Ontologically Correct Taxonomies by Construction

Jeferson O. Batista[a], João Paulo A. Almeida[a,*], Eduardo Zambon[a], Giancarlo Guizzardi[b,c]

[a] *Federal University of Espírito Santo, Brazil*
[b] *Free University of Bozen-Bolzano, Italy*
[c] *University of Twente, The Netherlands*

Abstract

Taxonomies play a central role in conceptual domain modeling, having a direct impact in areas such as knowledge representation, ontology engineering, and software engineering, as well as knowledge organization in information sciences. Despite this, there is little guidance on how to build high-quality taxonomies, with notable exceptions being the OntoClean methodology, and the ontology-driven conceptual modeling language OntoUML. These techniques take into account the ontological meta-properties of types to establish well-founded rules on the formation of taxonomic structures. In this paper, we show how to leverage the formal rules underlying these techniques in order to build taxonomies which are *correct by construction*. We define a set of correctness-preserving operations to systematically introduce types and subtyping relations into taxonomic structures. In addition to considering the ontological micro-theory of endurant types underlying OntoClean and OntoUML, we also employ the MLT (Multi-Level Theory) micro-theory of high-order types, which allows us to address multi-level taxonomies based on the *powertype* pattern. To validate our proposal, we formalize the model building operations as a graph grammar that incorporates both micro-theories. We apply automatic verification techniques over the grammar language to show that the graph grammar is *sound*, i.e., that all taxonomies produced by the grammar rules are correct, at least up to a certain size. We also show that the rules can generate all correct taxonomies up to a certain size (a *completeness* result).

Keywords: taxonomies, conceptual modeling, ontologies, graph grammars, correctness by construction

1. Introduction

Taxonomies are structures connecting types via *subtyping*, i.e., type specialization relations. These structures are fundamental for conceptual domain modeling, and have a central organizing role in areas such as knowledge representation, ontology engineering, object-oriented modeling, as well as in knowledge organization in information sciences (e.g., in the construction of vocabularies and other lexical resources). Despite taxonomies' key role in all these areas, there is little guidance in the literature on how to build high-quality taxonomies.

*Corresponding author
Email addresses: jeferson.batista@aluno.ufes.br (Jeferson O. Batista), jpalmeida@ieee.org (João Paulo A. Almeida), zambon@inf.ufes.br (Eduardo Zambon), gguizzardi@unibz.it (Giancarlo Guizzardi)

A notable exception is OntoClean [1], a pioneering methodology providing a number of guidelines for diagnosing and repairing taxonomic relations that are inconsistent from an ontological point of view. These guidelines are grounded on a number of *formal meta-properties*, i.e., properties characterizing types. Derived from these meta-properties, the methodology offers a number of formal rules governing how types characterized by different meta-properties can be associated to each other in well-formed taxonomies.

OntoClean has been successfully employed to evaluate and suggest repairs to several important resources, e.g., WordNet [2]. However, being a methodology, OntoClean does not offer a representation mechanism for building taxonomies according to its prescribed rules. To address this problem, a UML profile [3] was proposed containing modeling distinctions extending the meta-properties and rules of OntoClean. This UML profile would later become the basis of the OntoUML modeling language [4], incorporating syntactic rules to prevent the construction of incorrect taxonomies in conceptual models. In [5], the language has its full formal semantics defined in terms of a (proved-consistent) ontological theory, and its abstract syntax is defined in terms of an extension of the UML 2.0 metamodel, redesigned to reflect the ontological distinctions and axiomatizations put forth by that ontological theory.

As argued in [6], instead of leveraging on this axiomatization by proposing methodological rules (OntoClean) or semantically motivated syntactical constraints (OntoUML), a representation system based on this ontological theory could employ a more productive strategy, leveraging on the fact that formal constraints from the theory impose a correspondence between each particular type of type (characterized by ontological meta-properties) and certain modeling *structures* (or modeling *patterns*). In other words, a representation system grounded on this ontological theory is a *pattern language*, i.e., a system whose basic building blocks are not low-granularity primitives such as types and relations, but higher-granularity patterns *formed* by types and relations. In [7], the authors made a first attempt to formalize such representation system as a true *pattern grammar*, i.e., as a graph grammar that reflects the underlying theory of endurant types from OntoUML, with the grammar capturing the language patterns and their possible relations as graph transformation rules. A graph grammar is a formal way to specify an initial graph and a set of graph transformation rules. Each graph represents a model, in our case, a taxonomy. A transformation rule consists of preconditions that must be true for a model in order to the rule be applicable, and a set of creation and deletion operations for vertices and edges. The set of models reachable applying the grammar rules is called the grammar language.

One major advantage of capturing ontological patterns as a graph grammar is that it allows us to employ automatic verification tools and techniques over the grammar language [8]. Up to a certain number of types in each model, we are able to show that the graph grammar is *sound* and *complete*, i.e., that all models produced by the grammar rules are correct (*soundness*); and that the rules can generate all correct models (*completeness*). Such formal graph transformation rules thus comprise a framework that allows the user to build models which are *correct by construction*.

This paper is an extended version of [8] (presented in 2021 edition of the International Conference on Research Challenges in Information Science - RCIS). Here, we employ an additional foundational theory called MLT (Multi-Level Theory) [9, 10], which allow us to address taxonomies that also include high-order types, i.e., *multi-level* taxonomies. As shown in [11–13], multi-level taxonomies are observed at scale in practice and suffer from a large number of modeling errors that can be attributed to the inadequate use of subclassing and instantiation in tandem. By incorporating the rules of MLT in a graph grammar, we can prevent such errors from occurring. This new MLT-based grammar is then combined with the original OntoUML graph grammar from [8], to provide a comprehensive pattern grammar for multi-level ontology-based

conceptual modeling.

