resident might also be an employee. To achieve a bijective consistency definition, for each consistency relation CR its transposed relation $CR^T = \{\langle \mathfrak{c}_l, \mathfrak{c}_r\rangle \mid \langle \mathfrak{c}_r, \mathfrak{c}_l\rangle \in CR\}$ can be considered as well. Regarding Figure 2, if we consider the relation between employees and residents as well as its transposed, the model pair 4 would also be considered inconsistent, because an appropriate employee for each resident would be required by the transposed relation. We call sets of consistency relations that contain only bijective definitions of consistency *symmetric*.

Definition 4 (Symmetric Consistency Relation Set) Let $\mathbb{CR}$ be a set of consistency relations. We say that:

$$\mathbb{CR} \textit{ is symmetric } :\Leftrightarrow$$
$$\forall CR \in \mathbb{CR} : \exists CR' \in \mathbb{CR} : CR' = CR^T$$

Any description of bijective consistency relations can be achieved by defining a symmetric set of consistency relations. We chose to define consistency in a unidirectional way due to two reasons:

1. Some relevant consistency relations are actually not bijective. Apart from the simple example concerning residents and employees, this situation always occurs when objects at different levels of abstraction are related. Consider a relation between components and classes, requiring for each component an implementation class but not vice versa, or a relation between UML models and object-oriented code, requiring for each UML class an appropriate class in code but not vice versa. These relations could not be expressed if consistency relations were always considered bidirectional for determining consistency.
2. We consider networks of consistency relations, in which, as we will see later, a combination of multiple bijective consistency relations does not necessarily imply a bijective consistency relation again. Thus, we need a unidirectional notion of consistency relations anyway.

3.2.2 Implicit Consistency Relations

Each set of consistency relations defines binary consistency relations, each between two sets of classes. However, such consistency relations imply further *transitive* consistency relations. Having one relation between classes A and B and one between B and C implies an additional relation between A and C, for which we define a notion for the concatenation of relations. The goal of this notion is to provide a relation that is induced by the concatenated ones. This means, if a model is consistent to the concatenation, it should also be consistent to the single relations, as otherwise the concatenation would introduce additional consistency constraints. To achieve this, the following definition makes appropriate restrictions to the derived consistency relation pairs.

Definition 5 (Consistency Relations Concatenation) Let CR_1, CR_2 be two consistency relations. The concatenation is defined as follows:

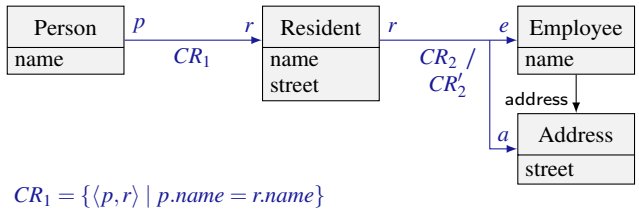
$$\begin{aligned} CR = CR_1 \otimes CR_2 := \{\langle \mathfrak{c}_l, \mathfrak{c}_r\rangle \mid \\ \exists \langle \mathfrak{c}_l, \mathfrak{c}_{r,1}\rangle \in CR_1 : \exists \langle \mathfrak{c}_{l,2}, \mathfrak{c}_r\rangle \in CR_2 : \mathfrak{c}_{r,1} \subseteq \mathfrak{c}_{l,2} \\ \wedge \forall \langle \mathfrak{c}_l, \mathfrak{c}'_{r,1}\rangle \in CR_1 : \exists \langle \mathfrak{c}'_{l,2}, \mathfrak{c}'_{r,2}\rangle \in CR_2 : \mathfrak{c}'_{l,2} \subseteq \mathfrak{c}'_{r,1}\} \end{aligned}$$

with $\mathfrak{C}_{l,CR} = \mathfrak{C}_{l,CR_1}$ and $\mathfrak{C}_{r,CR} = \mathfrak{C}_{r,CR_2}$

The concatenation of two consistency relations contains all pairs of object tuples that are related across common elements in the right respectively left side of the consistency relation pairs. Such a concatenation may be empty. Two requirements ensure that all models considered consistent to the concatenated relation are also consistent to the single relations: First, it is important that a pair of consistency relations CR_1, CR_2 is only combined if the left condition element of the consistency relation pair from CR_2 is a subset of the right condition element of the consistency relation pair $\langle \mathfrak{c}_l, \mathfrak{c}_r\rangle$ of CR_1. Second, it is necessary that for all elements $\mathfrak{c}_r$ in the right side of CR_1, to which a condition element $\mathfrak{c}_l$ is considered consistent, there must be a matching condition element, i.e. a subset of $\mathfrak{c}_r$, in the left condition of CR_2. Otherwise, in both cases the occurrence of $\mathfrak{c}_l$ in a model set would not necessarily impose any consistency requirement by CR_2. In the following, we explain these two requirements at an example.

Example 2 Figure 3 extends the initial example (Figure 1) with further classes in the consistency relations, such that they do not only relate single classes to each other. It defines an address for employees and in the second example also a location for the address of residents, which are represented in additional classes. Both examples contains a consistency relation CR_1 respectively CR_3 between persons and residents, which define that for each person a resident with the same name has to exist. The examples provide different options for consistency relation between residents (with locations) and employees with addresses (CR_2, CR'_2, CR_4), which exemplify the necessity for the restrictions in Definition 5:

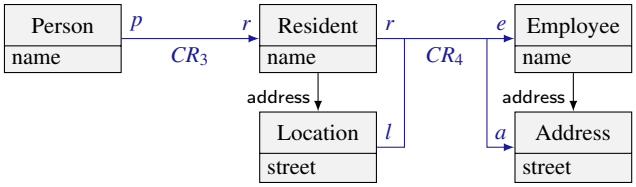
1. $CR_1 \otimes CR_2$: CR_2 requires for each resident an employee with the same name and an address with an arbitrary street name. In consequence, $CR_1 \otimes CR_2$ defines a relation for each person with an employee having the same name and all addresses with possible street names. All models that are consistent to the concatenation are also consistent to the single relations.



$CR_1 = \{\langle p, r\rangle \mid p.name = r.name\}$

$CR_2 = \{\langle r, (e,a)\rangle \mid r.name = e.name \wedge r.street = a.street\}$

$CR'_2 = \{\langle r, (e,a)\rangle \mid \langle r, (e,a)\rangle \in CR_2 \wedge r.street \neq ""\}$



$CR_3 = \{\langle p, r\rangle \mid p.name = r.name\}$

$CR_4 = \{\langle (r,l), (e,a)\rangle \mid r.name = e.name \wedge l.street = a.street\}$

Fig. 3: Two scenarios, each with two consistency relations: Consistency relations CR_1 and two options CR_2, CR'_2 with $CR_1 \otimes CR_2 \neq \emptyset$ and $CR_1 \otimes CR'_2 = \emptyset$, and consistency relations CR_3 and CR_4 with $CR_3 \otimes CR_4 = \emptyset$ and $CR_4^T \otimes CR_3^T \neq \emptyset$

2. $CR_1 \otimes CR'_2$: CR'_2 is similar to CR_2 but additionally requires that the street of a resident must not be empty. In consequence, for a resident with an empty address it is not required that an employee exists. This results in $CR_1 \otimes CR'_2 = \emptyset$, because for any person, there must not be an employee, as the person can be consistent to a resident with an empty street name. This shows the necessity of the second restriction in the definition.
3. $CR_3 \otimes CR_4$: The concatenation $CR_3 \otimes CR_4$ is obviously empty, because CR_3 requires a resident for each person, but CR_4 only requires an employee if there is also a location. Such a location does not necessarily exist if a person exists, thus if the models are consistent to CR_3 and CR_4 there must not necessarily be an employee for any contained person. This shows the necessity for the first restriction in Definition 5, which would require a left condition element from CR_4 (resident and location) to be a subset of a right condition element in CR_3 (resident).
4. $CR_4^T \otimes CR_3^T$: The concatenation of the transposed relations $CR_4^T \otimes CR_3^T$ is not empty, but actually contains all combinations of each possible employee with all addresses and relates them to a person with the same name. This is reasonable, because CR_4^T requires for all existing employees and addresses that an appropriate resident with the same name has to exist, which then requires a person with that name to exist due to CR_3^T. The definition does only cover that due to its first restriction, because $\mathfrak{c}_{l,2}$, i.e., the resident related to a person by CR_3^T is a subset of $\mathfrak{c}_{r,1}$, i.e., a tuple of resident and location.

We can formally show that the defined notion of concatenation does not lead to any restriction of consistency regarding the single relations:

Lemma 1 *Let CR_1, CR_2 be two consistency relations and let $CR = CR_1 \otimes CR_2$ be their concatenation. For all model sets $\mathbb{m} \in I_{\mathbb{M}}$ the following statement holds:*

$$\mathbb{m} \textit{ consistent to } \{CR_1, CR_2\} \Rightarrow \mathbb{m} \textit{ consistent to } CR$$

Proof For any set of models $\mathbb{m}$ that is consistent to CR_1 and CR_2, take the witness structure W_1 that witnesses consistency of $\mathbb{m}$ to CR_1 and W_2 that witnesses consistency of $\mathbb{m}$ to CR_2. Now consider the composed witness structure $W = W_1 \otimes W_2$. Let us assume there were $\langle \mathfrak{c}_l, \mathfrak{c}_r\rangle, \langle \mathfrak{c}'_l, \mathfrak{c}'_r\rangle \in W$ with $\mathfrak{c}_l = \mathfrak{c}'_l$ and $\mathfrak{c}_r \neq \mathfrak{c}'_r$. Per definition $\mathfrak{c}_l$ only occurs in one $\langle \mathfrak{c}_l, \mathfrak{c}_{r,1}\rangle \in W_1$. So there must be two $\langle \mathfrak{c}_{l,2}, \mathfrak{c}_r\rangle, \langle \mathfrak{c}'_{l,2}, \mathfrak{c}'_r\rangle \in CR_2$ with $\mathfrak{c}_{l,2} \subseteq \mathfrak{c}_{r,1}$ and $\mathfrak{c}'_{l,2} \subseteq \mathfrak{c}_{r,1}$. However, since $\mathfrak{c}_{l,2}$ and $\mathfrak{c}'_{l,2}$ contain instances of the same classes and are both subsets of the same other object tuples $\mathfrak{c}_{r,1}$, we have $\mathfrak{c}_{l,2} = \mathfrak{c}'_{l,2}$. So we know that:

$$\forall \langle \mathfrak{c}_{l,1}, \mathfrak{c}_{r,1}\rangle, \langle \mathfrak{c}_{l,2}, \mathfrak{c}_{r,2}\rangle \in W: \\ \langle \mathfrak{c}_{l,1}, \mathfrak{c}_{r,1}\rangle = \langle \mathfrak{c}_{l,2}, \mathfrak{c}_{r,2}\rangle \vee \mathfrak{c}_{l,1} \neq \mathfrak{c}_{l,2} \wedge \mathfrak{c}_{r,1} \neq \mathfrak{c}_{l,2}$$

Additionally, since W_1 and W_2 are witness structures for consistency of $\mathbb{m}$ to CR_1 and CR_2, the model set contains all condition elements in W_1 and W_2. Consequentially, $\mathbb{m}$ also contains the condition elements in W, as those in W are composed of the ones in W_1 and W_2. This implies that:

$$\forall \langle \mathfrak{c}_l, \mathfrak{c}_r\rangle \in W: \mathbb{m} \textit{ contains } \mathfrak{c}_l \wedge \mathbb{m} \textit{ contains } \mathfrak{c}_r$$

Finally, let us assume that:

$$\exists \mathfrak{c}'_l \in \mathfrak{C}_{l,CR}: \mathbb{m} \textit{ contains } \mathfrak{c}'_l \wedge \mathfrak{c}'_l \notin \mathfrak{C}_{l,W}$$

We know that $\mathfrak{C}_{l,CR} \subseteq \mathfrak{C}_{l,CR_1}$, because the left condition elements in CR are taken from the left condition elements in CR_1 per definition and thus also contained CR_1. Since $\mathbb{m}$ *contains* $\mathfrak{c}'_l$, there must be a consistency relation pair $\langle \mathfrak{c}'_l, \mathfrak{c}'_{r,1}\rangle \in W_1$, which witnesses consistency of $\mathfrak{c}'_l$ according to CR_1. There must be at least one consistency relation pair $\langle \mathfrak{c}'_{l,2}, \mathfrak{c}'_{r,2}\rangle \in CR_2$ with $\mathfrak{c}'_{l,2} \subseteq \mathfrak{c}'_{r,1}$, because otherwise $\mathfrak{c}'_l$ would per definition not occur in the left condition of CR. For all such tuples $\langle \mathfrak{c}'_{l,2}, \mathfrak{c}'_{r,2}\rangle$, we know that $\mathbb{m}$ *contains* $\mathfrak{c}'_{l,2}$, because $\mathbb{m}$ *contains* $\mathfrak{c}'_{r,1}$ due to its containment in W_1 and due to $\mathfrak{c}'_{l,2} \subseteq \mathfrak{c}'_{r,1}$. In consequence, consistency to CR_2 requires that for one of those $\mathfrak{c}'_{r,2}$ it holds that $\mathbb{m}$ *contains* $\mathfrak{c}'_{r,2}$ and that there is $\langle \mathfrak{c}'_{l,2}, \mathfrak{c}'_{r,2}\rangle \in W_2$ that witnesses this consistency. Summarizing, due to $\langle \mathfrak{c}'_l, \mathfrak{c}'_{r,1}\rangle \in W_1$ and $\langle \mathfrak{c}'_{l,2}, \mathfrak{c}'_{r,2}\rangle \in W_2$ with $\mathfrak{c}'_{l,2} \subseteq \mathfrak{c}'_{r,1}$ and due to the definition of W as the concatenation of W_1 and W_2, we know that $\langle \mathfrak{c}'_l, \mathfrak{c}'_{r,2}\rangle \in W$, which breaks our assumption. So we have shown that:

$$\forall \mathfrak{c}'_l \in \mathfrak{C}_{l,CR} \mid \mathbb{m} \textit{ contains } \mathfrak{c}'_l: \mathfrak{c}'_l \in \mathfrak{C}_{l,W}$$

Summarizing, we have shown that W fulfills all requirements to a witness structure according to Definition 3 for $\mathbb{m}$ being consistent to CR, so we know that $\mathbb{m}$ *consistent to* CR. □

We can use this notion of concatenation to define a transitive closure for sets of consistency relations, which contains all relations in that set complemented by all possible concatenations of them, i.e., *implicit relations* of that set. Having shown that our definition of consistency relations concatenation is well-defined in the sense that it does not introduce further restrictions for consistency, we are also able to show that the transitive closure does not restrict consistency in comparison to the set of consistency relation itself.

Definition 6 (Transitive Closure of Consistency Relations) Let $\mathbb{CR}$ be a set of consistency relations. We define its transitive closure $\mathbb{CR}^+$ as:

$$\mathbb{CR}^+ = \{CR \mid \exists CR_1,\ldots,CR_k \in \mathbb{CR} : CR = CR_1 \otimes \ldots \otimes CR_k\}$$

The transitive closure of a set of consistency relations $\mathbb{CR}$ contains all consistency relations of $\mathbb{CR}$ and all concatenations of relations in $\mathbb{CR}$. That means, the transitive closure contains consistency relations that relate all elements that are directly or indirectly related due to $\mathbb{CR}$.

The transitive closure of a consistency relation set does not further restrict consistency in comparison to the original set by construction of concatenation, i.e., if a model set is consistent to a set of consistency relations, it is also consistent to their transitive closure. We show that in the following by first extending the argument of Lemma 1, which shows that concatenation does not further restrict consistency, to the transitive closure, which is only a set of concatenations of consistency relations.

Lemma 2 *Let $\mathbb{CR}$ be a set of consistency relations for a set of metamodels $\mathbb{M}$. Then:*

$$\forall CR \in \mathbb{CR}^+ \setminus \mathbb{CR} : \exists CR_1,\ldots,CR_k \in \mathbb{CR} : \forall \mathbb{m} \in I_{\mathbb{M}} : \mathbb{m} \textit{ consistent to } \{CR_1,\ldots CR_k\} \Rightarrow \mathbb{m} \textit{ consistent to } CR$$

Proof Per definition, any $CR \in \mathbb{CR}^+$ is a concatenation of consistency relations in $\mathbb{CR}$, i.e.

$$\forall CR \in \mathbb{CR}^+ : \exists CR_1,\ldots,CR_k \in \mathbb{CR} : CR = CR_1 \otimes \ldots \otimes CR_k$$

We already know for any two consistency relations CR_1, CR_2 and all model sets $\mathbb{m}$ that if $\mathbb{m}$ *consistent to* $\{CR_1, CR_2\}$, then $\mathbb{m}$ *consistent to* $CR_1 \otimes CR_2$ due to Lemma 1. Inductively applying that argument to $CR_1,\ldots,CR_k$ shows that for all models $\mathbb{m}$ with $\mathbb{m}$ *consistent to* $\{CR_1,\ldots,CR_k\}$ we know that $\mathbb{m}$ *consistent to* CR. □

As a direct result of the previous lemma, we can now show that the transitive closure of a consistency relation set considers the same sets of models consistent as the consistency relation set itself.

Lemma 3 *Let $\mathbb{CR}$ be a set of consistency relations for a set of metamodels $\mathbb{M}$. Then for all sets of models $\mathbb{m} \in I_{\mathbb{M}}$ it is true that:*

$$\mathbb{m} \textit{ consistent to } \mathbb{CR} \Leftrightarrow \mathbb{m} \textit{ consistent to } \mathbb{CR}^+$$

Proof Adding a consistency relation to a set of consistency relations can never lead to a relaxation of consistency, i.e., models becoming consistent that were not considered consistent before. This is a direct consequence of Definition 3 for consistency, which requires models be consistent to all consistency relations in a set to be considered consistent, thus only restricting the set of consistent model sets by adding further consistency relations. In consequence, it holds that:

$$\mathbb{m} \textit{ consistent to } \mathbb{CR}^+ \Rightarrow \mathbb{m} \textit{ consistent to } \mathbb{CR}$$

Due to Lemma 3, we know that a set of models that is consistent to $\mathbb{CR}$ is always consistent to all transitive relations in $\mathbb{CR}^+$ as well. Thus, we know that:

$$\mathbb{m} \textit{ consistent to } \mathbb{CR} \Rightarrow \mathbb{m} \textit{ consistent to } \mathbb{CR}^+$$

In consequence, models are considered consistent equally for $\mathbb{CR}$ and its transitive closure $\mathbb{CR}^+$. □

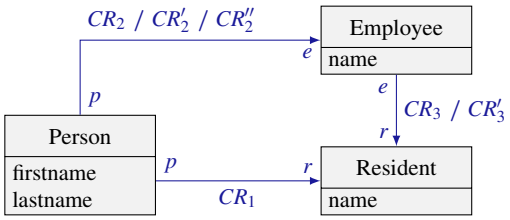
3.3 A Formal Notion of Compatibility

Based on the fine-grained notion of consistency in terms of consistency relations, we can know precisely formulate our initially informal notion of *compatibility* of consistency relations. We stated that we consider consistency relation incompatible if they are somehow contradictory, like the relation between names in our initial example in Figure 1. In that example, for residents with non-lowercase names no consistent set of models could be derived. To capture that in a definition, we consider relations compatible if for all condition elements in the consistency relations, i.e., for every tuple of objects for which consistency is somehow constrained by requiring further elements to exist in a set of models to consider it consistent, a consistent model containing those objects can be found. In consequence, a consistency relation is not allowed to prevent objects for which other relations specify consistency from existing in consistent models.

Definition 7 (Compatibility) Let $\mathbb{CR}$ be a set of consistency relations for a set of metamodels $\mathbb{M}$. We say that:

$$\mathbb{CR} \textit{ compatible } :\Leftrightarrow \forall CR \in \mathbb{CR} : \forall \mathbb{c} \in \mathbb{c}_{l,CR} : \exists \mathbb{m} \in I_{\mathbb{M}} : \mathbb{m} \textit{ contains } \mathbb{c} \wedge \mathbb{m} \textit{ consistent to } \mathbb{CR}$$

We call a set of consistency relation $\mathbb{CR}$ *incompatible* if it does not fulfill the definition of compatibility.



$$CR_1 = \{\langle p,r\rangle \mid r.name = p.firstname + "\text{␣}" + p.lastname\}$$

$$CR_2 = \{\langle p,e\rangle \mid e.name = p.firstname + "\text{␣}" + p.lastname\}$$
$$CR'_2 = \{\langle p,e\rangle \mid e.name = p.firstname + ",\text{␣}" + p.lastname\}$$
$$CR''_2 = \{\langle p,e\rangle \mid e.name = p.lastname + "\text{␣}" + p.firstname\}$$

$$CR_3 = \{\langle r,e\rangle \mid r.name = e.name\}$$
$$CR'_3 = \{\langle r,e\rangle \mid r.name = e.name.toLower\}$$

Fig. 4: Three metamodels with different consistency relations. The sets $\{CR_1, CR_1^T, CR_2, CR_2^T, CR_3, CR_3^T\}$ and $\{CR_1, CR_1^T, CR_2'', CR_2''^T, CR_3, CR_3^T\}$ are compatible, whereas the sets $\{CR_1, CR_1^T, CR_2', CR_2'^T, CR_3, CR_3^T\}$ and $\{CR_1, CR_1^T, CR_2, CR_2^T, CR_3', CR_3'^T\}$ are not.

Definition 7 formalizes the notion of *non-contradictory* relations by requiring that a relation may not restrict that an object tuple, for which consistency is defined in any consistency relation, cannot occur in a model set anymore. We exemplify this notion of compatibility on an extract of the initial example with different consistency relations.

Example 3 Figure 4 shows an extract of the three metamodels from Figure 1 and several consistency relations, of which different combinations are compatible or incompatible according to the previous definition. We always consider the actual relations together with their transposed ones to have a symmetric set of consistency relations.

$\{CR_1, CR_1^T, CR_2, CR_2^T, CR_3\}$**:** These relations are obviously compatible, because they relate *firstname* respectively *lastname* and *name* in the same way. Thus, for each object with any name, and thus any condition element in all of the consistency relations, a consistent model set can be found by adding instances of the other classes with appropriate names.

$\{CR_1, CR_1^T, CR_2', CR_2'^T, CR_3, CR_3^T\}$**:** These relations are obviously not compatible, because for each person with $firstname = f$ and $lastname = l$, another person with $firstname = f + ","$ and $lastname = l$ has to exist due to CR_2' and the transitive relations requiring the addition of a comma. Thus, each person would require an infinite number of further persons to exist in a consistent set of models. However, models are assumed to be finite, so there is no such model set and the relations are incompatible.

