Heterogeneous verification in the context of model driven engineering

Daniel Calegari\textsuperscript{a,}\textsuperscript{*}, Till Mossakowski\textsuperscript{b}, Nora Szasz\textsuperscript{c}

\textsuperscript{a} Facultad de Ingeniería, Universidad de la República, 11300 Montevideo, Uruguay  
\textsuperscript{b} Otto-von-Guericke University Magdeburg, 39106 Magdeburg, Germany  
\textsuperscript{c} Facultad de Ingeniería, Universidad ORT Uruguay, 11100 Montevideo, Uruguay

\section*{A R T I C L E   I N F O}

Article history:  
Received 1 May 2015  
Received in revised form 17 February 2016  
Accepted 19 February 2016  
Available online 26 February 2016

Keywords:  
Verification  
Formal methods  
Model-Driven Engineering  
Theory of Institutions  
Heterogeneous Tool Set

\section*{A B S T R A C T}

In some cases it may be useful to represent a problem in many logical domains, since they provide different perspectives for addressing formal verification. However, the maintenance of multiple representations in separate domains can be expensive if there is neither automated assistance nor a clear formal relation between these domains. We have addressed this problem in the context of Model-Driven Engineering (MDE). We defined solid foundations of a theoretical environment for formal verification using heterogeneous verification approaches. The environment is based on the Theory of Institutions which provides a sound basis for representing MDE elements and a way for specifying translations from these elements to other domains used for verification. In this paper we present how this environment can be supported in practice within the Heterogeneous Tool Set (Hets). Hets supports heterogeneous specifications and provides capabilities for monitoring the overall correctness of a heterogeneous proof. We first extend the theoretical environment with the inclusion of an institution for the Object Constraint Language (OCL), and then we define semantic-preserving translations from the OCL-constrained MDE elements to a core language of Hets. With this we can verify basic properties of our specification, and then use the existent connections between logical domains within Hets for broadening the spectrum of domains in which complementary verification properties can be addressed.

© 2016 Elsevier B.V. All rights reserved.

\section*{1. Introduction}

Software development costs may be reduced by adopting a model-centric approach in which different views of the system to be constructed are provided by models. Models are abstractions of the system to be built (or some aspects of it) and also allow us to deal with its intrinsic complexity in a simplified manner. Moreover, the use of automated mechanisms (model transformations) in which models are transformed from higher abstraction levels until an executable system is built, may improve efficiency on the whole process. The Model-Driven Engineering (MDE, [1]) paradigm is based on these practices, also encompassing other engineering efforts, such as maintenance and reverse engineering.

Every model (from now on SW-model to avoid conflicts) conforms to a metamodel which introduces the syntax and semantics of certain kind of SW-models. Sometimes if there are conditions that cannot be captured by the structural rules of this language, another constraints language must be used to specify them. These considerations imply defining conformance

\textsuperscript{*} Corresponding author. Tel.: +598 27114244x1125; fax: +598 27110469.  
E-mail addresses: dcalegari@fing.edu.uy (D. Calegari), mossakow@wvs.cs.uni-magdeburg.de (T. Mossakowski), szasz@ort.edu.uy (N. Szasz).

http://dx.doi.org/10.1016/j.scico.2016.02.003  
0167-6423/© 2016 Elsevier B.V. All rights reserved.
in terms of structural and semantical (or non-structural) conformance [2]. An SW-model is structurally conformant with respect to a metamodel if it is well-typed with respect to the metamodel and it also satisfies the multiplicity constraints. Semantical conformance requires, in addition to structural conformance, that the SW-model satisfies the invariants specified using the supplementary constraints language. A model transformation (or just transformation from now on) basically takes as input an SW-model conforming to certain metamodel and produces as output another SW-model conforming to its metamodel (possibly the same one).

Formal verification is often aided by a separation of duties between software developers. In general terms, MDE experts define models and transformations, while formal experts conduct the verification process, often aided by some (semi)automatic generation process which translates the MDE elements to their formal representation into the domain used for verification purposes. Nevertheless, there are multiple properties that can be verified, as well as a plethora of verification approaches with different objectives, formalisms and supporting tools [3]. Moreover, in some cases it may be useful to have a representation of MDE elements in different logical domains.

In [4,5] we proposed a theoretical environment for the formal verification of different MDE aspects using heterogeneous verification approaches [6]. The environment, based on the theory of institutions [7], proposes a generic representation of the syntax and semantics of the MDE elements by means of institutions. Although the environment can be potentially formalized for any transformation approach and language, our proposal is aligned with the MetaObject Facility (MOF, [8]), i.e. a standard language for metamodeling, and the Query/View/Transformation Relations (QVT-Relations, [9]), i.e. a relational language which defines transformation rules as mathematical relations between source and target elements. We defined institutions CSMOF for the structural conformance relation between SW-models and MOF-based metamodels, and CVTR for QVT-Relations model transformation. The theory allows the definition of semantic-preserving translations (co-morphisms) between semantic domains defined as institutions and re-use their entailment systems for developing proofs. The definition of comorphisms not only provides a way to choose the formalism in which formal experts are more skilled to address a formal proof, but also allows supplementing the former specification of MDE elements with additional properties using the target logic. To the extent that there are many logics connected through comorphisms, the capabilities of the environment increase.

The aim of this paper is to present how the environment can be supported in practice using the Heterogenous Tool Set (HETS, [6,10]), which is meant to support heterogeneous multi-logic specifications. It also provides proof management capabilities for monitoring the overall correctness of a heterogeneous specification whereas different parts of it are verified using (possibly) different semantic domains.

Semantical conformance is not addressed by the original CSMOF institution. Thus, as a proof of concepts of the whole environment, we first define an institution for a small subset of the Object Constraint Language (OCL, [11]), which is used as a constraint navigational expression language with MOF and QVT-Relations languages. The OCL and CSMOF institutions are intended for metamodeling only and not for the definition of software systems. To do the later, pre and post condition constraints must be introduced in the OCL institution as well as the concept of methods in CSMOF. We then define how MDE elements can be integrated into HETS by defining semantic-preserving translations to the Common Algebraic Specification Language (Casl, [12]), which is a core language of HETS. The existent connections between Casl and other formalisms broadens the spectrum of formal domains in which verification can be addressed. We also detail the implementation of a prototype which allows us to specify MDE elements, supplement them with multi-logic properties, and perform a heterogeneous verification.

This paper is a substantially extended and thoroughly revised version of [13], which is a continuation of the formal foundations introduced in [5]. Additional material includes:

- the definition of an institution for a subset of OCL to be used within our environment;
- a detailed formalization of the semantic-preserving translations to Casl, together with a running example;
- a deeper discussion about the scope of verification within the environment and of related work.

The remainder of the paper is structured as follows. In Section 2 we present the basic notions of the Theory of Institutions which will be useful for the theoretical understanding of the following sections. For a more detailed introduction to the topic refer to [7,14]. In Section 3 we briefly introduce the institution CSMOF for the structural conformance relation between SW-models and MOF-based metamodels, as defined in [5], and, based on it, we define an institution for OCL to be used within our environment. In Section 4 we briefly introduce the institution QVTR for QVT-Relations model transformation, as defined in [5] together with the use of the OCL institution. In Section 5 and Section 6 we present the semantic-preserving translation from the MDE elements into Casl. In Section 7 we give details about the implementation of our environment within HETS and discuss the capabilities of our environment. Finally, in Section 8 we discuss related work and in Section 9 we present some conclusions and future work.

2. Institutions and their (co)morphisms

The notion of Institution [7] relies on Category Theory [15] for formalizing the notion of “logical system”. It consists of vocabularies (called signatures) for constructing sentences in a logical system. A model (also called interpretation) provides semantics by assigning interpretations to the elements in the signature. A change of interpretation is done by the notion of
homomorphism, which consists of a mapping of elements between two models. Institutions also define formal translations (called signature morphisms) between signatures, allowing many different vocabularies at once. Since signatures can change through signature morphisms, we need to translate sentences and models accordingly. Sentences are translated along signature morphisms since symbols must be replaced in each sentence conforming to the signature morphism. In the case of models, they are translated in the opposite direction of signature morphisms, i.e. a model providing semantics to the target signature of a signature morphism is reduced to a model providing semantics to the source signature (target signatures potentially have more elements than source signatures). Finally, there is a satisfaction relation of sentences by models, such that when a signature is changed (by a signature morphism), satisfaction of sentences by models changes consistently.

**Definition 1 (Institution).** An institution (as defined in [14]) consists of:

1. a category $\mathbf{Sign}$ of signatures;
2. a functor $\text{Sen} : \mathbf{Sign} \to \mathbf{Set}$, giving a set $\text{Sen}(\Sigma)$ of $\Sigma$-sentences for each signature $\Sigma \in |\mathbf{Sign}|$ and a function $\text{Sen}(\sigma) : \text{Sen}(\Sigma_1) \to \text{Sen}(\Sigma_2)$ translating $\Sigma_1$-sentences to $\Sigma_2$-sentences for each signature morphism $\sigma : \Sigma_1 \to \Sigma_2$;
3. a functor $\text{Mod} : \text{Sign}^{\text{op}} \to \mathbf{Cat}$ (called reduct), giving a category $\text{Mod}(\Sigma)$ of $\Sigma$-models for each signature $\Sigma \in |\mathbf{Sign}|$ and a functor $\text{Mod}(\sigma) : \text{Mod}(\Sigma_2) \to \text{Mod}(\Sigma_1)$ translating $\Sigma_2$-models to $\Sigma_1$-models (and $\Sigma_2$-morphisms to $\Sigma_1$-morphisms) for each signature morphism $\sigma : \Sigma_1 \to \Sigma_2$;
4. for each signature $\Sigma \in |\mathbf{Sign}|$, a satisfaction relation $\vdash_\Sigma \subseteq |\text{Mod}(\Sigma)| \times \text{Sen}(\Sigma)$;

such that for any signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ the translation $\text{Mod}(\sigma)$ of models and $\text{Sen}(\sigma)$ of sentences preserve the satisfaction relation, that is, for any $\varphi \in \text{Sen}(\Sigma_1)$ and $M_2 \in |\text{Mod}(\Sigma_2)|$:

$$M_2 \vdash_{\Sigma_2} \text{Sen}(\sigma)(\varphi) \iff \text{Mod}(\sigma)(M_2) \vdash_{\Sigma_1} \varphi$$

In an arbitrary but fixed institution we can define a notion of *theory*, i.e. a pair $\mathcal{T} = \langle \Sigma, \Psi \rangle$ consisting of a signature $\Sigma$ and an arbitrary set $\Psi$ of axioms, which are $\Sigma$-sentences. Moreover, it is possible to define a *theory morphism* $\phi : \langle \Sigma_1, \Psi_1 \rangle \to \langle \Sigma_2, \Psi_2 \rangle$ as a signature morphism $\phi : \Sigma_1 \to \Sigma_2$ for which $\Psi_2 \vdash_{\Sigma_2} \phi(\Psi_1)$, i.e. a signature morphism that maps axioms to logical consequences. With obvious identities and composition we obtain a category $\mathbf{Th}$ of theories and theory morphisms. It is possible to extend $\text{Sen}$ and $\text{Mod}$ to start from the category $\mathbf{Th}$ of theories by putting $\text{Sen}(\langle \Sigma, \Psi \rangle) = \text{Sen}(\Sigma)$ and letting $\text{Mod}^{\text{TH}}(\langle \Sigma, \Psi \rangle)$ be the full subcategory of $\text{Mod}(\Sigma)$ induced by the class of those models satisfying $\Psi$. In this way, we get the institution of theories $\mathbf{Th} = (\text{Th}, \text{Sen}, \text{Mod}^{\text{TH}}, \vdash)$ over an institution [7].

In order to use an institution for verification purposes, there are two alternatives. The first one is to define a logic, i.e. an institution with an *entailment system* $\vdash$, that is a relation between sentences capturing logical consequence. Entailment is typically defined via a system of finitary derivation rules, giving a notion of proof that is absent when the institution is considered on its own. The second alternative is to translate the institution into another logic which has its own entailment system. Two institutions may be related through institution *morphism* [16] which come in several flavors, e.g. the so-called institution comorphisms.

**Definition 2 (Institution comorphism).** Given arbitrary institutions $I = \langle \mathbf{Sign}^I, \text{Sen}^I, \text{Mod}^I, \vdash^I \rangle$ and $J = \langle \mathbf{Sign}^J, \text{Sen}^J, \text{Mod}^J, \vdash^J \rangle$, an institution comorphism $\rho : I \to J$ consists of:

- a functor $\rho^\mathbf{Sign} : \mathbf{Sign}^I \to \mathbf{Sign}^J$;
- a natural transformation $\rho^\text{Sen} : \text{Sen}^I \Rightarrow \rho^\mathbf{Sign} ; \text{Sen}^J$;
- a natural transformation $\rho^\text{Mod} : (\rho^\mathbf{Sign})^{\text{op}} ; \text{Mod}^J \Rightarrow \text{Mod}^I$;

such that for any signature $\Sigma \in |\mathbf{Sign}^J|$ the translations $\rho^\text{Sen}_\Sigma$ of sentences, and $\rho^\text{Mod}_\Sigma$ of $J$-models, preserve the satisfaction relation, that is, for any $\varphi \in \text{Sen}^J(\Sigma)$ and $M \in \text{Mod}^J(\rho^\mathbf{Sign}(\Sigma))$:

$$M \vdash^J \rho^\text{Sen}_\Sigma(\varphi) \iff \rho^\text{Mod}_\Sigma(M) \vdash^I \varphi$$

The functor $\rho^\mathbf{Sign}$ translates signatures and morphisms from one institution into the other. The natural transformation $\rho^\text{Sen}$ translates sentences from one signature to the other in the same direction as the translation of signatures (as happens with signature morphisms). Moreover, the natural transformation $\rho^\text{Mod}$ translates models from one institution into the other in the opposite direction (as happens with models of an institution). The satisfaction condition of a comorphism states that a translated model satisfies a sentence in the source institution only if the original model satisfies the translated sentence.

---

1 $|C|$ is the collection of objects of a category $C$.
2 $\mathbf{Sign}^{\text{op}}$ is the opposite category of the category $\mathbf{Sign}$. $\mathbf{Cat}$ is the category of all categories. In order to avoid foundational difficulties, $\mathbf{Cat}$ needs to live in a set-theoretic universe that is higher than that of the categories it contains.
into the other institution. There are other variants of comorphisms. A simple theoroidal comorphism [16] from \( \mathcal{I} \) to \( \mathcal{J} \) is a comorphism from \( \mathcal{I} \) to \( \mathcal{J}^{th} \), where \( \mathcal{J}^{th} \) is the institution of theories over \( \mathcal{J} \).

The importance of comorphisms is the possibility (in some cases) to re-use the entailment systems of an institution into another one via a comorphism. This is possible thanks to the borrowing technique [17].

**Definition 3 (Borrowing of entailment).** Let \( \mathcal{I} \) and \( \mathcal{J} \) be two institutions, \( \rho = (\rho_{\text{Sign}}, \rho_{\text{Sen}}, \rho_{\text{Mod}}) : \mathcal{I} \rightarrow \mathcal{J} \) an institution comorphism, and \( \mathcal{T} \) a class of \( \mathcal{I} \)-theories. We say that \( \rho \) admits borrowing of entailment for \( \mathcal{T} \), if for any theory \( T = (\Sigma, \Psi) \in \mathcal{T} \) and any \( \Sigma \)-sentence \( \varphi \in \mathcal{I} \), we have

\[
\Psi \models_{\Sigma} \varphi \text{ iff } \rho_{\text{Sen}}(\Psi) \models_{\text{Sign}(\rho_{\text{Sen}}(\Sigma))} \rho_{\text{Sen}}(\varphi).
\]

**Theorem 1.** Let \( \mathcal{I} \) and \( \mathcal{J} \) be two institutions and \( \rho = (\rho_{\text{Sign}}, \rho_{\text{Sen}}, \rho_{\text{Mod}}) : \mathcal{I} \rightarrow \mathcal{J} \) an institution comorphism admitting model expansion. Then \( \rho \) admits borrowing of entailment for theories.

A comorphism admits model expansion if \( \rho_{\text{Mod}} \) is point-wise surjective on objects, i.e. each model of the source institution has a model representing it in the target institution. If the comorphism does not admit model expansion, then the proof calculus of the target institution can be borrowed only for disproving entailment (not for proving it).

In a word, if we have a sound (and complete) theorem prover for theories in the target institution of an institution comorphism admitting model expansion, we can re-use it as a sound (and complete) theorem prover for theories in the source institution, we just have to translate our proof goals.

2.1. Common Algebraic Specification Language

The Common Algebraic Specification Language (Casl, [12]) is a general-purpose specification language which plays an important role in this work. Thus, we briefly define the institution underlying Casl, which is sub-sorted partial first-order logic with equality and sort generation constraints SubPCFOL\(^{cc}\).

Signatures are many-sorted signatures enriched with predicate and function symbols of the form \( (S, \mathcal{T}F, \mathcal{P}F, P, \leq_S) \) where \( S \) is a set of sort names, \( \mathcal{T}F \) and \( \mathcal{P}F \) are disjoint families of total and partial function symbols, respectively, \( P \) is a family of predicate symbols, and \( \leq_S \) is a reflexive and transitive sort relation (embedding) on the set \( S \) of sorts. Signature morphisms consist of maps taking sort, function and predicate symbols to a symbol of the same kind, and they must preserve subsorting, typing of function and predicate symbols, and totality of function symbols.