This work contributes to the foundations of rigorous conceptual modeling by identifying the set of rules that should be considered as primitives in the design of correct taxonomies (including multi-level ones). Moreover, it does that in a metamodel-independent way, so the results presented here can be incorporated into different modeling languages (e.g., ORM [14], whose development was already influenced by the ontological distinctions underlying OntoUML) as well as different tools used by different communities (e.g., as a modeling plugin to Semantic Web tools such as Protégé[1]).

The remainder of this paper is structured as follows. In Section 2, we describe the graph grammar corresponding to the theory of endurant types underlying OntoClean and OntoUML. In particular, we present a number of ontological meta-properties, a typology of types derived from them, and the formal constraints governing the subtyping relations between these types. We then present and assess the graph transformation grammar with operations that take into account the presented theoretical foundations. In Section 3, we describe the graph grammar corresponding to the Multi-Level Theory, which is again formalized and assessed. In Section 4, both graph grammars are combined, leading us to correct ontology-based multi-level taxonomies. In Section 5, we discuss related work both concerning the application of graph grammars to models and concerning the use of foundational theories to support the construction of taxonomies. Finally, Section 6 presents our concluding remarks.

2. Taxonomies of Endurant Types

2.1. Ontological Foundations

In this section, we present some ontological distinctions that are the basis for the remainder of this paper. These notions, and the constraints governing their definitions and relations, correspond to a fragment of the foundational ontology underlying OntoUML, namely, the Unified Foundational Ontology (UFO) [4, 15]), which incorporates and extends the theory of types underlying OntoClean. For an in depth discussion, philosophical justification, empirical support, and full formal characterization of these notions, see [4, 5].

Types represent properties that are shared by a set of possible instances. The properties shared by those instances constitute what is termed the *intension* of a type; the set of instances that share those properties (i.e., the instances of that type) is termed the *extension* of that type. Types can change their extension across different circumstances, either because things come in and out of existence, or because things can acquire and lose some of those properties captured in the intension of a type.

Taxonomic structures capture *subtyping* relations among types, both from *intensional* and *extensional* points of view. In other words, subtyping is thus a relation between types that govern the relation between the possible instances of those types. So, if type B is a subtype of A, then we have that: (i) it is necessarily the case that all instances of B are instances of A, i.e., in all possible circumstances, the extension of B is a subset of the extension of A; and (ii) all properties captured by the *intension* of A are included in the *intension* of type B, i.e., B's are A's and, therefore, B's have all properties that are properties defined for type A.

[1] `https://protege.stanford.edu/`

Suppose all instances that exist in a domain of interest are *endurants* [5]. Endurants roughly correspond to what we call *objects* in ordinary language, i.e., things that (in contrast to occurrences, events) endure in time changing their properties while maintaining their identity. Examples include you, each author of this paper, Mick Jagger, the Moon, the Federal University of Espírito Santo.

Every endurant in our domain belongs to one KIND. In other words, central to any domain of interest we will have a number of object kinds, i.e., the genuine fundamental types of objects that exist in that domain. The term "kind" is meant here in a strong technical sense, i.e., by a kind, we mean a type capturing *essential* properties of the things it classifies. In other words, the objects classified by that kind could not possibly exist without being of that specific kind [5].

Kinds tessellate the possible space of objects in that domain, i.e., all objects belong to exactly one kind and do so necessarily. Typical examples of kinds include 'Person', 'Organization', and 'Car'. We can, however, have other static subdivisions (or subtypes) of a kind. These are naturally termed SUBKINDS. As an example, the kind 'Person' can be specialized in the (biological) subkinds 'Man' and 'Woman'.

Endurant kinds and subkinds represent essential properties of objects. They are examples of RIGID TYPES [5]. Rigid types are those types that classify their instances necessarily, i.e., their instances must instantiate them in every possible circumstance in which they exist. We have, however, types that represent *contingent* or *accidental* properties of endurants termed ANTI-RIGID TYPES [5]. For example, in the way that 'being a living person' captures a cluster of contingent properties of a person, that 'being a puppy' captures a cluster of contingent properties of a dog, or that 'being a husband' captures a cluster of contingent properties of a man participating in a marriage.

Kinds, subkinds, and the anti-rigid types specializing them are endurant SORTALS. In the philosophical literature, a sortal is a type that provides a uniform principle of identity, persistence, and individuation for its instances [5]. To put it simply, a sortal is either a kind (e.g., 'Person') or a specialization of a kind (e.g., 'Student', 'Teenager', 'Woman', 'Biological Mother'), i.e., it is either a type representing the essence of what things are or a sub-classification applied to the entities that "have that same type of essence", be it rigid, i.e., a SUBKIND, or anti-rigid, i.e., an ANTI-RIGID SORTAL. Figure 1 presents an example of sortal hierarchy for the kind 'Person'. The figure depicts this hierarchy as a directed graph, revealing the abstract syntax of taxonomies as adopted in this paper. Each node represents a class, whose ontological nature in UFO is represented in bold (here, 'Kind', 'SubKind' and 'AntiRigidSortal'). Edges are labeled to identify the relation between the elements; in this figure all edges represent specializations and are labeled 'subClassOf'.