$\{CR_1, CR_1^T, CR_2', CR_2'^T, CR_3, CR_3^T\}$**:** These relations are compatible, although one might not expect that. The relations define that for a resident with $firstname = f$ and $lastname = l$ another resident with $firstname = l$ and $lastname = f$ has to exist, so that the set of models is consistent. Although that behavior may not be intuitive, it does not violate the definition of compatibility, because for any object in the relations, a consistent model can be constructed. In general, such a behavior cannot be forbidden, because comparable behavior might be expected, such as that for a software component an implementation class as well a utility class with different names are created due to different relations, which leads to comparable behavior as in the example. Finally, such a relation would not prevent a consistency repair routine from finding a consistent set of models. So this can be seen as a semantic problem that requires further relation-specific knowledge, as it is necessary to know that a first name should never be mapped to a last name in our example.

$\{CR_1, CR_1^T, CR_2, CR_2^T, CR_3', CR_3'^T\}$**:** These consistency relations reflect the ones of our motivational example in Figure 1. According to the informal notion of incompatibility that we motivated in the introduction with that example, our formal definition of compatibility also considers these relations as incompatible, because it is not possible to create a resident with an uppercase name, such that the containing set of models is consistent. For a resident with $name = "A\text{␣}B"$, a person with $firstname = "A"$ and $lastname = "B"$ has to exist, which requires existence of an employee with $name = "A\text{␣}B"$. Now CR_3' requires a resident with $name = "a\text{␣}b"$ to exist, which in turn requires a resident with $firstname = "a"$ and $lastname = "b"$ and an employee with $name = "a\text{␣}b"$ to exist. In consequence, there are two employees, one with the uppercase and one with the lowercase name, for which a resident with the lowercase name has to exist according to the relation CR_3'. So there is no witness structure with a unique mapping between the elements that is required to fulfill Definition 3 for consistency.

To summarize, compatibility is supposed to ensure that consistency relations do not impose restrictions on other relations such that their condition elements, for which consistency is defined, can never occur in consistent models. The goal of ensuring compatibility of consistency relations is especially to prevent consistency repair routines of model transformation from non-termination, as may occur especially in the second scenario, where an infinitely large model would be required to fulfill the consistency relations.

Finally, analogously to the equivalence of a set of consistency relations $\mathbb{CR}$ and its transitive closure $\mathbb{CR}^+$ in regards to consistency of a set of models, we can show that a set of consistency relations and its transitive closure are always equal with regards to compatibility.

Lemma 4 *Let $\mathbb{CR}$ be a set of consistency relations for a set of metamodels $\mathbb{M}$. It holds that:*

$$\mathbb{CR} \textit{ compatible} \Leftrightarrow \mathbb{CR}^+ \textit{ compatible}$$

Proof The reverse direction of the equivalence is given by definition, since compatibility of a sub of consistency relations implies compatibility of any subset by definition. So we have to show the forward direction by considering the compatibility definition for all $CR \in \mathbb{CR}^+$. We partition $\mathbb{CR}^+$ into $\mathbb{CR}$ and $\mathbb{CR}^+ \setminus \mathbb{CR}$ and consider their consistency relations independently.

First, we consider $CR \in \mathbb{CR}^+ \setminus \mathbb{CR}$. According to Definition 6 for the transitive closure, each $CR \in \mathbb{CR}^+ \setminus \mathbb{CR}$ is a concatenation of consistency relations $CR_1, \ldots, CR_k \in \mathbb{CR}$. In consequence of that definition we know that $\mathbb{c}_{l,CR} \subseteq \mathbb{c}_{l,CR_1}$, so it is given that:

$$\begin{aligned} &\forall\, \mathfrak{c}_l \in \mathbb{c}_{l,CR} : \exists\, \mathfrak{c}'_l \in \mathbb{c}_{l,CR_1} : \forall\, \mathfrak{m} \in I_{\mathbb{M}} : \\ &\quad \mathfrak{m} \textit{ contains } \mathfrak{c}_l \Rightarrow \mathfrak{m} \textit{ contains } \mathfrak{c}'_l \end{aligned} \tag{1}$$

Since $\mathbb{CR}$ is compatible, we especially know from Definition 7 for compatibility that:

$$\begin{aligned} &\forall\, \mathfrak{c}'_l \in \mathbb{c}_{l,CR_1} : \exists\, \mathfrak{m} \in I_{\mathbb{M}} : \\ &\quad \mathfrak{m} \textit{ contains } \mathfrak{c}'_l \wedge \mathfrak{m} \textit{ consistent to } \mathbb{CR} \end{aligned} \tag{2}$$

Because of Equation 1 and Equation 2, we know that:

$$\begin{aligned} &\forall\, \mathfrak{c}_l \in \mathbb{c}_{l,CR} : \exists\, \mathfrak{m} \in I_{\mathbb{M}} : \\ &\quad \mathfrak{m} \textit{ contains } \mathfrak{c}_l \wedge \mathfrak{m} \textit{ consistent to } \mathbb{CR} \end{aligned} \tag{3}$$

Furthermore, Lemma 3 states that for all model sets $\mathfrak{m} \in I_{\mathbb{M}}$ it is true that:

$$\mathfrak{m} \textit{ consistent to } \mathbb{CR} \Leftrightarrow \mathfrak{m} \textit{ consistent to } \mathbb{CR}^+ \tag{4}$$

In consequence of equations 3 and 4, we know that:

$$\begin{aligned} &\forall CR \in \mathbb{CR}^+ \setminus \mathbb{CR} : \forall\, \mathfrak{c}' \in \mathbb{c}_{l,CR} : \exists\, \mathfrak{m} \in I_{\mathbb{M}} : \\ &\quad \mathfrak{m} \textit{ contains } \mathfrak{c}' \wedge \mathfrak{m} \textit{ consistent to } \mathbb{CR}^+ \end{aligned} \tag{5}$$

Second, we consider $CR \in \mathbb{CR}$. Due to the definition of compatibility of $\mathbb{CR}$ and Lemma 3 showing equality of consistency of $\mathfrak{m}$ regarding $\mathbb{CR}$ and $\mathbb{CR}^+$ it is true that:

$$\begin{aligned} &\forall CR \in \mathbb{CR} : \forall\, \mathfrak{c}' \in \mathbb{c}_{l,CR} : \exists\, \mathfrak{m} \in I_{\mathbb{M}} : \\ &\quad \mathfrak{m} \textit{ contains } \mathfrak{c}' \wedge \mathfrak{m} \textit{ consistent to } \mathbb{CR}^+ \end{aligned} \tag{6}$$

With Equation 5 and Equation 6, we have shown compatibility of $\mathbb{CR}^+$ if $\mathbb{CR}$ is compatible. □

4 A Formal Approach to Prove Compatibility

In this section, we use the definition of compatibility to derive a formal approach for proving compatibility of consistency relations. The approach bases on two ideas:

1. A set of consistency relations in which each pair of classes is only related across one concatenation of relations is inherently compatible, because there cannot be any contradictory relations. We precisely define this in a specific notion of *consistency relation trees*.
2. A consistency relation that is redundant in a set of relations, i.e., a relation that does not alter the notion of consistency for models regarding the other relations in that set, does not affect compatibility and can thus be removed from that set of relations.

Given a set of consistency relations, compatibility can be proven inductively if a consistency relation tree that is equivalent to the set of relations can be found by only removing redundant relations from that set. Finding such an equivalent consistency relation tree serves as a *witness* for compatibility of a set of relations. In the following, we formalize and prove this inductive approach to check compatibility of a set of consistency relations. This constitutes our contribution **C2**.

The sketched approach for witnessing compatibility is based on a definition of equivalence for sets of consistency relations. We consider two sets of consistency relations equivalent if they consider the same sets of models as consistent:

Definition 8 (Equivalence of Consistency Relations) Let $\mathbb{CR}_1, \mathbb{CR}_2$ be two sets of consistency relations defined for a set of metamodels $\mathbb{M}$. We say that:

$$\begin{aligned} &\mathbb{CR}_1 \textit{ equivalent to } \mathbb{CR}_2 :\Leftrightarrow \forall\, \mathfrak{m} \in I_{\mathbb{M}} : \\ &\quad \mathfrak{m} \textit{ consistent to } \mathbb{CR}_1 \Leftrightarrow \mathfrak{m} \textit{ consistent to } \mathbb{CR}_2 \end{aligned}$$

The goal of our approach is to find a set of consistency relations that is compatible and equivalent to a given consistency relation set. We will later use equivalence to introduce a specific notion of redundancy that is compatibility-preserving. In the following, we first consider structures of consistency relation sets that are inherently compatible and afterwards consider redundancy as a means to find an equivalent representation of a relation set that has such a structure.

4.1 Compatible Consistency Relation Set Structures

We first consider two essential properties of a consistency relation set that lead to its inherent compatibility:

1. Composability: We show that the union of independent, compatible sets of consistency relations is compatible.
2. Trees: We show that relations fulfilling a special notion of *consistency relation trees* are inherently compatible.

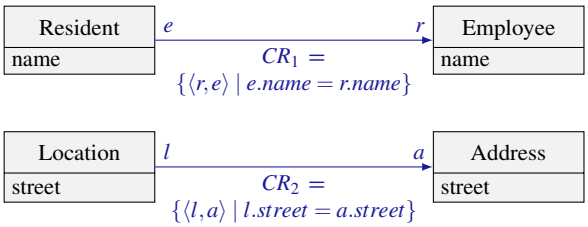


Fig. 5: Two independent (sets of) consistency relations

In consequence, we know that a consistency relation set that is composed of independent subsets of consistency relation trees is inherently compatible.

We consider consistency relation sets as independent if there are no transitive consistency relations induced by relations from both sets, i.e., for each object in a model consistency is only restricted by one of those sets.

Definition 9 (Independence of Consistency Relation Sets) Let $\mathbb{CR}_1$ and $\mathbb{CR}_2$ be two sets of consistency relations. We say that:

$$\begin{aligned}&\mathbb{CR}_1 \textit{ and } \mathbb{CR}_2 \textit{ are independent} :\Leftrightarrow\\ &\quad \forall CR \in \mathbb{CR}_1 : \forall CR' \in \mathbb{CR}_2 :\\ &\quad \forall CR_1,\ldots,CR_k \in \mathbb{CR}_1 \cup \mathbb{CR}_2 :\\ &\qquad CR \otimes CR_1 \otimes \ldots \otimes CR_k \otimes CR' = \emptyset\\ &\qquad \wedge CR' \otimes CR_1 \otimes \ldots \otimes CR_k \otimes CR = \emptyset\end{aligned}$$

We call $\mathbb{CR}$ *connected* if there is no partition of a consistency relation set $\mathbb{CR}$ into two subsets that are independent, i.e.

$$\begin{aligned}&\forall\, \mathbb{CR}_1, \mathbb{CR}_2 \subseteq \mathbb{CR} :\\ &\quad \mathbb{CR}_1 \cap \mathbb{CR}_2 = \emptyset \wedge \mathbb{CR}_1 \cup \mathbb{CR}_2 = \mathbb{CR}\\ &\qquad \Rightarrow \neg(\mathbb{CR}_1 \textit{ and } \mathbb{CR}_2 \textit{ are independent}),\end{aligned}$$

Example 4 Figure 5 depicts a simple example with two consistency relations CR_1 and CR_2, each relating instances of two disjoint classes with each other. Since there is no overlap in the objects that are related by the consistency relations, they are considered independent according to Definition 9.

An important property of independent sets of consistency relations is that computing their union is compatibility-preserving, i.e., the union of compatible, independent consistency relation sets is compatible as well:

Theorem 1 *Let $\mathbb{CR}_1$ and $\mathbb{CR}_2$ be two compatible sets of consistency relations. Then $\mathbb{CR}_1 \cup \mathbb{CR}_2$ is compatible.*

Proof Since $\mathbb{CR}_1$ is compatible, per definition there is a model set $\mathbb{m}$ for each condition element $\mathfrak{c}$ of the left condition of each consistency relation in $\mathbb{CR}_1$ that contains $\mathfrak{c}$ and that is consistent to $\mathbb{CR}_1$. Taking such an $\mathbb{m}$, we create a new $\mathbb{m}'$ by removing all elements from $\mathbb{m}$, which are contained in any condition elements in any consistency relation in $\mathbb{CR}_2$ and thus potentially require other elements to occur to be considered consistent to that consistency relation. In consequence, $\mathbb{m}'$ does not contain any condition elements from consistency relations in $\mathbb{CR}_2$ and is thus consistent to $\mathbb{CR}_2$ by definition. Additionally, $\mathbb{m}'$ is still consistent to $\mathbb{CR}_1$, because due to the independence of $\mathbb{CR}_1$ and $\mathbb{CR}_2$, there cannot be any consistency relations in $\mathbb{CR}_1$, which require the existence of the removed elements. In consequence, for each condition element $\mathfrak{c}$ of each consistency relation in $\mathbb{CR}_1$ there is a model set that contains $\mathfrak{c}$ and that is consistent to $\mathbb{CR}_1 \cup \mathbb{CR}_2$. The analogous argumentation applies to the consistency relations in $\mathbb{CR}_2$, which is why the definition of compatibility is fulfilled for all condition elements of all consistency relations in $\mathbb{CR}_1 \cup \mathbb{CR}_2$. □

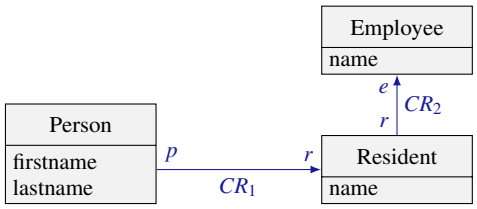
The constructive proof can also be reflected exemplarily in Figure 5: Take any set of models that, for example, contains a resident with an arbitrary name and is consistent to CR_1, i.e., that also contains an employee with the same name. If that set of models contains any addresses or locations, they can be removed without violating consistency to CR_1, because addresses and locations are independently related by CR_2.

Definition 10 (Consistency Relation Tree) Let $\mathbb{CR}$ be a symmetric, connected set of consistency relations. We say:

$$\begin{aligned}&\mathbb{CR} \textit{ is a consistency relation tree} :\Leftrightarrow\\ &\quad \forall CR = CR_1 \otimes \ldots \otimes CR_m \in \mathbb{CR}^+ :\\ &\quad \forall CR' = CR'_1 \otimes \ldots \otimes CR'_n \in \mathbb{CR}^+ \setminus CR :\\ &\qquad \forall s,t \mid s \neq t : CR_s \neq CR_t^T \wedge CR'_s \neq CR_t'^T\\ &\qquad \Rightarrow \mathfrak{C}_{l,CR} \cap \mathfrak{C}_{l,CR'} = \emptyset \vee \mathfrak{C}_{r,CR} \cap \mathfrak{C}_{r,CR'} = \emptyset\end{aligned}$$

The definition of a consistency relation tree requires that there are no sequences of consistency relations that put the same classes into relation, i.e. between all pairs of classes there is only one concatenation of consistency relations that puts them into relation. Since we assume a symmetric set of consistency relations, we exclude the symmetric relations from that argument, as otherwise there would always be two such concatenations by adding a consistency relation and its transposed relation to any other concatenation.

Example 5 Figure 6 depicts a rather simple consistency relation tree. Persons are related to residents and residents are related to employees, all having the same names respectively a concatenation of *firstname* and *lastname*, by the relations CR_1, CR_2, as well as their transposed relations CR_1^T, CR_2^T. There are no classes that are put into relation across different paths of consistency relations, thus the definition for a consistency relation tree is fulfilled. If an additional relation between persons and employees was specified, like in Figure 1, the tree definition would not be fulfilled.



$CR_1 = \{\langle p, r\rangle \mid r.name = p.firstname + "\_" + p.lastname\}$
$CR_2 = \{\langle r, e\rangle \mid r.name = e.name\}$

Fig. 6: A consistency relation tree $\{CR_1, CR_1^T, CR_2, CR_2^T\}$

The definition also covers the more complicated case in which multiple classes may be put into relation by consistency relations but there is only a subset of them that is put into relation by different consistency relations. We can now prove that a set of consistency relations that is a consistency relation tree is always compatible. We first present a lemma that shows that in a consistency relation tree you can always find an order of the relations such that the classes at the right side of a relation do not overlap with the classes at the left side of a relation that preceded in the order, i.e. there is no cycle in the relations between classes.

Lemma 5 *Let $\mathbb{CR} = \{CR_1, CR_1^T, \ldots, CR_k, CR_k^T\}$ be a symmetric, connected set of consistency relations. $\mathbb{CR}$ is a consistency relation tree if and only if for each CR there exists a sequence of consistency relations $\langle CR'_1, \ldots, CR'_k\rangle$ with $CR'_1 = CR$, containing for each i either CR_i or CR_i^T, i.e.,*

$$\forall i \in \{1, \ldots, k\} :$$
$$\left(CR_i \in \langle CR'_1, \ldots, CR'_k\rangle \wedge CR_i^T \notin \langle CR'_1, \ldots, CR'_k\rangle\right)$$
$$\vee \left(CR_i^T \in \langle CR'_1, \ldots, CR'_k\rangle \wedge CR_i \notin \langle CR'_1, \ldots, CR'_k\rangle\right)$$

such that:

$$\forall s \in \{1, \ldots, k-1\} : \forall t \in \{i+1, \ldots, k\} :$$
$$\mathfrak{C}_{r,CR'_s} \cap \mathfrak{C}_{r,CR'_t} = \emptyset \wedge \mathfrak{C}_{l,CR'_s} \cap \mathfrak{C}_{r,CR'_t} = \emptyset$$

Proof We start with the forward direction, i.e., given a consistency relation tree $\mathbb{CR}$ we show that there exists a sequence according to the requirements in Lemma 5 by constructing such a sequence $\langle CR'_1, \ldots, CR'_k\rangle$ for any $CR \in \mathbb{CR}$. Start with $CR'_1 = CR$ for any $CR \in \mathbb{CR}$. We now inductively add further relations to that sequence. Take any consistency relation $CR_s = CR_{s,1} \otimes \ldots \otimes CR_{s,m} \in \mathbb{CR}^+$ with $\mathfrak{C}_{l,CR_{s,1}} \subseteq \mathfrak{C}_{r,CR}$. Such a sequence must exist because of CR being connected. Now add all $CR_{s,1}, \ldots, CR_{s,m}$ to the sequence, which fulfills both requirements to that sequence in Lemma 5 by definition. The following addition of further consistency relations can be inductively applied. Take any other consistency relation $CR_t = CR_{t,1} \otimes \ldots \otimes CR_{t,n} \in \mathbb{CR}^+$ such that:

$$\exists CR' \in \{CR, CR_{s,2}, \ldots, CR_{s,m}\} : \mathfrak{C}_{l,CR_{t,1}} \subseteq \mathfrak{C}_{r,CR'}$$
$$\wedge CR_{t,1}, CR_{t,1}^T \notin \{CR, CR_{s,2}, \ldots, CR_{s,m}\}$$

In other words, take any concatenation in the transitive closure of $\mathbb{CR}$ that starts with a relation with a left class tuple that is contained in a right class tuple of a relation already added to the sequence. Again, such a sequence must exist because of $\mathbb{CR}$ being connected and, again, add all $CR_{t,1}, \ldots, CR_{t,n}$ to the sequence. Per construction, for each CR' in the sequence, there is a non-empty concatenation of relations within the sequence $CR \otimes \ldots \otimes CR'$, because relations were added in a way that such a concatenation always exists. Since all relations in the sequence are contained in $\mathbb{CR}$, such a concatenation was also contained in $\mathbb{CR}^+$. First, we show that the sequence still contains no duplicate elements (1.), i.e., that none of the $CR_{t,i}$ or $CR_{t,i}^T$ is already contained in the sequence $\langle CR, CR_{s,1}, \ldots, CR_{s,m}\rangle$. Second, we show that both further conditions for the sequence defined in Lemma 5 are still fulfilled for the sequence $\langle CR, CR_{s,1}, \ldots, CR_{s,m}, CR_{t,1}, \ldots, CR_{t,n}\rangle$ (2. ,3.).

1. Let us assume that the sequence $\langle CR, CR_{s,1}, \ldots, CR_{s,m}\rangle$ already contained one of the $CR_{t,i}$ or $CR_{t,i}^T$. If $CR_{t,i}$ is contained in the sequence, there is a concatenation $CR \otimes \ldots \otimes CR_{t,i}$ with relations in $\langle CR, CR_{s,1}, \ldots, CR_{s,m}\rangle$, as well as a concatenation $CR \otimes \ldots \otimes CR_{t,1} \otimes \ldots \otimes CR_{t,i}$. Since $CR_{t,1} \notin \{CR, CR_{s,2}, \ldots, CR_{s,m}\}$ by construction, these two concatenations relate the same class tuples, i.e., they contradict the definition of a consistency relation tree. If $CR_{t,i}^T$ was contained in the sequence $\langle CR, CR_{s,2} \otimes \ldots \otimes CR_{s,m}\rangle$, there is a concatenation $CR \otimes \ldots \otimes CR_w \otimes CR_{t,i}^T$ with relations in $\langle CR, CR_{s,1}, \ldots, CR_{s,m}\rangle$ and, like before, the concatenation $CR \otimes \ldots \otimes CR_{t,1}, \ldots, CR_{t,i}$. Due to $\mathfrak{C}_{r,CR_w} \cap \mathfrak{C}_{l,CR_{t,i}} \neq \emptyset$ and $CR_{t,1}^T \notin \{CR, CR_{s,2}, \ldots, CR_{s,m}\}$ by construction, the two concatenations $CR \otimes \ldots \otimes CR_w$ and $CR \otimes \ldots \otimes CR_{t,1} \otimes \ldots \otimes CR_{t,i}$ have an overlap in both their left and right class tuples, i.e., they contradict the definition of a consistency relation tree. In consequence, the sequence $\langle CR, CR_{s,1}, \ldots, CR_{s,m}\rangle$ cannot have contained any $CR_{t,i}$ or $CR_{t,i}^T$ before.
2. Let us assume there were any CR'_u and CR'_v in the sequence $\langle CR, CR_{s,1}, \ldots, CR_{s,m}, CR_{t,1}, \ldots, CR_{t,n}\rangle$ such that $\mathfrak{C}_{r,CR'_u} \cap \mathfrak{C}_{r,CR'_v} \neq \emptyset$. As discussed before, for each of these relations exists a concatenation of relations in the sequence $CR \otimes \ldots \otimes CR'_u$ and $CR \otimes \ldots \otimes CR'_v$, which is contained in $\mathbb{CR}^+$. This contradicts the definition of a consistency relation tree, so there cannot be two such relations with overlapping classes in the right class tuple.
3. Let us assume there were any CR'_u and CR'_v $(u < v)$ in the sequence $\langle CR, CR_{s,1}, \ldots, CR_{s,m}, CR_{t,1}, \ldots, CR_{t,n}\rangle$ such that $\mathfrak{C}_{l,CR'_u} \cap \mathfrak{C}_{r,CR'_v} \neq \emptyset$. Again per construction, there must be a non-empty concatenation $CR \otimes \ldots \otimes CR'_w \otimes CR'_u$ with $w < u$. Since $\mathfrak{C}_{l,CR'_u} \subseteq \mathfrak{C}_{r,CR'_w}$ per definition, it holds that $\mathfrak{C}_{r,CR'_w} \cap \mathfrak{C}_{r,CR'_v} \neq \emptyset$. In other words, the relation CR'_v introduces a cycle in the relations. We have already shown in (2.) that this contradicts the definition of a consistency relation tree.