Sentences are the usual partial many-sorted first-order logic formulas together with sort generation constraints. Many-sorted first-order logic formulas are built out of atomic formulas using the standard propositional connectives \( (\land, \lor, \Rightarrow, \Leftrightarrow, \neg) \) and quantifiers \( (\forall, \exists) \). The atomic formulas are applications of qualified predicate symbols to argument terms (variables or application of functions) of appropriate sorts, assertions about the definedness of fully-qualified terms, or existential \( t \equiv t' \) and strong \( t \equiv t' \) equations between fully-qualified terms of the same sort, the logical constants \( \text{true} \) and \( \text{false} \), and substort membership tests \( t \in s \). Sort generation constraints are triples \( (S', F', \sigma') \) such that \( \sigma' : \Sigma' \rightarrow \Sigma \) and \( S' \) and \( F' \) are respectively sort and function symbols of \( \Sigma' \), stating that a given set of sorts is generated by some set of functions. Formulas are translated along a signature morphism by replacing symbols.

Models are many-sorted first-order structures, i.e. consisting of a non-empty carrier set \( |M|_s \) for each sort name \( s \in S \), a partial function (or total) \( f_M \) for each function symbol \( f \in \mathcal{P}F \) (or \( \mathcal{T}F \)), and a relation \( p_M \) for each predicate symbol \( p \in P_w \), \( w \in S^w \), satisfying some axioms with respect to embedding, projection, and membership. Homomorphisms between models \( M \) and \( M' \) consist of a function \( h_s : |M|_s \rightarrow |M'|_s \) for each \( s \in S \) preserving values of functions, their definedness, and the truth of predicates. Reducts are defined by interpreting symbols of the signature in the reduct in the same way as their images under the signature morphism are interpreted.

Finally, the satisfaction relation is basically the usual satisfaction of a partial first-order formula in a first-order structure. An existential equation holds if both terms are defined and have the same interpretation; a strong equation also holds if both terms are undefined. A sort generation constraint holds in a model \( M \) if the carriers of the reduct of \( M \) of the generated sorts are generated by function symbols.

3. OCL-constrained metamodels

In [5] we defined an institution Csmof for the structural conformance relation between SW-models and metamodels specified with a simplified version of MOF. However, semantical conformance is not addressed by this institution. For this purpose, the Object Constraint Language (OCL, [11]) is commonly used. Moreover, OCL can be used for expressing constraints on QVT-Relations transformation rules, as well as for computing object values to be used in the definition of a transformation. Based on these interests, in what follows we define an institution Ocl for a very basic subset of OCL. We also present the main components of the institution Csmof in which the institution Ocl is based. For a complete definition and examples, please refer to [5].
3.1. The Institution CSMOF

Metamodels typically define syntax and (static) semantics of modeling languages like UML. This induces a notion of a model being conformant to its metamodel. Any MOF-based metamodel can be basically described as a set of classes which can belong to a hierarchical structure. Some of them may be defined as abstract (they do not have instances of their own). Any class has properties which can be attributes (named elements with an associated type which can be a primitive type or another class) and associations (relations between classes in which each class plays a role within the relation). Every property has a multiplicity which constrains the number of elements that can be related through the property, and it can be related with another property (known as its opposite) if the property corresponds to a bidirectional association between two classes.

**Example 1.** The metamodel in Fig. 1a defines a reduced but expressive version of UML-like class diagrams. Classifiers (classes and primitive types as string, boolean, integer, etc.) are contained in packages (association contains). Classes can contain attributes (association has) and may be declared as persistent (kind = ‘Persistent’), whilst attributes have a type that is a primitive type (association typeOf). The SW-model in Fig. 1c is structurally conformant to the metamodel depicted in Fig. 1a. It is composed by a persistent class of name ID within a package of name Package. The class has an attribute of name value and type String which is a primitive type. On the other hand, the relational SW-model in Fig. 1d is structurally conformant to the Relational diagrams metamodel depicted in Fig. 1b. Every schema contains a number of tables and each table has a number of columns. Each column has a name and a kind, and can be the primary key of the corresponding table.3

A signature in the CSMOF institution represents a metamodel, i.e. it defines hierarchical related classes, primitive types and type constructors. We consider a fixed set of primitive types and type constructors similar to those defined for the OCL [11].

**Definition 4 (CSMOF signature).** A CSMOF signature \( \Sigma = (C, \alpha, P) \) declares:

---

3 The SW-model in Fig. 1c can be expressed using UML class diagrams’ concrete syntax. However, we use the notation of object diagrams to stress the fact that it conforms to the metamodel in Fig. 1a.
A finite class hierarchy \( C = (C, \leq_C) \) where \( C \) is a set of class names, and \( \leq_C \subseteq C \times C \) is the subclass (inheritance) relation, extended with a subset \( \alpha \subseteq C \) denoting abstract classes. By \( T(C) \) we denote the type extension of \( C \) by primitive types (e.g., Boolean and String) and type constructors (e.g., List and Set).

A pair \( P = (R, P) \) representing property declarations (attributes and associations), where \( R \) is a finite set of role names with a default role name \("\)", and \( P \) is a finite set of properties of the form \((r_1 : c_1, r_2 : c_2)\) with \( r_1, r_2 \in R, c_1, c_2 \in T(C) \), such that for any class or type name \( c \in T(C) \), the role names of the properties in which \( c \leq_T C \) is involved are all different, i.e., if \((r_1 : c_1, r_2 : c_2)\) and \((s_1 : d_1, s_2 : d_2)\) are properties in \( P \) and \( c_k = d_j \in T(C) \), then \( r_i \neq s_j \) for any \( i \neq k \) and for any \( j \neq l \) (\( 1 \leq i \leq 2, 1 \leq j \leq 2 \)).

Any property declaration \((r_1 : c_1, r_2 : c_2) \in P\) represents a MOF property and its opposite, such that the type \( c_i \) attached to the role \( r_i \) represents the type of the property, as well the type in the opposite side represents its owned class. The default role name \("\)" is used if a property has no opposite.

Formulas represent multiplicity constraints, i.e. determining how the number of elements in a property end is bounded (upper and/or lower).

**Definition 5** (Csmor formula). Given a signature \( \Sigma = (C, \alpha, P) \) with \( C = (C, \leq_C) \) and \( P = (R, P) \), a \( \Sigma \)-formula representing a multiplicity constraint is defined by the following grammar:

\[
\Phi ::= \#\Pi = N \mid N \leq \#\Pi \mid \#\Pi \leq N \\
\Pi ::= C \cdot R
\]

We use \( \cdot \) as an operator representing the selection of the elements linked with an element of class \( c \in C \) through role \( r \in R \); there must exist a property \([r^c : c, r : d]\) or \([r : d, r^c : c]\) in \( P \). The \#-expression refers to the number of these elements.

**Example 2.** From the class metamodel in Fig. 1a we derive the signature \( \Sigma = (C, \alpha, P) \) with \( C = (C, \leq_C) \) and \( P = (R, P) \), such that:

\[
C = \{\text{UMLModelElement}, \text{Package}, \text{Classifier}, \text{PrimitiveDataType}, \text{Attribute}, \text{Class}\} \\
\leq_C = \{\text{Package} \leq_C \text{UMLModelElement}, \text{Classifier} \leq_C \text{UMLModelElement}, \text{Class} \leq_C \text{Classifier}, \text{PrimitiveDataType} \leq_C \text{Classifier}\} \\
\alpha = \{\text{UMLModelElement}\} \\
R = \{\text{namespace}, \text{elements}, \text{type}, \text{owner}, \text{attribute}, \text{name}, \text{kind}\} \\
P = \{[(\text{namespace} : \text{Package}, \text{elements} : \text{Classifier}), (\_, \text{UMLModelElement}, \text{name} : \text{String}), (\_, \text{UMLModelElement}, \text{kind} : \text{String}), (\text{attribute} : \text{Attribute}, \text{owner} : \text{Class}), (\_, \text{Attribute}, \text{type} : \text{PrimitiveDataType})]\}
\]

Moreover, we can express the multiplicity constraints as formulas, e.g. any model element must have exactly one name: \(#(\text{UMLModelElement} \cdot \text{name}) = 1\), and any attribute must have exactly one type: \(#(\text{Attribute} \cdot \text{type}) = 1\).

An interpretation (or model) provides a semantic representation for an SW-model, i.e. objects and links.

**Definition 6** (Csmor interpretation). Given a signature \( \Sigma = (C, \alpha, P) \) with \( C = (C, \leq_C) \) and \( P = (R, P) \), a \( \Sigma \)-interpretation \( \mathcal{I} \) consists of a tuple \( (V^I_C(O), A) \) where

- \( V^I_C(O) = (V_C)_{C \in T(C)} \) is the value extension of a \( C \)-object domain \( O \) by primitive values and value constructions where the classes in \( C \) must be interpreted as finite sets.
- \( A \) contains a relation \( (r_1 : c_1, r_2 : c_2)^I \subseteq V_{c_1} \times V_{c_2} \) for each relation name \( (r_1 : c_1, r_2 : c_2) \in P \) with \( c_1, c_2 \in T(C) \)
- \( c_2 \in \alpha \) implies \( O_{c_2} = \bigcup_{r_1 \leq c_2} O_{c_1} \)

**Example 3.** We can define an interpretation \( \mathcal{I} \) representing the SW-model in Fig. 1c. This interpretation has one element for each type in the signature, e.g. \( V_{\text{Class}} = \{c\} \) and \( V_{\text{Attribute}} = \{a\} \). It also contains a set of relations representing object links, one for each property in the signature, e.g. \( \mathcal{L}_{\text{UMLModelElement}, \text{name} : \text{String}}^I = \{(p, \text{Package}), (c, \text{ID}), (\text{pdt}, \text{String}), (a, \text{value})\} \) and \( \mathcal{L}_{\text{Attribute}, \text{type} : \text{PrimitiveDataType}}^I = \{(a, \text{pdt})\} \).

We must express that an SW-model conforms to a metamodel if it is well-typed and it also satisfies its multiplicity constraints. Well-typing holds by construction, since the interpretation representing an SW-model respects the signature which defines types within the metamodel, i.e. every object in the SW-model has a type defined within the metamodel (signature) and every link has a corresponding property, such that objects connected with such property link are well-typed according to the corresponding property. The satisfaction of multiplicity constraints (formulas) by an SW-model (interpretation) is thus the main concern of the satisfaction relation.
Definition 7 (Csmof satisfaction relation). Given a signature $\Sigma = (C, \alpha, P)$ with $C = (C, \leq C)$ and $P = (R, P)$, a $\Sigma$-formula $\phi$ representing a multiplicity constraint and a $\Sigma$-interpretation $I$, the interpretation satisfies $\phi$, if one of the following holds:

- $\phi$ is $\#(c \cdot r) = n$ and $|S| = n$ for all $S \in (c \cdot r)^I$
- $\phi$ is $n \leq \#(c \cdot r)$ and $n \leq |S|$ for all $S \in (c \cdot r)^I$
- $\phi$ is $\#(c \cdot r) \leq n$ and $|S| \leq n$ for all $S \in (c \cdot r)^I$

This means that for any object of class $c$, the number of elements within $I$ related through the role $r$ (of a property of the class $c$) satisfies the multiplicity constraints. With $(c_1 \cdot r_1)^I$ we denote the evaluation of an expression $c_1 \cdot r_1$ with respect to the interpretation $I$, which is defined as $\{t \in \langle r_1 : c_1, r_2 : c_2 \rangle^I \mid \pi(t) = o \mid o \in V_{c_i} \} (i = 1, 2)$.

Example 4. Back to the example, we can easily check that $I \models \Sigma \phi$ for every formula $\phi$ representing a multiplicity constraints defined before. As an example, we have that $\#(\text{UMLModelElement} \cdot \text{name}) = 1$ and $|S| = 1$ for all $S \in (\text{UMLModelElement} \cdot \text{name})^I = \{(p, \text{Package}), (c, \text{ID}), (\text{pdt}, \text{String}), (\text{a}, \text{value})\}$.

3.2. An Institution for OCL

Beyond the notion of truth or falsity of an expression there exists the notion of undefinedness, and in the last version of OCL [11] another element was added (null), thus OCL constitutes a many-valued logic. In order to represent the valuation of an OCL expression, we need to define how to handle the satisfaction condition of our institution. As discussed in [18] it is possible to define a generic institution dealing with this complexity in an abstract manner, which can be instantiated with the evaluation domain of interest. This means that for the valuation of OCL constraints the boolean domain can be used, giving a traditional satisfaction condition which states that an expression is satisfied if it evaluates to true, and it does not in any other case (i.e. evaluates to false or it is undefined). For invariants and model transformations we do not need such complex formal settings. As discussed in [19] the use of explicit null and invalid values in OCL complicate the logic of OCL and of transformation languages that use OCL, making it difficult to provide effective verification support for these languages. In this sense, the authors propose an alternative approach in which expressions are ensured by their context to be well-defined (not null or invalid) and determinate in value (null or invalid values should not be used).

Based on these considerations, we defined the institution $\text{Oct}$ for a very basic but powerful subset of OCL. As in [19], we impose some restrictions to the language which ensure that our institution is based on classical two-valued logic. The satisfaction condition is stated in terms of the formal semantics for OCL in [20]. The institution uses Csmof signatures and interpretations for representing metamodels and SW-models, respectively, and adds OCL expressions as formulas referencing elements within the metamodel (signature).

Although the institution is rather limited, it serves for a proof of concepts of the whole environment. A more complete OCL institution also addressing null and undefined values directly, e.g. along the lines of Featherweight OCL as encoded recently in Isabelle/HOL [21], is considered future work.

Definition 8 (OCL formula). Given a signature $\Sigma = (C, \alpha, P)$ with $C = (C, \leq C)$ and $P = (R, P)$, and a set of variables $X = (X^i)_{i \in T(C)}$, there are two kinds of OCL formulas:

- a formula context $T$ inv: $Expr$ representing that $Expr$ must hold in the context of any element of type $T \in C$
- a formula $\{x_j \mid 1 \leq j \leq n \mid x_p = Expr \text{ with } 1 \leq p \leq n \text{ and } x_j \in X$, representing the equality between the value of the variable $x_p$ within the set of variables $\{x_j\}$ and the valuation of $Expr$ (which may use those variables in $\{x_j\}$)

Formulas of the first type are used standalone when expressing constraints on metamodels, whereas formulas of the second type play an important role within model transformations, as explained in Section 4. The language of expressions $Expr$ is recursively defined by the following grammar.

\[
\text{Expr ::= Literal | self | Var | } \\
\text{Expr and Expr | Expr or Expr | Expr implies Expr | not Expr | } \\
\text{Expr < Expr | Expr > Expr | Expr = Expr | Expr <> Expr | } \\
\text{Expr + Expr | if Expr then Expr else Expr endif | } \\
\text{Type.allInstances() | Expr.Name | } \\
\text{Expr -> exists(TVars || Expr) | Expr -> forAll(TVars || Expr) | } \\
\text{Expr -> collect(TVars || Expr) | Expr -> select(TVars || Expr) | Expr -> size() | } \\
\text{Expr -> includes(Var) | Expr -> excluding(Var) | Expr -> including(Var) | } \\
\text{TVars ::= Var | Type | Var [: Type | TVars | Literal | BooleanLiteral | StringLiteral}}
\]

Our expressions syntax allows to represent integer, boolean and string literal values, variables $\text{Var}$ for existent types, boolean connectives (and, or, not and implies), operators for integers (< and >), the append operator for strings (+),
operators for equality = and inequality <>, if-then-else expressions, the OCL built-in allInstances for the retrieving of every instance of a given type, the navigation through a property (.), and functions over collections (exists, forAll, collect, select, size, excluding and including). We leave out some OCL constructions like let for pseudefeature definitions, other simple and collection types (e.g. Real and Set), and the definition and use of auxiliary operations. As in [19], we impose that OCL expressions must be well-defined (not null or invalid), and no null or invalid values should be used. In general terms, these aspects can be ensured by construction of the OCL expressions using some conditions and operators. Moreover, we assume that expressions are type-safe, i.e. any sub-expression evaluates to the type according to the expression containing it, e.g. allInstances does not apply to arbitrary expressions but only to existent types in the signature, boolean connectives only take two boolean expressions, + is only defined for the String type, the expression Expr.Name is valid only for Expr evaluating to a single element (in other case collect must be used), etc. We also consider a general Bag type for representing collections with potentially repeated elements. Finally, we restrict allInstances to be used only on types representing metamodel elements (within C) and not on primitive types, i.e. finite domains. For the definition of a wider institution for OCL, a type inference system must be considered, as the one in [20].