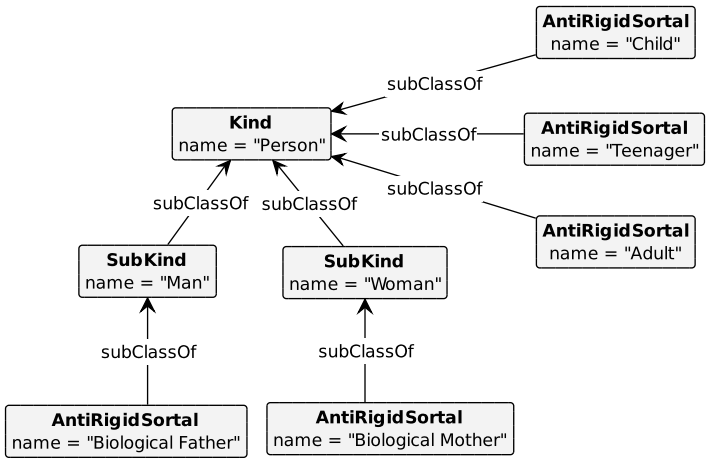


Figure 1: An example of sortal hierarchy for the kind 'Person'.

In general, types that represent properties shared by entities of *multiple kinds* are termed NON-SORTALS, i.e., non-sortals are types whose extension possibly intersect with the extension of more than one kind. Non-sortals too can also be further classified depending on whether the properties captured in their intension are essential (i.e., rigid) properties or not.

Now, before we proceed, we should notice that the logical negation of rigidity is not anti-rigidity but *non-rigidity*. If being rigid for a type A means that all instances of A are *necessarily* instances of A, the negation of that (i.e., non-rigidity) is that there is at least one instance of A that can *cease to be* an instance of A; anti-rigidity is much stronger than that, it means that *all* instances of A can cease to be instances of A, i.e., A's intension describes properties that are contingent for all its instances. Finally, we call a type A *semi-rigid* iff it is non-rigid but not anti-rigid, i.e., if it describes properties that are essential to some of its instances but contingent to some other instances. Because non-sortal types are dispersive [16], i.e., they represent properties that behave in very different ways with respect to instances of different kinds, among non-sortal types, we have: those that describe properties that apply *necessarily to the instances of all kinds* it classifies (i.e., Rigid Non-Sortals, which are termed CATEGORIES); those that describe properties that apply *contingently to the instances of all kinds* it classifies (ANTI-RIGID NON-SORTALS); those that describe properties that apply *necessarily to the instances of some of the kinds it classifies* but that also apply *contingently to the instances of some other kinds it classifies* (i.e., Semi-Rigid Non-Sortals, termed MIXINS). An example of a category is 'Physical Object' representing properties of all kinds of entities that have masses and spatial extensions (e.g., people, cars, notebooks, buildings); an example of an anti-rigid non-sortal is 'Customer' representing contingent properties for all its instances (i.e., no customer is necessarily a customer), which can be of the kinds 'Person' and 'Organization'; an example of a mixin is 'Insured Item', which describe properties that are essential to entities of given kinds (e.g., suppose that cars are necessarily insured) but which are contingent to things of other kinds (e.g., houses can be insured but they are not necessarily insured). Figure 2 presents examples of the different types of non-sortals discussed.

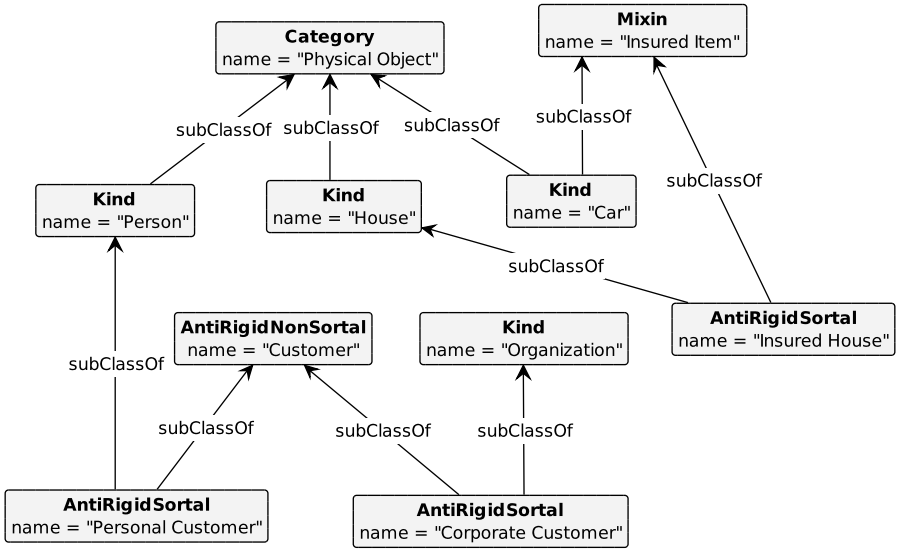


Figure 2: A taxonomy with Non-Sortals.

Figure 3 represents (with a UML class diagram) the typology of endurant types generated by the possible values of the two properties, sortality and rigidity. As usual, UML arrows with a closed head connect subtypes to their supertypes (the arrowhead pointing to the supertype). Two subtyping relations joined in their arrowheads form a *generalization set*, which here we assume to tessellate the extension of the supertype (pointed to by the joint arrowhead), i.e., these are disjoint and complete generalization sets. There are two superimposed generalization trees, one formed by first considering the *sortality* meta-property (in red) and the other considering first the *rigidity* meta-property (in blue). As a result of the combination of these two meta-properties, we have the following six (exhausting and mutually disjoint) types of types (i.e., metatypes): KINDS, SUBKINDS, ANTI-RIGID SORTALS, CATEGORIES, ANTI-RIGID NON-SORTALS, and MIXINS (shaded in Figure 3).