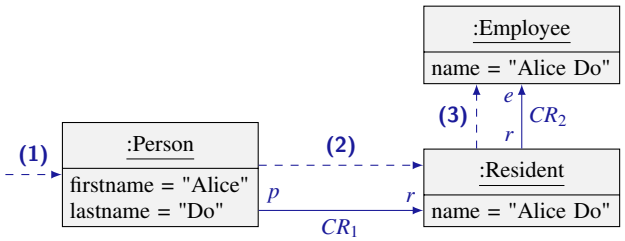
The previous strategy for adding relations to the sequence can be continued inductively by adding relations of the transitive closure of $\mathbb{CR}$ if their relations were not already added to the sequence. This process can be continued until finally all relations in $\mathbb{CR}$ are added to the sequence. Inductively applying the same arguments as before, the final sequence still fulfills all requirements for the sequence in Lemma 5.

We proceed with the reverse direction, i.e., given that a sequence according to the requirements in Lemma 5 exists for all $CR \in \mathbb{CR}$, we show that the set of consistency relations fulfills the definition of a consistency relation tree. Let us assume that the tree definition was not fulfilled, i.e., that there were two consistency relations $CR_s = CR_{s,1} \otimes \ldots \otimes CR_{s,m} \in \mathbb{CR}^+$ and $CR_t = CR_{t,1} \otimes \ldots \otimes CR_{t,n} \in \mathbb{CR}^+$ such that $\mathfrak{C}_{l,CR_s} \cap \mathfrak{C}_{l,CR_t} \neq \emptyset$ and $\mathfrak{C}_{r,CR_s} \cap \mathfrak{C}_{r,CR_t} \neq \emptyset$. Without loss of generality, we assume that $CR_{s,m} \neq CR_{t,n}$, because otherwise we could instead consider the sequence without those last relations and still fulfill the defined requirements. Any sequence according to Lemma 5 containing both $CR_{s,m}$ and $CR_{t,n}$ would contradict the assumption, because $\mathfrak{C}_{r,CR_{s,m}} \cap \mathfrak{C}_{r,CR_{t,n}} \neq \emptyset$ in contradiction to the assumptions regarding the sequence. Thus, the sequence has to contain either $CR^T_{s,m}$ or $CR^T_{t,n}$. Let us assume that the sequence contains $CR^T_{s,m}$. Then the sequence cannot contain $CR_{s,m-1}$, because $\mathfrak{C}_{r,CR^T_{s,m}} \cap \mathfrak{C}_{r,CR_{s,m-1}} \neq \emptyset$, which, again, would contradict the assumptions regarding the sequence. This argument can be inductively applied to all $CR_{s,i}$, such that the sequence has to contain all $CR^T_{s,i}$. Since the sequence contains $CR^T_{s,1}$, it must contain $CR_{t,1}$, because $\mathfrak{C}_{r,CR^T_{s,1}} \cap \mathfrak{C}_{r,CR^T_{t,1}} \neq \emptyset$. In consequence of $CR_{t,1}$ being contained in the sequence, all $CR_{t,i}$ have to be contained as well, due to the same reasons as before. So we have these conditions, which introduce a cycle in the overlaps of the class tuples of the relations within the sequence:

$$\mathfrak{C}_{l,CR^T_{s,i-1}} \cap \mathfrak{C}_{r,CR^T_{s,i}} \neq \emptyset \wedge \mathfrak{C}_{l,CR_{t,1}} \cap \mathfrak{C}_{r,CR^T_{s,1}} \neq \emptyset$$
$$\wedge\, \mathfrak{C}_{l,CR_{t,i}} \cap \mathfrak{C}_{r,CR_{t,i-1}} \neq \emptyset \wedge \mathfrak{C}_{l,CR^T_{s,m}} \cap \mathfrak{C}_{r,CR_{t,n}} \neq \emptyset$$

Because of that cycle in the overlap of class tuples, there is no order of these relations $CR''_1, \ldots, CR''_{m+n}$ such that for all of them it holds that $\mathfrak{C}_{l,CR''_u} \cap \mathfrak{C}_{r,CR''_v} \neq \emptyset$ $(u < v)$, which contradicts the assumptions regarding the sequence in Lemma 5. The analog argument holds when we assume that the sequence contains $CR^T_{t,n}$ instead of $CR^T_{s,m}$. In consequence, there cannot be two such concatenations CR_s and CR_t without breaking the assumptions for the sequence in Lemma 5. □

The previous lemma shows that the definition of consistency relation trees based on unique concatenations of the same class tuples is equivalent the possibility to find sequences of the relations that do not contain cycles in the related class tuples. The definition is supposed to be easier to check in practice. However, we can now show that a consistency relation tree is always compatible with a constructive proof that requires the equivalent definition from Lemma 5.



$CR_1 = \{\langle p, r\rangle \mid r.name = p.firstname + \text{"␣"} + p.lastname\}$

$CR_2 = \{\langle r, e\rangle \mid r.name = e.name\}$

Fig. 7: An example for constructing a model with the condition element of CR_1 containing the person named "Alice Do" for a consistency relation tree according to the consistency relations in Figure 6.

Theorem 2 *Let $\mathbb{CR}$ be a consistency relation tree, then $\mathbb{CR}$ is compatible.*

Proof We prove the statement by constructing a set of models for each condition element in the left condition of each consistency relation that contains the condition element and is consistent, i.e., that fulfills the compatibility definition. The basic idea is that because $\mathbb{CR}$ is a consistency relation tree, we can simply add necessary elements to get a model set that is consistent to all consistency relations, by following an order of relations according to Lemma 5. Thus, we explain an induction for constructing such a model set, which is also exemplified for a simple scenario in Figure 7, based on the relations in the consistency relation tree in Figure 6.

Base case: Take any $CR \in \mathbb{CR}$ and any of its left side condition elements $\mathfrak{c}_l = \langle o_{l,1}, \ldots, o_{l,m}\rangle \in \mathbb{c}_{l,CR}$. Select any $\mathfrak{c}_r = \langle o_{r,1}, \ldots, o_{r,n}\rangle \in \mathbb{c}_{r,CR}$, such that $\mathfrak{c}_l$ and $\mathfrak{c}_r$ constitute a consistency relation pair $\langle \mathfrak{c}_l, \mathfrak{c}_r\rangle \in CR$. Now construct the model set $\mathbb{m}$ that contains only $o_{l,1}, \ldots, o_{l,m}$ and $o_{r,1}, \ldots, o_{r,n}$. In consequence, we have a minimal model set $\mathbb{m}$, such that $\mathbb{m}$ *contains* $\mathfrak{c}_l$ and $\mathbb{m}$ *consistent to* CR. Additionally, $\mathbb{m}$ is consistent to CR^T due to symmetry of CR and CR^T: It is $\mathfrak{c}_r \in \mathbb{c}_{l,CR^T}$ and $\langle \mathfrak{c}_r, \mathfrak{c}_l\rangle \in CR^T$ and no other condition element of $\mathbb{c}_{l,CR^T}$ is contained in $\mathbb{m}$ by construction, thus $\mathbb{m}$ is consistent to CR^T. In consequence, we know that for all $CR \in \mathbb{CR}$, $\{CR, CR^T\}$ is compatible. Considering the example in *Figure* 7, for the selection of any person as a condition element in $\mathbb{c}_{l,CR_1}$ (1), we select a resident in $\mathbb{c}_{r,CR_1}$ with the same name (2), such that the elements are consistent to CR_1.

Induction assumption: According to Lemma 5, there is a sequence $\langle CR_1, \ldots, CR_k\rangle$ of the relations in $\mathbb{CR}$ with $CR_1 = CR$, such that:

$$\forall s \in \{1, \ldots, k-1\} : \forall t \in \{i+1, \ldots, k\} :$$
$$\mathfrak{C}_{r,CR'_s} \cap \mathfrak{C}_{r,CR'_t} = \emptyset \wedge \mathfrak{C}_{l,CR'_s} \cap \mathfrak{C}_{r,CR'_t} = \emptyset$$

Considering the example in Figure 7, such a sequence would be $\langle CR_1, CR_2 \rangle$, because the elements in the right condition of CR_2 are not represented in the left condition of CR_1. If, in general, we know that $\{CR_1, CR_1^T \ldots, CR_i, CR_i^T\}$ $(i < k)$ is compatible, for every $\mathfrak{c}_l \in \mathbb{C}_{l,CR}$, we can find a model set $\mathbb{m}$ that contains $\mathfrak{c}_l$ and is consistent to $\{CR_1, CR_1^T, \ldots, CR_i, CR_i^T\}$ by definition. We can especially create a minimal model according to our construction for the base case and the following inductive completion.

Induction step: Consider CR_{i+1}. There is at most one condition element $\mathfrak{c}_l \in \mathbb{c}_{l,CR_{i+1}}$ with $\mathbb{m}$ *contains* $\mathfrak{c}_l$. If there were at least two condition elements $\mathfrak{c}_l, \mathfrak{c}'_l \in \mathbb{c}_{l,CR_{i+1}}$, both contained in $\mathbb{m}$, then by construction there is a consistency relation CR_s $(s < i+1)$ with $\mathfrak{c}_l, \mathfrak{c}'_l \in \mathbb{c}_{r,CR_j}$. Let us assume there were two consistency relations CR_s, CR_t, each containing one of the condition elements in the right condition, then there would be non-empty concatenations $CR \otimes \ldots \otimes CR_s$ and $CR' \otimes \ldots \otimes CR_t$ with $\mathfrak{C}_{l,CR} \cap \mathfrak{C}_{l,CR'} \neq \emptyset$, because we started the construction with elements from the left condition of CR, so every element is contained because of a relation to those elements, and with $\mathfrak{C}_{r,CR_s} \cap \mathfrak{C}_{r,CR_t} \neq \emptyset$, because both condition elements $\mathfrak{c}_l$ and $\mathfrak{c}'_l$ instantiate the same classes, as they are both contained in $\mathbb{c}_{l,CR_{i+1}}$. This would violate Definition 10 for a consistency relation tree, thus there is only one such consistency relation CR_s. Consequently, there must be two condition elements $\mathfrak{c}_{ll}, \mathfrak{c}'_{ll} \in \mathbb{c}_{l,CR_s}$ with $\langle \mathfrak{c}_{ll}, \mathfrak{c}_l \rangle, \langle \mathfrak{c}'_{ll}, \mathfrak{c}'_l \rangle \in CR_s$, because per construction $\mathbb{m}$ was consistent to CR_s and so there must be a witness structure with a unique mapping between condition elements contained in $\mathbb{m}$. The above argument can be applied inductively until we finally find that there must be two condition elements $\mathfrak{c}_{lll}, \mathfrak{c}'_{lll} \in \mathbb{c}_{l,CR}$, which are contained in $\mathbb{m}$. This is not true by construction, as we started with only one element from $\mathbb{c}_{l,CR}$, so there is only one such condition element $\mathfrak{c}_l \in \mathbb{c}_{l,CR_{i+1}}$ with $\mathbb{m}$ *contains* $\mathfrak{c}_l$.

For this condition element $\mathfrak{c}_l \in \mathbb{c}_{l,CR_{i+1}}$, select an arbitrary $\mathfrak{c}_r = \langle o_1, \ldots, o_s \rangle \in \mathbb{c}_{r,CR_{i+1}}$, such that $\langle \mathfrak{c}_l, \mathfrak{c}_r \rangle \in CR_{i+1}$. Now create a model set $\mathbb{m}'$ by adding the objects $o_1, \ldots, o_s$ to $\mathbb{m}$. Since $\mathfrak{c}_l$ is the only of the left condition elements of CR_{i+1} that $\mathbb{m}$ contains, model set $\mathbb{m}'$ is consistent to CR_{i+1} per construction. $\mathbb{m}'$ is also consistent to CR_{i+1}^T, because due to the symmetry of CR_{i+1} and CR_{i+1}^T, it is $\mathfrak{c}_r \in \mathbb{c}_{l,CR_{i+1}^T}$ and due to $\langle \mathfrak{c}_r, \mathfrak{c}_l \rangle \in CR_{i+1}^T$, a consistent corresponding element exists in $\mathbb{m}'$. Furthermore, there cannot be any other $\mathfrak{c}' \in \mathbb{c}_{l,CR_{i+1}^T}$ with $\mathbb{m}'$ *contains* $\mathfrak{c}'$, because otherwise there would have been another consistency relation CR' that required the creation of $\mathfrak{c}'$, which means that there are two concatenations of consistency relations $CR \otimes \ldots \otimes CR'$ and $CR \otimes \ldots \otimes CR_{i+1}$ that both relate instances of the same classes, which contradicts Definition 10 for a consistency relation tree.

Additionally, due to Lemma 5, for all CR_s $(s < i+1)$, we know that $\mathfrak{C}_{l,CR_s} \cap \mathfrak{C}_{r,CR_{i+1}} = \emptyset$. Since the newly added elements $\mathfrak{c}_r$ are part of $\mathbb{c}_{r,CR_{i+1}}$, these elements cannot match the left conditions of any of the consistency relations CR_s $(s < i+1)$. So $\mathbb{m}'$ is still consistent to all CR_s $(s < i+1)$. Finally, due to Lemma 5, for all CR_s $(s < i+1)$, we know that $\mathfrak{C}_{r,CR_s} \cap \mathfrak{C}_{r,CR_{i+1}} = \emptyset$. Again, since the newly added elements $\mathfrak{c}_r$ are part of $\mathbb{c}_{r,CR_{i+1}}$, these elements cannot match the left conditions of any of the consistency relations CR_s^T $(s < i+1)$. So $\mathbb{m}'$ is still consistent to all CR_s^T $(s < i+1)$. In consequence, we know that $\mathbb{m}'$ *consistent to* $\{CR_1, CR_1^T \ldots, CR_{i+1}, CR_{i+1}^T\}$.

Considering the example in Figure 7, we would select CR_2 and add for the resident, which is in the left condition elements of CR_2, an appropriate employee to make the model set consistent to CR_2 (3).

Conclusion Taking the base case for CR and the induction step for CR_{i+1}, we have inductively shown that

$$\mathbb{m}' \textit{ consistent to } \{CR_1, CR_1^T \ldots, CR_k, CR_k^T\} = \mathbb{CR}$$

Since the construction is valid for each condition element in every consistency relation in $\mathbb{CR}$, we know that a consistency relation tree $\mathbb{CR}$ is compatible. □

Summarizing, Theorem 1 and Theorem 2 have shown that consistency relation sets fulfilling a special notion of trees are compatible and that combining compatible independent sets of relations is compatibility-preserving. In consequence, having a consistency relation set that consists of independent subsets that are consistency relation trees, this set of relations is inherently compatible. An approach that evaluates whether a given set of consistency relations fulfills Definition 9 and Definition 10 for independence and trees can be used to prove compatibility of those relations.

However, consistency relations fulfill such a structure only in specific cases. In general, like in our motivational example in Figure 1, there may be different consistency relations putting the same elements into relation, such that the definition for consistency relation trees is not fulfilled. In the following, we discuss how to find a consistency relation tree that is equivalent to a given set of consistency relations, such that this equivalence witnesses compatibility.

4.2 Redundancy as Witness for Compatibility

We have introduced specific structures of consistency relations that are inherently compatible. If a given set of consistency relations does not represent one of those structures, especially because there are multiple consistency relations putting the same classes into relation, it is unclear whether such a set is compatible.

In the following, we present an approach to reduce a set of consistency relations to a structure of independent consistency relation trees. The essential idea is to find relations within the set, which do not change compatibility of the consistency relation set whether or not they are contained

in it. An approach that finds such relations and—virtually—removes them from the set until the remaining relations form a set of independent consistency relation trees, proves compatibility of the given set of relations. We first define the term of a *compatibility-preserving* relation.

Definition 11 Let $\mathbb{CR}$ be a compatible set of consistency relations and let CR be a consistency relation. We say that:

$$CR \text{ compatibility-preserving to } \mathbb{CR} :\Leftrightarrow \\ \mathbb{CR} \cup \{CR\} \text{ compatible}$$

To be able to find such a compatibility-preserving relation, we introduce the notion of *redundant* relations and prove the property of being compatibility preserving. Informally speaking, a relation is redundant if it is expressed transitively across others, i.e., if it does not restrict or relax consistency compared to a combination of other relations. We precisely specify a notion of redundancy in the following.

Definition 12 (Redundant Consistency Relation) Let $\mathbb{CR}$ be a set of consistency relations for a set of metamodels $\mathbb{M}$. For a consistency relation $CR \in \mathbb{CR}$, we say that:

$$CR \text{ redundant in } \mathbb{CR} :\Leftrightarrow \\ \exists CR' \in (\mathbb{CR} \setminus \{CR\})^+ : \forall \mathrm{m} \in I_{\mathbb{M}} : \\ \mathrm{m} \text{ consistent to } CR' \Rightarrow \mathrm{m} \text{ consistent to } CR$$

The definition of redundancy of a consistency relation CR ensures that there is another consistency relation, possibly transitively expressed across others, such that if a model is consistent to that other relation, it is also consistent to CR. This means that there are no model sets that are considered inconsistent to CR, but not to another relation, thus CR does not restrict consistency. Actually, the definition of redundancy implies that the set of consistency relations with and without the redundant one are equivalent according to Definition 8, thus both consider the same model sets as consistent.

Lemma 6 *Let $CR \in \mathbb{CR}$ be a redundant consistency relation in a relation set $\mathbb{CR}$. Then $\mathbb{CR}$ is equivalent to $\mathbb{CR} \setminus \{CR\}$.*

Proof Like discussed in Lemma 3, adding a consistency relation to a set of consistency relations can never lead to a relaxation of consistency, i.e., models becoming consistent that were not considered consistent before. This is a direct consequence of Definition 3 for consistency, which requires models be consistent to all consistency relations in a set to be considered consistent, thus restricting the set of consistent model sets by adding further consistency relations. In consequence, it holds that:

$$\mathrm{m} \text{ consistent to } \mathbb{CR} \Rightarrow \mathrm{m} \text{ consistent to } \mathbb{CR} \setminus \{CR\}$$

Additionally, a direct consequence of Definition 12 for redundancy is that a redundant consistency relation does not restrict consistency, as it considers all models to be consistent that are also considered consistent to another consistency relation in the transitive closure of the consistency relation set. Thus, all models that are considered consistent to the transitive closure of $\mathbb{CR} \setminus \{CR\}$ are also consistent to CR and thus to all relations in $\mathbb{CR}$:

$$\mathrm{m} \text{ consistent to } (\mathbb{CR} \setminus \{CR\})^+ \Rightarrow \mathrm{m} \text{ consistent to } \mathbb{CR}$$

According to Lemma 3, each set of models that is consistent to a consistency relation set is also consistent to its transitive closure an vice versa. In consequence, the previous implication is also true for $\mathbb{CR} \setminus \{CR\}$ rather than $(\mathbb{CR} \setminus \{CR\})^+$. Summarizing, $\mathbb{CR}$ and $\mathbb{CR} \setminus \{CR\}$ are equivalent. □

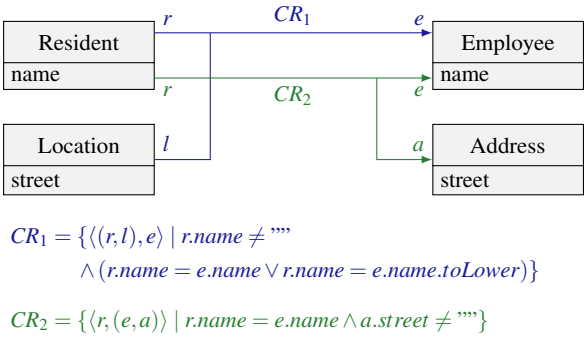
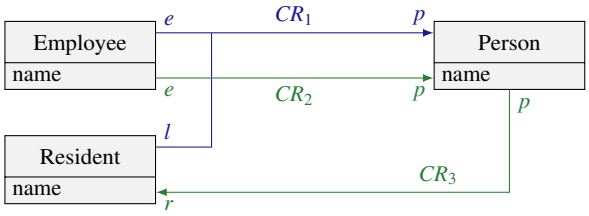


Fig. 8: Redundant consistency relation CR_1 in $\{CR_1, CR_2\}$

In general, to consider a consistency relation redundant in $\mathbb{CR}$, it has to define equal or weaker requirements for consistency than one of the other relations in $\mathbb{CR}$. Informally speaking, such weaker requirements mean that the redundant relation must have weaker conditions, i.e., it must require consistency for less objects and consider the same or more objects consistent to each of the left condition elements.

Example 6 Such weaker consistency requirements are exemplified in Figure 8, which shows a consistency relation CR_1 that is redundant in $\{CR_1, CR_2\}$. A redundant consistency relation, such as CR_1, must have weaker requirements in the left condition, such that it requires consistent elements to exist in less cases. This means that it may have a larger set of classes that are matched and that there may be less condition elements for which consistency is required. In case of CR_1, the left condition contains both a resident and a location, whereas the left condition of CR_2 only contains residents. Thus CR_1 requires consistent elements, i.e., employees, only if a resident and a location exists, whereas CR_2 requires that already for an existing resident. Furthermore, the residents for which CR_1 defines any consistency requirements are a subset of those for which CR_2 defines consistency requirements, as CR_1 does not make any statements about residents having an empty name. Thus, the left condition elements of CR_1 are a subset of those of CR_2. In consequence, if CR_1



$CR_1 = \{\langle (e,r), p\rangle \mid e.name = r.name.toUpper \wedge e.name = p.name\}$

$CR_2 = \{\langle e, p\rangle \mid e.name = p.name\}$

$CR_3 = \{\langle p, r\rangle \mid r.name = p.name.toLower\}$

Fig. 9: A consistency relation CR_1 being redundant in $\{CR_1, CR_2, CR_3\}$, with $\{CR_2, CR_3\}$ being compatible and $\{CR_1, CR_2, CR_3\}$ being incompatible.

requires consistency for a resident and a location, CR_2 requires it anyway, because it already defines consistency for the contained resident.

Additionally, a redundant consistency relation, such as CR_1, must have weaker requirements for the elements at the right side, such that one of the consistent right condition elements is contained anyway because another relation already required them. This means that the relation may have a smaller set of classes, of whom instances are required to consider the models consistent, and there may be more condition elements of the right side that are considered consistent with condition elements of the left side to not restrict the elements considered consistent. In case of CR_1, it only requires an emploee to exist for a resident compared to CR_2, which also requires a non-empty address to exist. Additionally, CR_1 does not restrict the employees that are considered consistent to employees compared to CR_2, as it also considers employees with the same name as consistent, but additionally those having the name of the resident in lowercase.

Our goal is to have a compatibility-preserving notion of redundancy, i.e., adding a redundant relation to a compatible relation set should preserve compatibility. Unfortunately, our intuitive redundancy definition is not compatibility-preserving.