**Example 5.** We can express the following invariants over the class metamodel in Fig. 1: (a) there cannot be two attributes with the same name within the same class, and (b) every table has a key which is a column of its own.

context Class inv: self.attribute->forAll(a1 : Attribute; a2 : Attribute | a1 <> a2 implies a1.name <> a2.name)
context Table inv: Table.allInstances()->forAll(t : Table | t.key.column.owner = t)

Signature morphisms are also taken from the original Csmof institution. Given a set of variables \( X_2 = (X_2^i)_{c_2 \in T(C_2)} \), we define a set \( X_1 = (X_1^i)_{c_1 \in T(C_1)} \) by \( X_1^i = X_2^{c_1 \cdot c_2} \). Signature morphisms extend to formulas over \( \Sigma_1 \) and \( \Sigma_2 \) as follows: given a \( \Sigma_1 \)-formula \( \psi \), \( \sigma(\psi) \) is the canonical application of the signature morphism to the types and roles in the formula, i.e.

\[
\sigma(\text{context Type inv : Expr}) = \text{context } \sigma(\text{Type}) \text{ inv : } \sigma(\text{Expr})
\]

\[
\sigma([x_j]_{1 \leq j \leq n} \cdot x_p = \text{Expr}) = [\sigma(x_j)]_{1 \leq j \leq n} \cdot \sigma(x_p) = \sigma(\text{Expr})
\]

With respect to OCL expressions, the signature morphism only affects the following expressions:

\[
\sigma(\text{Type.allInstances()}()) = \sigma(\text{Type}).\text{allInstances()}()
\]

\[
\sigma(\text{Expr.Name}) = \sigma(\text{Expr}).\sigma(\text{Name})
\]

\[
\sigma(\text{context Type inv : Expr}) = \text{context } \sigma(\text{Type}) \text{ inv : } \sigma(\text{Expr})
\]

\[
\sigma(\text{Var : Type}) = \text{Var : } \sigma(\text{Type})
\]

**Lemma 2.** There is a functor \( \text{Sen} \) giving a set of formulas \( \psi \) (object in the category \( \text{Set} \)) for each signature \( \Sigma \) (object in the category \( \text{Sign} \)), and a function \( \sigma : \text{Sen}(\Sigma_1) \rightarrow \text{Sen}(\Sigma_2) \) (arrow in the category \( \text{Set} \)) translating formulas for each signature morphism \( \sigma : \Sigma_1 \rightarrow \Sigma_2 \) (arrow in the category \( \text{Sign} \)).

**Proof.** The Csmof signature morphism changes types and roles consistently. Its application to any formula in \( \text{Sen}(\Sigma_1) \) gives a formula in \( \text{Sen}(\Sigma_2) \) with the types and roles translated with respect to the signature morphism \( \sigma : \Sigma_1 \rightarrow \Sigma_2 \). In this sense, the domain and codomain of the image of an arrow in \( \text{Set} \) are the images of domain and codomain, respectively, of the arrow in \( \text{Sign} \). Moreover, since types and roles are translated consistently with respect to a signature morphism, and signature morphism can be composed, the composition with respect to formulas is also preserved. For the same reason, identities are preserved. Finally, the functor \( \text{Sen} \) is defined. \( \square \)

Since Csmof is an institution (as proved in [5]), signatures and signature morphisms define a category \( \text{Sign} \). Lemma 2 proves that there is a functor \( \text{Sen} \) from this category to the category of sets of formulas and their translations. Interpretations, homomorphisms and reducts, are also taken from the Csmof institution, thus, interpretations and homomorphisms define a category \( \text{Mod}(\Sigma) \), the reduct defines a functor, and there is a functor \( \text{Mod} \) giving a category of interpretations for each signature and a reduct functor for each signature morphism.

Given a signature \( \Sigma \), a set of variables \( X = (X^i)_{c \in T(C)} \), and a \( \Sigma \)-interpretation \( I = (V^T(O), A) \), a variable assignment is a finite sequence \( \gamma \) of variable assignments of the form \( [x_1 \mapsto v_1, \ldots, x_n \mapsto v_n] \), with \( x_i \in X \) and \( v_i \in V^T(O) \) for all \( 1 \leq i \leq n \). The empty variable assignment is denoted by \( \emptyset \), concatenation of variable assignments \( \gamma \) and \( \gamma' \) is denoted \( \gamma \cdot \gamma' \). Moreover, given a signature morphism \( \sigma : \Sigma_1 \rightarrow \Sigma_2 \), \( \Sigma_2 \)-variables \( X_2 = (X_2^i)_{c_2 \in T(C_2)} \) and a variable assignment \( \gamma_2 \) for \( X_2 \) in a \( \Sigma_2 \)-interpretation \( I_2 \), the reduct of \( \gamma_2 \) along \( \sigma \) is a variable assignment for \( X_2^{\sigma(c)} \) in \( I_2 \) defined by \( \gamma_2^{\sigma(c)} \cdot x_1 \mapsto v_1, \ldots, x_n \mapsto v_n = \gamma_2(x_1) \mapsto \gamma_2(\sigma(x_1)) \mapsto \gamma_2(x_2) \mapsto \gamma_2(\sigma(x_n)) \mapsto v_n \).

The last thing we need is to provide a definition for the satisfaction relation between an Ocl formula and an interpretation. Then, by proving that the satisfaction condition holds for OCL formulas we can state that Ocl defines an institution.
Table 1
Operational semantics.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Spec)</td>
<td>((\omega; \gamma, \text{self} \rightarrow e \downarrow \nu))_{\text{self}(\xi)}</td>
</tr>
<tr>
<td>(Var)</td>
<td>((\omega; \gamma \rightarrow x \downarrow \gamma(x)))</td>
</tr>
<tr>
<td>(And)</td>
<td>((\omega; \gamma \rightarrow e \land e_2 \downarrow \nu_1 \land \nu_2))</td>
</tr>
<tr>
<td>(Not)</td>
<td>((\omega; \gamma \rightarrow e \rightarrow \neg v))</td>
</tr>
<tr>
<td>(Comp)</td>
<td>((\omega; \gamma \rightarrow e \downarrow v; \nu_{1,2}))</td>
</tr>
<tr>
<td>(Cond)</td>
<td>((\omega; \gamma \rightarrow e \downarrow \text{true} \land e_1 \downarrow \nu_1))</td>
</tr>
<tr>
<td>(Feat)</td>
<td>((\omega; \gamma \rightarrow e \downarrow v))</td>
</tr>
</tbody>
</table>

(\(\text{Lit^1} \omega; \gamma \rightarrow l \downarrow l\))
(\(\text{Self^1} \omega; \gamma \rightarrow \text{self} \downarrow \gamma(\text{self})\))
(\(\text{Or^1} \omega; \gamma \rightarrow e_1 \lor e_2 \downarrow \nu_1 \lor \nu_2\))
(\(\text{Impl^1} \omega; \gamma \rightarrow e_1 \rightarrow e_2 \downarrow \nu_1 \rightarrow \nu_2\))
(\(\text{Inst^1} \omega; \gamma \rightarrow \zeta\text{.allInstances()} \downarrow \omega(\zeta)\))

Our satisfaction relation is defined in terms of the operational semantics for OCL expressions given in [20]. This semantics evaluates an OCL term in the context of a dynamic basis. A dynamic basis \(\omega\) delivers the information on the actual state of the object system, i.e., it provides a semantic representation of an SW-model (instances and their types, values of structural features, the implementations of the built-in OCL properties, etc.) as our interpretations does. Thus, we define \(\omega(I)\) as the translation of an interpretation \(I\) into a dynamic basis \(\omega\). Our variable assignment can also be interpreted in terms of \(\omega\) instead of \(I\). The operational semantics derives judgments of the form \(\omega; \gamma \vdash t \downarrow \rho\), i.e., an OCL term \(t\) evaluates to \(\rho\) in the dynamic basis \(\omega\) and with the variable assignment \(\gamma\). Using these settings, the satisfaction relation with respect to OCL expressions is stated as follows.

**Definition 9 (OCL satisfaction relation).** Given a signature \(\Sigma = (C, \alpha, P)\) with \(C = (C, \leq_C)\) and \(P = (R, P)\), a set of variables \(X = (X_i)_{i \in T(C)}\), a \(\Sigma\)-formula \(\varphi\) representing an OCL expression, a \(\Sigma\)-interpretation \(I\), and a variable assignment \(\gamma\) over \(I\), the interpretation satisfies \(\varphi\), if one of the following holds:

\[
I, \gamma \vdash_{\Sigma} \text{context Type inv: Expr} \quad \text{if} \quad \omega(I); \emptyset \vdash \text{context Type inv: Expr} \downarrow \text{true} \\
I, \gamma[x_1 \mapsto v_1, \ldots, x_p \mapsto v_p, \ldots, x_n \mapsto v_n] \vdash_{\Sigma} \{x_j\}_{1 \leq j \leq n} x_p = \varphi(I); \omega(I); \gamma \vdash \text{Expr} \downarrow v_p
\]

Then, \(I, \Sigma, \varphi\) is defined as for all \(I, \gamma, \varphi \vdash_{\Sigma} \varphi\).

Judgments are derived by the rules in Table 1 which are a modified version of those in [20] with respect to our syntax. Every function over within our syntax (e.g., select, forAll) can be defined in terms of the iterate constructor (we need only one derivation rule (Iter1)). Moreover, the operator \(<=>\) is a syntactic sugar for the negation of the equality. The semantics defines \(\omega(\varphi)\) as the finite set of elements of type \(\zeta\) within \(\omega\). Finally, the retrieval function \(\text{impl}_{\nu_\gamma}\) takes as parameters the name of a property, the type in which it is declared, and the semantic element for which the property is to be evaluated, and returns the semantic elements connected through the property with such element. In the case of multiple elements, it returns a Bag of them.

We can prove that the valuation of an OCL expression is invariant under change of notation, i.e., that the application of a signature morphism does not change the values of expressions with respect to a dynamic basis built from an interpretation. This property can be proved by induction with respect to the operational semantics in Table 1.

**Lemma 3 (Invariance under change of notation).** Given signatures \(\Sigma_i (i = 1, 2)\), a signature morphism \(\sigma : \Sigma_1 \rightarrow \Sigma_2\), a \(\Sigma_2\)-interpretation \(I\), a set of \(\Sigma_2\)-variables \(X_2 = (X_2^i)_{i \in T(C_2)}\), a variable assignment \(\gamma\) over \(I\), and a \(\Sigma_1\) OCL expression \(e\) over \(X_1 = X_2\sigma\), the following property holds.

\[
\omega(I)\sigma; \gamma|\sigma \vdash e \downarrow v \iff \omega(I); \gamma \vdash \sigma(e) \downarrow v
\]

Proof. This property can be proved by induction in the structure of OCL, i.e., with respect to the operational semantics in Table 1. The case (Lit1) is trivial since literal values do not change with respect to a signature morphism. In the case of (Var1) we have that \(\omega|\sigma; \gamma|\sigma \vdash \sigma(x) \downarrow \gamma(\sigma(x))\) since \(\gamma|\sigma[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n] = \gamma[\sigma(x_1) \mapsto v_1, \ldots, \sigma(x_n) \mapsto v_n]\) by definition of reduction of the assignment of variables. The case of (Self1) is a specific case of the last one. In the case of (Spec1) we have that \(\omega|\sigma; \gamma|\sigma \vdash \text{context } \zeta \text{ inv: } e \downarrow v \nu\) holds if \(\omega|\sigma; \gamma|\sigma\),
self → ν | e → v e, v ∈ ≤ v

By inductive hypothesis we have that (ω; γ, σ (self) ↦ v) ⊢ σ (e) ↓ ≤ v, v ∈ ≤ ω (σ (γ)) which proves that ω; γ | e, v ∈ ≤ v

The cases (And), (Or), (Not), (Impl), (Comp1), (Cond1) and (Cond2) are all proved in the same way. Let us take for example the case (And) in which we have that ω | e | e1 e2 | v1 v2. By inductive hypothesis we have that (ω; γ | e | e1, e2) | v1 v2 which proves that ω; γ | e, v ∈ ≤ v1 v2. In the case of (Cond2) we have that ω | e | σ (e1) v1, allInstances () | v2. We also know that ω | e | σ (γ) since, by definition of the reduc of interpretations in CSMoF, the interpretation of element of γ do not change with respect a signature morphism and thus the dynamic basis defined from both interpretations is the same. Then, we have that ω; γ | e | σ (γ). allInstances () | v. In the case of (Feat) we know that ω | e | σ (e) ⊢ e' | v. By inductive hypothesis we have that ω | e | σ (e) | v. Moreover, we know that Impl | e | σ (e) | v since, by definition of the reduc of interpretations in CSMoR, the semantic elements related through properties do not change. Using both results we have that ω | e | σ (e) | v'.

In the case of (Iter) we know that ω | e | σ (e) → iterate (x; x' = e' | e'') | v1 v2 if ω; γ | e | Bag [v1, v2].

Using hypothesis we have that ω | e | Bag [v1, v2], and (ω; γ | v1 | x → v1, x' = v1' | v1'' | v1 | v1' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1'' | v1"
Every relation has a set \(<R\text{\_var}\_set>\) of variables occurring in the relation, which are particularly used within the domains \(<<\text{domain\_k}\text{\_var}\_set>\text{domain\_k}\_\text{pat}>\) and in the when clause \(<<\text{when}\_\text{var}\_set>\text{when}\_\text{cond}>\). Relations define source/target domain patterns \(<<\text{domain\_k}\_\text{pat}>\). A pattern can be viewed as a graph of typed elements (which will be matched by objects) and relations (which will be matched by links), together with a predicate (boolean OCL expression) which must hold. The predicate may refer to variables other than the pattern elements; these are the free variables of a pattern. Relations can also contain when \(<<\text{when}\_\text{cond}>\text{and}\>\) and where \(<<\text{where}\_\text{cond}>\) clauses. A when clause specifies the conditions under which the relationship holds, whilst the where clause specifies the condition that must be satisfied by all SW-model elements participating in the relation, and it may constrain any of the variables in the relation and its domains. The when and where clauses, as well as the predicate of a pattern, may contain arbitrary boolean OCL expressions in addition to the relation invocation expressions. Any relation can define a set of primitive domains which are data types used to parameterize the relation \(<<\text{R}\text{\_par}\_set>\). In this sense, top-level relations can be parametric when called from a when clause, whereas non-top-level relations are always parametric since they are called for given source and target domains elements.

Finally, the transformation can also define keys on metamodel elements, i.e. a definition of which properties of an element, in combination, can uniquely identify an instance of that class. Keys must be ensured by the model transformation.

**Example 6.** Consider the following example which is a simplified version of the well-known Class to Relational transformation [9]. The transformation uml2rdbsms basically describes how persistent classes within a package (specified with the metamodel of Fig. 1a and referred as UML) are transformed into tables within a schema (specified with the metamodel of Fig. 1b and referred as RDBMS). The relation PackageToSchema states that any UML package is mapped into a relational schema. Moreover, the relation ClassToTable states that classes marked as persistent are mapped into tables with the same name, a primary key and an identifying column, such that the package to which the class belongs is in the relation with the schema to which the table belongs. The relation AttributeToColumn is called from the where clause of ClassToTable and maps primitive attributes of the persistent class to columns of the corresponding table. There are also keys, e.g. stating that the transformation must ensure that there cannot be two Tables with the same name within the same Schema.

The integer property numC within a Schema is involved with the transformation since we add in the where clause an Ocl formula stating that this property must be equal to the number of persistent classes within the corresponding package.

```plaintext
transformation uml2rdbsms ( uml : UML , rdbms : RDBMS ) { 
  key RDBMS::Table {name, schema}; 
  ...

top relation PackageToSchema ( 
  pn : String;
  checkonly domain uml p:UML::Package { name = pn };
  enforce domain rdbms s:RDBMS::Schema { name = pn, numC = pClasses };
  where { pCl = Class.allInstances->select(c : Class | 
    c.kind='Persistent' and c.namespace = p)->size(); } 
)

top relation ClassToTable ( 
  cn, prefix : String;
  checkonly domain uml c:UML::Class { 
    namespace = p:UML::Package (), kind = 'Persistent', name = cn
  };

  enforce domain rdbms t:RDBMS::Table ( 
    schema = s:RDBMS::Schema (), name = cn,
    column = cl:RDBMS::Column { name = 'TID', typeT = 'NUMBER' },
    key = k:RDBMS::Key { name = 'PK', column = cl }
  );
  when { PackageToSchema(p, s); }
  where { AttributeToColumn(c, t, prefix); prefix = ''; }
)

relation AttributeToColumn ( ... )
}
```
A signature defines the source and target metamodels that are involved in a specific model transformation.

**Definition 10 (Qvtr signature).** A Qvtr signature is a pair \((\Sigma_1^{\text{CSMOF}}, \Sigma_2^{\text{CSMOF}})\) of CSMOF signatures \(\Sigma_i^{\text{CSMOF}} = (C_i, \alpha_i, P_i)\) \((i = 1, 2)\) representing the source and target metamodels of the transformation. From such a signature, we can derive an Ocl signature which is defined as the union of both Csmof signatures, i.e. \((C_1 \cup C_2, \alpha_1 \cup \alpha_2, P_1 \cup P_2)\).

**Example 7.** With respect to the example, the signature \(\Sigma = (\Sigma_1^{\text{CSMOF}}, \Sigma_2^{\text{CSMOF}})\) contains the signature \(\Sigma_1^{\text{CSMOF}}\) of the source metamodel, which is presented in Example 2, and the signature \(\Sigma_2^{\text{CSMOF}}\) of the target metamodel, which is not shown here but can be derived in the same way as the other one.

Formulas represent key constraints defined on source and target metamodel elements and transformation rules.