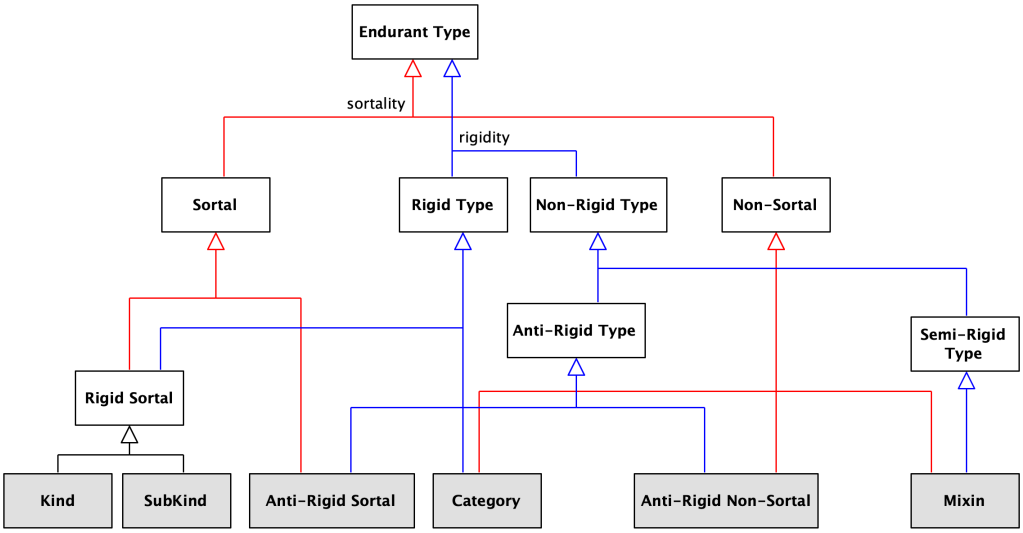


Figure 3: A taxonomy for Endurant Types.

The ontological meta-properties that characterize these different types of types also impose constraints on how they can be combined to form taxonomic structures [5]. As we have already seen, since kinds tessellate our domain and, because all sortals are either kinds or specializations thereof, we have that: (i) no kind can specialize another kind; and (ii) every sortal that is not a kind specializes a unique kind. In other words, every sortal hierarchy has a unique kind at the top. Moreover, from these, we have that any type that is a supertype of a kind must be a non-sortal. But also that, given that every specialization of a kind is a sortal, non-sortals cannot specialize sortals. Finally, given the formal definitions of rigidity (including anti-rigidity), it just follows logically that anti-rigid types (sortals or not) cannot be supertypes of semi-rigid and rigid types (sortals or not – see proof in [5]). For example, if we determine that 'Customer' applies contingently to persons in the scope of business relationships, then a taxonomy in which a rigid type 'Person' specializes an anti-rigid type 'Customer', as shown in Figure 4, is logically incorrect. Intuitively, a person will be at the same time required through the specialization to be statically classified as a 'Customer' while at the same time, being defined dynamically classified as a 'Customer', in virtue of the definition of that type. So, either: (i) the definition of 'Customer' should be revised to capture only essential properties, becoming a rigid type and thus solving the incorrect specialization problem; or (ii) a different organization of the taxonomy is required, with 'PersonalCustomer' as an anti-rigid specialization of the rigid type 'Person', 'Corporate Customer' an anti-rigid specialization of 'Organization', and 'Customer' as an anti-rigid supertype of 'PersonalCustomer' and 'Corporate Customer' as shown in Figure 2 (this solution is identified as the 'roles with disjoint allowed types' design pattern in [4]).

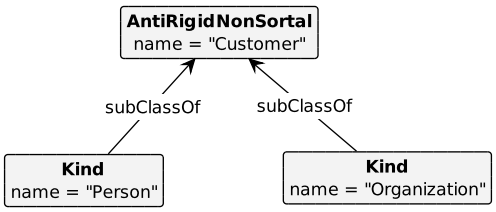


Figure 4: Incorrect representation of type 'Customer'.

2.2. Graph Transformation Rules to Build Ontologically Correct Taxonomies

Graph transformation (or *graph rewriting*) [17] has been advocated as a flexible formalism, suitable for modeling systems with dynamic configurations or states. This flexibility is achieved by the fact that the underlying data structure, that of graphs, is capable of capturing a broad variation of systems. Some areas where graph transformation is being applied include visual modeling of systems, the formal specification of model transformations, and the definition of graph languages, to name a few [18, 19].

The core concept of graph transformation is the rule-based modification of graphs, where each application of a rule leads to a graph transformation step. A transformation rule specifies both the necessary preconditions for its application and the rule effect (modifications) on a *host graph*. The modified graph produced by a rule application is the result of the transformation.

In this work, we use graph transformations to formally model the operations for the construction of a taxonomy. A set of graph transformation rules can be seen as a declarative specification of how the construction can evolve from an initial state, represented by an initial (empty) host graph. This combination of a rule set plus an initial graph is called a *graph grammar*, and the (possibly infinite) set of graphs reachable from the initial graph constitute the *grammar language*.

Our main contribution in this paper is to formally define a graph grammar that, starting from an empty taxonomy, allow us to build only correct taxonomies. To put this more precisely: in the area of formal verification, statements about a system are usually split between *soundness* and *completeness* properties. The soundness of a modeled system ensures that only desirable models are possible. In our setting, this means that only correct taxonomies can be part of the grammar language. On the other hand, *completeness* ensures that if a desirable system configuration can exist "in the real world", then a corresponding model is reachable in the formalization. In our setting, this means that any correct taxonomy can be created using the proposed graph grammar.