Proposition 1 *Let $\mathbb{CR}$ be a compatible set of consistency relations and let CR be a consistency relation that is redundant in $\mathbb{CR} \cup \{CR\}$. Then CR is not necessarily compatibility-preserving, i.e., $\mathbb{CR} \cup \{CR\}$ is not necessarily compatible.*

Proof We prove the proposition by providing a counterexample. Consider the example in Figure 9. CR_2 relates each employee to a person with the same name and CR_3 relates each person to a resident with the same name in lowercase. The consistency relation set $\{CR_2, CR_3\}$ is obviously compatible, because for each employee and each person, which constitute the left condition elements of the consistency relations, a consistent model set containing the person respectively employee can be created by adding the appropriate person or employee with the same name and a resident with the name in lowercase. Furthermore, CR_1 is redundant in $\{CR_1, CR_2, CR_3\}$ according to Definition 12, because if a model is consistent to CR_2 it is also consistent to CR_1, since CR_1 also requires persons with the same name as an employee to be contained in a model set but in less cases, precisely only those in which the model also contains a resident such that the employee name is the one of the resident in uppercase.

However, $\{CR_1, CR_2, CR_3\}$ is not compatible. Intuitively, this is due to the fact that CR_1 and CR_3 define an incompatible mapping between the names of residents and persons. This is also reflected by Definition 7 for compatibility. Take a model with an employee and a resident named A. This is a condition element in $\mathbb{c}_{l,CR_1}$. Consequentially, CR_1 requires a person A to exist. Furthermore CR_3 requires a resident with name a to exist. In consequence, there are two tuples of employees and residents, both with employee A and one with resident A respectively resident a each, for which a consistent person with name A is required by CR_1. However, CR_1 actually forbids to have two residents, one having the lowercase name of the other, because both are condition elements in CR_1 requiring an appropriate person to occur in a consistent model, but there is only one person that to which both can be mapped, namely the one with the uppercase name, so there is no witness structure with a unique mapping as required by Definition 3 for consistency. This example shows that adding a redundant consistency relation to a compatible set of consistency relations does not lead to a compatible consistency relation set. □

In consequence of Proposition 1, we need a stronger definition of redundancy that is compatibility-preserving. In the example in Figure 9 showing Proposition 1, we have seen that it is problematic if a redundant consistency relation considers more classes in its left condition than the relation it is redundant to. Therefore, we restrict the left class tuple.

Definition 13 (Left-equal Redundant Consistency Relation) Let $\mathbb{CR}$ be a set of consistency relations for a metamodel set $\mathbb{M}$. For a consistency relation $CR \in \mathbb{CR}$, we say:

$$\begin{aligned} &CR \text{ left-equal redundant in } \mathbb{CR} :\Leftrightarrow \\ &\quad \exists CR' \in (\mathbb{CR} \setminus \{CR\})^+ : \forall \mathbb{m} \in I_{\mathbb{M}} : \\ &\qquad \mathbb{m} \text{ consistent to } CR' \Rightarrow \mathbb{m} \text{ consistent to } CR \\ &\qquad \wedge\, \mathfrak{C}_{l,CR} = \mathfrak{C}_{l,CR'} \end{aligned}$$

The definition of left-equal redundancy is similar to the redundancy definition but restricts the notion of redundancy to cases in which the left condition of the redundant consistency relation CR considers the same classes than the other relation in the set of consistency relations that induces consistency of a model set to CR. As discussed before, redundancy in general allows that the left condition of a redundant consistency relation can consider a superset of those classes.

Lemma 7 *Let CR be a consistency relation that is left-equal redundant in a set of consistency relations* $\mathbb{CR}$*. Then CR is redundant in* $\mathbb{CR}$*.*

Proof Since the definition of left-equal redundancy is equal to the one for redundancy, apart from the additional restriction for the class tuples, redundancy of a left-equal redundant relation is a direct implication of the definition. □

Before showing that left-equal redundancy is compatibility-preserving, we introduce an auxiliary lemma that shows that if a model set contains any left condition element of a left-equal redundant relation, i.e., if that redundant relation requires the model set to contain corresponding elements for that object tuple to be consistent, there is also another relation that requires corresponding elements for that object tuple.

Lemma 8 *Let CR be a consistency relation that is left-equal redundant in a set of consistency relations* $\mathbb{CR}$ *for a set of metamodels* $\mathbb{M}$*. Then it holds that:*

$$\exists CR' \in (\mathbb{CR} \setminus \{CR\})^+ : \forall \mathfrak{c}_l \in \mathfrak{c}_{l,CR} : \exists \mathfrak{c}'_l \in \mathfrak{c}_{l,CR'} : \\ \forall \mathbb{m} \in I_{\mathbb{M}} : \mathbb{m} \textit{ contains } \mathfrak{c}'_l \Rightarrow \mathbb{m} \textit{ contains } \mathfrak{c}_l$$

Proof Due to left-equal redundancy of *CR* in $\mathbb{CR}$, we know per definition that:

$$\exists CR' \in (\mathbb{CR} \setminus \{CR\})^+ : \forall \mathbb{m} \in I_{\mathbb{M}} : \\ \mathbb{m} \textit{ consistent to } CR' \Rightarrow \mathbb{m} \textit{ consistent to } CR \\ \wedge \mathfrak{C}_{l,CR} = \mathfrak{C}_{l,CR'}$$

This implies that:

$$\exists CR' \in (\mathbb{CR} \setminus \{CR\})^+ : \forall \mathfrak{c}_l \in \mathfrak{c}_{l,CR} : \mathfrak{c}_l \in \mathfrak{c}_{l,CR'}$$

Because if there was a $\mathfrak{c}_l \in \mathfrak{c}_{l,CR}$ so that $\mathfrak{c}_l \notin \mathfrak{c}_{l,CR'}$, then the model set $\mathbb{m}$ only consisting of $\mathfrak{c}_l$ would be consistent to CR', because it does not require any other elements to exist for considering the model set consistent, whereas there is at least one $\langle \mathfrak{c}_l, \mathfrak{c}_r \rangle \in CR$, so that $\mathbb{m}$ needs to contain $\mathfrak{c}_r$ for considering $\mathbb{m}$ consistent to *CR*, which is not given by construction. This shows that $\mathfrak{c}_{l,CR'}$ contains all elements in $\mathfrak{c}_{l,CR}$, so there is always at least one element from $\mathfrak{c}_{l,CR'}$ that a model set $\mathbb{m}$ contains if it contains an element from $\mathfrak{c}_{l,CR}$, which proves the statement in the lemma. □

Theorem 3 *Let* $\mathbb{CR}$ *be a compatible set of consistency relations for a set of metamodels* $\mathbb{M}$ *and let CR be a consistency relation that is left-equal redundant in* $\mathbb{CR} \cup \{CR\}$*. Then* $\mathbb{CR} \cup \{CR\}$ *is compatible.*

Proof Due to left-equal redundancy of *CR* in $\mathbb{CR} \cup \{CR\}$, which also implies general redundancy according to Definition 12, $\mathbb{CR}$ and $\mathbb{CR} \cup \{CR\}$ are equivalent, according to Lemma 6. Due to that equivalence, we know that for any model set $\mathbb{m} \in I_{\mathbb{M}}$:

$$\mathbb{m} \textit{ consistent to } \mathbb{CR} \Leftrightarrow \mathbb{m} \textit{ consistent to } \mathbb{CR} \cup \{CR\} \quad (1)$$

It follows from Definition 7 for compatibility and Equation 1:

$$\forall CR' \in \mathbb{CR} : \forall \mathfrak{c}_l \in \mathfrak{c}_{l,CR'} : \exists \mathbb{m} \in I_{\mathbb{M}} : \\ \mathbb{m} \textit{ contains } \mathfrak{c}_l \wedge \mathbb{m} \textit{ contains } \mathbb{CR} \cup \{CR\} \quad (2)$$

This already shows that for $\mathbb{CR}$ the compatibility definition is fulfilled, so we need to prove that the compatibility definition is fulfilled for *CR* as well. Due to compatibility of $\mathbb{CR}$ and Lemma 4 showing equality of compatibility for a consistency relation set and its transitive closure, we know that:

$$\forall CR' \in \mathbb{CR}^+ : \forall \mathfrak{c}_l \in \mathfrak{c}_{l,CR'} : \exists \mathbb{m} \in I_{\mathbb{M}} : \\ \mathbb{m} \textit{ contains } \mathfrak{c}_l \wedge \mathbb{m} \textit{ consistent to } \mathbb{CR}^+ \quad (3)$$

Due to left-equal redundancy of *CR* in $\mathbb{CR} \cup \{CR\}$, we have shown in Lemma 8 that the following is true:

$$\exists CR' \in \mathbb{CR}^+ : \forall \mathfrak{c}_l \in \mathfrak{c}_{l,CR} : \exists \mathfrak{c}'_l \in \mathfrak{c}_{l,CR'} : \forall \mathbb{m} \in I_{\mathbb{M}} : \\ \mathbb{m} \textit{ contains } \mathfrak{c}'_l \Rightarrow \mathbb{m} \textit{ contains } \mathfrak{c}_l \quad (4)$$

The combination of Equation 3 and Equation 4 gives:

$$\exists CR' \in \mathbb{CR}^+ : \forall \mathfrak{c}_l \in \mathfrak{c}_{l,CR} : \exists \mathfrak{c}'_l \in \mathfrak{c}_{l,CR'} : \\ (\forall \mathbb{m} \in I_{\mathbb{M}} : \mathbb{m} \textit{ contains } \mathfrak{c}'_l \Rightarrow \mathbb{m} \textit{ contains } \mathfrak{c}_l) \\ \wedge (\exists \mathbb{m} \in I_{\mathbb{M}} : \mathbb{m} \textit{ contains } \mathfrak{c}'_l \wedge \mathbb{m} \textit{ consistent to } \mathbb{CR}^+)$$

A simplification by combining the two last lines of that statement leads to:

$$\forall \mathfrak{c}_l \in \mathfrak{c}_{l,CR} : \exists \mathbb{m} \in I_{\mathbb{M}} : \\ \mathbb{m} \textit{ contains } \mathfrak{c}_l \wedge \mathbb{m} \textit{ consistent to } \mathbb{CR}^+$$

Due to Equation 1 and Lemma 3, which shows equality of consistency for a consistency relation set and its transitive closure, this is equivalent to:

$$\forall \mathfrak{c}_l \in \mathfrak{c}_{l,CR} : \exists \mathbb{m} \in I_{\mathbb{M}} : \\ \mathbb{m} \textit{ contains } \mathfrak{c}_l \wedge \mathbb{m} \textit{ consistent to } \mathbb{CR} \cup \{CR\} \quad (5)$$

The combination of Equation 2 and Equation 5 shows that $\mathbb{CR} \cup \{CR\}$ fulfills Definition 7 for compatibility. □

Corollary 1 *Let* $\mathbb{CR}$ *be a compatible set of consistency relations and let* $CR_1, \ldots, CR_k$ *be consistency relations with:*

$$\forall i \in \{1, \ldots, k\} : \\ CR_i \textit{ left-equal redundant in } \mathbb{CR} \cup \{CR_1, \ldots, CR_i\}$$

Then $\mathbb{CR} \cup \{CR_1, \ldots, CR_k\}$ *is compatible.*

Proof This is an inductive implication of Theorem 3, because $\mathbb{CR}$ is compatible and sequentially adding CR_i to $\mathbb{CR} \cup \{CR_1, \ldots, CR_{i-1}\}$ ensures that $\mathbb{CR} \cup \{CR_1, \ldots, CR_i\}$ is compatible, because $\mathbb{CR} \cup \{CR_1, \ldots, CR_{i-1}\}$ was compatible as well. □

With Corollary 1, we have shown that if we have a set of consistency relations $\mathbb{CR}$ and are able to find a sequence of redundant consistency relations $CR_1, \ldots CR_k$ according to Corollary 1 such that we know that $\mathbb{CR} \setminus \{CR_1, \ldots CR_k\}$ is compatible, then it is proven that $\mathbb{CR}$ is compatible.

4.3 Summary

In the previous sections, we have proven the following three central insights:

1. Compatibility is composable: If independent sets of consistency relations are compatible, then their union is compatible as well (Theorem 1).
2. Consistency relation trees are compatible: If there are no two concatenations of consistency relations in a consistency relation set that relate the same classes, then that set is compatible (Theorem 2).
3. Left-equal redundancy is compatibility-preserving: Adding a left-equal redundant consistency relation to a compatible set of consistency relations, that set unified with the redundant relation is still compatible (Corollary 1).

These insights enable us to define a formal approach for proving compatibility of a set of consistency relations. Given a set of relations for which compatibility shall be proven, we search for consistency relations in that set that are left-equal redundant to it. If iteratively removing such redundant relations—virtually—from the set leads to a set of independent consistency relation trees, it is proven that the initial set of consistency relations is compatible.

Such an approach to prove compatibility of consistency relations is *conservative*. If the approach finds redundant relations, such that a consistency relation set can be reduced to a set of independent consistency relations trees, the set is proven compatible, as we have shown by proof. If the approach is not able to find such relations, the set may still be compatible, but the approach is not able to prove that. Conceptually, this can be due to the fact that there may be compatibility-preserving relations that do not fulfill the definition of left-equal redundancy. Furthermore, an actual technique to identify left-equal redundant relations may not be able to find all of them automatically, as we will see later.

In the following, we discuss how such an approach can be operationalized. First, we discuss how actual transformations, at the example of QVT-R, can be represented in a graph-based structure, such that it conforms to our formal notion and allows to check whether the structure is an independent set of consistency relation trees. Second, we present an approach for finding consistency relations that are left-equal redundant, by the means of an SMT solver applied to the constraints defined in QVT-R relations.

5 Decomposing Transformations

The formal approach adopted in this article demonstrates that deriving a consistency relation tree from a set of consistency relations $\mathbb{CR}$ is an effective way to prove compatibility. It is especially a consequence of Theorem 2. Given that proving compatibility amounts to the construction of a consistency relation tree, this result lends itself well to an operationalization. To this end, we propose an algorithm that turns the proof of compatibility into an operational procedure. This constitutes our contribution **C3**. For the most part, this algorithm is based on results previously developed and described in detail in a master's thesis [51].

Constructing a consistency relation tree can be achieved by finding and virtually removing every redundant consistency relation in $\mathbb{CR}$. Such a result facilitates the development of software systems. First, developers build transformations independently, resulting in a transformation network. Then, they regularly run a procedure that assesses the compatibility of consistency relations specified in transformations by checking the existence of a consistency relation tree.

Removing redundant relations in a consistency relation set to generate a tree is called *decomposition*. Designing a decomposition procedure requires to represent consistency relations in actual model transformation languages and to provide a way to test the redundancy of a consistency relation. We first highlight a mapping between the consistency framework developed in this article and the QVT-R transformation language through the use of *predicates*. As a consequence, results achieved with consistency relations become also applicable with QVT-R. Then, we design a fully automated decomposition procedure that takes a *consistency specification*, i.e., a set of QVT-R transformations, as an input and virtually removes as many redundant consistency relations as possible. In the decomposition procedure, each consistency relation removal is a two-step process. First, a potentially redundant relation and an alternative concatenation of consistency relations are identified. Then, a redundancy test is performed: it answers whether it is possible or not to remove the candidate relation using the alternative concatenation. Uncoupling the search for candidates from the decision-making makes it possible to plug in different strategies to test redundancy. This section focuses on the first step, i.e., setting up a structure suited to the detection of possibly redundant relations and finding candidates for the redundancy test.

5.1 Consistency Relations in Transformation Languages

According to Definition 2, consistency relations are built by enumerating valid co-occurring condition elements. However, developers do not enumerate valid models when writing transformations. They rather describe patterns for models to be considered consistent and sometimes how consistency is restored after a model was modified. In relational transformation languages, developers define consistency as a set of criteria that models must fulfill. Criteria are expressed using metamodel elements (i.e., class properties), as objects are only distinguished by their contents. For example, an `Employee` object and a `Person` object are considered consistent if their `name` attributes are equal.

Criteria are equivalent to predicates, i.e., Boolean-valued filter functions: consistency relations are then defined as sets of pairs of condition elements for which the predicate evaluates to TRUE. Therefore, we move from an extensional to an intensional, programming-like definition of consistency relations, making it easier to link consistency relations with QVT-R transformations.

5.1.1 Properties, Property Values and Predicates

We first define concepts that allow the intensional construction of consistency relations. The main idea is to select some properties in each metamodel and to define a predicate that filters values of these properties.

Definition 14 (Property Set) A property set for a class C is a subset $\mathbb{P}_C$ of properties of C, i.e., $\mathbb{P}_C = \{P_{C,1}, \ldots, P_{C,n}\}$ such that $P_{C,i} \in C$.

A property set models a choice of properties that play a role in the definition of a predicate in order to distinguish consistent and non-consistent condition elements. Not all properties have to be used to describe consistency, particularly in incremental model transformations.

Definition 15 (Tuple of Property Sets) For a class tuple $\mathfrak{C}$, it is possible to build a tuple of property sets by defining a property set for every class, i.e., $\mathfrak{P}_\mathfrak{C} = \langle \mathbb{P}_{C_1}, \ldots, \mathbb{P}_{C_n} \rangle = \langle \{P_{C_1,1}, \ldots, P_{C_1,m}\}, \ldots, \{P_{C_n,1}, \ldots, P_{C_n,k}\} \rangle$.

Tuples generalize the use of property sets to class tuples, because conditions themselves are made up of class tuples.

Definition 16 (Property Value Set) A property value set $\mathbb{p}_C$ for a property set $\mathbb{P}_C$ is a set in which each property in $\mathbb{P}_C$ is instantiated, i.e., $\mathbb{p}_C = \{p_{C,1}, \ldots, p_{C,n}\}$ with $p_{C,i} \in I_{P_{C,i}}$. Similarly, a tuple of property value sets can be built from a tuple of property sets by instantiating each property set in it.

Just as a property set is a subset of properties of a class C, a property value set is a subset of property values of an object o that instantiates C. The property value set is a fragment of o that provides enough information to evaluate consistency.

Definition 17 (Predicate) A predicate for two class tuples $\mathfrak{C}_l$ and $\mathfrak{C}_l$ is a triple $\pi = (\mathfrak{P}_{\mathfrak{C}_l}, \mathfrak{P}_{\mathfrak{C}_r}, f_\pi)$ where $\mathfrak{P}_{\mathfrak{C}_l}$ (resp. $\mathfrak{P}_{\mathfrak{C}_r}$) is a tuple of property sets of $\mathfrak{C}_l$ (resp. $\mathfrak{C}_r$) and f_π is a Boolean-valued function that takes instances of $\mathfrak{P}_{\mathfrak{C}_l}$ and $\mathfrak{P}_{\mathfrak{C}_r}$ as an input, i.e., $f_\pi \colon I_{\mathfrak{P}_{\mathfrak{C}_l}} \times I_{\mathfrak{P}_{\mathfrak{C}_r}} \to \{\text{TRUE}, \text{FALSE}\}$.

For readability purposes, it is sometimes useful to group all the properties used by a predicate within the same set. As a consequence, the property collection P_π of a predicate $\pi = (\mathfrak{P}_{\mathfrak{C}_l}, \mathfrak{P}_{\mathfrak{C}_r}, f_\pi)$ is defined as:

$$\mathrm{P}_\pi = (\bigcup_j \mathfrak{P}_{\mathfrak{C}_l,j}) \cup (\bigcup_k \mathfrak{P}_{\mathfrak{C}_r,k})$$

The definition of a predicate involves the choice of some properties for each class that occurs in one of the two class tuples of a consistency relation *CR*. It also involves the definition of an appropriate function f_π that answers whether instances of these properties, i.e., property values, evaluate to TRUE or FALSE. In the former case, objects containing these property values match the predicate: the associated consistency relation pair is in *CR*. In the latter case, objects do not match the predicate and are not considered consistent, i.e., do not occur in *CR*. The expression of f_π is the choice of the developer who defines it according to consistency criteria.

Predicates model the way consistency relations are defined in model transformation languages. Objects can only be distinguished by their property values. Thus, the distinction between consistent and non-consistent pairs of condition elements is always based on some attribute or reference values.

5.1.2 Predicate-Based Consistency Relations

Definition 18 (Property Matching) A property value set $\mathbb{p}_C = \{p_{C,1}, \ldots, p_{C,n}\}$ matches an object o if and only if

$$o \in I_C \wedge \forall\, p_{C,i} : p_{C,i} \in o$$

Similarly, a tuple of property value sets $\mathfrak{p}_\mathfrak{C} = \langle \mathbb{p}_{C_1}, \ldots, \mathbb{p}_{C_n} \rangle$ matches a tuple of objects $\mathfrak{o} = \langle o_1, \ldots, o_k \rangle$ if and only if $|\mathfrak{p}_\mathfrak{C}| = |\mathfrak{o}|$ and $\forall\, i : \mathbb{p}_{C_i}$ matches o_i.

Definition 19 (Predicate-Based Consistency Relation) Let $\mathbb{c}_l$ and $\mathbb{c}_r$ be two conditions for two class tuples $\mathfrak{C}_{\mathbb{c}_l}$ and $\mathfrak{C}_{\mathbb{c}_r}$. Let Π be a set of predicates for $\mathfrak{C}_{\mathbb{c}_l}$ and $\mathfrak{C}_{\mathbb{c}_r}$. A Π-based consistency relation CR_Π is a subset of pairs of condition elements such that:

$$\begin{aligned} CR_\Pi = \{(c_l, c_r) \mid \forall (\mathfrak{P}_{\mathfrak{C}_{\mathbb{c}_l}}, \mathfrak{P}_{\mathfrak{C}_{\mathbb{c}_r}}, f_\pi) \in \Pi : \\ \exists \mathfrak{p}_{\mathfrak{C}_{\mathbb{c}_l}} \in \mathfrak{P}_{\mathfrak{C}_{\mathbb{c}_l}}, \mathfrak{p}_{\mathfrak{C}_{\mathbb{c}_r}} \in \mathfrak{P}_{\mathfrak{C}_{\mathbb{c}_r}} : \\ \mathfrak{p}_{\mathfrak{C}_{\mathbb{c}_l}} \text{ matches } c_l \\ \wedge\, \mathfrak{p}_{\mathfrak{C}_{\mathbb{c}_r}} \text{ matches } c_r \\ \wedge\, f_\pi(p_{\mathfrak{C}_{\mathbb{c}_l}}, p_{\mathfrak{C}_{\mathbb{c}_r}}) = \text{TRUE}\} \end{aligned}$$

The construction of a predicate-based consistency relation is of importance for the practicality of model transformation languages. The developer can produce a consistency specification by retaining some object properties and imposing conditions on values of these properties via a predicate function. Then, the construction of the consistency relation fully amounts to the evaluation of the predicate function.