**Definition 11 (Qvtr formula).** Given a signature \((\Sigma_1^{\text{CSMOF}}, \Sigma_2^{\text{CSMOF}})\) such that \(\Sigma_i^{\text{CSMOF}} = (C_i, \alpha_i, P_i)\) with \(C_i = (C_i, \leq C_i)\) and \(P_i = (R_i, P_i)\), \(\Sigma\)-formulas are defined as follows:

- A formula \(\varphi^K\) representing a key constraint of the form \(\{c, [r_1, \ldots, r_n]\} (1 \leq n)\) with \(c \in C_i \ (i = 1..2)\) a class in one of the metamodels, \(r_j \in R_i \ (j = 1..n)\) roles defined in properties in which such class participates (having such role at either side of it), i.e. for each \(r_j\) there is a property \(\{r_j : c_j, r_j : c_j\} \in P_i\) such that \(c = c_i\) (the property is non-navigable from \(c\)) or \(c = c_j\) (\(r_j\) is navigable from \(c\)). Roles determine the elements within these properties that together can uniquely identify an instance of the class.
- A formula \(\varphi^K\) representing a set of interrelated transformation rules with variables \(X^i = (X^i)_s \in l(J_i, T(C_i)))\), is a finite set of tuples representing rules of the form \((\top, \text{VarSet}, \text{ParSet}, \text{Pattern}_1, \ldots, \text{Pattern}_n)\) when, where: where: \(\top \in \{\text{true}, \text{false}\}\) defines if the rule is a top-level relation or not; \(\text{VarSet} \subseteq X^i\) is the set of variables used within the rule; \(\text{ParSet} \subseteq \text{VarSet}\) representing the set of variables taken as parameters when the rule is called from another one; \(\text{Pattern}_i \ (i = 1, 2)\) are the source and target patterns, i.e. tuples \((E_i, A_i, P_{r1})\) such that \(E_i \subseteq X^C\) is a set of class-indexed variables, \(A_i\) is a set of elements representing associations of the form \(\text{rel}(p, x, y)\) with \(p \in P_i\) and \(x, y \in E_i\), and \(P_{r1}\) is an Ocl-formula over these elements; when/where are the \(\text{when/where}\) clauses of the rule, respectively, when/where is a pair of an Ocl-formula with variables in \(\text{VarSet}\), and a set of pairs of transformation rule names (within \(\varphi^R\)) and set of variables which are the parameters of the rules.

**Example 8.** The key definition in the example is represented by the following formula: \((\text{Table, } \text{name, schema}).\) There is also a formula representing the whole transformation with one rule for each rule. As an example, for the rule \(\text{PackageToSchema} \) we have \((\top, \text{VarSet}, \text{ParSet}, \text{Pattern}_1, \ldots, \text{Pattern}_n)\) when, where: such that

\[
\begin{align*}
\text{top} & = \ \text{true} \\
\text{VarSet} & = \{(p, n, p, \text{pCl}) \mid p \in X^\text{String}, p \in X^\text{Package}, \text{and } s \in X^\text{Schema} \text{ and } \text{pCl} \in X^{\text{Integer}}
\]
\text{ParSet} & = \{p, s\} \\
\text{Pattern}_1 & = (E_1, A_1, P_{r1}) \text{ with } E_1 = \{p, pn\}, A_1 = [\text{rel}(\text{name}, p, pn)], \text{ and } P_{r1} = \text{true} \\
\text{Pattern}_2 & = (E_2, A_2, P_{r2}) \text{ with } E_2 = \{s, pn, \text{pCl}\}, A_2 = [\text{rel}(\text{name}, s, pn), \text{rel}(\text{numC}, s, \text{pCl})], \text{ and } P_{r2} = \text{true} \\
\text{when} & = (\emptyset, \emptyset) \\
\text{where} & = (\emptyset, \emptyset)
\end{align*}
\]

Within the \(\text{where}\) clause, \(\varphi\) is the following Ocl formula with variables \(\text{pCl} \in X^{\text{Integer}}\) and \(p \in X^{\text{Package}}\)

\[
\text{pCl} = \text{Class.allInstances->select(c : Class } \mid \text{ c.kind='Persistent' } \text{ and } \text{c.namespace = p)->size()}
\]

An interpretation contains a semantic representation for the source and target SW-models.

**Definition 12 (Qvtr interpretation).** Given a signature \((\Sigma_1^{\text{CSMOF}}, \Sigma_2^{\text{CSMOF}})\), an interpretation is a tuple \((M_1^{\text{CSMOF}}, M_2^{\text{CSMOF}})\) of CSMOF interpretations \((V_1^C(O_i), A_i)\) that agree on the shared part \(M_1^{\text{CSMOF}} \cap M_2^{\text{CSMOF}}\). From this interpretation we can derive an Ocl interpretation, defined over the union of both signatures (in Definition 10), as the amalgamated sum of the \((V_1^C(O_i), A_i)\). This works because the union of signatures is a pushout w.r.t. their intersection, and the institution admits amalgamation of models along pushouts, see [22].

**Example 9.** We can define an interpretation \(M = (M_1^{\text{CSMOF}}, M_2^{\text{CSMOF}})\) such that \(M_1^{\text{CSMOF}}\) is the one defined in Example 3, and \(M_2^{\text{CSMOF}}\) is an interpretation with a direct correspondence with the SW-model in Fig. 1d.

---

4 We denote by \(k_\text{VarSet} \ (k = \{1, 2\})\) the variables used in pattern \(k\) that do neither occur in the other domain nor in the when clause, and by \(k_\text{WhenVarSet}\) the set of variables occurring in the when clause.
The satisfaction relation of our institution is defined following the checking semantics presented in the QVT standard [9], i.e. it checks whether the source and target SW-models (represented within the interpretation) satisfy the relations defined by the transformation rules. It also checks whether key constraints hold (both represented as formulas). The semantics states that a rule holds if for each valid binding of variables of the when clause and variables of domains other than the target domain, that satisfy the when condition and source domain patterns and conditions, there must exist a valid binding of the remaining unbound variables of the target domain that satisfies the target domain pattern and where condition.

**Definition 13 (Qvtr satisfaction relation).** Given a signature \( \left( \Sigma_1^{Csmof}, \Sigma_2^{Csmof} \right) \) such that \( \Sigma_2^{Csmof} = (C_i, a_i, P_i) \) with \( C_i = (C_i, \leq C_i) \) and \( P_i = (R_i, P_i), \) and an interpretation \( \mathcal{M} = (\mathcal{M}_1^{Csmof}. \mathcal{M}_2^{Csmof}) \), we define that \( \mathcal{M} \) satisfies a formula \( \varphi \), written \( \mathcal{M} \models \varphi \), by distinguishing the following cases:

- A formula \( \varphi^R = (c, \{r_1, \ldots, r_n\}) \) with \( c \in C_1 (j = 1..n), r_j \in R_1 (j = 1..n), \) is satisfied in \( \mathcal{M} \), if in the corresponding metamodel \( \mathcal{M}_1^{Csmof} \) there are no two elements of type \( c \) with the same set of elements related through properties involving roles \( r_j \) (they must differ in at least one element). Formally, for each \( r_j \) the corresponding property \( p_j \) is \( \{r : c, r_j : d\} \) if \( r_j \) is navigable, or \( \{r_j : c, \ldots : d\} \) if the opposite role of \( r_j \) is non-navigable. We can define that an element \( x \in (V_{Cl} \cup C_1) \), the set of semantic elements linked with \( x \) in \( p_1 \) is \( v(x, p_j) = \{\mathcal{P}_2(t) | \mathcal{P}_1(t) = x, \ t \in p_j \} \). The definition is straightforward in the case of \( c \) in the second component of the property. The formula is satisfied if for all \( x, y \in (V_{Cl} \cup C_1), x \neq y \) implies \( (\nabla_j v(x, p_j) \neq (\nabla_j v(y, p_j)) \).
- A formula \( \varphi^R \) is satisfied if every top-level relation holds. Formally, given an Ocl interpretation \( \mathcal{M}^{Ocl} \) built from the interpretation \( \mathcal{M}, \varphi^R \) is satisfied if for every top relation \( \text{Rule} \in \varphi^R, \) we have that \( \mathcal{M}^{Ocl}, \emptyset \models \text{Rule} \). We use \( \emptyset \) as the empty variable assignment, only filled in the case of explicit called relations. A relation is satisfied if there are matching elements in the source and target SW-models in the relation. The complete formal definition of the satisfaction of Rule can be found in [5].

**Example 10.** In the example, we need to prove that our interpretation satisfies both kinds of formulas. In the case of the key (Table, name, schema), we have only one table \( r \) and only one key \( k \), thus the condition trivially holds. We need to prove now that \( \mathcal{M}^{Ocl}, \emptyset \models \text{PackageToSchema}. \) We know that \( \text{[p]} = \text{Vpackage} = \{p\} \) and \( \text{[pn]} = \text{Vpackage, String, ID, value...} \), thus \( \text{[p, pn]} = \{\text{[Package, p]}, \text{[String, p]}, \text{[ID, p]}, \ldots\} \). We also have that \( \text{[s]} = \text{Vschema} = \{s\} \) and \( \text{[pCl]} = \{1, 2, 3,...\} \). Thus, \( \mathcal{M}^{Ocl}, \emptyset \models \text{PackageToSchema} \) if

\[ \forall \mu^1[pn, p] \in \{\text{[Package, p]}, \text{[String, p]}, \text{[ID, p]}, \ldots\}, \]
\[ (\mathcal{M}^{Ocl}|_{\varphi}, \mu^1 \models \text{Pattern1} \rightarrow \exists \mu^2[s, pCl] \in \{(s, 1), (s, 2), (s, 3), \ldots\}, \]
\[ (\mathcal{M}^{Ocl}|_{\varphi}, (\mu^1 \cup \mu^2) \models \text{Pattern2} \wedge \mathcal{M}^{Ocl}|_{\varphi}, (\mu^1 \cup \mu^2) \models \text{where}) \]

For every \( \mu^1[pn, p] \) different from \( \text{[Package, p]} \) we have that Pattern1 does not hold, since it depends on the relation \( \text{rel(name, p, pn)} \). Thus, in these cases the implication holds. Now, in the case of \( \text{[Package, p]} \), we have that Pattern1 holds, and that there exists a valid combination for \( \mu^2[s, pCl] \) which is \( (s, 1) \). In this case we have that \( \mathcal{M}^{Ocl}, (\mu^1 \cup \mu^2) \models \text{Pattern2} \) since the relations \( \text{rel(name, s, pn)} \) and \( \text{rel(numC, s, pCl)} \) hold. Note in Fig. 1d that the schema has the same name as the package, which is semantically represented as Package. Moreover, we have that \( \mathcal{M}^{Ocl}, (\mu^1 \cup \mu^2) \models \text{where} \) since the Ocl formula holds (according to the Ocl satisfaction relation in Definition 9). As a proof of it, we have that the variable environment is \( \gamma[pCl \rightarrow 1, \ldots, p \rightarrow p] \) and that in this context the expression in the right hand side of equality evaluates to 1, i.e. there is only one persistent class within the package \( p \). In conclusion, we have that \( \mathcal{M}^{Ocl}, \emptyset \models \text{PackageToSchema} \).

5. Encoding conformance into Casl

We do not define our own entailment system for verification within the Csmof and Qvtr institutions, but borrow Casl’s entailment system through suitable institution comorphisms. In what follows we define an extension of Csmof (and Qvtr), named Csmof\textsuperscript{Ocl} and Csmof\textsuperscript{Qvtr}, respectively, which is of interest for the inclusion of SW-models to be considered by any possible entailment system devised to derive the satisfiability of other formulas. Moreover, we define a simple theoreoidal comorphism between Csmof\textsuperscript{Ocl} and the institution SubPCF\textsuperscript{Ocl}\textsuperscript{O}. Along the definition, we illustrate the main concepts with the example presented in Section 3.

5.1. Extending the Qvtr and Csmof Institutions

We would like to be able to verify properties of individual SW-models, for example, whether they satisfy the multiplicity constraints of their metamodel. However, in Csmof, we cannot do this, because we cannot speak about SW-models. Therefore, we now define an institution  \( \mathcal{I}^D \) extending the Csmof institution, and providing a new kind of formulas, namely syntactic representations of individual SW-models.
Definition 14 ($\mathcal{I}^{\Sigma}$-formulas). Given a signature $\Sigma = (C, \alpha, P)$ with $C = (C, \subseteq C)$ and $P = (R, P)$ as defined for the CsmOf institution, and variables $X = (X^c)_{c \in T(C)}$, we define formulas as follows:

$$\Omega ::= X^c \mid \{r_1, x_1^{c_1}, r_2, x_2^{c_2}\} \mid \Omega \oplus \Omega$$

with $X^c \subseteq X^\Sigma$, $x_1^{c_1} \in X_1^{c_1}$, $x_2^{c_2} \in X_2^{c_2}$, $c_1, c_2 \in T(C)$, $\langle r_1 : c_1, r_2 : c_2 \rangle \in P, r_1, r_2 \in R$.

A variable $x^c$ represents a typed element, $\{r_1, x_1^{c_1}, r_2, x_2^{c_2}\}$ represents a link between two typed elements with their respective roles, and $\Omega \oplus \Omega$ allows to compose these elements to represent a whole SW-model.

Example 11. As an example, the formula $\Omega$ corresponding to the class SW-model in Fig. 1c is defined as follows:

$$\mathcal{P}^{\text{Package}} \oplus C^{\text{Class}} \oplus a^{\text{Attribute}} \oplus \mathcal{P}^{\text{PrimitiveDataType}} \oplus \mathcal{P}^{\text{String}} \oplus \mathcal{I}^{\text{String}} \oplus \mathcal{P}^{\text{Persistent}} \oplus \ldots \oplus$$

$$\langle \text{namespace, p, elements, c} \rangle \oplus (\ldots \implies \langle \text{p, name, Package} \rangle \oplus (\text{namespace, p, elements, pdt}) \oplus (\ldots \implies \langle \text{a, type, pdt} \rangle \ldots)$$

The satisfaction of a $\Omega$ formula with respect to an interpretation $\mathcal{I}$ is defined as the existence of an isomorphism between $\Omega$ and the reduction of the interpretation with respect to the types in $\Omega$ [4], i.e., a bijective function mapping each element $x^c$ to an element of $(V^c)_{c \in T(C)}$ for each $c \in T(C)$, such that syntactic links in $\Omega$ and semantic links in $\mathcal{I}$ coincide.

As with this supporting institution $\mathcal{I}^{\Sigma}$, our OCL institution presented in Section 3 takes basic definitions from the CsmOf institution, e.g., signatures and interpretations, and introduces OCL expressions as formulas. In this sense, the institutions CsmOf, $\mathcal{I}^{\Sigma}$, and Ocl coincide in the definition of signatures and interpretations (as in other institution elements) but differ in the formulas. As demonstrated in [14], by taking these elements and defining formula sets (and the satisfaction relation) as the disjoint union of CsmOf, $\mathcal{I}^{\Sigma}$ and Ocl formula sets (and of satisfaction relations), we can define an institution CsmOfOcl. We can also define an extension of the Qvtr institution QvtrOcl such that formulas are either Qvtr or CsmOfOcl formulas.

These extensions are of interest for the inclusion of SW-models to be considered by any possible entailment system intended to derive the satisfiability of other formulas. If we have a set of formulas $\Psi$ composed by multiplicity constraints and OCL formulas, we need to prove that they hold in a concrete SW-model $\Omega$, which is the context in which the verification must be done. This is expressed as $\Omega \models_\Psi \Psi$. In the same way, we also need to verify whether a key constraint (or a set of them), represented as a formula $\varphi^R$, is derived from the same $\Omega$, i.e. $\Omega \models_\Sigma \varphi^R$, or whether a transformation rule $\varphi^R$ (or the whole model transformation) is derived from a pair of SW-models, i.e. $\Omega_1 \cup \Omega_2 \models_\Sigma \varphi^R$. A sound entailment system will ensure semantic entailment, i.e. $\Omega \models_\Sigma \Psi$ implies $\Omega \models_\Sigma \Psi$. Semantic entailment is defined by the satisfaction relations of the corresponding institutions. In Section 7 we exemplify how we can address these proof requirements when moving into Casl.

5.2. Translation of signatures

As defined in Section 2, the translation is given in terms of a function $\rho^{\text{Sign}}$ between CsmOfOcl signatures and SubPCFOLm theories, a natural transformation $\rho^{\text{Sen}}$ such that SW-model formulas, multiplicity constraints and OCL formulas are translated to SubPCFOLm formulas, and a natural transformation $\rho^{\text{Mod}}$ translating interpretations and homomorphisms from SubPCFOLm to CsmOfOcl.

We consider that SubPCFOLm is equipped with functions and predicates defined for built-in types, as already defined in the Casl standard library. In particular, we assume the existence of a type $\text{BagElem}$ representing a bag of generic elements with functions, e.g. $\text{size} : \text{BagElem} \to \text{Nat}$ and $+: \text{Elem} \times \text{BagElem} \to \text{BagElem}$ for querying the number of elements and adding an element to a bag, respectively; and predicates, e.g. $\text{eps} : \text{Elem} \times \text{BagElem}$ stating that an element is part of a bag (see the Casl standard libraries [23]).