The grammar described in this section was created with GROOVE [19], a graph transformation tool suitable for the (partial) enumeration of a grammar language, which the tool calls the *state space exploration* of the graph grammar. We will use the tool later to demonstrate that our proposed taxonomy grammars are *sound* and *complete*, at least up to a certain taxonomy size.

2.2.1. Introducing New Types

We start by defining transformation rules to introduce a new type in the taxonomy. Types for four of the leaf ontological metatypes given in Figure 3 can be introduced in the taxonomy without being related with a previously introduced type: these include all Kinds and all the non-sortals: Categories, Mixins and Anti-Rigid Non-Sortals.

Figure 5 shows the four rules that introduce these 'independent' types, using the GROOVE visual notation for presenting rules. Each rule is formed only by a green thick lined box representing the type that will be *created* during rule application. A type has an ontological metatype

(the label inside the box). No rule in Figure 5 has preconditions. Therefore, types for these four ontological metatypes can be introduced without requiring the existence of other types or relations in the taxonomy.

(a) new-kind (b) new-category (c) new-mixin (d) new-antirigid-nonsortal

Figure 5: Transformation rules to introduce an independent type.

2.2.2. Introducing Dependent Types

In contrast to non-sortals and kinds, SubKinds and Anti-Rigid Sortals have preconditions upon their introduction.

In the case of SubKinds, their introduction requires the existence of a previous sortal, from which the subkind will inherit a principle of identity. In addition, this sortal must be rigid, to respect the ontological principle that a rigid type cannot specialize an anti-rigid one. These preconditions for the introduction of a new SubKind are captured in the rule shown in Figure 6. The *existing* Rigid Sortal is shown as a gray box in the figure. It represents any Kind or SubKind, since a *type graph* in GROOVE was established to reflect the taxonomy presented in Figure 3. The green thick "subClassOf" arrow states that a new direct subtyping relation will be introduced in the model.

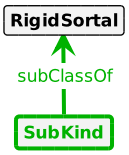


Figure 6: Transformation rule to introduce a SubKind type.

In the case of an Anti-Rigid Sortal type, the only precondition is the existence of a previous sortal, from which the newly introduced Anti-Rigid Sortal will inherit a principle of identity. This rule is shown in Figure 7. Differently from a SubKind, an Anti-Rigid Sortal can specialize any Sortal (and not only rigid ones).

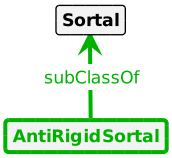


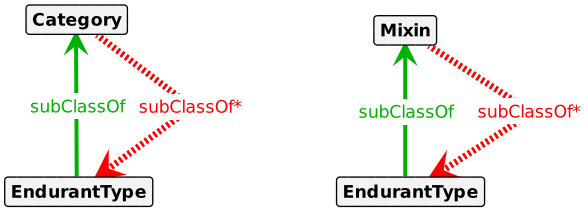
Figure 7: Transformation rule to introduce an Anti-Rigid Sortal type.

2.2.3. Introducing Specializations for Existing Non-Sortal Types

Having defined rules for the introduction of types, we proceed with rules to insert subtyping relations between two types already present in the model. We start with Category and Mixin

specializations, as both of these ontological metatypes have meta-properties that allow their types to be specialized in any ENDURANT TYPE, without breaking formal ontology principles.

Figure 8(a) shows a rule that creates a subtyping relation between an existing CATEGORY supertype and an existing ENDURANT subtype. The red dashed arrow in the figure prevents the introduction of a circularity in the subtyping relations. Note that circularity of specializations may be tolerated in taxonomies structured with *improper specialization relations*, such as the case of `rdfs:subClassOf` used in RDF and OWL. However, for the purposes of this work, we represent only *proper specializations*, i.e., those in which the subtype is different from its supertype (in that some possible instances of the supertype are not instances of the subtype). Red elements in GROOVE rules indicate *forbidden* patterns, i.e., elements that, if present, prevent the rule application. The label "subClassOf*" indicates "subClassOf" paths of any size (including zero, which would amount to the equality between the related classes.) Figure 8(b) shows the analogous rule for the specialization of a MIXIN.



(a) category-specialization (b) mixin-specialization

Figure 8: Transformation rules to specialize a CATEGORY or a MIXIN.

Finally, the rule depicted in Figure 9 allows the specialization of an ANTI-RIGID NON-SORTAL by another ANTI-RIGID TYPE.

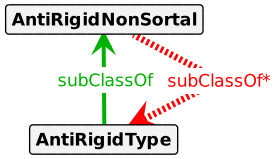


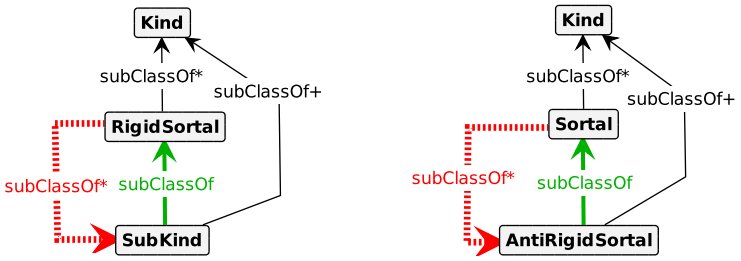
Figure 9: Transformation rule to specialize an ANTI-RIGID NON-SORTAL type.

2.2.4. Introducing Generalizations for Existing Sortal Types

KIND types appear on the top of SORTAL hierarchies because kinds provide a principle of identity for all their instances. By definition, kinds cannot specialize other kinds. Therefore, they can only specialize NON-SORTAL types, more specifically CATEGORIES and MIXINS. These specializations can already be constructed with the rules presented in Section 2.2.3.