Example 7 The following example demonstrates how to build a consistency relation CR_{PR} based on predicates between `Person` and `Resident` metamodels, according to the example in Figure 1. CR_{PR} ensures that the name of a `Resident` object is the concatenation of the first name and the last name of a `Person` object. It also ensures that both objects have

```
import M1 : 'path_m1.ecore';
import M2 : 'path_m2.ecore';

transformation T(M1, M2) {
    [top] relation R1 {
        [variable declarations]
        domain M a : A { πM }
        domain N b : B { πN }
        [when { PRECOND }] [where { INVARIANT }]
    }

    [top] relation R2 { ... }
}
```

Listing 1: Simplified structure of a QVT-R transformation

```
fstn: String; lstn: String;
inc: Integer;

domain pers p:Person {
    firstname=fstn, lastname=lstn,
    income=inc
};

domain emp e:Employee {
    name=fstn + '␣' + lstn,
    salary=inc
};
```

Listing 2: Two domains, each with one domain pattern

the same address. First, CR_{PR} involves one class in each metamodel, resulting in two class tuples: $\mathfrak{C}_P = \langle Person \rangle$ and $\mathfrak{C}_R = \langle Resident \rangle$. There are two conditions to achieve consistency, which are equal names and equal addresses, so CR_{PR} will be made up of two predicates. The first predicate needs the `firstname` and `lastname` attributes in `Person` and the `name` in `Resident`, so $\mathfrak{P}_{\mathfrak{C}_P,1} = \langle \{firstname, lastname\} \rangle$ and $\mathfrak{P}_{\mathfrak{C}_R,1} = \langle \{name\} \rangle$. Similarly, $\mathfrak{P}_{\mathfrak{C}_P,2} = \langle \{address\} \rangle$ and $\mathfrak{P}_{\mathfrak{C}_R,2} = \langle \{address\} \rangle$. The functions of the predicate, shortly denoting `name` as n, `firstname` as fn, `lastname` as ln, as well as `address` of `Person` as a_P and of `Resident` as a_R, look as follows:

$$f_{\pi,1}(\langle \{n\} \rangle, \langle \{fn, ln\} \rangle) = \begin{cases} \text{TRUE} & \text{if } n = fn + \text{`` } + ln \\ \text{FALSE} & \text{otherwise} \end{cases}$$

$$f_{\pi,2}(\langle \{a_P\} \rangle, \langle \{a_R\} \rangle) = \begin{cases} \text{TRUE} & \text{if } a_P = a_R \\ \text{FALSE} & \text{otherwise} \end{cases}$$

CR_{PR} is a Π-based consistency relation where Π is the set of predicates $\{(\mathfrak{P}_{\mathfrak{C}_P,1}, \mathfrak{P}_{\mathfrak{C}_R,1}, f_{\pi,1}), (\mathfrak{P}_{\mathfrak{C}_P,2}, \mathfrak{P}_{\mathfrak{C}_R,2}, f_{\pi,2})\}$.

5.1.3 Consistency Relations in QVT-R Transformations

There is a variety of model transformation languages [13]. Like programming languages, transformation languages can be divided into two main paradigms: *declarative* languages that focus on *what* transformations should perform and *imperative* languages that describe *how* transformations should be performed. The QVT standard [45] provides three transformation languages: Operational, Core and Relations.

The most relevant language for consistency specification is QVT Relations (QVT-R). It is a declarative and relational language that shares many concepts with the consistency framework developed in this article. It lends itself well to mathematical formalization [63]. As with consistency relations, QVT-R supports bidirectionality. Transformations written in QVT-R can be used with two execution modes. First, a *checkonly* mode to check that models fulfill consistency relations. Second, an *enforce* mode to repair consistency in a given direction if not all relations are fulfilled. The simplified structure of a QVT-R transformation is as follows and also depicted in Listing 1.

A QVT-R `transformation` can check or repair consistency of models it receives as parameters. Models are typed models, i.e., their structure conforms to a type defined by the metamodel. Each `transformation` is composed of `relations`, which define the rules for objects of both models to be consistent. Relations are only invoked if they are prefixed by the `top` keyword, if they belong to the precondition (`when`) of a relation to be invoked, or if they belong to the invariant (`where`) of a relation already invoked. The QVT-R mechanism for checking consistency is based on pattern matching. Shared information between objects of different models is represented by variables assigned to class properties. These variables contain values that must remain consistent from one object to another. Therefore, there must exist some assignment that matches all patterns at the same time for classes in a relation to be consistent.

More precisely, each QVT-R `relation` contains two `domains` that contain themselves *domain patterns*. In QVT terminology, a domain pattern is a variable instantiating a class. Values that this variable can take are constrained by conditions on its properties. These conditions, known as *property template items* (PTIs), are OCL constraints [47]. OCL operations provide the ability to describe more complex constraints than equalities between property values and variables. In Listing 2, each domain has one pattern. These patterns filter `Person` objects (with three PTIs) and `Employee` objects (with two PTIs), respectively. For two objects to be consistent, there must exist values of `fstn`, `lstn` and `inc` that match property values of these objects, thus ensuring the fact that the name of the employee equals the concatenation of the first name and the last name of the person and the fact that both instances have the same income. If objects are inconsistent, e.g., if the person and the employee have different incomes, then there is no such variable assignment.

QVT-R relations are defined intensionally. In *checkonly* mode, a `relation` does not check that metamodel instances

are consistent by looking for them in an existing set of pairs of consistent models. It rather evaluates the existence of a value that fulfills all property template items in domain patterns. These patterns can be regarded as predicates. Thus, it is relevant to correlate QVT-R relations and predicate-based consistency relations. One relation in QVT-R can be translated into one or more predicates. The main idea is to extract properties that are bound to the same QVT-R variables: having QVT-R variables in common means that values of these properties are interrelated. Properties are separated to build two tuples of property sets, one for each metamodel. Then, a predicate function is generated by extracting OCL constraints. The triplet that groups these objects together is a predicate. A formal construction of predicates from QVT-R will be presented in the subsequent section.

As a result, QVT-R is a language suitable for writing consistency relations according to our formalism. The decomposition procedure presented in this section treats a set of (binary) QVT-R `transformations` as a consistency relation set and checks its compatibility. The QVT-R transformations for the example in Figure 1 are depicted in Figure 10.

5.2 Decomposition Procedure

In this section, we introduce a fully automated procedure to achieve the decomposition of transformation networks. The procedures takes a consistency relation set as an input and virtually deletes as many redundant consistency relations as possible. The result is a simplified, possibly tree-like transformation network, which is equivalent to the input network. The topology of the resulting network gives information about compatibility of its relations. If the resulting network is a consistency relation tree, compatibility is inherent. Otherwise, developers can focus on consistency relations in remaining cycles to detect possible incompatibilities. The procedure considers transformations written in QVT-R, whose mapping to consistency relations was exposed above.

The decomposition procedure relies on an algorithmic way to detect redundant consistency relations. Redundancy occurs in two ways. First, it requires the existence of (sequences of) relations that relate the same metamodels. Second, it indicates that definitions of these consistency relations overlap in some way. In fact, this captures both aspects of consistency specifications. First, transformation networks give an insight into the specification structure, i.e. *which* metamodels are related with each other. This global point of view helps to check compatibility: a cycle in the network may indicate contradictory transformations, whereas compatibility is inherent in a tree topology. However, identifying redundant transformations requires to know exactly *how* metamodels are related to each other. It is necessary to see how consistency is defined with class properties at the metamodel element level to find out if some transformation definitions are contradictory. In predicate-based consistency relations, these definitions are made explicit through the use of predicates. This is a local point of view: given a set of transformations forming a cycle in a transformation network, each transformation definition is retrieved and compared to others so as to assess compatibility. Such a comparison is called a *redundancy test*. As a result, transformations are both edges of a graph and sets of consistency relations with exact definitions. Although both aspects are essential for decomposing transformations, the graph can be generated from consistency relation definitions. That is, vertices of the graph are metamodels and whenever a consistency relation definition relates two elements from two different metamodels, there is an edge between these metamodels. To take advantage of both aspects, we propose to perform redundancy tests on a single structure: a graph of class properties labeled by predicates that define consistency relations. The procedure operates in two phases. First, the decomposition procedure creates this structure out of QVT-R transformations. Then, it refers to consistency relation definitions inside it to detect redundant relations and check compatibility of the consistency specification.

5.2.1 From Consistency Specification to Property Graph

The decomposition procedure uses an intermediate representation of transformation networks, i.e., a single data structure that combines both aspects of transformations. This data structure, called a *property graph*, brings the graph characterization of transformations at the level of class properties and predicates. Such a structure can be represented as a hypergraph with a labeling.

Definition 20 (Property Graph) Let $\mathbb{CR} = \{CR_i\}_{i=1}^{n}$ be a set of consistency relations where each consistency relation CR_i is based on a set of predicates Π_i. A property graph is a couple $\mathcal{M} = (\mathcal{H}, l)$, such that $\mathcal{H} = (V_\mathcal{H}, E_\mathcal{H})$ is a hypergraph and $l\colon E_\mathcal{H} \to \{\text{TRUE}, \text{FALSE}\}^{I_{\mathfrak{P}_{\mathfrak{C}_l}} \times I_{\mathfrak{P}_{\mathfrak{C}_r}}}$ is a hyperedge labeling:

- $V_\mathcal{H}$ is the set of vertices, i.e., the set of all properties used in all predicates:

  $$V_\mathcal{H} = \bigcup_{i=1}^{n} \bigcup_{\pi \in \Pi_i} \mathrm{P}_\pi$$

- $E_\mathcal{H}$ is the set of hyperedges, i.e., $E_\mathcal{H} \subseteq \mathcal{P}(V_\mathcal{H}) \setminus \{\varnothing\}$. For a property graph, hyperedges are made up of properties that occur in the same predicate:

  $$E_\mathcal{H} = \bigcup_{i=1}^{n} \bigcup_{\pi \in \Pi_i} \{\mathrm{P}_\pi\}$$

- l is a function that labels each hyperedge with its corresponding predicate function:

  $$\forall i \in \{1,\ldots,n\}, \forall \pi = (\mathfrak{P}_{\mathfrak{C}_l}, \mathfrak{P}_{\mathfrak{C}_r}, f_\pi) \in \Pi_i : \\ l(\mathrm{P}_\pi) = f_\pi$$

```
import personMM : 'personmm.ecore';
import employeeMM : 'employeemm.ecore';

transformation PersonEmployee(
    person: personMM,
    employee: employeeMM) {

    top relation PE {
        fstn: String;
        lstn: String;
        inc: Integer;

        domain person p:Person {
            firstname=fstn,
            lastname=lstn,
            income=inc};
        domain employee e:Employee {
            name=fstn + '␣' + lstn,
            salary=inc};
    }
}
```

```
import personMM : 'personmm.ecore';
import residentMM : 'residentmm.ecore';

transformation PersonResident(
    person: personMM,
    resident: residentMM) {

    top relation PR {
        fstn: String;
        lstn: String;
        addr: String;

        domain person p:Person {
            firstname=fstn,
            lastname=lstn,
            address=addr};
        domain resident r:Resident {
            name=fstn + '␣' + lstn,
            address=addr};
    }
}
```

```
import employeeMM : 'employeemm.ecore';
import residentMM : 'residentmm.ecore';

transformation EmployeeResident(
    employee: employeeMM,
    resident: residentMM) {

    top relation ER {
        n: String;
        ssn: Integer;

        domain employee e:Employee {
            name=n,
            socsecnumber=ssn};
        domain resident r:Resident {
            name=n,
            socsecnumber=ssn};
    }
}
```

Fig. 10: Three binary QVT-R transformations forming a consistency specification, based on the relations in Figure 1

The idea behind the property graph is to group properties that participate in the definition of the same predicate. When the consistency relation set is not a tree, some properties may be used in the definition of multiple consistency relations. For example, an employee's name must be consistent with a resident's name and a person's first and last name. When properties are vertices, such groups form hyperedges. For a consistency relation to be redundant, there must be other relations that share properties with it. As a consequence, the property graph is useful to detect independent sets of consistency relations as well as cycles of hyperedges (which form alternative concatenations for redundant relations). Hyperedges only address the structural aspect of consistency relation definitions. For this reason, each hyperedge is labeled with its corresponding predicate function.

The need for hypergraphs arises from the fact that predicates can relate more than two properties: the predicate in the relation CR_{PE}, which ensures equality of an employee's name and the concatenation of first and last name of a person, contains three properties. The first phase of the procedure is to set up such a structure using QVT-R transformations.

Input and Transformation Parsing The procedure takes a set of well-formed QVT-R files as inputs. Each file contains one or more transformations. Transformations and metamodels are then parsed to provide a logical view of the consistency specification. Decomposition is intended to help the developer by either proving compatibility or highlighting possible causes of incompatibility. It does, however, not update the specification, thus access to the specification is read-only.

Traversal Order Among Transformations In order to build the property graph, each transformation must be processed to retrieve relations it contains. The decomposition procedure

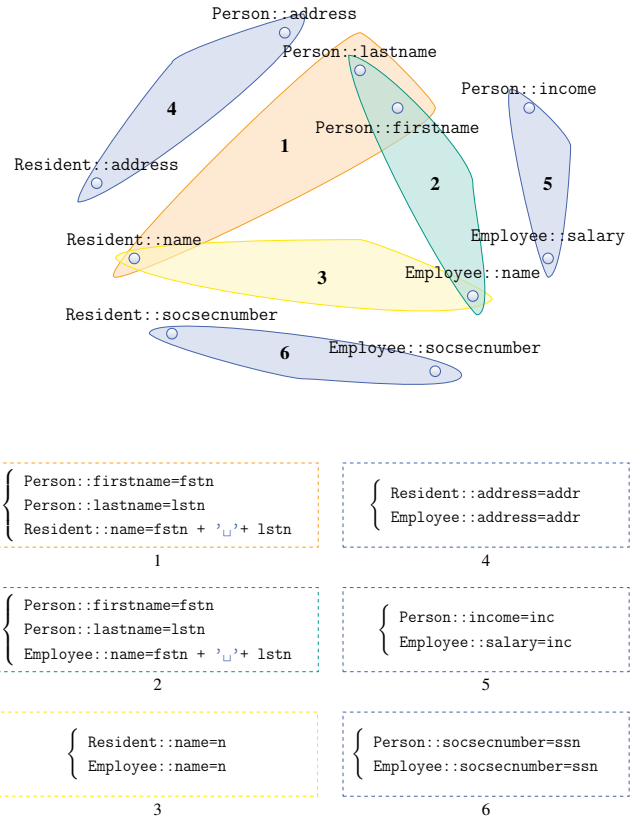


Fig. 11: Property graph for the QVT-R example in Figure 10 based on the relations in Figure 1

processes one transformation at a time. For compatibility checking purposes, transformations are independent of each other. They can be processed separately and in any order. The reason behind this is that QVT-R relations are not executed on models but only read. Therefore, they are side-effect free.

```
relation R {
    [variable declarations]

    domain MM1 a : A {P_A,1=e_A,1 , P_A,2=e_A,2}
    domain MM2 b : B {P_B,1=e_B,1}
    [when { PRE }] [where { INV }]
}
```

Listing 3: Structure of a QVT-R relation with property template items

Traversal Order Inside Transformations A transformation is a set of QVT-R relations. In general, each relationship deals with the consistency of only a small part of each metamodel, for example the consistency between two classes. Unlike QVT-R transformations, QVT-R relations cannot be processed in any order. There are two types of relations: top-level relations and non-top-level relations. Top-level relations are always invoked, whereas non-top-level relations are only invoked in `where` or `when` clauses of other relations through a mechanism similar to a function call. Only relations that would be invoked during the execution of transformations have to be processed for the decomposition, as non-invoked relations cannot cause incompatibilities. To determine a relevant processing order and retain only QVT-R relations that can be invoked, one solution is to create the call graph of the transformation. We impose a restriction on the specification to make this representation easier: relations may only be invoked in `where` clauses. Starting from top-level relations, relations are visited using a depth-first traversal. A relation R_2 can be visited from a relation R_1 if R_1's `when` clause invokes R_2. As a result, a relation is only visited if it is top-level or if a relation invoking it was itself visited before.

From QVT-R Relation to Hyperedge At the beginning, the property graph is an empty object. Each time a QVT-R relation is processed, the property graph may get new vertices and a new hyperedge. In accordance with Definition 20, hyperedges can be generated from predicates. Therefore, it is relevant to translate each QVT-R relation into a set of predicates. Listing 3 depicts the structure of an abstract QVT-R relation between two metamodels MM_1 and MM_2. Defining a predicate from a QVT-R relation amounts to find important properties for each metamodel and definitions that bind them. Class tuples are composed of classes that occur in each domain, i.e., $\mathfrak{C}_{MM_1} = \langle A \rangle$ and $\mathfrak{C}_{MM_2} = \langle B \rangle$. Each class in each class tuple is associated with a set of property template items. Important properties for the consistency specification are in the left-hand side of each property template item. For example, the property template item $P_{A,1} = e_{A,1}$ indicates that the property $P_{A,1}$ must match the OCL expression $e_{A,1}$ in which there are QVT-R variables. Not all properties are related to each other within the same QVT-R relation. For example, constraints on `Employee.salary` and `Employee.name` are independent, because consistency of one does not depend on consistency of the other. There is a simple criterion in QVT-R to identify interrelated properties. Pattern matching indicates which properties have to be grouped together to build a predicate. If two properties depend on the same QVT-R variable, they are interrelated, because a value assignment must satisfy both property template items. Predicates can then be generated from sets of interrelated properties. OCL expressions can also occur in `when` and `where` clauses. As with relation invocations, we focus on invariants (`where`), which we limit to the manipulation of QVT-R variables, given that properties can be limited to domain patterns without loss of generality. The processing of an invariant is similar to that of property template items: properties that depend on QVT-R variables occurring in the same invariant have to be grouped together.

Algorithm 1 Merge algorithm

```
1  procedure MERGE-CONSISTENCY-VARIABLES({(p, V_{p}) | p ∈ P})
2      stopMerge ← TRUE
3      entries ← ⟨({p}, V_{p}) | p ∈ P⟩
4
5      do
6          stopMerge ← TRUE
7          results ← ⟨⟩
8
9          while entries ≠ ⟨⟩ do
10             ref = (P_ref, V_{p_ref}) ← entries[0]
11             others ← entries[1:]
12             entries ← ⟨⟩
13
14             for (P, V_P) ∈ others do
15                 if V_P ∩ V_{P_ref} = ∅ then
16                     entries ← entries ∪ {(P, V_P)}
17                 else
18                     stopMerge ← FALSE
19                     ref ← (P ∪ P_ref, V_P ∪ V_{P_ref})
20             results ← results ∪ {ref}
21         entries ← results
22     while not stopMerge
23
24     return set(entries)
```

Algorithm 1 formalizes the way properties are grouped to form predicates. At the beginning of the algorithm, each property is associated with QVT-R variables that occur in the corresponding property template item. This association, called an *entry*, is a couple $(\{p\}, V_{\{p\}})$ where $\{p\}$ is a singleton containing the property p and $V_{\{p\}}$ a set of QVT-R variables. The entry of an invariant is composed of variables in it and all properties associated with these variables through property template items. At each iteration, the algorithm chooses a reference entry and merges all other entries with it if the intersection of their sets of QVT-R variables is nonempty. The algorithm stops when all sets of QVT-R variables are pairwise disjoint.

Example 8 There are five properties in the relation PE of the QVT-R transformation PersonEmployee in Figure 10, which can be described with the following entries:

$$(\{\texttt{firstname}\},\{fstn\}),(\{\texttt{lastname}\},\{lstn\}),$$
$$(\{\texttt{income}\},\{inc\}),(\{\texttt{name}\},\{fstn,lstn\}),$$
$$(\{\texttt{salary}\},\{inc\})$$

After the execution of the algorithm, properties are merged as follows:

$$(\{\texttt{firstname},\texttt{lastname},\texttt{name}\},\{fstn,lstn\}),$$
$$(\{\texttt{income},\texttt{salary}\},\{inc\})$$

This results in two sets of properties.

At the end of the algorithm, each entry can be transformed into a hyperedge. To do so, properties of the entry are assigned to the classes out of which they originate to form property sets. These property sets are grouped into two tuples. The predicate function is the conjunction of all OCL expressions associated with properties of the entry. When all transformations and all QVT-R relations have been processed, the property graph is correctly initialized. It is now invariable, in the sense that the procedure cannot add new vertices or hyperedges to the graph. Only hyperedges identified as redundant can then be removed. Moreover, all the information needed to assess compatibility in the consistency specification is translated into the property graph. There is no need to query metamodels or QVT-R transformations anymore.

5.2.2 From Property Graph to Decomposition

Given a property graph $\mathcal{M} = (\mathcal{H}, l)$, decomposition is accomplished by removing redundant consistency relations (hyperedges of $\mathcal{H}$) until all relations have been tested once or until the property graph is only composed of trees. The hypergraph $\mathcal{H}$ provides valuable information about the nature of the consistency specification. First, a necessary condition for a consistency relation between two metamodels M_1 and M_2 to be redundant according to Definition 12 is the existence of an alternative concatenation of relations that links M_1 and M_2 too. In terms of graph, there must exist a path between M_1 and M_2. Second, consistency relation definitions can be independent from each other, in the sense that they share no properties. Following Theorem 1, the union of two independent and compatible consistency relations is also compatible. Independence in the hypergraph is given by connected components. An important aspect of the decomposition procedure is to find consistency relation definitions that can be tested for redundancy. Taking advantage of the structure of the graph is useful for listing these relations.

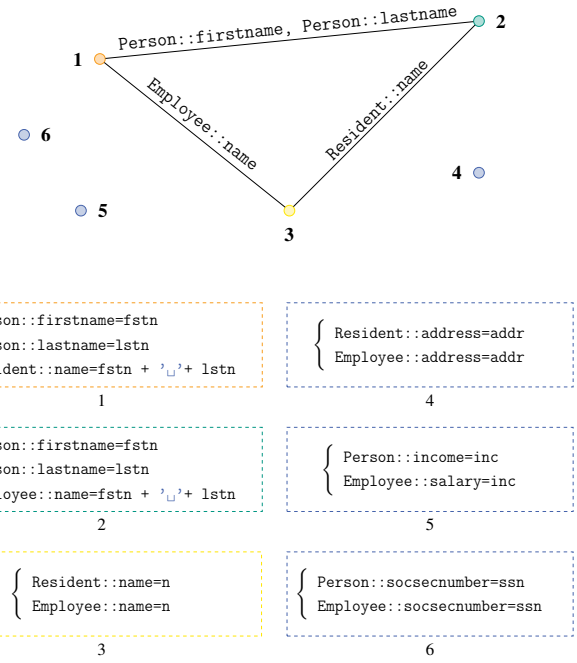


Fig. 12: Dual of the property graph for the QVT-R example in Figure 10 based on the relations in Figure 1

Consistency Relation Preprocessing As a special kind of hypergraph, a property graph has the advantage of being expressive when it comes to model consistency relation definitions involving multiple properties. The downside is that common graph algorithms (such as graph traversal) become harder to define and to apply. The choice between graphs and hypergraphs is a balance between abstraction and usability. For purposes of implementation, the property graph is replaced by its dual, i.e., a simple graph that is equivalent to it. The dual of the property graph is the graph whose vertices are hyperedges of the property graph. If hyperedges of the property graph share at least one property, their corresponding vertices in the dual are linked. An example for the dual of a property graph is given in Figure 12.