The class hierarchy represented within a CsmOfOcl signature is directly represented as a subsort hierarchy; properties are translated to predicates, and an axiom is introduced to relate predicates derived from bidirectional properties. Every CsmOfOcl signature $\Sigma = (C, \alpha, P)$ with $C = (C, \subseteq C)$ and $P = (R, P)$ is translated to a theory $(\mathcal{S}, \mathcal{T}, \mathcal{F}, \mathcal{P}, \subseteq, \in)$ such that:

- For every class name $c$ in $C$, there is a sort name $c \in \mathcal{S}$, and a total function $c_{\text{allInstances}} : \text{Bag}[c]$ together with an axiom in $E$ stating the $\forall x \in c \cdot x \in c_{\text{allInstances}}$.
- For every $c_1 \subseteq c_2$ with $c_1, c_2 \in C$, we have $c_1 \subseteq c_2 \iff c_1, c_2 \in S$.
- For every abstract class $c_2 \in \alpha$, we introduce an axiom $\forall x : c_2 \cdot \exists c_1 \subseteq c_2 : x \in c_1$ (where $x \in c_1$ is a Casl sort membership test).
- For every $\langle r_1 : c_1, r_2 : c_2 \rangle \in P$, there are two predicates $r_1 : c_1 \times c_1$ and $r_2 : c_1 \times c_2 \in \mathcal{I}$ together with an axiom in $E$ stating the equivalence of the predicates, i.e. $r_1(x, y) \iff r_2(y, x)$ with $x : c_1, y : c_2$. Moreover, there are also two functions $r_1 : c_2 \to \text{Bag}[c_1]$ and $r_2 : c_1 \to \text{Bag}[c_2] \in \mathcal{T}$ for accessing elements related through properties, together with an axiom in $E$ stating that these functions are implied by the corresponding predicates, e.g. $\forall x \in c_2, y \in c_1 \cdot y \in r_1(x) \iff r_1(x, y)$. In the case of predicates with the default role name _, we only generate the predicate (and the total function) in the opposite direction of the default role, i.e. if $\langle \_ : c_1, r_2 : c_2 \rangle$ or $\langle r_1 : c_1, \_ : c_2 \rangle$ we only have $r_2 : c_1 \times c_2$ or $r_1 : c_2 \times c_1$, respectively.
Example 12. The translation of the signature corresponding to the UML class diagrams metamodel in Fig. 1a gives the following theory (expressed using CASL syntax).³

```
sorts Class, PrimitiveDataType < Classifier; Attribute, Classifier, Package < UMLModelElement

pred name : UMLModelElement × String
op name : UMLModelElement → Bag[String]
• ∀ x : UMLModelElement; y : String • y ∈ name(x) ⇔ name(x, y)
pred attribute : Class × Attribute
op attribute : Class → Bag[Attribute]
• ∀ x : Class; y : Attribute • y ∈ attribute(x) ⇔ attribute(x, y)
pred owner : Attribute × Class
op owner : Attribute → Bag[Class]
• ∀ x : Attribute; y : Class • y ∈ owner(x) ⇔ owner(x, y)
op Class_allInstances : Bag[Class]
• ∀ x : Class • x ∈ Class_allInstances

∀ x : Class; y : Attribute • attribute(x, y) ⇔ owner(y, x)

∀ x : UMLModelElement • x ∈ Attribute ∨ x ∈ Classifier ∨ x ∈ Package
```

Now we can define how signature morphisms are translated. Given \( \Sigma_i = (C_i, \alpha_i, P_i) \) \((i = 1, 2)\) with \( C_i = (C_i, \leq_{C_i}) \) and \( P_i = (R_i, P_i) \), and a signature morphism \( \sigma : \Sigma_1 \rightarrow \Sigma_2 = (\sigma_T, \sigma_R) \), its translation \( \rho^{\text{Sign}}(\sigma) \) is a SubPCFOL \( \text{signatures and signature morphisms to the category of theories in SubPCFOL}^\text{®} \).

We can prove now that \( \rho^{\text{Sign}} \) is indeed a functor.

Lemma 6. The function \( \rho^{\text{Sign}} : \text{CSMOF}_{\text{OCL}} \rightarrow \text{Th}^\text{SubPCFOL}^\text{®} \) is a functor from the category of CSMOF\( \text{OCL} \) signatures and signature morphisms to the category of theories in SubPCFOL\( \text{®} \).

Proof. (a) By definition, the image of an arrow \( \sigma : \Sigma_1 \rightarrow \Sigma_2 \) in the category of CSMOF\( \text{OCL} \) theories is an arrow \( \rho^{\text{Sign}}(\sigma) : \rho^{\text{Sign}}(\Sigma_1) \rightarrow \rho^{\text{Sign}}(\Sigma_2) \) in the category of theories in SubPCFOL\( \text{®} \). Thus, domain and codomain of the image of an arrow are the images of domain and codomain, respectively, of the arrow.

(b) We have to prove that composition is preserved, i.e. \( \rho^{\text{Sign}}(\sigma_2 \circ \sigma_1) = \rho^{\text{Sign}}(\sigma_2) \circ \rho^{\text{Sign}}(\sigma_1) \). Let \( \Sigma_i \) \((i = 1..4)\) be signatures, and let \( \sigma_i : \Sigma_i \rightarrow \Sigma_{i+1} \) \((i = 1, 2)\) be signature morphisms. As defined in [5], the composition \( \sigma_2 \circ \sigma_1 \) is a tuple \( \langle \sigma_T, \sigma_R \rangle \) such that \( \sigma_T(c) = \sigma_{T_2}(\sigma_{T_1}(c)) \), and \( \sigma_R(c) = \sigma_{R_2}(\sigma_{R_1}(c)) \). Its translation \( \rho^{\text{Sign}}(\sigma_2 \circ \sigma_1) \) is a SubPCFOL \( \text{signature morphism} \langle \sigma_S, \sigma_F, \sigma_P \rangle \) such that:

- \( \sigma_S(\rho^{\text{Sign}}(c)) = \rho^{\text{Sign}}(\sigma_{T_2}(\sigma_{T_1}(c))) \) for every \( c \in T(C_i) \)
- \( \sigma_T(\sigma(r_1, r_2)) = \sigma_{T_2}(\sigma_{T_1}(\sigma_{T_1}(c_1))), \sigma_{T_2}(\sigma_{T_1}(c_1))) \), and \( \sigma_F(\sigma(r_1, r_2)) = \sigma_{F_2}(\sigma_{F_1}(\sigma_{F_1}(c_1))), \sigma_{F_2}(\sigma_{F_1}(c_1))) \),
- for every \( r_1 : c_1, r_2 : c_2 \) \( P \) and \( r_1 : c_1, r_1 : c_1, r_2 : c_2 \) the predicates generated from \( \rho^{\text{Sign}}(\sigma(\tau_1, \tau_2 : c_2)) \).

Moreover, the translations \( \rho^{\text{Sign}}(\sigma_1) \) and \( \rho^{\text{Sign}}(\sigma_2) \) are SubPCFOL\( \text{®} \) signature morphisms \( \langle \sigma_S, \sigma_F, \sigma_P \rangle \) \((i = 1, 2)\). Using this information and the properties of signature morphisms, we can conclude that \( \rho^{\text{Sign}}(\sigma_2 \circ \sigma_1) = \rho^{\text{Sign}}(\sigma_2) \circ \rho^{\text{Sign}}(\sigma_1) \), since

- \( \sigma_S(\rho^{\text{Sign}}(\sigma_1)) = \rho^{\text{Sign}}(\sigma_{T_2}(\sigma_{T_1}(c_1))) = \rho^{\text{Sign}}(\sigma_{T_2}(\sigma_{T_1}(c_1))) \)
- \( \rho^{\text{Sign}}(\sigma_1(r_1, r_2, c_1)) = \rho^{\text{Sign}}(\sigma_{T_2}(\sigma_{T_1}(c_1))), \sigma_{T_2}(\sigma_{T_1}(c_1))) = \sigma_{T_2}(\sigma_{T_1}(c_1))), \sigma_{T_2}(\sigma_{T_1}(c_1)) \).
- \( \sigma_{S_2} \circ \sigma_{S_1} \), \( \sigma_{F_2} \circ \sigma_{F_1} \), \( \sigma_{P_2} \circ \sigma_{P_1} \) are SubPCFOL\( \text{®} \) signature morphisms \( \langle \sigma_S, \sigma_F, \sigma_P \rangle \) \((i = 1, 2)\). The other case is analogous.

³ For space reasons we show only a excerpt. We omit many predicates corresponding to properties in the metamodel, their corresponding axioms stating their equivalence, and the allInstances axioms for the other sorts.
(c) We have to prove that identities are preserved. As defined in [5], the identity signature morphism $\idS$ in $\text{Sign}^\text{CSMOF}$ is a tuple of identity functions for types and roles. Its translation $\rho^\text{Sign}(\idS)$ is the identity signature morphism in $\text{SubPCFOL}_m$, since by definition of $\rho^\text{Sign}$ we have that:

- $\sigma^x(\rho^\text{Sign}(c)) = \rho^\text{Sign}(\idS(c)) = \rho^\text{Sign}(c)$ for every $c \in T(C)$.
- $\rho^\text{Sign}(r)(1)(c_1, c_2) = \idS(r)(\idS(c_1), \idS(c_2)) = r(\idS(c_1), \idS(c_2)) = r(c_1, c_2)$, for every $(r(1), c_1, c_2) \in P_1$ and $r(1, c_1, c_2)$ the predicates generated from $\rho^\text{Sign}(r(1), c_1, c_2)$.

Finally, $\rho^\text{Sign}$ is a functor. □

5.3. Translation of formulas

An SW-model formula $\Omega$ is translated to an existentially quantified formula such that each variable within the formula (representing an object) is translated to a variable of the corresponding type, and several properties are added in order to represent implicit constraints in the institution CSMOR$^{\text{DS}}$ that are not necessarily kept when representing the basic elements in $\text{SubPCFOL}_m$. For example, we express the need of distinguishing between two different variables and we specify the cases in which a property holds (when there is a syntactic link represented within the formula $\Omega$). Formally,

- For every element $x'$ representing a typed element within a formula $\Omega$, there is an existential quantified variable $x : c$ with $c \in S$.
- The following formulas constrain the existentially quantified variables:
  - Distinguishability: $\{x_1 \neq x_2 \mid i \neq j, x_i : c_1, x_j : c_j, c_i \leq c, c_j \leq c \text{ for some } c\}$ stating that different variables have different values (unique name assumption), provided that they have comparable domains (otherwise, the inequality would be ill-typed).
  - Completeness of elements: for all $x : c$ we have that $x = o_1$ for some variable $o_1 : c$. It states that for a given sort representing a type, elements of that sort are one of the variables representing objects of that type. When $c$ is a non-abstract class having sub-classes, completeness must be defined for $o_1 : c'$ for all $c' \leq c$.
  - Completeness of relations: for all $x : c_1, y : c_2$ we have that the predicate $r_2 : c_1 \times c_2$ holds only if $x = o_1$ and $y = o_2$ for those variables $o_1 : c_1, o_2 : c_2$ for which $r_1, o_1, o_2$ representing a link between two typed elements is defined within $\Omega$. It completely specifies the extension of the relation $r$.

The “distinguishability” and “completeness of elements” axioms correspond to the so-called “no junk, no confusion” principle: there are no other values than those denoted by the variables $x : c$, and all distinct variables denote different values.

Example 13. The translation of the $\Omega$ formula corresponding to the Class SW-model in Fig. 1c gives the following formula (expressed using Cast syntax).

$$\exists c : \text{Class}; a : \text{Attribute}; p : \text{Package}; pdt : \text{PrimitiveDataType}$$

$$\{\text{distinguishability }\%\}$$

$$\exists \ a = c \land \lnot \exists \ a = pdt \land \lnot \exists \ c = pdt \land \lnot \exists \ p = pdt$$

$$\{\text{completeness of elements }\%\}$$

$$\land \left( \forall x : \text{Class} \bullet x = c \right) \land \left( \forall x : \text{Attribute} \bullet x = a \right) \land \left( \forall x : \text{Package} \bullet x = p \right) \land \left( \forall x : \text{PrimitiveDataType} \bullet x = pdt \right) \land \left( \forall x : \text{Classifier} \bullet x = c \lor x = pdt \right) \land \left( \forall x : \text{UMLModelElement} \bullet x = c \lor x = p \lor x = pdt \right)$$

$$\{\text{completeness of relations }\%\}$$

$$\land \left( \forall x : \text{Attribute}; y : \text{PrimitiveDataType} \bullet \text{typeT}(x, y) \iff x = a \land y = pdt \right)$$

$$\land \left( \forall x : \text{UMLModelElement}; y : \text{String} \bullet \text{kind}(x, y) \iff (x = a \land y = "") \lor (x = c \land y = "Persistent") \lor (x = p \land y = ") \lor (x = pdt \land y = ") \right)$$

$$\land \left( \forall x : \text{UMLModelElement}; y : \text{String} \bullet \text{name}(x, y) \iff (x = a \land y = "value") \lor (x = c \land y = "ID") \lor (x = p \land y = "Package") \lor (x = pdt \land y = "String") \right)$$

$$\land \left( \forall x : \text{Package}; y : \text{Classifier} \bullet \text{elements}(x, y) \iff (x = p \land y = c) \lor (x = p \land y = pdt) \right)$$

$$\land \left( \forall x : \text{Class}; y : \text{Attribute} \bullet \text{attribute}(x, y) \iff x = c \land y = a \right)$$

For the translation of a multiplicity constraint formula we define the following $\text{SubPCFOL}_m$ formulas for constraining the size of the set of elements in a relation with some other:

- $\min(n, R : D \times C)$ holds if for all $y : D$ exists $x_1, \ldots, x_n : C$ such that $R(y, x_i)$ for all $i \in \{1..n\}$, and $x_i \neq x_j$ for all $i, j \in \{1..n\}, i \neq j$.
- $\max(n, R : D \times C)$ holds if for all $y : D$ and $x_1, \ldots, x_{n+1} : C$, $\text{Rel}(y, x_i)$ for all $i \in \{1..n+1\}$ implies there is some $x_i = x_j$, $i, j \in \{1..n+1\}, i \neq j$.  

The first formula states that there are at least \( n \) different elements related to every element \( y \) by the relation \( R \), which represents a minimal cardinality for the relation. The other formula states that there are no more than \( n \) elements related to any element \( y \) by the relation \( R \), which represents a maximal cardinality for the relation. Using these \( \text{SubPCFOL}^\text{m} \) formulas, we can translate any multiplicity constraint formula as follows:

- \( n \leq \#D \cdot R \) is translated to \( \min(n, R : D \times C) \)
- \( \#D \cdot R \leq n \) is translated to \( \max(n, R : D \times C) \)
- \( \#D \cdot R = n \) is translated to \( \min(n, R : D \times C) \wedge \max(n, R : D \times C) \)

such that \( R : D \times C \in P \) are the predicates generated by the signature translation of the institution cohomism. In the case of \( C \cdot Q \) the predicate \( Q : C \times D \) is used instead of \( R : D \times C \).

**Example 14.** The formula \( \#(\text{UMLModelElement} \cdot \text{name}) = 1 \), defining the multiplicity constraint of property name, is translated to the conjunction of \( \min(1, \text{name} : \text{UMLModelElement} \times \text{String}) \) and \( \max(1, \text{name} : \text{UMLModelElement} \times \text{String}) \) which is represented in CASL syntax as follows

- \( (\forall x_1 : \text{UMLModelElement} \cdot \exists y_1 : \text{String} \cdot \text{name}(x_1, y_1)) \)
- \( \wedge \forall x_1 : \text{UMLModelElement} ; y_2, y_1 : \text{String} \cdot \text{name}(x_1, y_2) \wedge \text{name}(x_1, y_1) \Rightarrow y_2 = y_1 \)

The translation of OCL formulas is based on the mapping from OCL to first-order logic in [24]. Let define by \([Ex]\) the result of translating an OCL formula. The translation is defined by structural recursion on the expressions, but some of the cases contribute to the enclosing outermost existential quantification:

- a formula context Type inv: Ex is translated to a \( \text{SubPCFOL}^\text{m} \) formula \( \forall self : Type . [Ex] \)
- a formula \( [x_1]_{1 \leq i \leq n} x_p = Ex \) is translated to a \( \text{SubPCFOL}^\text{m} \) formula \([x_1]_p = [Ex] \)

For the translation of \([Ex]\) we must consider that collection expressions and role navigations can be composed. In these cases, the translation is done from left to right. The left hand side of the expression should define a bag with type according to the type of the expression, which is considered within the right hand side, as shown in **Example 15.** The translation is defined as follows.