SUBKIND types, on the other hand, carry a principle of identity from their supertypes and, ultimately, from exactly one KIND type. The rule shown in Figure 10(a) properly captures this restriction. If there is a RIGID SORTAL distinct from a SUBKIND and not specializing the latter (as defined by the "subClassOf*" red dashed edge) and both carry a principle of identity from the same KIND, then a direct subtyping relation can be created between the two. The black

thin lined edges with labels "subClassOf*" and "subClassOf+" indicate that, for the rule to be applied, a specialization relation from the new supertype and from the SubKind to the same Kind must already be present, or at least that the new (direct) supertype of the SubKind is its own Kind. Subkinds can also specialize any rigid or semi-rigid non-sortal, but these cases are already covered by the rules presented in Section 2.2.3. A similar construction for Anti-Rigid Sortal types can be seen in Figure 10(b).



(a) subkind-generalization (b) antirigid-sortal-generalization

Figure 10: Transformation rules to generalize a SubKind or an Anti-Rigid Sortal.

2.3. Formal Verification

We use the GROOVE graph transformation tool to carry out a formal verification of the graph grammar presented in Section 2.2. To do so, we employ verification conditions in GROOVE, which formally define the ontological constraints described in Section 2.1, and allow us to perform an analysis over the graph state model (a state in this case corresponds to a *taxonomy shape*, representing an equivalence class of isomorphic taxonomies). We then use the state space exploration functionality of the tool to examine the taxonomy shapes that the grammar generates.

As stated in Section 2.2, our objective with the verification is two-fold: to demonstrate the *soundness* and *completeness* of the proposed graph grammar. Soundness ensures that the grammar rules only produce correct taxonomies, i.e., those that do not invalidate well-formedness constraints. Completeness ensures that any and all correct taxonomies can be produced by a sequence of rule applications. (Certain caveats concerning taxonomy size apply to our formal verification tasks, these will be discussed in Sections 2.3.1 and 2.3.2.)

A *graph condition* in GROOVE is represented diagrammatically in the same way as transformation rules, albeit without creator (green thick lined) elements. A graph condition is satisfied by a taxonomy model if all reader (black thin lined) elements of the condition are present in the model, and all forbidden (red dashed) elements are absent.

Figure 11 shows our first graph condition, capturing the restriction that Kinds must appear at the top of sortal hierarchies, hence not specializing another Sortal. It is important to note that restrictions are stated *positively* but are checked *negatively*. Thus, the condition in Figure 11 characterizes an undesired model violation (a Kind specializing a Sortal), and therefore, by verifying that such condition never occurs in any taxonomy model, we can determine the grammar is sound. This same rationale holds for all other conditions shown in this section.

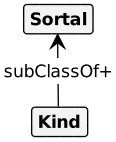


Figure 11: Restrictive condition of a KIND specializing another SORTAL.

Figure 12 formalizes a second restrictive condition, stating that a SORTAL cannot inherit its principle of identity from more than one KIND. A third condition, shown in Figure 13, captures the situation in which the *rigidity* meta-property is contradicted, that is, when a rigid or semi-rigid type specializes an anti-rigid one. Similarly, the fourth restrictive condition, depicted in Figure 14, represents the situation in which the *sortality* meta-property is contradicted, that is, when a NON-SORTAL type specializes a SORTAL one.

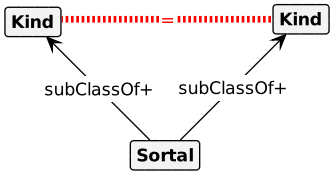


Figure 12: Restrictive condition of a SORTAL with more than one KIND.

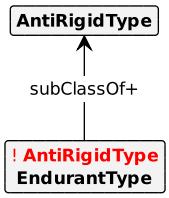


Figure 13: Restrictive condition of a rigid or semi-rigid type specializing an anti-rigid one.

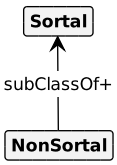


Figure 14: Restrictive condition of a NON-SORTAL type specializing a SORTAL one.

To specify a fifth and final restrictive condition, we consider that all SORTALS ultimately should specialize (or be) a KIND, from which they inherit a principle of identity. The violating situation, in which a SORTAL does not specializes a KIND, is shown in Figure 15.

Figure 15: Restrictive condition of a SORTAL without a KIND.

2.3.1. Verifying Soundness

The first step in verifying the soundness of the graph grammar proposed is to enumerate its language, i.e., construct all possible taxonomies reachable by any sequence of rule applications. Subsequently, the graph conditions just presented are checked against these constructed taxonomies. If any model triggers one or more graph conditions, then we know the model violates some ontological restrictions, and therefore it is incorrect. Consequently, the goal of the soundness analysis is to verify that no taxonomy in the language is incorrect. To perform the grammar state space exploration we use the GROOVE Generator, a command-line tool designed for this task. Details of GROOVE usage can be found at the tool manual (available at `https://sourceforge.net/projects/groove/`). Additional case studies that illustrate the tool functionalities are presented in [19].

A major caveat in the first step above is that the grammar language is *infinite*, thus preventing a complete enumeration in a finite amount of time. To cope with this situation, we need to perform a *bounded* exploration with the GROOVE tool. In this setting, our bound N is the number of types present in a taxonomy shape. When performing the exploration, the tool managed to generate a total of 2,123,196 correct taxonomy shapes up to a bound $N = 6$, with a breakdown of this total per bound value shown in Table 1. The table also shows that our soundness goal was validated (at least up to $N = 6$), with no taxonomy shapes being flagged as incorrect by the graph conditions.