Definition 21 (Dual of a Property Graph) Let $\mathcal{M} = (\mathcal{H}, l)$ be a property graph. The dual of the property graph $\mathcal{M}$, denoted $\mathcal{M}^*$ is a tuple $(\mathcal{G}, v, l)$ with a simple graph $\mathcal{G}$ and two functions v and l such that:

- $V_\mathcal{G} = E_\mathcal{H}$
- $E_\mathcal{G} = \{\{E_1, E_2\} \mid \forall (E_1, E_2) \in E_\mathcal{H}^2 : E_1 \cap E_2 \neq \varnothing\}$
- $\forall (E_1, E_2) \in E_\mathcal{G} : v(\{E_1, E_2\}) = E_1 \cap E_2$

Each edge $\{E_1, E_2\}$ in the dual is labelled with the set of properties that occur both in E_1 and E_2. The dual contains all the information necessary to build the property graph again. Given a dual $\mathcal{M}^* = (\mathcal{G}, v, l)$, the property graph $\mathcal{M} = (\mathcal{H}, l)$ can be built by defining $V_\mathcal{H} = \bigcup_{V \in V_\mathcal{G}} V$ and $E_\mathcal{H} = V_\mathcal{G}$.

Independent Subsets of Consistency Relations In a consistency specification, consistency relations can form independent sets, in the sense that the consistency of one set can be checked and repaired independently without affecting the consistency of the other set. This occurs at two levels. First, at the metamodel level, when there exist two sets of metamodels such that no metamodel of one set is bound to a metamodel of the other set through a consistency relation. Second, at the metamodel element level, when two consistency relation sets of the same metamodel relate objects that are independent of each other. In terms of consistency relations and predicates, such sets are made up of relations that do not share any property. Independence is characterized in the same way for both levels in the property graph. This results in two subhypergraphs[1] such that there is no path (i.e., sequence of incident hyperedges) from one to the other. In the dual of the property graph, this results in two subgraphs that are not connected to each other as well. Otherwise, there would exist two consistency relations (one from each subgraph) that share at least one property, thus contradicting the hypothesis of independence. Independent subsets of hyperedges in the property graph can be processed independently, since one's compatibility has no influence on compatibility of the others.

Once the property graph is converted to its dual, the decomposition procedure computes independent subsets of relations. This can be achieved by computing connected components in the dual. Connected components are maximal subgraphs such that there exists a path between any two vertices in it. We use Tarjan's algorithm to compute them in linear time [66]. Incompatibilities can then only occur within a connected component. Therefore, we prove that a consistency relation set is compatible by proving compatibility of every connected component of the dual of the property graph.

Generation of Candidate Relations For a connected component to be compatible, there must be no redundant predicate in it. Like consistency relations at the metamodel level, predicates at the metamodel element level are said to be redundant if consistency specifications with and without it are equivalent. In the property graph, this requires the existence of an alternative sequence of hyperedges that relate the same properties as the possibly redundant hyperedge. Note that the existence of an alternative path is a necessary but not a sufficient condition. For instance, a predicate ensuring that two `String` attributes are equal could not be replaced by a sequence of predicates only ensuring that these strings have the same length. That is why the possibly redundant predicate and the alternative path of predicates are subject to a redundancy test (see Section 6).

Given that connected components are independent, a predicate can only be replaced by other predicates in the same component. Moreover, Theorem 2 also applies to the dual of the property graph: once the component is a tree, it is inherently compatible. As a result, the dual proves compatibility of the consistency specification if it is only composed of independent trees. Such a graph is called a *forest*.

In the property graph, an alternative path for a hyperedge E (i.e., a predicate) is a sequence of pairwise incident hyperedges such that the first and the last are also incident to E. Hyperedges of the property graph become vertices in the dual. Therefore, an alternative path for a predicate E in the dual is characterized by a cycle that contains E. If the vertex sequence of such a cycle is $\langle E, E_1, \ldots, E_n, E\rangle$, the alternative path is $\langle E_1, \ldots, E_n\rangle$. Ultimately, the generation of candidate relations for a redundancy test amounts to the enumeration of pairs $(E, \langle E_i\rangle)$, where E is a possibly redundant predicate (i.e., a hyperedge in the property graph and a vertex in its dual) and $\langle E_i\rangle$ is an alternative sequence of predicates that may replace E. There may be multiple alternative paths for a given predicate in the dual of the property graph, hence the need to find multiple cycles. The problem of finding all simple cycles in an undirected graph is called *cycle enumeration*.

The cycle enumeration algorithm used in the decomposition procedure relies on a *cycle basis*. In an undirected graph, a cycle basis is a set of simple cycles that can be combined to generate all other simple cycles of the graph. The cycle basis is first computed using Paton's algorithm [50]. For a given predicate, the enumeration starts from the cycle basis and merges two or more cycles in each iteration. In the context of the decomposition procedure, a cycle must also go through the predicate to analyze. Algorithm 2 is a slightly modified version of Gibb's algorithm to enumerate simple cycles in an undirected graph using a cycle basis [21]. In this algorithm, every cycle is represented as a set of edges. We denote the

Algorithm 2 Enumeration of alternative paths

```
1:  procedure ENUMERATE-CYCLES(Dual M*, v ∈ V_M*)
2:      {B_1, ..., B_n} ← PATON-ALGORITHM(M*)
3:      Q ← {B_1}, R ← ∅, R* ← ∅
4:
5:      for B ∈ {B_2, ..., B_n} do
6:          for T ∈ Q do
7:              if T ∩ B ≠ ∅ then
8:                  R ← R ∪ {T ⊕ B}
9:              else
10:                 R* ← R* ∪ {T ⊕ B}
            // Remove non-simple cycles from R
11:         for U, V ∈ R do
12:             if U ⊂ V then
13:                 R ← R \ {V}, R* ← R* ∪ {V}
            // New valid cycles are in R ∪ {B}
14:         for C ∈ R ∪ {B} do
15:             if v ∈ C and REPLACE-HYPEREDGE(v, C) then
16:                 remove v and its incident edges from M*
17:                 break
18:         Q ← Q ∪ R ∪ R* ∪ {B}
19:         R ← ∅, R* ← ∅
```

[1] A subhypergraph of a hypergraph $\mathcal{H} = (V_\mathcal{H}, E_\mathcal{H})$ is a hypergraph $\mathcal{S} = (V_\mathcal{S}, E_\mathcal{S})$ such that $E_\mathcal{S} \subseteq E_\mathcal{H}$ and $V_\mathcal{S} = \bigcup_{E \in E_\mathcal{S}} E$

symmetric difference with the $\oplus$ sign, i.e., $A \oplus B$ is the set of edges that are in A or in B but not in both. The set Q contains all linear combinations of cycles. Merged with cycles of the basis, these linear combinations are used to merge more than two cycles of the basis. At each iteration of Algorithm 2, new simple cycles are in $R \cup \{B\}$. Note that redundancy tests can be performed as new cycles are generated, as shown on line 14. By doing so, it is not necessary to store all cycles and wait for the end of the algorithm before starting redundancy tests. More interestingly, if the redundancy test is positive for one alternative sequence of predicates, there is no need to test others. The initial predicate can be removed and the algorithm can be used with another possibly redundant predicate.

Stopping Criterion The decomposition procedure stops when each predicate has been tested once. Note that if the connected component becomes a tree after a few removals of predicates, then the last tests of remaining predicates are trivial. As there are no more cycles in the dual of the connected component, no redundancy test is performed.

5.3 Summary

In this section, we have presented an algorithm for proving compatibility of relations in a consistency specification written in QVT-R. We defined property graphs and their dual as a representation of consistency relations and explained how they can be derived from a specification in QVT-R. We discussed how a consistency relation tree manifests in such a representation and how candidates for redundancies in connected components of such a representation can be found by computing cycle basis. Based on Theorem 1 and Theorem 2, as well as Corollary 1, this algorithm is able to prove compatibility by removing redundant relations, such that the resulting network is a composition of independent trees. However, we still need to discuss how redundancy of relations, in terms of redundant predicates in the property graph, can be identified, which we will discuss in the subsequent section.

6 Finding Redundancies in Transformations

In the decomposition procedure, enumerating possibly redundant predicates and proving that such predicates are redundant are uncoupled tasks. The latter task can be regarded as a black box embedded in the former one: only the result of the redundancy test matters. As a consequence, the decomposition procedure allows the use of various strategies to prove that a predicate is redundant. There is no perfect strategy due to the limitations on the decidability of OCL expressions. In this article, we opt for a strategy that allows the decomposition procedure to be fully automated. We first discuss how predicates can be compared to prove redundancy. Then, an approach that translates OCL constraints (in predicates) to first-order logic formulae and uses an automated theorem prover is introduced. Finally, the translation rules from OCL to first-order logic are presented along with their limitations.

6.1 Intensional Comparison of Predicates

Whatever the strategy, the *redundancy test* takes a couple $(E, \langle E_1, \ldots, E_n \rangle)$ as an input and returns TRUE if the predicate E was proven redundant because of the sequence of predicates $\langle E_1, \ldots, E_n \rangle$, FALSE otherwise. As stated in Definition 13, a consistency relation is considered left-equal redundant if its removal from the set of consistency relations leads to an equivalent set and relates the same kinds of elements at the left side. For the relation to be redundant, there must be an alternative sequence of relations that already fulfills the role of the initial relation. This also applies to the property graph: for a predicate to be removed, there must exist another sequence of predicates relating the same properties that is strict enough not to weaken the consistency specification. Weakening the consistency specification means allowing models that would have been considered non-consistent before the removal of the predicate. Since we only consider predicates that relate the same properties the additional requirement of left-equal redundancy in comparison to general redundancy in Definition 12 is always fulfilled. In the following, we thus only discuss redundancy rather then left-equal redundancy, as it is always given by construction. As illustrated in Figure 13, a predicate E can only be removed if all instances matching the predicate also match predicates $\langle E_1, \ldots, E_n \rangle$.

Therefore, a redundancy test is equivalent to the comparison of two sets of instances. However, a predicate may be fulfilled by infinitely many model elements. For example, the predicate ensuring that the income of a person and the salary of an employee are equal is valid for infinitely many integer pairs. It is impossible to compare these sets element per element (i.e., extensionally). Since consistency relations in the decomposition procedure are defined intensionally, by means of predicates, anyway, the redundancy test compares sets in their intensional specification. As a result, the redundancy test uses the description of the possibly redundant predicate and the candidate alternative sequence of predicates to decide whether the predicate is redundant. In QVT-R, predicates are expressed as OCL constraints. As part of the construction of the property graph, these constraints are already represented as hyperedge labels. That is, comparing predicate definitions in the decomposition procedure amounts to perform a static analysis of these labels and QVT-R relation conditions (`when` and `where` clauses). In order to prove redundancy, the static analysis has to rely on a rigorous framework to reason about OCL constraints. This is provided by various formal methods in the field of software verification.

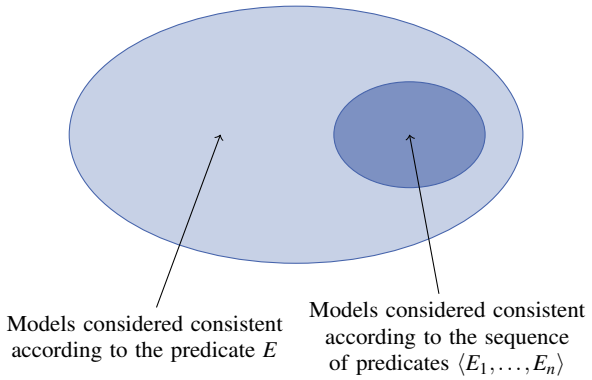


Fig. 13: The hyperedge E is redundant, because all instances valid according to E are already valid according to the alternative sequence $E_1,\ldots,E_n$ of hyperedges.

6.2 Encoding Redundancy in First-Order Formulae

In this article, the strategy we use to prove redundancy is based on first-order logic, a well-suited and expressive mathematical language for decision procedures. The idea is to set up a first-order formula that is valid (i.e., true under every interpretation) if and only if the redundancy test is positive. To this end, the formula embeds OCL expressions (translated into first-order logic as well) contained in predicates. Then, a theorem prover evaluates the validity of the formula.

The choice of first-order logic is motivated by the nature of OCL: there exists a general translation of OCL into first-order logic [5]. This result was later refined in [6] in order to show that OCL formulae are essentially full first-order formulae. What this means is that OCL does not form a fragment of first-order logic and needs all its expressiveness. First-order logic being undecidable in general, this also implies that not all redundant formulae can be proven valid. Therefore, the results obtained by the decomposition procedure are highly dependent on the performance of the theorem prover. Even with quantifiers and variables, the gap between programming languages and first-order sentences remains significant. OCL is composed of arithmetic operations, strings, arrays, etc. The meaning of language constructs must also be integrated into formulas. To this effect, it is possible to replace binary variables in first-order formulae with a Boolean sentence expressed in a given theory. Theories use all kinds of objects such as strings, floats, sequences, etc. With theories, the satisfiability problem equates to assign values to variables in first-order sentences such that the evaluation of sentences makes the whole formula TRUE. For instance, the formula $(a\times b=10)\wedge(a+b>0)$ is satisfiable given the assignment $\{a=2,b=5\}$. This extension is known as *satisfiability modulo theories* (SMT). First-order formulae for the SMT problem are called *SMT instances*. Some theorem provers come with built-in theories, they are thus called *theory-based theorem provers*. In the context of the decomposition procedure, we translate constructs in OCL constraints with corresponding constraints into built-in theories of the prover. By doing so, the mapping between OCL and first-order logic is easier to achieve.

6.2.1 Modeling as a Horn Clause

For any two models, consistency depends on condition elements in predicate-based consistency relations, which themselves depend entirely on property values for which predicate functions evaluate to TRUE. As a result, redundancy can be tested by comparing descriptions of predicate functions. This information is contained in the input of the redundancy test. Let $\pi=(\mathfrak{P}_{\mathfrak{C}_l},\mathfrak{P}_{\mathfrak{C}_r},f_\pi)$ be a predicate for two class tuples $\mathfrak{C}_l$ and $\mathfrak{C}_l$. During the construction of the property graph, a hyperedge composed of all properties in $\mathfrak{P}_{\mathfrak{C}_l}$ and $\mathfrak{P}_{\mathfrak{C}_r}$ is labeled with the description of the predicate function f_π.

In terms of predicate functions, the predicate E can be replaced by a sequence of predicates $\langle E_1,\ldots,E_n\rangle$ under the following condition: for any set of models, f_E evaluates to TRUE whenever $f_{E_1}\wedge\cdots\wedge f_{E_n}$ evaluates to TRUE. If this condition is met, the consistency specification is neither strengthened nor weakened after the removal of E. The specification is not strengthened because the removal of a predicate can only allow more sets of models to be consistent. It is not weakened either because all property values that match $\bigwedge f_{E_i}$ also match E. As a consequence, E is redundant. That is, solutions of $\bigwedge f_{E_i}$ form a subset of solutions of E. This is consistent with Figure 13. The redundancy test can be encoded as a formula in the following way:

$$(f_{E_1}\wedge\cdots\wedge f_{E_n})\Rightarrow f_E$$

The formula above is called a *Horn clause*. Horn clauses form an important fragment of logic in the field of automated reasoning. Terms on the left-hand side of the clause are called *facts*, whereas the term on the right-hand side is called *goal*. The implication represents the deduction of the goal from the facts. The assignment of values to variables in the Horn clause also models the instantiation of properties (i.e., the assignment of property values). If the Horn clause is valid, then the alternative sequence of predicates can replace the initial predicate whatever the instantiation of metamodel elements (i.e., whatever the models).

One last detail is to be taken into consideration for the translation of predicate functions. Horn clauses are usually described without quantifiers. In a Horn clause, all variables are implicitly universally quantified. Given that predicate functions are made up of OCL expressions, they contain local QVT-R variables. Consistency depends upon pattern matching, i.e., the existence of a valid assignment of variables. Therefore, QVT-R variables in the goal clause have to be existentially quantified.

Example 9 Figure 12 depicts the dual of the property graph derived from the motivational example in Figure 1. The dual contains four connected components, including three with one predicate only. Compatibility is already proven for these three components because they are trivial trees. The other component is made up of three predicates and contains a cycle ($\{1,2,3\}$). Let 3 be the possibly redundant predicate. Then, the alternative combination of predicates is composed of 1 and 2. This leads to the following formula in which 3 is the goal and 1 and 2 are the facts:

$$\begin{aligned}&[(\texttt{Person::firstname} = f1) \wedge (\texttt{Person::lastname} = l1)\\ &\quad \wedge (\texttt{Resident::name} = f1 + \text{“␣”} + l1)]\\ \wedge\,&[(\texttt{Person::firstname} = f2) \wedge (\texttt{Person::lastname} = l2)\\ &\quad \wedge (\texttt{Employee::name} = f2 + \text{“␣”} + l2)]\\ \Rightarrow\,&(\exists n : (\texttt{Resident::name} = n) \wedge (\texttt{Employee::name} = n))\end{aligned}$$

QVT-R variables have been renamed to avoid conflicts. This is necessary because they are no longer isolated as they were before in two distinct QVT-R relations. According to the SMT solver, the formula above is valid. Therefore, predicate 3 can be removed from the property graph and its dual. There are then only two predicates left in this component. It is inherently compatible. As all independent subsets of predicates are compatible, the consistency specification is compatible.

6.2.2 Redundancy Test

Redundancy can be proven by checking that the Horn clause derived from predicate functions is valid (i.e., true under every interpretation). Because the whole decomposition procedure is automated, the theorem prover, namely an SMT solver, embedded in the procedure is also automated. The solver takes an SMT instance as an input and answers whether it is satisfiable insofar as possible. Proving that a Horn clause H is valid is actually equivalent to proving that its negation $\neg H$ is unsatisfiable. Therefore, we prove that the SMT instance $f_{E_1} \wedge \cdots \wedge f_{E_n} \wedge \neg f_E$ is unsatisfiable. The SMT solver can provide three possible outcomes:

Satisfiable. If $\neg H$ is satisfiable, then H is not valid. This means that an interpretation exists (i.e., an instantiation of properties) that fulfills the possibly redundant predicate but not the alternative sequence of predicates. Thus, the predicate is not redundant and cannot be removed.

Unsatisfiable. If $\neg H$ is unsatisfiable, then H is valid. When the alternative sequence is fulfilled, so is the predicate. It is redundant and can be removed.

Unknown. First-order logic being undecidable, not all formulae can be proven valid. When a theorem prover is unable to evaluate the satisfiability of a formula, it returns *Unknown*. By application of the conservativeness principle, the redundancy test is considered negative. As a result, the predicate is not removed.

OCL Data Type	Ecore Data Type	SMT Data Type
Integer	`EInt`	IntSort
Real	`EDouble`	RealSort
Boolean	`EBoolean`	BoolSort
String	`Estring`	StringSort
UnlimitedNatural	`EInt`	IntSort *(without infinity)*

Table 2: Mapping between primitive types representations

6.3 Translation

Translation refers to the process of mapping OCL expressions of QVT-R relations to SMT instances. In the context of the decomposition procedure, this is facilitated by the fact that QVT-R uses a subset of OCL called *EssentialOCL* [45], a side-effect free sublanguage that provides primitives data types, data structures and operations to express constraints on models. Many constructs of OCL have a direct equivalent in theories of the theorem prover. More complex constructs can often be mapped through the combination of primitive constructs. Note that there also exist constructs that cannot be translated. Language constructs of SMT solvers are described using the SMT-LIB specification, a standard that provides among others an input language for solvers [4]. This language uses a syntax similar to that of Common Lisp. The translation is recursive: each OCL expression depends on the translation of its subexpressions. A complete reference of translated constructs has been developed in a master's thesis [51].

6.3.1 Primitive Data Types

OCL defines five primary data types: integers, reals, booleans, strings and unlimited naturals. These data types are mapped with Ecore when parsing QVT-R transformations. The mapping between Ecore data types and Z3 data types (called *sorts*) is described in Table 2. It is straightforward, except for *UnlimitedNatural*, a data type to represent multiplicities. *UnlimitedNatural* and *Integer* are different in that the former can take an infinite value whereas the latter cannot. IntSort in Z3 cannot be infinite. In this case, a workaround is to represent an *UnlimitedNatural* as a couple (IntSort, BoolSort) where the value equals ∞ if the Boolean is TRUE.

6.3.2 Data Structures

Primitive data structures in OCL are called *collections*. There are four types of collections based on the combination of two features, the first defining whether elements are ordered and the second whether duplicate elements are allowed. Among those four collection types, two are currently supported: sequences (ordered, duplicates allowed) and sets (non-ordered, no duplicates). In QVT-R, collections are used either as literals or as types for a special kind of properties: role names.

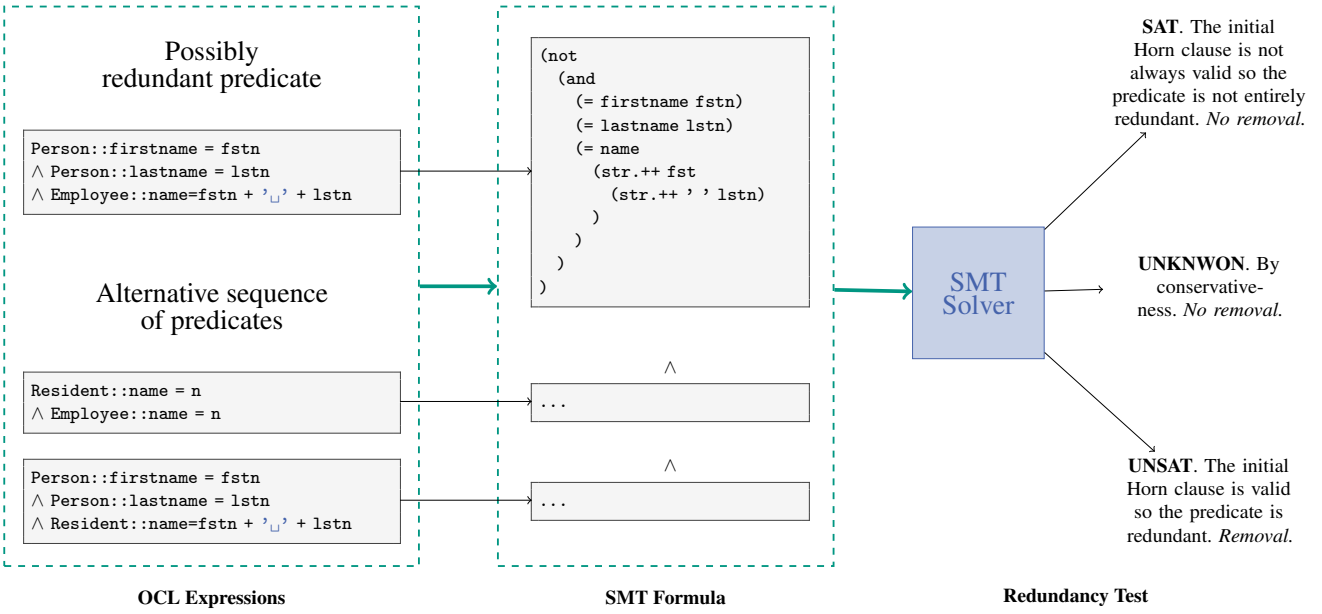


Fig. 14: Redundancy test, from OCL expressions to the SMT solver

Collection Literals Collection literals are OCL expressions that represent data structures with constant and predefined values (e.g., `Sequence{1, 4, 9}` or `Set{2, 5}`). In SMT solving, the fundamental theory to represent a collection of values is the theory of *arrays*. Arrays are maps that relate a set of indexes (domain) and a set of values (codomain). They are immutable, purely functional data structures. Unlike OCL data structures, there is no notion of size in arrays. To overcome this limitation, we translate collections to algebraic data types in the SMT input language. Data types are composed of an array (collection values) and an integer storing the collection size. It is noteworthy that collection literals rarely occur in consistency relations. In general, collections are groups of objects resulting from references in metamodels.