- Literal values and variables (including self) are basically copied.
- Expressions involving \( and, or, not, and implies \) are translated to formulas mirroring their structure.
- Expressions involving \( <, >, =, <=, \) and \( + \) are also translated to formulas mirroring their structure but with respect to the literal values, variables or bags defined by the expressions in both sides of the symbols, e.g. \([Ex_1 = Ex_2]\) is translated to \([Ex_1] \wedge [Ex_2] \rightarrow s_1 = s_2\) when \( s_1 \) and \( s_2 \) are the main bags defined by \([Ex_1]\) and \([Ex_2]\); it is translated to \([Ex_1] \rightarrow s_1 = [v]\) when \( s_1 \) is the main bag defined by \([Ex_1]\), and \([Ex_2]\) is translated to a variable \( v\); and it is translated to \( f = l\) when \([Ex_1]\) is translated to a function \( f\), and \([Ex_2]\) is translated to a literal value \( l\).
- if \( Ex_1\) then \( Ex_2\) else \( Ex_3\) endif is translated to the expression \([Ex_2]\) when \([Ex_1]\) else \([Ex_3]\).
- Type.allinstances() is translated to the function Type.allinstances already defined.
- If \( Ex\) is a variable, \( Ex.Name\) is translated to \( Name([Ex])\) which is the function defined for accessing elements related through properties. If not, it is translated to \([Ex] \rightarrow \exists s_2 : \text{Bag}[T_2]. \forall x : T_2 \cdot x \text{ eps} s_2 \leftrightarrow \exists y : T . y \text{ eps} s \wedge \text{Name}(y, x)\) such that \( s : \text{Bag}[T]\) is the main bag defined by \([Ex]\), and \( T_2\) is the target type of the property \( Name\). If \([Ex]\) is translated to a function, then \([Ex] \rightarrow \) is omitted and the function is used instead of \( s\).
- The collection expressions \( Ex->size, Ex->includes, Ex->excluding,\) and \( Ex->including\) are translated to expressions of the form \([Ex] \rightarrow \text{colexp}(s)\), with \( \text{colexp}\) the corresponding built-in function and \( s : \text{Bag}[T]\) the main bag defined by \([Ex]\). If \([Ex]\) is translated to a function, then \([Ex] \rightarrow \) is omitted and the function is used instead of \( s\).
- Finally, the other collection expressions \( ([Ex], \exists, \text{select, and collect})\) are translated as follows. Assuming that \( s : \text{Bag}[T]\) is the main bag defined by \([Ex]\), we have that:
- \( Ex->forall\ e \mid b\) is translated to \([Ex] \rightarrow \forall e : T . e \text{ eps} s \rightarrow [b]\).
- \( Ex->exists\ e \mid b\) is translated to \([Ex] \rightarrow \forall e : T . e \text{ eps} s \wedge [b]\).
- \( Ex->select\ e \mid b\) is translated to \([Ex] \rightarrow \exists s_2 : \text{Bag}[T]. \forall e : T . (e \text{ eps} s \wedge [b]) \leftrightarrow e \text{ eps} s_2\).
- \( Ex->collect\ b\) is translated to \([Ex] \rightarrow \exists s : \text{Bag}[T]. (\forall e : T . e \text{ eps} s \leftrightarrow [b])\).
- If \([Ex]\) is translated to a function, then \([Ex] \rightarrow \) is omitted and the function is used instead of \( s\). If there are more than one variable \( e_i\) within the collection expressions, we write \( \bigwedge_i e_i . \text{eps} s \) in the above translations.

**Example 15.** The OCL formulas defined in **Example 5** are translated step-by-step as follows:

```plaintext
context Class inv: self.attribute->forall(a1 : Attribute; a2 : Attribute | 
   a1 <> a2 implies a1.name <> a2.name)
```
This formula has the following general structure: context Class inv: Ex→forall(e | b). It is translated to a SubPCFOL™ formula following these steps

1. ∀ self: Class . [Ex→forall(e | b)]
2. ∀ self: Class . [Ex] → ∀ e : T . e eps s → [b]
   - s : Bag[T] is the main bag that must be defined by [Ex] and is of the same type T than the variables e (type Attribute).
3. ∀ self: Class . ∀ a1, a2: Attribute . a1 eps attribute(self) ∧ a2 eps attribute(self) → [b]
   - [Ex] = [self.attribute] = attribute(self) since self is a variable and attribute is a function for accessing a property. Moreover, [Ex] → is omitted and attribute(self) is used instead of s
   - there is more than one variable a_1 (represented as e ) of type Attribute within the collection expression, thus we write \( \bigwedge_i a_i \) eps attribute(self).
4. ∀ self: Class . ∀ a1, a2: Attribute.
   - a1 eps attribute(self) ∧ a2 eps attribute(self) → (¬ a1 = a2 → ¬ name(a1) = name(a2))
   - [b] = [a1 -> a2 implies a1.name -> a2.name] = ¬ a1 = a2 → ¬ name(a1) = name(a2). This is a straightforward translation, using the facts that [a1.name] = name(a1), [a2.name] = name(a2) and that both sides of the inequalities are functions.

```
context Table inv: Table.allInstances() -> forall(t : Table | t.key.column.owner = t)
```

This formula has the same structure as before and it is translated to a SubPCFOL™ formula following these steps

1. ∀ self: Table . [Ex→forall(e | b)]
2. ∀ self: Table . [Ex] → ∀ e : T . e eps s → [b]
   - s : Bag[T] is the main bag that must be defined by [Ex] and is of the same type T than the variables e (type Table).
3. ∀ self: Table . ∀ t : Table . t eps Table.allInstances() → [b]
   - [Ex] = [Table.allInstances()] = Table.allInstances which is the function for accessing every element of type Table. Moreover, [Ex] → is omitted and Table.allInstances is used instead of s.
4. ∀ self: Table . ∀ t : Table . t eps Table.allInstances() → (t.key.column.owner) s = [t]
   - [b] = [t.key.column.owner = t] = [t.key.column.owner] s = [t]. This is an equality such that the left hand side provides a bag s : Bag[Table] and t is a variable.
5. ∀ self: Table . ∀ t : Table . t eps Table.allInstances() →
   - (∃ s1 : Bag[Column]. ∀ x : Column . x eps s1 ≤t y : Key . y eps key(t) ∧ column(y, x))
   - (∃ s2 : Bag[Table]. ∀ x2 : Table . x2 eps s2 ≤t y2 : Column . y2 eps s1 ∧ owner(y2, x2)) → s2 = [t]
   - [t.key.column.owner] = [[t.key].column].owner]. The composition is translated from left to right.
   - [t.key] = key(t) since t is a variable and key is a function for accessing a property.
   - [[t.key].column] = ∃ s1 : Bag[Column]. ∀ x : Column . x eps s1 ≤t y : Key . y eps key(t) ∧ column(y, x) such that key(t) is the main bag defined in the left hand side of the expression [[t.key]]; since it is a function, the expression [[t.key]] → is omitted and key(t) is used instead of s.
   - [[Ex].owner] = [Ex] → ∃ s2 : Bag[Table]. ∀ x2 : Table . x2 eps s2 ≤t y2 : Column . y2 eps s1 ∧ owner(y2, x2) such that s1 is the main bag defined in the left hand side of the expression (in the above item). Since [Ex] is not a function, it is not omitted from the expression [Ex] →, and s2 is used in the right hand side.

Finally, we can prove that \( ρ^\text{Sen} \) is indeed a natural transformation.

**Lemma 7.** The function \( ρ^\text{Sen} \) is a natural transformation from the functor of CSMoF^OCL formulas and translations to the functor of formulas and translations in SubPCFOL™, composed with the signature translation functor.

**Proof.** This means that there is a family of arrows \( ρ^\text{Sen}_A : \text{Sen}^\text{CSoMf}^\text{OCL} (A) → \rho^\text{Sign}_\text{Sen}^\text{SubPCFOL} (A) \), one for each signature \( A \) of \( \text{Sign}^\text{CSoMf}^\text{OCL} \), such that, for every signature morphism \( σ : A → B \) it holds: \( ρ^\text{Sign}_\text{Sen}^\text{SubPCFOL} \circ ρ^\text{Sen}_A = ρ^\text{Sen}_B \circ \text{Sen}^\text{CSoMf}^\text{OCL} \).

Consider any formula \( φ \) in \( \text{Sen}^\text{CSoMf}^\text{OCL} \) (multiplicity constraint, SW-model formula or OCL formula). By definition, we have that any signature morphism \( σ \), applied to a formula in \( \text{Sen}^\text{CSoMf}^\text{OCL} (σ) \) is the canonical application of the signature morphism to every type and role in the formula. Moreover, its composition with \( ρ^\text{Sen}_B \) gives a formula which uses the translation of types and roles given by the original signature morphism. In the other case, if the translation \( ρ^\text{Sen}_A \) is applied first, we got a similar formula (with the same structure) but with the translation of types and roles in the source signature. Then, its composition with the signature morphism \( ρ^\text{Sign}_\text{Sen}^\text{SubPCFOL} (σ) \) only changes the type and role names according to the signature morphism, as defined, which corresponds to the same formula obtained in the other case. Thus, the property holds. Finally, \( ρ^\text{Sen} \) is a natural transformation. □
5.4. Translation of interpretations

Given a $\text{CsmofOcl}$ signature $\Sigma$, an $\text{SubPCFOL}^\Sigma$ model $M$ of its translated theory ($\Sigma', E$) is reduced to a $\Sigma$-interpretation denoted $I = (V_C^T(O), A)$ where:

- Each non-empty carrier set $|M|_c$ with $c \in T(C)$, is translated to the set $V_c := |M|_c$ in the object domain $V_C^T(O)$. The axiom that the functor $\rho^\Sigma$ adds for every $c \in \alpha$ ensures that $c_2 \in \alpha$ implies $O_{c_2} = \bigcup c_1 \subseteq c_2 O_{c_1}$.
- Each relation $p_M$ of a predicate symbol $r_2(c_1, c_2) \in P$ derived from the translation of a predicate $(r_1 : c_1, r_2 : c_2)$, is translated to the relation $p^T \subseteq V_{c_1} \times V_{c_2} \in A$.

We can prove now that $\rho^\text{Mod}$ is indeed a natural transformation, and also that the satisfaction condition holds.

**Lemma 8.** The function $\rho^\text{Mod} : (\rho^\text{Sign})^{\text{op}}, \text{Mod}^{\text{SubPCFOL}} \rightarrow \text{Mod}^{\text{CsmofOcl}}$, with $\text{Mod} : \text{Th} \rightarrow \text{Cat}$ the functor giving the category of models of a theory, is a natural transformation.

**Proof.** For any model in $\text{Mod}^{\text{SubPCFOL}} (\rho^\text{Sign}(B))$, the translation $\rho^\text{Mod}_B$ gives an interpretation denoted $I = (V_C^T(O), A)$ such that:

- Each non-empty carrier set $|M|_c$ with $c \in T(C)$, is translated to the set $V_c := |M|_c$ in the object domain $V_C^T(O)$.
- Each relation $p_M$ of a predicate symbol $r_2(c_1, c_2) \in P$ derived from the translation of a predicate $(r_1 : c_1, r_2 : c_2)$, is translated to the relation $p^T \subseteq V_{c_1} \times V_{c_2} \in A$.

By definition of reduct, the application of $\text{Mod}^{\text{CsmofOcl}}$ to this interpretation gives the same interpretation. In the other side, the reduct $\text{Mod}^{\text{SubPCFOL}} (\rho^\text{Sign}(\alpha))$ gives an interpretation of symbols of the translated signature $A$, and its composition with the translation $\rho^\text{Mod}_A$ produces the same interpretation as before, since the carries sets $|M|_s$ with $s \in S$, and the relations $p_M$ are those derived from the translation of elements in the translates signature, which was reduced to the elements in the signature $A$.

In the case of homomorphisms the property also holds. The translation of a homomorphism is defined in conformance with the original homomorphism, and the reduct gives the same homomorphism. For a homomorphism in $\text{Mod}^{\text{SubPCFOL}}$ we have that $h'_\varphi(c) = h^\text{Sign}(\varphi(c))$ and the composition with the reduct gives a homomorphism $h_c = h^\text{Sign}(\varphi)$. In the other side, the reduct of a homomorphism in $\text{Mod}^{\text{SubPCFOL}}$ gives a homomorphism $h^\text{Sign}(\varphi) = h_{\rho^\text{Sign}(\varphi)}$, and its translation gives a homomorphism $h_c = h^\text{Sign}(\varphi)$ which is the same as before. Finally, $\rho^\text{Mod}$ is a natural transformation. □

**Theorem 9** (Preservation of the Satisfaction Relation). For any signature $\Sigma \in |\text{Sign}^{\text{CsmofOcl}}|$ the translations $\rho^\text{Sen}_\Sigma$ of sentences, and $\rho^\text{Mod}_\Sigma$ of models, preserve the satisfaction relation, that is, for any $\varphi \in \text{Sen}^{\text{CsmofOcl}}(\Sigma)$ and $M \in \text{Mod}^{\text{SubPCFOL}} (\rho^\text{Sign}(\Sigma))$:

$$M \vdash_{\text{SubPCFOL}} \varphi \iff \rho^\text{Sign}_\Sigma(\varphi) \vdash \rho^\text{Mod}_\Sigma(M) \vdash_{\text{CsmofOcl}} \varphi$$

**Proof.** Let’s consider that, according to the translation of $\text{SubPCFOL}^\Sigma$ models into $\text{CsmofOcl}$ models, there is a direct correspondence between the elements in $M$ and $\rho^\text{Mod}_\Sigma(M)$ and they are related in the same way.

In the case of a multiplicity formula, we know that $\rho^\text{Mod}_\Sigma(M)$ satisfies $\varphi$ if one of the following conditions holds:

- $\varphi$ is $\#(c \bullet r) = n$ and $|S| = n$ for all $S \subseteq (c \bullet r)^\text{mod}(M)$
- $\varphi$ is $\#(c \bullet r) \leq n$ and $|S| \leq n$ for all $S \subseteq (c \bullet r)^\text{mod}(M)$
- $\varphi$ is $\#(c \bullet r) \leq n$ and $|S| \leq n$ for all $S \subseteq (c \bullet r)^\text{mod}(M)$

But since the elements in the original and the translated model are similar, we have that the cardinality $|S|$ is conserved after the translation. Since this happens, we know that, for example in the second case, the translated formula $\text{min}(n, r : c \times d)$ holds in the original model, since for all $y : c$ exists at least $n$ different elements $x_1, \ldots, x_n : d$ related through the translated property $r(y, x_i)$. The other two cases are similar.

In the case of a SW-model formula, we know that $\rho^\text{Mod}_\Sigma(M)$ satisfies $\varphi$ if there is an isomorphism between $\varphi$ and the explicit scope defined by the reduced interpretation with respect to the types in the SW-model (see [4]). We also know that the translation $\rho^\text{Sign}_\Sigma(\varphi)$ defines existential quantified variables together with formulas constraining $M$, e.g. the “distinguishability” and “completeness of elements” axioms. In this sense, $M$ satisfies $\rho^\text{Sen}_\Sigma(\varphi)$ only if there is also an isomorphism between them.

In the case of an OCL formula, we have that $\rho^\text{Mod}_\Sigma(M)$ satisfies context Type inv: Ex if for every semantic element of type Type, we have that the expression Exp evaluates to true according to the operational semantics in Table 1. This
is exactly the requirement we impose when translating to SubPCFOL\(^m\). In the case of a formula \(|x_j|_{1 \leq j \leq n} \chi_x = \chi_{\chi_x}\), we require that the variable has the same value than the expression \(\chi_x\), which is the requirement we impose after translating the formula. Thus, the problem is reduced to prove that an OCL expression evaluates to an element in \(\rho_{\Sigma^m}^\text{Mod}(M)\) iff their translated version evaluates to the same element in \(M\). In the case of literal values and variables (including self) which are basically copied, and expressions involving and, or, not, implies, if-then-else, \(<\>, \leq, \geq\), and \(+\), which are translated to formulas mirroring their structure, we have that both formulas have the same semantics. In the case of Type\_allInstances\(), since it is translated to the function Type\_allInstances\) stating that every element of type Type belongs to the bag defined by the function, both expressions refer to the same set of elements. In the case of Ex.Name we have that the expression evaluates to the set of elements related to those from the result of evaluating \(\chi_x\) through the role Name. This is the same we declare with the existence of a bag of elements with every element related through the property Name with every other within the main bag defined by \(|\chi_x|\). In the case of the collection expressions, they all respect the semantics of the iterate constructor. This constructor basically iterates over a bag, performs a calculation with each element of the bag, and stores the results in an accumulator. As an example, in the case of size it keeps a counter which is increased with each element within the bag, in the case of forAll it returns the conjunction of evaluating a boolean expression in every element, and in the case of select it collects the elements satisfying a boolean expression. The translation of these kind of formulas follows this idea, i.e. the translated expression states the expected result of the accumulator. Since there is a direct correspondence between the elements in \(M\) and \(\rho_{\Sigma^m}^\text{Mod}(M)\), we can conclude that the accumulator returns the same elements.

Finally, we conclude that the satisfaction relation is preserved. \(\square\)

The comorphism admits model expansion since \(\rho_{\Sigma^m}^\text{Mod}\) is pointwise surjective on objects, i.e. each model of an CsmoR\(\text{Ocl}\)\-theory has a corresponding model in the translated theory within the SubPCFOL\(^m\) institution. Just consider any model \((V^T_1(O), A)\), we can construct an SubPCFOL\(^m\) model such that \(|M|_s\) corresponds to \(V_c\) with \(s\) the translation of type \(c \in T(C)\), and for each relation \(|r_1: c_1, r_2: c_2| \in A\) a pair of relations \(\rho_M\) (or just one if the relation is unidirectional). \(M\) also satisfies the extra axioms generated by the signature translation since elements coincide in both models respecting the subclass/subsort relation, and relations are generated in pairs satisfying the equivalence of the predicates. With this final result, we conclude that the comorphism admits borrowing of entailment for theories.

### 6. Encoding model transformations into Casl

We define now a simple theoroidal comorphism between the institution Qvtr\(\text{Ocl}\) introduced in Section 5.1 and SubPCFOL\(^m\). As before, the translation is given in terms of a functor \(\rho_{\Sigma^m}^\text{Sign}\) between Qvtr\(\text{Ocl}\) signatures and SubPCFOL\(^m\) theories, a natural transformation \(\rho_{\Sigma^m}^\text{Sen}\) such that key constraints and transformation rules formulas, as well as Ocl. formulas are translated to SubPCFOL\(^m\) formulas, and a natural transformation \(\rho_{\Sigma^m}^\text{Mod}\) translating interpretations and homomorphisms from SubPCFOL\(^m\) to Qvtr\(\text{Ocl}\). Along the definition, we illustrate the main concepts with the example presented in Section 4.