# types (N)	**Produced taxonomy shapes**	**Incorrect taxonomy shapes**
1	4	0
2	21	0
3	160	0
4	2,032	0
5	46,885	0
6	2,074,094	0

Table 1: Results of soundness analysis for the endurant types graph grammar.

Given the inherently exponential growth of the number of possible taxonomy shapes with respect to bound N, it was not possible to continue the exploration for $N = 7$ and beyond due to memory limitations (the execution was halted after several million models partially produced.) This *state space explosion* is a common problem for all explicit state model checkers, such as GROOVE [19].

To support that the soundness results in Table 1 are significant, we rely on the *small scope hypothesis*, which basically claims most design errors can be found in small counterexamples [20]. Experimental results suggest that exhaustive testing within a small finite domain does indeed

catch all type system errors in practice [21], and many case studies using the formal language and tool Alloy have confirmed the hypothesis by performing an analysis in a variety of scopes and showing, retrospectively, that a small scope would have sufficed to find all the bugs discovered [22]. A more detailed discussion concerning the implications of the small scope hypothesis for our verification tasks is presented in Section 2.3.3.

2.3.2. Verifying Completeness

The verification described in the previous section shows us that all taxonomies produced are correct, but does nothing to persuade us that any and all possible correct taxonomies can be produced. To provide this kind of assurance is the goal of the completeness verification described in this section.

To perform the completeness analysis we need to consider not only correct taxonomies but also the incorrect ones. To this end, we developed another, completely permissible, graph grammar that allows the creation of both correct and incorrect models. The grammar is quite simple, with six rules for the unrestricted creation of the leaf types of types in Figure 3, and one rule allowing the introduction of a subtyping relation between any two endurant types.

The results of the exploration with this new permissible grammar are presented in Table 2. As expected, the rate of growth in this scenario is even steeper, given that more models can be produced. The tool was able to perform a bounded exploration up to $N = 5$, with larger bounds exceeding the available memory. The second column of Table 2 lists all taxonomies created with the new grammar, both correct and incorrect. We again use the same graph conditions to flag violations of ontology restrictions in the models. If a taxonomy triggers *any* of the graph conditions, then it is considered incorrect. Conversely, if *no* graph condition is triggered by a model, then it certainly describes a correct taxonomy. The last two columns in the table summarize this classification.

# types (N)	**All taxonomy shapes**	**Incorrect shapes**	**Correct shapes**
1	6	2	4
2	57	36	21
3	956	796	160
4	30,741	28,709	2,032
5	1,958,538	1,911,653	46,885

Table 2: Results of completeness analysis for the endurant types graph grammar.

The completeness goal can be verified by a comparison between the **Correct shapes** column of Table 2 and the **Produced taxonomy shapes** column of Table 1. It can be seen immediately that all values up to $N = 5$ match. Given that the permissible grammar produces all possible models (correct and incorrect), this allows us to conclude that the taxonomy grammar of Section 2.2 produces *all* correct taxonomies up to that size, and *only* the correct ones.

2.3.3. Verification Scope Matters

The small scope hypothesis asserts that a problem in the grammar would be revealed in violating taxonomies of small size. To lend credence to this hypothesis, we need to consider the nature of well-formedness violations (according to the restrictive conditions shown in Figures 11–15). For example, in the case of rigidity and sortality, violations arise from specializations relating

classes with particular meta properties: rigid classes cannot specialize anti-rigid classes (Figure 13) and non-sortals cannot specialize sortals (Figure 14). The violations of rules concerning rigidity and sortality can therefore arise with taxonomies of size 2 (e.g., a single rigid class specializing a single anti-rigid class and a single non-sortal specializing a single sortal). Taxonomies of size 2 also reveal violations of the rules involving kinds (a kind specializing another kind or any other sortal, Figure 11). These sorts of violations are all covered by our analysis (and are included in the 36 incorrect taxonomy shapes of size 2 in the second row of Table 2). So, a problem involving these violations would have been revealed if the proposed grammar could produce them. Other problems involving kinds are manifested with taxonomies of size 3, e.g., when a sortal specializes two kinds (Figure 12), or even size 1, in a taxonomy with a single sortal other than a kind (Figure 15). Given the nature of the restrictive conditions, they can only be violated in taxonomies larger than 3 classes either due to: (i) the occurrence of a violating fragment of size 1–3 (corresponding to one of the Figures 11–15) or (ii) due to the transitivity of the subClassOf relation. Problems involving transitivity would already be revealed with taxonomies of size 3 (in most cases) and 5 (in the case of Figure 12). In conclusion, all restrictive conditions can be possibly violated with taxonomy shapes of size 5 or less, which gives us confidence that the exhaustive exploration up to that size (1,958,538 taxonomy shapes as reported in Table 2) is relevant to our verification goals.

With respect to completeness, there is an inherent limitation of the exhaustive enumeration of states as proposed here. In our future work, we intend to address this limitation by exploring other techniques, such as the partial exploration of the grammars state space or the use of proof assistants.

3. Multi-Level Taxonomies

So far, we have covered conventional taxonomies built up by establishing specialization relations between types. However, there are several knowledge domains in which types are also considered *instances of* other types. For example, in the biological domain, types of animals such as 'Dog' and 'Cat' may be considered instances of 'Species', and types such as 'Greyhound' and 'Siamese Cat' may be considered instances of 'Breed' specializing 'Dog' and 'Cat' respectively. In these domains, metatypes or high-order types appear (such as 'Species' and 'Breed'). The taxonomy can thus be considered a *multi-level* one, with a classification level of types whose instances are individuals (in this example, 'Dog' and 'Cat') and higher classification levels, with types whose instances are types (in this example, 'Species' and 'Breed'). This example in the biological domain is shown in Figure 16. Other examples of multiple classification levels come from domains such as that of organizational roles (or professional positions) [23], software engineering [24] and product types [25].