Collections from Role Names In QVT-R property template items, properties are either attributes (like `Person::name`) or role names. A role name is an alias for objects at the end of the reference owned by the class of the pattern. If the upper bound of the reference multiplicity is greater than one (e.g., `0..*`), then the role name may represent a collection of objects. The nature of the collection depends on whether the end is ordered or unique or both. Even if the content of the collection is unknown, it is possible to reason about role names by means of the theory of *uninterpreted functions* (UF). A role name r of a class c can be represented as a function of c (e.g., $r(c)$). By abstracting the semantics of functions, uninterpreted functions help to reason about model elements without knowing all their details. For example, two role names are equal if both belong to objects that have been proven to be equal themselves.

6.3.3 Operations

OCL also provides many operations on primitive data types and data structures, such as arithmetic operations or string operations. Following the object-oriented structure of OCL, every operation has a source and zero or more arguments. For example, the + operation denotes an addition when the source is an integer but a concatenation when the source is a string. We translated operations regarding arithmetics, booleans, conversion operators, equality operators, order relations, collections and strings [51].

Some OCL operations are said to be *untranslatable*, because it is impossible to find a mapping between them and features of state-of-the-art SMT solvers. As a result, there are QVT-R relations that cannot be processed by the decomposition procedure. For instance, the string operations `toLower` and `toUpper` cannot be easily translated without numerous user-defined axioms in current SMT solvers. Although decision procedures for such a case exist [71], they are not yet integrated into solvers.

6.4 Summary

In this section, we have presented an approach to evaluate redundancy of a predicate in a property graph (respectively its dual) for the decomposition procedure, also depicted in Figure 14. The approach translates the OCL expressions of predicates into logic formulae and generates Horn clauses for a potentially redundant predicate and its alternative predicates. If an SMT solver proves unsatisfiability of that clause, the checked predicate is redundant and can be removed.

7 Evaluation

We have conducted a case study to evaluate correctness and applicability of our approach. The evaluation focuses on the appropriate operationalization of the formal approach, which is proven correct, and its practical applicability in terms of providing an appropriate level of conservativeness. This defines our contribution **C4**.

7.1 Goals

Correctness Correctness of our approach means that it is able to classify a given set of consistency relations as compatible or otherwise does not reveal a result. This especially means that it operates conservatively. The formal approach presented in Section 4 is proven correct by Theorem 3, Theorem 2 and Corollary 1, such that we do not need to further evaluate its correctness. For that reason, the correctness evaluation focuses on the operationalized approach presented in Section 5 and Section 6. We investigate whether the operationalized approach reveals expected results, indicating that the mapping of our formalism to QVT-R is correct, and especially that it operates conservatively.

Applicability Since the approach defines a fully automated algorithm, which does not require further input apart from the QVT-R relations to check, applicability may only be restricted by inadequate outputs, which are correct but not useful for the user. In consequence, we consider applicability of our approach especially in terms of the practicality regarding its degree of conservativeness. If the approach is not able to identify compatibility in too many cases, in which relations are actually compatible, applicability would be limited. For that reason, we aimed to identify in how many cases the approach is not able to prove compatibility although compatibility is given, and what the reasons for those results are. It is of special interest whether those are conceptual issues of the formal approach or a limitation of the operationalization that may be fixed by other realization approaches.

7.2 Methodology

To empirically evaluate correctness and applicability of our approach, we developed a prototypical implementation and applied it to exemplary case studies. We give an overview of that prototype in the subsequent subsection. Table 3 summarizes the case studies to which we applied the approach. Each of those scenarios consists of three or four metamodels and especially comprises primitive data types and operations. They were specifically developed to evaluate our approach by defining as many kinds of relations that can be expressed with QVT-R as possible, thus also reflecting edge cases.

#	Scenario Description	Compatible
1	Three equal String attributes of three metamodels	✓
2	Six equal String attributes of three metamodels	✓
3	Concatenation of two String attributes	✓
4	Double concatenation of four String attributes	✓
5	Substring in a String attribute	✓
6	Substring in a String attribute with precondition	✓
7	Precondition with all primitive datatypes	✓
8	Absolute value of Integer attribute with precondition	✓
9	Transitive equality for three Integer attributes	✓
10	Inequalities for three Integer attributes	✓
11	Contradictory equalities for three Integer attributes	✗
12	Contradictory inequalities for three Integer attributes	✗
13	Constant property template items	✓
14	Linear equations with three Integer attributes	✓
15	Contradictory linear equations for three Int. attributes	✗
16	Emptiness of various OCL sequence and set literals	✗
17	Equal String attributes for four metamodels	✓
18	Transitive inclusions in sequences	✓
19	Comparison of role names in three metamodels	✓

Table 3: Example scenarios of consistency relations and their compatibility property, from [51]

We developed 14 compatible and four incompatible transformations, according to our Definition 7 for compatibility. Thus, we know the ground truth regarding compatibility of transformations for each scenario by construction. Applying our prototypical implementation to those scenarios classifies them as compatible (*positives*) or makes no statement about compatibility (*negatives*), i.e., they could either be compatible or not. Considering which of the results are actually correct gives us insights on correctness and applicability.

The approach is correct, which especially means that it operates conservatively, if it does not classify any transformation networks as compatible although they are not. This means that no *false positives* are allowed to occur or otherwise the approach would, per definition, be incorrect. In other words, the *precision* of the approach has to be 1:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives}$$

Applicability of the approach depends on the degree of conservativeness, i.e., in how many cases it does not identify a transformation network as compatible although it is. This is reflected by the number of *false negatives* and, when compared to the *true positives* known as the *recall*, gives insights on the degree of conservativeness:

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

A high recall value indicates high applicability of the approach in terms of not being too conservative.

7.3 Prototypical Implementation

The decomposition procedure presented in Sections 5 and 6 resulted in the implementation of a prototype, which is available on GitHub [17]. The formal approach of this paper addresses a common problem in the development of cyber-physical systems: inconsistencies lead to the development of incompatible artifacts, which in turn can lead to unexpected behavior. Therefore, the practicality of our approach matters. A tool for proving compatibility could be easily integrated into the development process of a transformation network in order to assist developers and domain experts.

7.3.1 Features

The implementation of the procedure takes a set of QVT-R transformations as an input and outputs a list of redundant QVT-R relations. This list must be compared to the initial consistency specification. There are two scenarios: either the delivered specification forms a consistency relation tree or there are still cycles left. In the former case, compatibility is proven. In the latter case, remaining cycles require the developer's attention. This may be due to incompatibility or the inability of the procedure to prove redundancy.

In addition to the features of the procedure, the prototype provides an input validation. There are two reasons why a consistency specification may cause the procedure not to operate correctly. First, specifications can be composed of not well-formed transformations, i.e., QVT-R transformations that are syntactically incorrect. In this case, the specification is not usable and the procedure immediately exits. There is another scenario: specifications that are well-formed but not valid. For example, this occurs when two transformations have the same name or when a QVT-R domain pattern uses a nonexistent class. Although this scenario is non-blocking, i.e., the decomposition procedure still produces a result, the output must be interpreted with caution. To assist the developer, the procedure displays semantic errors in the specification at the beginning of the parsing. In the end, the procedure is intended to be non-intrusive, i.e., it is does not alter any artifact and can be used at any moment during the development process, i.e., by logging its results.

7.3.2 Implementation

Technical choices are mostly driven by the support of model-driven engineering technologies. One important initiative to this end is the Model Driven Architecture (MDA) [46]. The decomposition procedure makes use of many specifications recommended by the MDA, including QVT-R for the definition of transformations, Essential MOF (EMOF) for metamodels [48] and OCL for constraints over metamodels [47]. Eclipse provides an implementation of these languages within the Eclipse Modeling Framework (EMF) [61].

	Classified Compatible	**Unclassified**
Compatible	12	4
Incompatible	0	3

Table 4: Number of scenarios from Table 3 regarding actual compatibility and their classification by our approach.

As a consequence, metamodels of the decomposition procedure are implemented using *Ecore*, a meta-metamodel that is compliant with EMOF. EMF supports a number of model-to-model transformation languages through the *Eclipse MMT* project. In particular, the *QVT Declarative* (QVTd) component provides a parser for QVT-R transformations. As QVT-R relies on OCL, QVTd makes use of *Eclipse OCL*, an implementation of the OCL language.

Regarding the strategy for redundancy testing, the implementation of the decomposition procedure requires the use of an SMT solver. Most SMT solvers are based on SMT-LIB, an initiative that provides a common input/output language for SMT instances. The prototype relies on the Z3 theorem prover, an SMT solver with a Java binding and a large number of theories supported [16].

7.4 Results

In the following, we present the results of our evaluation regarding the methodology proposed in Subsection 7.2 applied to the prototypical implementation introduced in Subsection 7.3. The classification results are summarized in Table 4.

7.4.1 Correctness

As discussed before, the correctness of our approach in terms of conservative behavior is proven for the formal approach by construction. Since the operationalized approach is based on that formalization, correctness is also given by construction provided that the following requirements are fulfilled:

1. All relevant QVT-R relations are considered, i.e., all QVT-R relations are represented in the property graph to be considered as consistency relations to be checked.
2. Consistency rules in QVT-R are defined using variables, so all constructs referring to these variable have to be considered. This especially means that all template expressions need to be considered for the property graph construction, namely property template items, preconditions and invariants.

We ensured that all these relevant elements are considered by construction of the approach discussed in Section 5 and Section 6. However, the results of the case study further validate that we did not miss any relevant parts of QVT-R

relations. In fact, the results summarized in Table 4 show that we have a precision of 1, thus having no incompatible scenarios classified as compatible by error:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} = \frac{12}{12+0} = 1$$

7.4.2 Applicability

Applicability of the presented approach depends on its degree of conservativeness. If it is not able to prove compatibility of compatible transformation networks in too many cases, applicability is reduced. In general, the conservative behavior results especially from two reasons:

1. Definition 13 for left-equal redundancy, which is used to prove compatibility of a network, may be a too strong requirement for identifying compatibility-preserving consistency relations.
2. Consistency relations as defined in Definition 2 are extensional specifications and thus usually enumerate infinite sets of elements, which are impossible to compare programmatically. For that reason, our operationalized approach relies on intensional specifications, which describe how consistent pairs of elements can be derived. These specifications are written in OCL. However, OCL in general is undecidable, because it can be transformed into first-order logic [5].

Especially formulae that contain many quantifiers are hard to analyze. For that reason, the number of variables used in a consistency relation is crucial, as these variables are translated to existentially quantified formulae. Although not all available OCL constructs might be necessary to describe relevant consistency relations, constructs involving operations on sets and strings are problematic. Operation collections are transferred to quantified formulae, which are hard to analyze. Reasoning about strings is problematic, because some OCL operations like `toUpper` and `toLower` cannot be easily transferred to state-of-the-art SMT solvers like Z3 and thus cannot be considered for detecting redundancy. Furthermore, SMT solvers use heuristics, so it not possible to formally evaluate which kinds of relations can be analyzed.

Applying the prototypical implementation of our approach to the scenarios introduced in Subsection 7.2 led to the result that twelve of the 15 compatible transformation networks were correctly classified as compatible, whereas three were not. This leads to a recall value of 80%.

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} = \frac{12}{12+3} = 0.8$$

The three scenarios that were not classified, although they are actually compatible, are Scenarios 8, 18 and 19 from Table 3. In all cases, the SMT solver returned *unknown*, although it should have returned *unsatisfiable*. In consequence, in each case an actually consistent consistency relation was not removed, thus the set of relations was not considered compatible although it is. More precisely, in scenario 8, a precondition ensures that an element is included in the intersection of two set literals, which the solver was not able to check properly. Scenarios 18 and 19 were problematic due to comparable reasons. While in Scenario 18 the transitive inclusion of sets was defined, Scenario 19 considers role names of classes with equivalent identifiers, which both the solver was not able to check properly. All observed false negatives were due to reasons of undecidability of the translation of OCL constructs to first-order logic.

To summarize, we found that basic operations on primitive data types, even with non-trivial constraints involving integer equations and string operations, were treated correctly. More complex operations and structures requiring many quantifiers led to unprovability by the SMT solver, especially concerning collection operations and role names. Thus, the approach is especially applicable for consistency relations concerning attributes and primitive types. However, this limitation does only concern the chosen SMT solver approach, but neither the concept of operationalization nor of the formal framework behind it. We did especially not find a scenario, in which our definition of left-equal redundancy was too strict for proving compatibility.

7.5 Discussion

The presented approach aims to support developers of transformation networks to independently develop individual transformations, i.e., parts of the network, without aligning the consistency relations on which the transformations are based a priori, but allows them to check their compatibility during or after development. For that reasons, it provides a benefit by automating a process that currently requires manual effort by either aligning consistency relations with each other or by defining test cases, which are able to validate but not to verify compatibility, i.e., which cannot make any all-quantified statements about compatibility. Even if the approach had a high degree of conservativeness, the approach would be beneficial for the combination of independently developed transformation. First, there is still a chance that the approach is able to prove compatibility for a given set of relations. Second, if the approach is not able to prove compatibility, it may still find some redundant relations and thus reduces the effort for the user to investigate the remaining relations for contradictions. It would even be possible to define an interactive approach, combining the removal of redundant relations by proof and by user decision, as we will propose in Subsection 9.2. In the following, we discuss threats to the validity of our evaluation and discuss limitations of both the approach and our evaluation.

7.5.1 Threats to Validity

Although we designed our evaluation in a way such that it gives us appropriate insights on correctness and applicability of the approach, there may be limitations regarding its internal and external validity.

Scenario Selection We developed the scenarios specifically for the evaluation of our approach. Due to that reason, they may not be sufficiently representative for actual transformation networks. However, the scenarios were specifically designed to test different aspects of the approach. They represent an extensive set of consistency relations and especially different types of relations, also considering edge cases that may be rare in practical scenarios. That even provides a benefit regarding practical consistency relation specifications.

Scenario Complexity The scenarios only comprise OCL constructs that are currently supported by our approach. This may be a bias, because unsupported constructs are not covered by the evaluation. However, the algorithm would not deliver any results in such scenarios anyway, thus applying it to them would not give further insights. Additionally, the unsupported constructs are only a limitation of the current implementation and not a conceptual limitation of the approach. Finally, the limitation in complexity of the relations may also lead to the fact that we do not cover cases that actually occur in practice but for which our definition for redundancy is too strong to prove compatibility. This is an actual limitation that has to be considered in further evaluations.

Scenario Size The considered scenarios are rather small, as they only consider up to four metamodels and only few consistency relations. Actual consistency relations will involve larger metamodels and consistency relations. However, the inductive characteristics of our approach makes it independent from the number of metamodels and relations to consider. One property affected by the scenario size is the performance of the approach, which we discuss in Subsection 7.5.2.

Conclusion In consequence, our evaluation only gives an initial indicator for the applicability of our approach due to the limited set and complexity of scenarios. To improve evidence in external validity, applying the approach to further, more practical transformation networks would be beneficial. However, acquiring such a networks is difficult. At least existing networks contain transformations that are aligned with each other and thus do not allow to validate cases in which consistency relations are actually incompatible. It may be possible to reduce that problem by taking existing sets of consistency relations and manually extending them with either redundant or incompatible consistency relations, checking whether the approach is able to correctly remove the added redundant relations or detect incompatibility.

7.5.2 Limitations

Current limitations of our approach especially arise from its operationalization and the limitations of SMT solvers. Additionally, we are currently only able to argue for the benefits and performance of our approach, but further evaluation would be necessary to validate these arguments.

Operationalized Approach The operationalized approach has both fundamental and technical limitations. First, SMT solvers are limited in a way that they are not able to analyze all types of expressions regarding satisfiability. In consequence, even if all kinds of QVT-R respectively OCL constructs can be transformed into appropriate logic formulae, it may not be able to check them for satisfiability, as we have seen in the applicability evaluation. Second, we do not yet provide a translation for all kinds of OCL constructs to logic formulae, such that not all QVT-R relations are supported. However, this is a technical limitations that can be solved by implementing these translations.

Benefits Evaluation We did not provide an evaluation for the claimed benefits of our approach. This is due to two reasons. First, we already argued why the approach provides a benefit anyway due to being fully automated and not requiring further input. Second, to the best of our knowledge, there are no competitive approaches to compare our approach with. In consequence, only an empirical study in which the approach is practically applied would give further insights to its benefits for a user, apart from the obvious benefits given by the automation of a currently manual process.

Performance Evaluation We did neither formally evaluate nor measure performance of our approach. If the approach required to much time to be executed on a set of transformations, its applicability would be reduced. SMT solvers, such as the used Z3 solver, depend on heuristics, which makes their performance unpredictable. Thus, it would be important to evaluate performance of the approach in a case study. In our case study, we did observe any time-consuming scenarios. However, transformation networks with more and larger transformations and especially many cycles need to be investigated to make generalizable statements on the performance.

8 Related Work

In this article, we have presented an approach for proving compatibility of transformation network. Thus, our work contributes to the goal of achieving consistency respectively consistency preservation between multiple models and is related to other approaches with that goal. It is highly related to the area of transformation networks and multi-directional transformations, especially to validation techniques for them.

Combining transformations to a network is also related to transformation composition and transformation chain construction, as it is a more general case of these specific problems. Finally, we used formal techniques including a theorem prover to make statements about OCL expressions in QVT-R relations, which is why other comparable formal techniques are related to our work. We discuss the relation of our work to work in these areas in the following.

8.1 Consistency Preservation of Multiple Models

Preserving consistency of software artifacts (i.e., models) has been long researched. Starting with approaches for specific modeling languages, such as the UML [15], the relevance of model-driven engineering, accompanied by OMG's Model Driven Architecture [46] process specification, rose. Several approaches provide domain-specific solutions for consistency problems, such as for consistency between SysML [49] and AUTOSAR [59] in the automotive domain [22]. Modeling frameworks, such as EMF [61], enabled the definition of tools that are independent from concrete models, such as transformation languages, model merging tools and so on.

Based on such modeling framework, different approaches considering model consistency have been developed. They can be distinguished into approaches that are only able to check consistency of models [53, 29] and those that are able to also enforce consistency. Consistency-enforcing approaches are sometimes also referred to as *model repair* approaches, which were surveyed by Macedo et al. [40]. They also considered whether the approaches are able to handle multiple models or only pairs, but found that only one of the considered approaches handles that case by considering the pairwise relations between models. Consistency preservation approaches are based on heterogeneous ideas, ranging from model merging [41, 54], macro- and megamodeling [56, 55], model finding and constraint solving [36, 39] and model transformations [14, 33, 57, 40]. Most of these approaches, if supporting the case of multiple models at all, assume that there is a common knowledge about how all involved models shall be related. With modular knowledge, like assumed when creating transformation networks, incompatibilities in the way consistency is considered always lead to problems, regardless of the approach chosen, so the finding of our work is relevant for all these approaches.

8.2 Multi-directional Transformations

Of the previously presented approaches for consistency preservation, model transformations is the approach that provides the highest degree of freedom to influence the way in which consistency is restored. The area of incremental, bidirectional transformations is most relevant for consistency preservation purposes. The concept of bidirectional transformation can be generalized to multi-directional transformations [62, 11], i.e., specification with relations as well as consistency repair routines between multiple models. So consistency preservation between multiple models can be achieved with two transformation approaches, first with multi-directional transformations, and second by combining bidirectional transformations to networks. However, those topics have only been considered in research since recently [11].

Only few approaches presented in the recent years explicitly consider the case where multiple models shall be kept consistent. Several transformation languages have been proposed in the recent years, surveyed by Kusel et al. [33]. Among popular languages such as QVT [45], Atlas Transformation Language (ATL) [25, 76], VIATRA [7] and Triple Graph Grammars (TGGs) [2, 1], originally developed by Schürr [60], only the QVT-R standard explicitly considers the case in which more than two models shall be transformed into each other by allowing the definition of multi-directional transformations. However, Macedo et al. [38] revealed several limitations of its applicability. Extensions of TGGs to multiple models called Multi Graph Grammars (MGGs) [30] and Graph Diagram Grammars [68, 67] consider the specification of multi-directional rules, but focus on the specification concept and do not yet consider what happens if several such rules are conflicting. Although multi-directional transformation approaches are inherently less prone to compatibility problems, we already discussed drawbacks of the necessity to have no modular specification of consistency.

The case that transformations are combined to networks is not considered by any existing transformation language. Most of the existing considerations for such networks are rather theoretical. For a single bidirectional transformation, several relevant properties, such as correctness, hippocraticness or undoability have been found and researched [63]. In our work, in contrast, we are interested in further properties that are relevant when combining transformations to networks. Stevens [62] started to discuss problems that arise from the combination of several transformations, such as potential non-termination or the problem of not finding a consistent solution. She defined in which situations it is not possible to express a multiary relation by means of binary relations at all. She also discussed orchestration problems for the execution order of transformations [64]. However, compatibility of relations have not been considered yet.

An approach to emulate multi-directional transformations in terms of bidirectional transformation networks are commonalities models. They introduce further models that contain the information that is shared between models and thus has to be kept consistent. They serve as a hub with bidirectional transformations to the actual models, acting like a multi-directional transformations. This concept has been considered on a rather theoretical basis [65, 18], discussing

which kinds of relations can be expressed with such an approach, and from an engineering perspective [27], discussing the modular specification and composition of commonalities. However, all these approaches do not allow a combination of independently developed consistency specifications for subsets of the models, which is the goal of our work.

8.3 Transformation (De-)Composition

Our approach can be seen as a technique to decompose transformations into sets of transformations that are either essential or redundant. Transformation composition has especially been researched in terms of creating chains of transformations, composing larger transformations from smaller ones and finding and extracting common parts in different transformations, known as *factorization*.

A transformation chain defines a sequence of transformations, which transforms one abstract, high-level model into one low-level model across one or more others of different abstraction levels. Languages like FTG+PM [35] and UniTI [70] allow to specify the combination of transformations to chains. However, tools like UniTI derive compatibility from additional, external specifications of the transformations, for which conformance to the actual transformation is not guaranteed. Additionally, transformation chains are only a special case of transformation networks, as each transformation network is also aware of the individual transformation chains between all pairs of models. They are, by construction, not that prone to compatibility problems, because there cannot be any cycles in the transformations.