#### 6.1. Translation of signatures and formulas

Every Qvtr\(\text{Ocl}\) signature \((\Sigma^m_1, \Sigma^m_2)\) is translated by the functor \(\rho_{\Sigma^m}^\text{Sign}\) to a theory that disjointly unites the translations (as defined in Section 5) of the signatures \(\Sigma^m_1\). Since Qvtr\(\text{Ocl}\) is built over the institution Ocl, we can use the encoding of the latter into the Casl institution SubPCFOL\(^m\) defined in Section 5. For each signature morphism \((\sigma^m_1, \sigma^m_2)\), there is a signature morphism which disjointly unites the translations of \(\sigma^m_1\).

**Lemma 10.** The function \(\rho_{\Sigma^m}^\text{Sign} : \text{Qvtr}\text{Ocl} \rightarrow \text{Th}\text{SubPCFOL}^m\) is a functor from the category of Qvtr\(\text{Ocl}\) signatures and signature morphisms to the category of theories in SubPCFOL\(^m\).

**Proof.** Using the fact that signature morphisms are defined as the disjoint union of CsmoR\(\text{Ocl}\) signature morphisms, in which composition is preserved (by Lemma 6), and that the translation is defined componentwise, we can conclude that the functor preserves the composition of signature morphisms. Reasoning in the same way, we have that the translated signature morphism preserves the identities in SubPCFOL\(^m\). Thus, \(\rho_{\Sigma^m}^\text{Sign}\) is a functor. \(\square\)

In the case of a formula \(\phi^K\), the translated formula defines that there are not two different instances of that class with the same combination of properties conforming the key of such class. Formally, every formula \(\phi^K = (C, \{r_1, \ldots, r_n\})\) is translated to a formula of the form \(\forall x, y \in C, v_j : T_j, x \neq y \rightarrow \bigvee_i r_i(x, v_j) \rightarrow \bigvee_j r_i(y, v_j)\), with \(r_i(\ldots)\) one of the two predicates obtained from the translation of the property \(|r_1: c_1, r_2: c_2| \in P_j\) such that one of the roles is of type \(C\) and the other of type \(T_j\). In the case that \(r_i\) is the role of \(C\) in the property (because the opposite role is not navigable), we use \(r_i(v_j, x)\) instead of \(r_i(x, v_j)\).

**Example 16.** The translation of the key formula \((\text{Table}, \{\text{name}, \text{schema}\})\) is expressed using Casl syntax as follows:
Finally, the changed.

A formula $\varphi^R$ is translated as the conjunction of the translation of each top rule. Rules are translated according to the set of conditions stated by the semantics of QVT-Relations, which was explained in Section 4, i.e. a relation holds if for each valid binding of variables of the when clause and variables of domains other than the target domain, that satisfy the when condition and source domain patterns and conditions, there must exist a valid binding of the remaining unbound variables of the target domain that satisfies the target domain pattern and where condition. Formally, any rule $R = (\text{top}, \text{VarSet}, \text{ParSet}, \text{Pattern}, (i = 1, 2), \text{when}, \text{where}) \in \varphi^R$ is translated according to one of these cases:

1. If $\text{WhenVarSet} = \emptyset$
   \[
   \forall x_1, \ldots, x_n \in (\text{VarSet} \setminus \text{VarSet}) \setminus \text{ParSet}.
   (\rho^{\text{Sen}}(\text{Pattern}_1)) \rightarrow \exists y_1, \ldots, y_m \in 2_{\text{VarSet}} \setminus \text{ParSet}. (\rho^{\text{Sen}}(\text{Pattern}_2) \land \rho^{\text{Sen}}(\text{where})))
   \]

2. If $\text{WhenVarSet} \neq \emptyset$
   \[
   \forall z_1, \ldots, z_o \in \text{WhenVarSet} \setminus \text{ParSet}. (\rho^{\text{Sen}}(\text{when}) \rightarrow \forall x_1, \ldots, x_n \in (\text{VarSet} \setminus (\text{WhenVarSet} \cup 2_{\text{VarSet}})) \setminus \text{ParSet}.
   (\rho^{\text{Sen}}(\text{Pattern}_1) \rightarrow \exists y_1, \ldots, y_m \in 2_{\text{VarSet}} \setminus \text{ParSet}. (\rho^{\text{Sen}}(\text{Pattern}_2) \land \rho^{\text{Sen}}(\text{where})))))
   \]

The translation $\rho^{\text{Sen}}(\text{Pattern}_i)$ ($i = 1, 2$) of Pattern $i = (E_i, A_i, Pri_i)$ is the formula $\bigwedge_{i=1,2} (r_i(x, y) \land \rho^{\text{Sen}}(\text{Pri}_i))$ such that $r_2(x, y)$ is the translation of predicate $p = (r_1 : C, r_2 : D)$ for every $rel(p, x, y) \in A_1$ where $x : C, y : D$, and $\rho^{\text{Sen}}(\text{Pri}_i)$ is the translation of the $\text{Constr}^{\text{OCL}}$ formula into $\text{SubPCFOL}^\omega$. Moreover, the translation $\rho^{\text{Sen}}(\text{when})$ of when $= (\text{when}_{e}, \text{when}_{n})$ is the formula $\bigwedge \rho^{\text{Sen}}(\text{Rule}, v) \land \rho^{\text{Sen}}(\text{when}_{e})$ with $\text{Rule}, v \in \varphi^R$, such that $\rho^{\text{Sen}}(\text{Rule}, v)$ is the translation of Rule using variables in $v$ as the set of parameters $\text{ParSet}$; and $\rho^{\text{Sen}}(\text{when}_{e})$ is the translation of the $\text{Qvt}^{\text{OCL}}$-formula into $\text{SubPCFOL}^\omega$. The translation $\rho^{\text{Sen}}(\text{where})$ is straightforward.

**Example 17.** The rule PackageToSchema is translated to a formula stating that for each package there must be a schema with the same name, such that the schema satisfies the OCL expression. This is expressed using Casl syntax as follows.

\[
\forall p : \text{Package}; \forall pn : \text{String} \rightarrow \exists s : \text{Schema}; \forall cl : \text{Integer} \rightarrow \forall n(s, pn) \land \text{numC}(s, cl) \lor [\varphi]
\]

The OCL formula $\varphi$ defined in **Example 8** is translated to a $\text{SubPCFOL}^\omega$ formula following these steps. Notice that it cannot be evaluated by their own, but within some scope giving values to variables $p$ and $\text{pCl}$. This scope is defined by the pattern of the transformation rule to which this expression belongs.

\[
\text{pCl} = \text{Class.allInstances} \rightarrow \text{select}(c : \text{Class} \mid c.\text{kind} = \text{Persistent} \land c.\text{nameSpace} = p) \rightarrow \text{size}()
\]

1. $\text{pCl} = [[\text{Ex}_1] \rightarrow \text{select}(e \mid b] \rightarrow \text{size}()$)
2. $\exists s_2 : \text{Bag(Class)} \land \forall c : \text{Class} \rightarrow (c \in \text{Class.allInstances} \land [b]) \rightarrow c.\text{eps} s_2) \rightarrow \text{size}(s_2) = \text{pCl}$
   - $\text{Ex}_1] = [\text{Class.allInstances}] = \text{Class.allInstances}$ which is the function for accessing every element of type Class.
   - $[\text{Ex}_1] \rightarrow \text{size}() = [\text{Ex}_2] \rightarrow \text{size}(s_2)$ with $s_2 : \text{Bag(Class)}$ the main bag defined by $[\text{Ex}_2]$.
3. $\exists s_2 : \text{Bag(Class)} \land \forall c : \text{Class} \rightarrow (c \in \text{Class.allInstances} \land [b]) \rightarrow c.\text{eps} s_2$
   - $[b] = [c.\text{kind} = \text{Persistent} \land c.\text{nameSpace} = p] = \text{kind}(c) = \text{Persistent} \land \text{nameSpace}(c) = p$

**Lemma 11.** The function $\rho^{\text{Sen}}$ is a natural transformation from the functor of $\text{Qvt}^{\text{OCL}}$ formulas and translations to the functor of $\text{SubPCFOL}^\omega$ formulas and translations with the signature transformation functor.

**Proof.** This means that there is a family of arrows $\rho^{\text{Sen}}_A : \text{Sen}^{\text{Qvt}^{\text{OCL}}}(A) \rightarrow \rho^{\text{Sen}}_B : \text{Sen}^{\text{SubPCFOL}^\omega}(A)$, one for each signature $A$ of $\text{Sig}^{\text{OCL}}$, such that, for every signature morphism $\sigma : A \rightarrow B$ it holds: $\rho^{\text{Sen}} : \text{Sen}^{\text{SubPCFOL}^\omega} \circ \rho^{\text{Sen}}_A = \rho^{\text{Sen}}_B \circ \text{Sen}^{\text{Qvt}^{\text{OCL}}}$. We already proved in **Lemma 7** that $\rho^{\text{Sen}}$ is a natural transformation with respect to $\text{Constr}^{\text{OCL}}$ formulas and translations. Now we need to prove it in the case of formulas $\varphi^R$ and $\varphi^R$ in $\text{Sen}^{\text{Qvt}^{\text{OCL}}}$. By definition of signature morphism $\sigma$ and translation of formulas and morphisms $\rho^{\text{Sen}}$, we have that any morphism applied to any formula in $\text{Sen}^{\text{Qvt}^{\text{OCL}}} (\sigma)$ is the canonical transformation of the signature morphism $\rho^{\text{Sen}}$. If this morphism is composed with the translation $\rho^{\text{Sen}}_B$, we have a formula in $\text{Sen}^{\text{SubPCFOL}^\omega}$ preserving the names. In the other side, if we apply first $\rho^{\text{Sen}}_A$ we obtain the same formula but with the original names, and thus after applying the translated morphism (which is the canonical application of the translated signature morphism to every element in the translated formula) we get the same formula with the translated names. In conclusion, we have that the property holds. Finally, $\rho^{\text{Sen}}$ is a natural transformation. $\square$
6.2. Translation of interpretations

We need now to define how the natural transformation \( \rho_{\text{Mod}} \) is defined, i.e., how \( \text{SubPCFOL}^\omega \) models and homomorphisms are translated to \( \text{Qvtr}^{\text{Ocl}} \) interpretations and homomorphisms. Given an \( \text{Qvtr}^{\text{Ocl}} \) signature \( \Sigma \), a model \( M \) of its translated theory \( (\Sigma', E) \) is translated to a \( \Sigma \)-model \( M' = (M^1, M^2) \) by constructing two models with an interpretation of elements for each corresponding \( \text{Qvtr}^{\text{Ocl}} \) signature. Each \( M_i^j \) \((i, j) \) is defined as in Section 5. Moreover, given two models of its translated theory \( (\Sigma', E) \), and let \( h' : M \to M' \) be a \( \Sigma \)-homomorphism. Let us denote \( N' = \rho_{\text{Mod}}(M) \) and \( N' = \rho_{\text{Mod}}(M') \) and let us define \( h : N \to N' \) as the disjoint translation of \( h' \), as in Section 5, with respect to the elements in the corresponding signatures. The disjoint union is a homomorphism since each translated function is a homomorphism.

We can prove that \( \rho_{\text{Mod}} \) is indeed a natural transformation, and that the satisfaction condition holds.

Lemma 12. The function \( \rho_{\text{Mod}} : (\Phi)^{\text{ap}}, \text{Mod}_{\text{SubPCFOL}^\omega} \to \text{Mod}_{\text{Qvtr}^{\text{Ocl}}} \), with \( \text{Mod} : \text{Th} \to \text{Cut} \) the functor giving the category of models of a theory, is a natural transformation.

Proof. Given any model in \( \text{Mod}_{\text{SubPCFOL}^\omega}(\rho_{\text{Sign}}(B)) \), the translation \( \rho_{\text{Mod}} \) gives a pair of disjoint models. By definition of reduct, the application of \( \text{Mod}_{\text{Qvtr}^{\text{Ocl}}} \) gives the same pair of models. Moreover, the reduct \( \text{Mod}_{\text{SubPCFOL}^\omega}(\rho_{\text{Sign}}(\sigma)) \) gives an interpretation of symbols of the translated signature \( A \), and the translation \( \rho_{\text{Mod}} \) just divide the model to two disjoint ones with respect to the two parts of the signature \( B \). Thus, the property holds. In the case of homomorphisms the property also holds, since the translation of a homomorphism is defined as a disjoint translation with respect to the elements in the corresponding signatures, and the reduct of a homomorphism is defined component-wise. This means that the translation gives the same homomorphism between elements as two disjoint functions, and it composition with the reduct gives a reduced homomorphism with respect to the elements in the source signature. This is equal to the reduct of the homomorphism with respect to the elements in the source signature and then its separation to two disjoint functions. Finally, \( \rho_{\text{Mod}} \) is a natural transformation.

Theorem 13 (Preservation of the satisfaction relation). For any signature \( \Sigma \in |\text{Sign}_{\text{Qvtr}^{\text{Ocl}}} \) the translations \( \rho_{\text{Sign}}(\sigma) \) of sentences, and \( \rho_{\Sigma}^{\text{Mod}} \) of models, preserve the satisfaction relation, that is, for any \( \varphi \in \text{Sen}_{\text{Qvtr}^{\text{Ocl}}}(\Sigma) \) and \( M \in \text{Mod}_{\text{SubPCFOL}^\omega}(\rho_{\text{Sign}}(\Sigma)) \):

\[
M \models_{\text{Sen}_{\text{Qvtr}^{\text{Ocl}}}^{\text{SubPCFOL}^\omega}(\Sigma)} \rho_{\Sigma}^{\text{Mod}}(M) \models_{\text{Qvtr}^{\text{Ocl}}} \rho_{\Sigma}^{\text{Sen}}(\varphi)
\]

Proof. We already proved in Theorem 9 that the satisfaction relation is preserved in the case of \( \text{Csmof}^{\text{Ocl}} \) formulas. Now we need to prove it in the case of formulas \( \varphi^K \) and \( \varphi^R \) in \( \text{Sen}_{\text{Qvtr}^{\text{Ocl}}} \).

By definition of the satisfaction of a formula \( \varphi^K = (c, [r_1, \ldots, r_n]) \), we have that it is satisfied by a model \( \rho_{\Sigma}^{\text{Mod}}(M) \) if the sets of elements within the model related with any two elements of type \( c \) through relations interpreting properties \( \langle r : c, r_j : c_2 \rangle \), differ in at least one element. The translation \( \rho_{\Sigma}^{\text{Mod}}(M) \) basically split the elements in two \( \text{Csmof}^{\text{Ocl}} \) models such that there is a direct correspondence between the elements in \( M \) and \( \rho_{\Sigma}^{\text{Mod}}(M) \) and they are related in the same way (according to the translation of \( \text{SubPCFOL}^\omega \) models into \( \text{Csmof}^{\text{Ocl}} \) models). Thus, we have that the sets of elements within the original model \( M \), related with any two elements in the carrier set \( |M|_c \) through relations derived from translated properties \( \langle r : c, r_j : c_2 \rangle \), also differ in at least one element. Finally, we can conclude that the property is satisfied for the translated formula \( \rho_{\Sigma}^{\text{Sen}}(\varphi^K) \).

In the case of a formula \( \varphi^R \) we have a similar situation with respect to the translation of models. Moreover, in this case, the formula is satisfied if for every top rule, one of the cases of the satisfaction relation holds. Since the translated formula is the conjunction of the translation of top rules with respect to a direct representation of this satisfaction relation, we have equivalent proof obligation. By considering both results, we can conclude that the satisfaction relation is also preserved for \( \varphi^R \) formulas.

Since each model of an \( \text{Qvtr}^{\text{Ocl}} \) theory has a correspondent model in the translated \( \text{SubPCFOL}^\omega \) theory, as proved in Section 5, the property holds for the disjoint union of models. Thus, the comorphism admits model expansion and thus borrowing of entailment.

7. The environment in action

We have implemented a prototype of our environment using the Heterogeneous Tools Set (Hers [6,10]), an open source software providing a general framework for formal methods integration and proof management, based on the Theory of Institutions. Hers consists of logic-specific tools for the parsing and static analysis of basic logical theories written in the different involved logics (e.g. our \( \text{Csmof}^{\text{Ocl}} \) and \( \text{Qvtr}^{\text{Ocl}} \) institutions), as well as a logic-independent parsing and static analysis tool for structured theories and theory relations. Proof support for other logics can be obtained by using logic translations defined by comorphisms (e.g. from \( \text{Csmof}^{\text{Ocl}} \) to \( \text{SubPCFOL}^\omega \)). We provided Hers with specific institutions for the
specification of MDE elements. With this approach we have only one generic representation of the MDE elements which is formally (and automatically) translated into SubPCFOL™ or any other connected logic when needed. Within this prototype, MDE experts can specify model transformations in their domain and such specifications can be complemented by verification experts with other properties to be verified. All this information is taken by Hets, which performs automatic translations of proof obligations into other logics and allows selecting the corresponding prover to be used, whilst a graphical user interface is provided for visualizing the whole proof. In other words, we provided to MDE practitioners the “glue” they need for connecting their domain with the logical domains needed for verification.

7.1. Structured and heterogeneous specification and refinement

In Section 2, we have defined theories for an arbitrary institution. Larger theories should be written in a structured, modular way. A simple kernel language for structured specifications over an arbitrary institution has been introduced in [25] and extended to heterogeneous specifications in [26,27]:

\[ SP ::= \langle \Sigma, \Psi \rangle | SP \cup SP | SP \text{ with } \sigma | SP \text{ hide } \sigma | SP \text{ with } \rho | SP \text{ hide } \mu \]

where \( \sigma \) is a signature morphism, \( \rho \) an institution comorphism and \( \mu \) a so-called institution morphism [16], i.e. a translation between institutions as in Definition 2, but with the natural transformation \( \rho^{\text{sem}} \) and the natural transformation \( \rho^{\text{Mod}} \) going in the opposite direction. This simple but powerful kernel language also provides the heart of the structuring constructs of Casl [23], as well as the heterogeneous constructs of the distributed ontology, modeling and specification language (DOL [28]), which provides an extension of Casl.

On top of structured (and possibly heterogeneous) specifications, it is easy to define a notion of simple refinement as model class inclusion, i.e. a specification \( SP_1 \) refines to \( SP_2 \), written \( SP_1 \rightarrow SP_2 \), if \( \text{Mod}(SP_2) \subseteq \text{Mod}(SP_1) \). Note that more complex forms of refinement (e.g. involving signature morphisms or institution comorphisms) can be obtained by using suitable structuring constructs in \( SP_1 \) and/or \( SP_2 \). Still more complex refinements involve branching, i.e. decomposition of an implementation task into several subtasks, as well as refinements of networks of specifications, see [29,30,28] for details.

Further work is needed in order to evaluate the use of these capabilities to support model refinement and hierarchical modeling requirement in MDE.

7.2. How the environment works

Our problem is stated as a heterogeneous specification using Casl and DOL structuring constructs [23]. Within such specifications it is possible to select the current institution using the keyword logic. We have at least three logics: Casl, CSMOF and QVTR, the last two corresponding to the institutions presented in Section 5.1. When a logic is used, Hets applies logic-specific parsers for parsing and static analysis. We also perform logic translations through the theoroidal comorphisms defined in Section 5 and Section 6, which are CSMOF2CASL and QVTR2CASL, respectively. Next, we show an excerpt of the heterogeneous specification of the example.

(1) \textbf{logic CSMOF}
(2) \textbf{from UML get UML} \Leftrightarrow UMLMODEL
\textbf{from UML\_WMult get UML} \Leftrightarrow UMLCONSTRAINTS
(3) \textbf{spec UMLPROOF = UMLMODEL}
\textbf{then \%implies}
\textbf{UMLCONSTRAINTS}
\textbf{end}
(4) \textbf{logic QVTR}
(5) \textbf{from uml2rdbms get UML2RDBMS} \Leftrightarrow QVTMODELS
\textbf{from uml2rdbms\_WForm get UML2RDBMS} \Leftrightarrow QVTTRANSFORMATION
(6) \textbf{spec ModelTransformation = QVTMODELS}
\textbf{then \%implies}
\textbf{QVTTRANSFORMATION}
\textbf{end}
(7) \textbf{logic CASL}
(8) \textbf{spec MoreProofs = UMLMODEL with logic CSMOF2CASL}
\textbf{then \%implies}
\forall x,y : Classifier; str : String \bullet \text{name}(x, str) \land \text{name}(y, str) \Rightarrow x = y
\textbf{end}

In (1) we state that we are using the CSMOF logic. In (2) we create two CSMOF-theories (specifications) from XMI files with the information of the class metamodel and the class SW-model in Fig. 1. This implies the creation of a Haskell
representation of signatures and formulas according to the institution CsmofOcl. Both theories only differ in the formulas, i.e. the first theory UMLModel does not have any OCL invariant or multiplicity constraint, but a concrete SW-model. The same is done with the RDBMS information, which is not shown in this example. In (3) another specification is created by extending UMLModel and stating that UMLConstraints is implied. This means that every formula (OCL expressions and multiplicity constraints) in the second specification can be derived from the SW-model in the first specification, thus there must be a proof of UMLModel $\implies$ UMLConstraints. This is how logical entailment in the institution CsmofOcl is checked. Notice that for developing the proof, the comorphism CsmofOcl $\implies$ (or any other if defined) must be called since CsmofOcl does not have any specific proof system.

In (4) we state the use of the QVTR logic and in (5) we create two specifications from a standard .qvt file according to the institution QVTROcl. The model transformation is specified using the same language defined within the QVT standard (e.g. as the one in the running example) together with our OCL expressions language. This step also loads the XML files containing the information of source and target metamodels for constructing the signature of the institution, as done in (2). We use the name of the source and target metamodel in the transformation specification for finding the corresponding files. These specifications differ in the formulas. QVTModels define concrete source and target SW-models whilst QVTTransformation define keys and transformation rule formulas. In (6) we define another specification by extending QVTModels and stating that QVTTransformation is implied. This means that keys and transformation rules can be derived from the SW-models within QVTModels. As in (3), we need a proof for each implied formula.

Finally, we can also translate our specifications and complement them with other constraints which cannot be stated as formulas of the former institutions. As an example, in (7) we move into Casl and in (8) we state that there cannot be two Classifiers with the same name in the UMLMetamodel specification. For this purpose we are using the CsmofCASL comorphism. Again, as in (3), a proof of it must be given. Notice that we can use any other logic within the logics graph of Hets through existing comorphisms, improving proof capabilities.

Once our heterogeneous specification is processed, Hets constructs a heterogeneous development graph in which nodes correspond to specifications, some of them with open proof obligations, and arrows correspond to dependencies between them. Proof goals can be discharged using a logic-specific calculus. In the example we have three proof obligations which correspond to those formulas marked as % implies within the specifications. Since we do not have any specific proof system for our institutions, Hets allows to apply our comorphisms to translate the proof goals into Casl. Finally, those proof goals can be discharged using any theorem proving system for Casl (e.g. SPASS [31]).

7.3. Verification properties

When verifying transformations, there are several properties of interest, some of them related to the computational nature of transformations and target properties of transformation languages, and other to the modeling nature of transformations, as we exhaustively studied in [3].

The minimal requirement is conformance, i.e. that the source and target SW-models (resp. the transformation specification) are syntactically well-formed instances of the source and target metamodels (resp. the transformation language). Our framework provides this verification in several places. During the construction of the CsmofOcl and QVTOCL theories, parsing and static analysis check whether signatures and formulas are well-formed. Moreover, an SW-model within a signature is a structurally well-formed instance of the metamodel in the same signature, as well as a transformation specification given in a formula is well-formed with respect to the signature containing both source and target metamodels. Finally, our framework provides CsmofOcl formulas which contain the conformance with respect to OCL invariants and multiplicity constraints.

One interesting point is that Hets also allows for disproving goals using consistency checkers, which provides an additional point of view in the verification process. In this sense, a goal can also be marked as disproved or a theory as inconsistent. In particular, an inconsistent set of transformation rules means that there are contradictory conditions which could inhibit the execution of the transformation. A very simple example of this is the following specification in which we state that every package contains only one classifier. This can be disproved since package P has two classifiers: class c and primitive datatype pdc.

spec INVALIDPROPERTY = UMLModel with logic Csmof2CASL
then %implies
  \forall x : Package; y,z : Classifier \in elements(x, y) \land elements(x, z) \implies y = z
end

In most cases a general-purpose logic, as the one provided by Casl, is enough to cover most of the verification approaches presented in [3]. However, the verification process may depend on the problem to verify, since it is well-known that automatic proofs are not always possible. In this sense, in Hets it is possible to choose the tool we want to use, e.g. we can choose not to use an automated theorem proving system, but select for example an interactive theorem prover, e.g. Isabelle.

Sometimes when verifying a model transformation, we want to consider its elements as a whole and not individually. The notion of a transformation model is used, i.e. a SW-model composed by the source and target metamodel, the transformation specification and the well-formedness rules. A QVTOCL theory (QVTTransformation in the example) is a
transformation model which allows to add additional properties by combining elements from the source and target metamodels and SW-models. With this we can state model syntax relations, trying to ensure that certain elements of any input SW-model will be transformed into other elements of the output SW-model. This need arises when, for example, these relations cannot be inferred by just looking at the individual transformation rules. We can also state model semantics relations, e.g. temporal properties and refinement. Although further work is needed to evaluate the alternatives, there are languages and tools already in Hets, as Modal Casl and VSE (based on dynamic logics) commonly used for verifying these kind of things.

We could also be interested in working at another abstraction level, i.e. not considering specific SW-models but only metamodels and the transformation specification. This can be useful, for example, for proving that a transformation guarantees some model syntax relations when transforming any valid source SW-model. One example of this is when proving termination and determinism of a transformation: the existence of a target SW-model for any execution and the uniqueness of such SW-model, respectively. In particular, termination and determinism properties can be basically stated as in the following Casl code.

\[
\forall \text{ma : ClaM} \bullet \text{Pre ma} \Rightarrow (\exists \text{mb : RelM} \bullet \text{Rules ma mb} \land \text{Post mb})
\]

\[
\forall \text{ma : ClaM} \bullet \text{Pre ma} \Rightarrow (\exists! \text{mb : RelM} \bullet \text{Rules ma mb} \land \text{Post mb})
\]

For any class SW-model satisfying the source constraints (Pre), there must be a relational SW-model satisfying the target constraints (Post) and the transformation rules (Rules). In the case of determinism we require that this SW-model is unique. These kind of problems are hard to verify automatically since the space of solutions for \( \text{ma : ClaM} \) is almost always infinite.

The problem here is that we need another institutional representation, somehow related to [32] in which our Csmof institution is based, in which models of the institution are not constrained by the SW-model within the signature. Moreover we need to generate as part of the comorphism an abstract representation of an SW-model (e.g. ClaM in the example). This is subject of future work.

8. Related work

There are some works that define environments for the comprehensive verification of MDE elements based on a unified mathematical formalism. As an example, in [33,34] rewriting logic is used to analyze MOF-like and QVT-like elements. Since Hets integrates rewriting logic, we can specify MDE elements directly using this proposal instead of using our logic-independent institution and then performing the translation into SubPCFOL\(\text{^m}\). Nevertheless, logic-independence provides more flexibility for the definition of further specific comorphisms into other logics and languages (e.g. UML). In general, the use of a fixed unified mathematical formalism serving as a unique semantic basis can be quite restrictive.

In [35] the authors define a language-independent representation of metamodels and model transformations supporting many transformation languages. They also define mappings to the B and Z3 formalisms. Since they use only one generic language, only one semantic mapping needs to be defined for each target formalism. However, the semantic mapping should be semantics-preserving, an aspect not formally addressed in such work. In our case, comorphisms already preserve the semantics with respect to the satisfaction relation. Moreover, our comorphism into SubPCFOL\(\text{^m}\) and the corresponding implementation in Hets, provides the possibility of connecting our institutions to several logics and tools.

Other works are based on the construction of a transformation model [36] which is a unified representation of the MDE elements using OCL contracts. In [37–39] the authors present how to build an OCL-based transformation contract for several model transformation languages. These SW-models can be easily checked and analyzed using readily available OCL model finders.

This work follows a general program to give a semantics to heterogeneous UML diagrams through institutions and comorphisms, which has been set up in [40–42]. In [40] the authors propose to represent metamodels as institutions, and the correctness of a transformation is stated in terms of the existence of an intermediate institution that relates the source and target institutions through some institution morphism. In [41] the authors define a heterogeneous approach to the semantics of UML, which provides the basis of our Csmof institution. In [42], we extend this approach to a relevant number of institutions; however, MOF and the details of OCL are not covered.

In [43] the authors provide an algebraic representation of MDE elements in which metamodels are represented as algebras and transformations as triple patterns. They devise two options for the implementation of this approach: either using a tool like Maude that would allow them to directly work on algebras, or specializing the approach to the case of graphs and using a graph transformation tool.

There are some works in the context of graph-based transformations, which are quite similar to other relational approaches such as QVT-Relational. In [44,45] the authors follow algebraic approaches for the definition of formal semantics. In [46,47] the authors define the application of triple graph grammars (TGG) for model transformations, and its verification through a translation into Isabelle/HOL.

With respect to the comorphisms, there are works representing the semantics of UML class diagrams with first-order logic, as in [48]. Since there are not so many alternatives for this representation, these works have similarities with our
representation of CSMOFOCL into SubPCFOL\textsuperscript{\textregistered}. However, unlike these works, we provide a formal proof of the correctness of the mapping with respect to a well-defined formal semantics for OCL [20]. In particular, the work in [48] is the nearest to ours from which we take many aspects, e.g. the “distinguishability” and “completeness of elements” axioms. In [49] the authors explain how class diagrams with OCL constraints can be translated into CASL. However, their definition is informally presented, while our use of a comorphism provides a correctness proof of the translation it encodes. In [50] the authors define a comorphism from UML class diagrams with rigidity constraints to ModalCASL (an extension of CASL). Since our CSMOFOCL institution is an adaptation of the institution for UML class diagrams, the comorphisms have some aspects in common, as the translation of formulas, but without the modal logic particularities. In [51] the authors present a formalization of metamodels in the NEREUS language and they explain how this formalization is translated into CASL. The final representation differs a lot from our, since for example each class and association is represented as a CASL specification.

Although ATL is another transformation language, it is somehow related to QVT-Relations. In this sense, the mappings from ATL to constructive type theory [52] and first-order logic [53] have some similarities with our mapping since they are also expressed as \( \forall \exists \) formulas following the standard checking semantics.

With respect to OCL, the semantics of OCL defined in the standard leads to many interpretations and problems which were widely studied in the last decade. Most proposals define the semantics in terms of a shallow embedding of OCL into others formal languages, e.g. rewriting logic [33], first-order logic [24,54] and higher-order logic [21], in a search for automatic checking tools. In most cases, formal semantics are given for an expressive subset of the language but not for the complete language. The most recent and complete semantic of OCL is Featherweight OCL [21]. Although it does not define a complete OCL library, it formalizes a powerful core of OCL 2.3 (also considering invalid and null values) as an embedding into Isabelle/HOL. In [18] the authors define institutional basis for the definition of OCL-like expression languages in a compositional manner. Besides an institution for OCL is not completely defined, there is a valuable approach to be associated with our institutional settings to remove the restrictions we have imposed. In [55] the author develops institutional basis to combine many-valued logics with other logical systems. In particular they define a generic institution which can be used as a semantic oriented framework for defining in a uniform way new concrete many-valued logical systems. This could be the basis for an alternative approach for the definition of an institution for OCL. We translated OCL formulas following the mapping defined in [24] and, unlike this work, we provided a formal proof of the correctness of the mapping.

Finally, there are several approaches to heterogeneous specification for traditional software development, but there is little tool support. CafeOBJ [56] is an institution-based approach providing a fixed cube of eight logics and twelve projections (formalized as institution morphisms), not allowing logic encodings (formalized as comorphisms). Thus, it is not an option for the definition of our environment. Moreover, HeteroGenius [57] is another institution-based framework allowing the interaction between different external tools for performing hybrid analysis of a specification. However, the framework is not formally defined nor available.

9. Conclusions and future work

We have presented the implementation of an environment for the formal verification of different MDE aspects using heterogeneous verification approaches, which is based on the ideas introduced in [5]. The environment was integrated into Herts by defining comorphisms from institutions representing MDE elements to CASL, a core language of Herts. The existent connections between Casl and other logics within Herts broadens the spectrum of logical domains in which the verification of MDE elements can be addressed. The environment supports a separation of duties between software developers (MDE and formal methods experts) such that a formal perspective is available whenever it is required. A developer imports MDE elements, supplement them with verification properties specified using any other logic supported by Herts, and perform the heterogeneous verification assisted by the tool. Since the implementation can generate a heterogeneous specification from the same files used by MDE practitioners, and there is no need of rewriting MDE building block in each logic involved, the environment is scalable without human assistance. Although our proposal is aligned with OMG standards, this idea can be potentially formalized for any transformation approach and language, which allows extending the approach as far as necessary. Finally, the environment is reliable since it is supported by a well-founded theory and by a mature tool in which there are several logics already defined.

We have also defined an institution for a small subset of OCL by extending our original CSMOF\textsuperscript{\textregistered} institution. However, we imposed certain restrictions. Although they seem reasonable when considering the institution OCL as part of a proof of concepts of the whole environment, they must be eliminated for a wider consideration of OCL, in particular addressing null and undefined values. As a future work, we need to consider the big picture of OCL, and in this sense, it could be useful to study how to integrate the institution setting for OCL-like expression languages defined in [18], as well as to derive a more complete OCL institution along the lines of Featherweight OCL [21], which provides semantics for a very powerful core of OCL.

We also expect to extend the institutions to include some elements not considered before and give them tool support, besides exploring other options for the verification of transformation properties. This will strengthen the formal environment for MDE. Since our institutions formalize languages strongly related with those in the UML ecosystem, it will be interesting to explore the possibility of integrating them with other languages, as those already defined as institutions in [41,42].

Finally, we need to continue bridging the gap between MDE and formal verification in terms of tool development in order to practitioners really be able to benefit from our approach. We can connect the definition of the MDE elements in any
popular tool with an automatic generation of the heterogeneous specification. Moreover, we could perform an automated verification of some properties (if possible) by running Hets in the background. This will be possible in the model-hub.org web platform based on Hets. We also need to improve feedback from existing formal tools, which needs better traceability between the problem definition and the results given by a verification tool. We can define some traceability links from comorphisms, interpret the output of the verification tool and return something that the MDE practitioner can interpret.

Moreover, the environment deals with many verification properties, but a deeper understanding of this (as for example about the behavior of models) is a must. In this sense, we can use the knowledge in [3] to provide a guide for the selection of the “right” verification approach for the problem which is of interest to verify. We also need to apply our approach to industrial, real-size examples for strengthening the results.

Acknowledgements

This work has been partly funded by the Agencia Nacional de Investigación e Innovación (ANII, Uruguay). We thank Alexander Knapp for cooperation on institutional semantics of UML.

References