Transformation composition techniques can be seen as a means to build transformation networks. Internal composition techniques can be separated into white-box approaches, which are integrated into languages [72, 74, 73], e.g., inheritance or superimposition techniques, and external techniques, which consider the transformations as black boxes. For such transformation compositions, Lano et al. [34] present a catalog of patterns that foster correct composition. Our approach considers the transformations as white boxes, or at least requires knowledge about the defined consistency relations, but is, in contrast to existing work, not integrated into a transformation language. Additionally, existing approaches have the goal of enhancing composition of transformations between the same metamodels, thus providing benefits like improved reusability, whereas we combine transformations between different metamodels. However, our findings on compatibility can also be applied to composition of transformations between the same metamodels. Finally, factorization approaches identify common parts of transformations and extract them into a base transformation from which the individual parts are extended [58]. Such approaches use intrusive operators that adapt the transformations for composition, whereas we only non-intrusively analyze the transformations.

8.4 Formal Methods in Consistency Preservation

Some approaches consider consistency preservation as a constraint solving problem rather than a transformation problem. They use constraints to represent consistency relations, like we do for the relations of transformations, and then try to find valid solutions after an inconsistency-introducing modification was made by model finding. For example, some approaches use Answer Set Programming (ASP) to preserve consistency of models [10, 19]. For QVT-R and Echo, implementations with Alloy were proposed to resolve inconsistencies [36, 37], which were also implemented in the transformation tool Echo. However, these approaches find consistent models based on the defined constraints rather than checking whether those constraints can be fulfilled under specific conditions, like our definition of compatibility specifies and our presented approach is able to prove.

Finally, there are several approaches for the validation of OCL constraints used to define conditions on valid models or to define model transformations. To validate the existence of models that fulfill certain OCL constraints, Kuhlmann et al. [32] and González et al. [23] propose an approach using SAT solvers. For the validation of model transformations, different approaches have been proposed. Cabot et al. [9] derive invariants from transformations, which they use for verification purposes, such as to find whether a model exists that can fulfill a transformation rule. Comparably, Cuadrado et al. [12] analyze ATL transformations to find errors in transformations and to find out whether a source model exists that may trigger a transformation. Rather than using constraint logic for verifying a transformation, Azizi et al. [3] verify correctness of an ETL transformation using the symbolic execution of the transformation. Instead of checking a transformation on its own, Vallecillo et al. [69] propose to define a formal specification of transformations, against which they can be validated. Finally, Büttner et al. [8] propose an approach for proving correctness of ATL transformations against pre- and post-conditions using SMT solvers. Most approaches use some kind of constraint logic or theorem proving for validating correctness of transformations, which is comparable to our approach. Our defined notion of compatibility is comparable to correctness notions in the approaches of Cuadrado et al. [12] and Cabot et al. [9], as they try to figure out if a rule can be triggered by any model. However, all these approaches consider correctness of a single transformation. In contrast, we consider correctness of a transformation network.

9 Future Work

Based on the presented work, there are plenty of possibilities and necessities for follow-up research. In the following, we present an overview of the topics that are most relevant to be considered next from our point of view. We first discuss

conceptual extensions by considering weaker redundancy notions and processes to use the approach in, as well as its impact on other correctness requirements in terms of consistency repair. Afterwards, we shortly discuss the possibility to evaluate alternatives for the realization of our formal approach, as well as ideas for completing our realization.

9.1 Relaxation of Redundancy Notion

In Subsection 4.2, we have introduced the notion of *left-equal redundancy*, since an intuitive notion of redundancy is not strong enough to be compatibility-preserving. We based the decision for that stronger definition on insights from a counterexample for redundancy being compatibility-preserving. However, there might be a weaker notion than left-equal redundancy that is still strong enough to be compatibility-preserving. In that case, it would be interesting to investigate application scenarios in which compatibility-preserving relations that are not left-equal redundant occur and how the definition can be appropriately relaxed, such that our approach supports that notion as well.

9.2 Interactive Process

Our approach enables a user to check a network of consistency relations regarding compatibility. If the approach identifies a given network as compatible, it is actually compatible as the algorithm operates conservatively. However, the approach is not able to prove incompatibility. If the approach does not identify a network as compatible, it may be incompatible or not. For that reason, we should define a holistic process for the usage of the approach, which integrates further information given by the user into the process of proving compatibility. If the algorithm is not able to prove compatibility, it can present the network, in which it removed some redundant relations, to the user. The user could then be asked to declare a cycle of relations as compatible, for which the algorithm is not able to prove it, or which are actually not compatible, but whose restriction of relations regarding consistency according to other relations is intended. Afterwards, the algorithm could proceed with finding further redundant relations to prove compatibility, based on the decision of the user. As a result, the approach would be applicable to more cases in which compatibility is intentionally not given or in which the algorithm on its own is not able to prove it.

9.3 Impact on Consistency Repair

In Subsection 3.1, we introduced different levels at which a transformation network can be incorrect [28]. While the approach in this paper is concerned with the correctness of consistency relations, correctness can also be considered at the level of consistency repair routines that ensure consistency according to the relations. Correctness of consistency relations is a necessary requirement for consistency repair routines to work properly, as otherwise, for example, non-terminating loops that try to fulfill consistency relations that can never be fulfilled may occur. Based on the formal notion of compatibility presented in this article, we will evaluate in future work how compatibility affects correctness of consistency repair routines, and whether all remaining correctness issues can be avoided by construction, as proposed in [28]. This would result in a holistic approach based on construction guidelines and our technique to prove compatibility that enables developers to build correct transformation networks.

9.4 Validation of Operationalization Alternatives

We have chosen to translate OCL expressions in QVT-R relations into first-order logic formulae and to use an SMT solver, namely Z3, to evaluate Horn clauses of those expressions that represent the potential redundancy in cycles of consistency relations. However, such a theorem prover is not able to evaluate satisfiability of clauses in all cases, as we already discussed in Subsection 7.5.2. It is possible to implement the approach in Section 5 by means of other formal methods. For example, interactive theorem provers may be able to prove redundancy of consistency relations in more cases. This hypothesis can be evaluated in future work. Another possibility is the use of multiple formal methods in the decomposition procedure. Although this requires translating OCL expressions into multiple languages, some formal methods can sometimes provide proofs where others cannot. Thus, the simultaneous use of different symbolic computation tools can increase the chances of finding redundancy proofs.

9.5 Completion of Operationalized Approach

The operationalization of our approach presented in Section 5 and Section 6 is currently limited to the parts of QVT-R relations and OCL operations that we presented in those sections. In future work, we will extend the set of supported OCL operations, which the approach is able to translate into first-order logic formulae. This will allow us to apply the approach to more sophisticated case studies and provide further evaluation to indicate general applicability of the approach. Moreover, SMT solvers come with heuristics (sometimes called *strategies*) to fine-tune their performances. Strategies should be chosen according to the nature of tested SMT instances, i.e.. consistency specifications. Thus, a better integration of the SMT solver can improve the realization of the current approach for proving compatibility in transformation networks.

10 Conclusion

In this article, we presented an approach to prove compatibility of consistency relations in transformation network. We introduced a formal notion of compatibility, describing when consistency relations are considered contradictory, and we proved correctness of a formal approach that checks whether a transformation network is compatible. We defined an operationalization of that approach for QVT-R and OCL, which uses the translation of OCL to first-order logic formulae and an SMT solver to prove compatibility.

Applying the approach to different scenarios in an evaluation, we found that the approach operates correctly in the sense that it produces *conservative* results. Further, we found that conservativeness is rather low, i.e., only few actually compatible transformation networks were not identified as such. More precisely, only 20% of the compatible transformations were not identified as such, which indicates the practical applicability of the approach. The current limitations of the approach and the degree of conservativeness especially arise from limitations due to undecidability of OCL and missing translations of OCL constructs to first-order logic formulae. We did not identify conceptual issues that limit the expressiveness or applicability of our approach.

The presented approach enables developers of transformations to independently define their transformations and combine them afterwards, without the necessity to align the underlying consistency relations a priori or to check their compatibility manually when combining them. This is an important contribution to the overall goal of being able to build properly working networks of independently developed transformations to foster the development of large software and cyber-physical systems that involve several models and views to describe that system under construction.

Verfiability

The prototypical implementation of our approach, as well as the metamodels and transformations of the case studies, which we used for the evaluation presented in Section 7, can be found on GitHub [17].

References

[1] A. Anjorin. *Synchronization of Models on Different Abstraction Levels using Triple Graph Grammars*. PhD thesis, Technische Universität Darmstadt, 2014.

[2] A. Anjorin, S. Rose, F. Deckwerth, and A. Schürr. "Efficient model synchronization with view triple graph grammars". In *Modelling Foundations and Applications*. Volume 8569, LNCS, pages 1–17. Springer International Publishing, 2014.

[3] B. Azizi, B. Zamani, and S. Kolahdouz-Rahimi. *Contract verification of ETL transformations*. In *2017 7th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 154–160, 2017.

[4] C. Barrett, P. Fontaine, and C. Tinelli. *The SMT-LIB Standard: Version 2.6*. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[5] B. Beckert, U. Keller, and P. H. Schmitt. *Translating the Object Constraint Language into first-order predicate logic*. In *VERIFY Workshop (VERIFY 2002) at FLoC 2002: Federated Logic Conferences*, pages 113–123, 2002.

[6] D. Berardi, D. Calvanese, and G. D. Giacomo. "Reasoning on uml class diagrams". *Artificial Intelligence*, 168(1):70–118, 2005.

[7] G. Bergmann, I. Dávid, Á. Hegedüs, Á. Horváth, I. Ráth, Z. Ujhelyi, and D. Varró. "Viatra 3: a reactive model transformation platform". In *Theory and Practice of Model Transformations*, pages 101–110. Springer, 2015.

[8] F. Büttner, M. Egea, and J. Cabot. *On Verifying ATL Transformations Using 'off-the-shelf' SMT Solvers*. In *Model Driven Engineering Languages and Systems*, pages 432–448. Springer Berlin Heidelberg, 2012.

[9] J. Cabot, R. Clarisó, E. Guerra, and J. de Lara. "Verification and Validation of Declarative Model-to-Model Transformations through Invariants". *Journal of Systems and Software*, 83(2):283–302, 2010.

[10] A. Cicchetti, D. Di Ruscio, and R. Eramo. *Towards Propagation of Changes by Model Approximations*. In *2006 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06)*, page 24. IEEE Computer Society, 2006.

[11] A. Cleve, E. Kindler, P. Stevens, and V. Zaytsev. "Multidirectional Transformations and Synchronisations (Dagstuhl Seminar 18491)". *Dagstuhl Reports*, 8(12):1–48, 2019.

[12] J. S. Cuadrado, S. Guerra, and J. de Lara. "Static Analysis of Model Transformations". *IEEE Transactions on Software Engineering*, 43(9):868–897, 2017.

[13] K. Czarnecki and S. Helsen. *Classification of Model Transformation Approaches*. In *OOPSLA 2003 Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.

[14] K. Czarnecki and S. Helsen. "Feature-based Survey of Model Transformation Approaches". *IBM Systems Journal*, 45(3):621–645, 2006.

[15] C. R. Dantas, L. G. P. Murta, and C. M. L. Werner. *Consistent evolution of UML models by automatic detection of change traces*. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, pages 144–147, 2005.

[16] L. de Moura and N. Bjørner. *Z3: an efficient smt solver*. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer Berlin Heidelberg, 2008.

[17] *Decomposition GitHub Repository*. URL: `https://github.com/aurelienpepin/KIT_ConsistencyPreservation_Decomposition` (visited on 02/28/2020).

[18] Z. Diskin, H. König, and M. Lawford. *Multiple Model Synchronization with Multiary Delta Lenses*. In *Fundamental Approaches to Software Engineering*, pages 21–37. Springer International Publishing, 2018.

[19] R. Eramo, A. Pierantonio, J. R. Romero, and A. Vallecillo. *Change Management in Multi-Viewpoint System Using ASP*. In *Enterprise Distributed Object Computing Conference Workshops, 2008 12th*, pages 433–440, 2008.

[20] ETAS Group. *ASCET-DEVELOPER*. URL: `https://www.etas.com/ascet` (visited on 02/12/2020).

[21] N. E. Gibbs. "A cycle generation algorithm for finite undirected linear graphs". *Journal of the ACM (JACM)*, 16(4):564–568, 1969.

[22] H. Giese, S. Hildebrandt, and S. Neumann. "Model synchronization at work: keeping sysml and autosar models consistent". In *Graph Transformations and Model-Driven Engineering*. Vol-

ume 5765, LNCS, pages 555–579. Springer Berlin / Heidelberg, 2010.

[23] C. A. González, F. Büttner, R. Clarisó, and J. Cabot. *EMFtoCSP: A tool for the lightweight verification of EMF models*. In *2012 First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, pages 44–50, 2012.

[24] ITEA. *AMALTHEA4public – An Open Platform Project for Embedded Multicore Systems*. URL: `http://www.amalthea-project.org/`. Accessed 2020-02-12.

[25] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez. *ATL: A QVT-like Transformation Language*. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 719–720. ACM, 2006.

[26] H. Klare. *Multi-model Consistency Preservation*. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS 2018*, pages 156–161, 2018.

[27] H. Klare and J. Gleitze. *Commonalities for Preserving Consistency of Multiple Models*. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 371–378, 2019.

[28] H. Klare, T. Syma, E. Burger, and R. Reussner. "A Categorization of Interoperability Issues in Networks of Transformations". *Journal of Object Technology*, 18(3):4:1–20, 2019.

[29] H. König and Z. Diskin. *Efficient Consistency Checking of Interrelated Models*. In *Modelling Foundations and Applications*, pages 161–178. Springer International Publishing, 2017.

[30] A. Königs and A. Schürr. "MDI: A Rule-based Multi-document and Tool Integration Approach". *Software and Systems Modeling (SoSyM)*, 5(4):349–368, 2006.

[31] M. E. Kramer. *Specification Languages for Preserving Consistency between Models of Different Languages*. PhD thesis, Karlsruhe Institute of Technology (KIT), 2017. 278 pages.

[32] M. Kuhlmann, L. Hamann, and M. Gogolla. *Extensive Validation of OCL Models by Integrating SAT Solving into USE*. In *Objects, Models, Components, Patterns*, pages 290–306. Springer Berlin Heidelberg, 2011.

[33] A. Kusel, J. Etzlstorfer, E. Kapsammer, P. Langer, W. Retschitzegger, J. Schoenboeck, W. Schwinger, and M. Wimmer. *A Survey on Incremental Model Transformation Approaches*. In *ME 2013 – Models and Evolution Workshop Proceedings*, pages 4–13, 2013.

[34] K. Lano, S. Kolahdouz-Rahimi, I. Poernomo, J. Terrell, and S. Zschaler. "Correct-by-construction synthesis of model transformations using transformation patterns". *Software & Systems Modeling*, 13(2):873–907, 2014.

[35] L. Lúcio, S. Mustafiz, J. Denil, H. Vangheluwe, and M. Jukss. *FTG+PM: An Integrated Framework for Investigating Model Transformation Chains*. In *SDL 2013: Model-Driven Dependability Engineering*, pages 182–202. Springer Berlin Heidelberg, 2013.

[36] N. Macedo and A. Cunha. "Implementing QVT-R Bidirectional Model Transformations Using Alloy". In *Fundamental Approaches to Software Engineering*. Volume 7793, LNCS, pages 297–311. Springer Berlin Heidelberg, 2013.

[37] N. Macedo and A. Cunha. "Least-change bidirectional model transformation with QVT-R and ATL". *Software & Systems Modeling*, 15(3):783–810, 2016.

[38] N. Macedo, A. Cunha, and H. Pacheco. *Towards a framework for multi-directional model transformations*. In *3rd International Workshop on Bidirectional Transformations - BX*, volume 1133. CEUR-WS.org, 2014.

[39] N. Macedo, T. Guimarães, and A. Cunha. *Model Repair and Transformation with Echo*. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013)*, pages 694–697. IEEE Computer Society, 2013.

[40] N. Macedo, T. Jorge, and A. Cunha. "A Feature-based Classification of Model Repair Approaches". *IEEE Transactions on Software Engineering*, 43(7):615–640, 2017.

[41] U. Mansoor, M. Kessentini, P. Langer, M. Wimmer, S. Bechikh, and K. Deb. *MOMM: Multi-objective model merging*. In *Journal of Systems and Software*, volume 103, 2015.

[42] MathWorks. *Simulink – Simulation and Model-Based Design – MATLAB & Simulink*. URL: `https://www.mathworks.com/products/simulink.html` (visited on 02/12/2020).

[43] M. Mazkatli. *Consistency Preservation in the Development Process of Automotive Software*. Master's thesis, Karlsruhe Institute of Technology (KIT), 2016.

[44] M. Mazkatli, E. Burger, A. Koziolek, and R. H. Reussner. *Automotive systems modelling with vitruvius*. In *15. Workshop Automotive Software Engineering* (Chemnitz), volume P-275 of *Lecture Notes in Informatics (LNI)*, pages 1487–1498. GI, Bonn, 2017.

[45] Object Management Group (OMG). *Meta Object Facility (MOF) 2.0 – Query/View/Transformation Specification*. https://www.omg.org/spec/QVT/1.3/PDF, 2016.

[46] Object Management Group (OMG). *Model Driven Architecture (MDA) – MDA Guide rev. 2.0*. https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf, 2014.

[47] Object Management Group (OMG). *Object Constraint Language – Version 2.4*. http://www.omg.org/spec/OCL/2.4/PDF, 2014.

[48] Object Management Group (OMG). *OMG Meta Object Facility (OMG MOF^{TM}) – Version 2.5.1*. https://www.omg.org/spec/MOF/2.5.1/PDF, 2016.

[49] Object Management Group (OMG). *OMG System Modeling Language (OMG SysML^{TM}) – Version 1.6*. https://www.omg.org/spec/SysML/1.6/PDF, 2019.

[50] K. Paton. "An algorithm for finding a fundamental set of cycles of a graph". *Communications of the ACM*, 12(9):514–518, 1969.

[51] A. Pepin. *Decomposition of Relations for Multi-model Consistency Preservation*. Master's Thesis, Karlsruhe Institute of Technology (KIT), 2019.

[52] M. Petrenko, V. Rajlich, and R. Vanciu. *Partial Domain Comprehension in Software Evolution and Maintenance*. In *2008 16th IEEE International Conference on Program Comprehension*, pages 13–22, 2008.

[53] A. Reder and A. Egyed. *Incremental Consistency Checking for Complex Design Rules and Larger Model Changes*. In *Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS 2012)*, volume 7590 of *LNCS*, pages 202–218. Springer-Verlag, 2012.

[54] J. Rubin and M. Chechik. *N-way model merging*. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*, 2013.

[55] R. Salay, S. Kokaly, A. Di Sandro, and M. Chechik. *Enriching megamodel management with collection-based operators*. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 236–245, 2015.

[56] R. Salay, J. Mylopoulos, and S. Easterbrook. *Managing Models through Macromodeling*. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 447–450, 2008.

[57] L. Samimi-Dehkordi, B. Zamani, and S. Kolahdouz-Rahimi. *Bidirectional Model Transformation Approaches – A Comparative Study*. In *2016 6th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 314–320, 2016.

[58] J. Sánchez Cuadrado and J. García Molina. *Approaches for Model Transformation Reuse: Factorization and Composition*. In *Theory and Practice of Model Transformations*, pages 168–182. Springer Berlin Heidelberg, 2008.

[59] O. Scheid. *AUTOSAR Compendium - Part 1: Application and RTE*. AUTOSAR - Compendium Series. CreateSpace Independent Publishing Platform, 2015.

[60] A. Schürr. "Specification of graph translators with triple graph grammars". In *Graph-Theoretic Concepts in Computer Science*. Volume 903, LNCS, pages 151–163. Springer Berlin Heidelberg, 1995.

[61] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF - Eclipse Modeling Framework*. Addison-Wesley Professional, 2nd edition, 2009.

[62] P. Stevens. *Bidirectional transformations in the large*. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 1–11, 2017.

[63] P. Stevens. "Bidirectional model transformations in QVT: semantic issues and open questions". *Software & Systems Modeling*, 9(1):7–20, 2010.

[64] P. Stevens. *Towards sound, optimal, and flexible building from megamodels*. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 301–311. ACM, 2018.

[65] P. Stünkel, H. König, Y. Lamo, and A. Rutle. *Multimodel Correspondence Through Inter-model Constraints*. In *Conference Companion of the 2Nd International Conference on Art, Science, and Engineering of Programming*, Programming'18 Companion, pages 9–17. ACM, 2018.

[66] R. Tarjan. "Depth-first search and linear graph algorithms". *SIAM journal on computing*, 1(2):146–160, 1972.

[67] F. Trollmann and S. Albayrak. *Extending Model Synchronization Results from Triple Graph Grammars to Multiple Models*. In *Proceedings of the 9th International Conference on Theory and Practice of Model Transformations*, pages 91–106. Springer International Publishing, 2016.

[68] F. Trollmann and S. Albayrak. *Extending Model to Model Transformation Results from Triple Graph Grammars to Multiple Models*. In *Proceedings of the 8th International Conference on Theory and Practice of Model Transformations*, pages 214–229. Springer International Publishing, 2015.

[69] A. Vallecillo, M. Gogolla, L. Burgueño, M. Wimmer, and L. Hamann. "Formal Specification and Testing of Model Transformations". In *Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures*, pages 399–437. Springer Berlin Heidelberg, 2012.

[70] B. Vanhooff, D. Ayed, S. Van Baelen, W. Joosen, and Y. Berbers. *UniTI: A Unified Transformation Infrastructure*. In *Model Driven Engineering Languages and Systems*, pages 31–45. Springer Berlin Heidelberg, 2007.

[71] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjorner. "Symbolic finite state transducers: algorithms and applications". *SIGPLAN Not.*, 47(1):137–150, 2012.

[72] D. Wagelaar. *Composition Techniques for Rule-Based Model Transformation Languages*. In *Theory and Practice of Model Transformations*, pages 152–167. Springer Berlin Heidelberg, 2008.

[73] D. Wagelaar, M. Tisi, J. Cabot, and F. Jouault. *Towards a General Composition Semantics for Rule-Based Model Transformation*. In *Model Driven Engineering Languages and Systems*, pages 623–637. Springer Berlin Heidelberg, 2011.

[74] D. Wagelaar, R. Van Der Straeten, and D. Deridder. "Module superimposition: a composition technique for rule-based model transformation languages". *Software & Systems Modeling*, 9(3):285–309, 2010.

[75] C. Wolff, L. Krawczyk, R. Höttger, C. Brink, U. Lauschner, D. Fruhner, E. Kamsties, and B. Igel. *AMALTHEA – Tailoring tools to projects in automotive software development*. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 2, pages 515–520, 2015.

[76] Y. Xiong, D. Liu, Z. Hu, H. Zhao, M. Takeichi, and H. Mei. *Towards Automatic Model Synchronization from Model Transformations*. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 164–173. ACM, 2007.