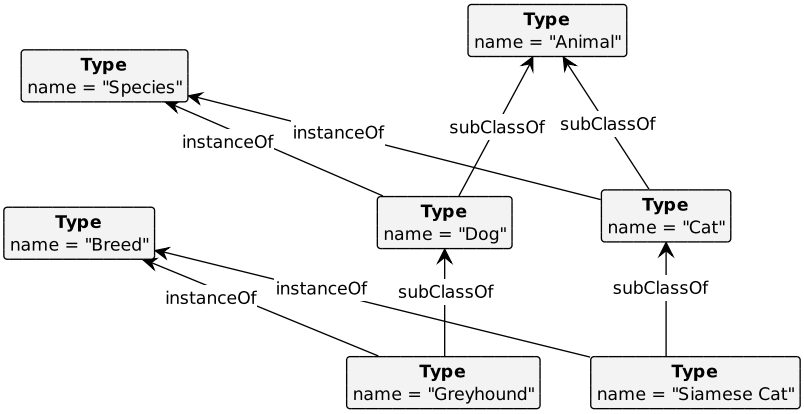


Figure 16: An example of multi-level taxonomy in biological domain.

3.1. The Multi-Level Theory

Over the last decades, the importance of this modeling phenomenon has justified a number of works under the banner of "multi-level modeling" [10, 24–27]. Techniques for multi-level modeling must provide modeling concepts to deal with types in various classification levels and the relations that may occur between those types. These approaches embody conceptual notions that are key to the representation of multi-level models, such as the existence of entities that are simultaneously types and instances, the iterated application of instantiation across an arbitrary number of levels, etc [9].

These fundamental notions for multi-level modeling have been captured formally in the MLT Multi-Level Theory, described in various publications over the last years [10, 27, 28]. Similarly to the endurant types theory described in Section 2.1, MLT was formalized [10] and applied in the design of a well-founded profile for UML [29]. The theory was also applied fruitfully to uncover problematic taxonomies in Wikidata [12, 13], to design the ML2 multi-level modeling language embodying the theory's rules as syntactic constraints [11], to support multi-level models using Semantic Web technologies [30], etc.

The following key definitions are proposed for MLT [9]: *Individuals* are those entities which cannot possibly play the role of type in the instantiation relation (i.e., those entities that cannot have instances). Examples include Albert Einstein, Laika the Soviet space dog, The Earth. *First-order types* are those types whose instances are individuals. Examples include Person, Dog, Planet, Car. *Second-order types* are those types whose instances are first-order types. Examples include Species, Breed, but also Astronomical Object Type, Car Model. *Third-order types* are those types whose instances are second-order types (e.g., Taxonomic Rank whose instances may include Species and Breed), and so on. The topmost order can be established as required by applications, and the scheme can thus be extended to cope with an arbitrary number of levels.

Classification levels in MLT are generated by the iterative application of the notion of *powertype* in line with the definition of Cardelli [31]. A type *pt is powertype of* a (base) type *t* iff all instances of *pt* are (improper) specializations of *t* and all possible specializations of *t* are instances of *pt*. Powertypes in this sense are analogous to powersets. The powerset of a set *A* is a set that includes as members all subsets of *A* (including *A* itself). Using this definition, we can clarify how classification levels are related: if Individual is the type that classifies all possible individuals, then the type that classifies all first-order types—First-order Type—can be defined as

the *powertype of* INDIVIDUAL. SECOND-ORDER TYPE can be defined as the *powertype of* FIRST-ORDER TYPE, and so on. The types defined in this way, i.e., INDIVIDUAL, FIRST-ORDER TYPE, SECOND-ORDER TYPE, etc., are called *basic types* in MLT. It follows from Cardelli's definition of powertype that a powertype is unique for a base type, i.e., a base type has one and only one powertype. Further, the base type is unique for a given powertype (for theorems and their proofs see [10]).

Using the definition of basic types, MLT partitions a domain taxonomy into strictly stratified levels by establishing that all domain types are specializations of these basic types. This amounts to enforcing the *strict metamodeling principle* [32]. Since basic types form a line connected by powertype relations, a first-order type (such as ANIMAL) is at the same time an instance of the basic type FIRST-ORDER TYPE and a specialization of INDIVIDUAL (since its instances are individuals). A second-order type (such as SPECIES) is at the same time an instance of the basic type SECOND-ORDER TYPE and a specialization of FIRST-ORDER TYPE (since its instances are first-order types), and so on for higher-order types.

MLT also accounts for an important variant of the powertype pattern proposed by Odell [33] (that inspired the homonymous construct in UML class diagrams). Odell stated simply that a *powertype* is a type whose instances are subtypes of a base type. This means, that, differently from Cardelli, not all subtypes of the base type are required to be instances of the powertype. (In fact, as pointed out by [34], the relation defined by Odell is misnamed *powertype* since, in fact, it is analogous to a *subset* of the *powerset*.) MLT calls the relation between an Odell powertype and its base type *categorization*. For example, we can say that ANIMAL SPECIES *categorizes* ANIMAL; this is because some specializations of ANIMAL are instances of SPECIES (DOG, CAT), but not all of them are (e.g., FEMALE DOG, SIAMESE CAT, while specializations of ANIMAL, are not instances of SPECIES). Figure 17 enriches the model of Figure 16 revealing the 'First-Order Type' and 'Individual' basic types as well as the categorization relations. (The 'instanceOf' edges between the various first-order types specializing 'Individual' are omitted for the sake of readability.)

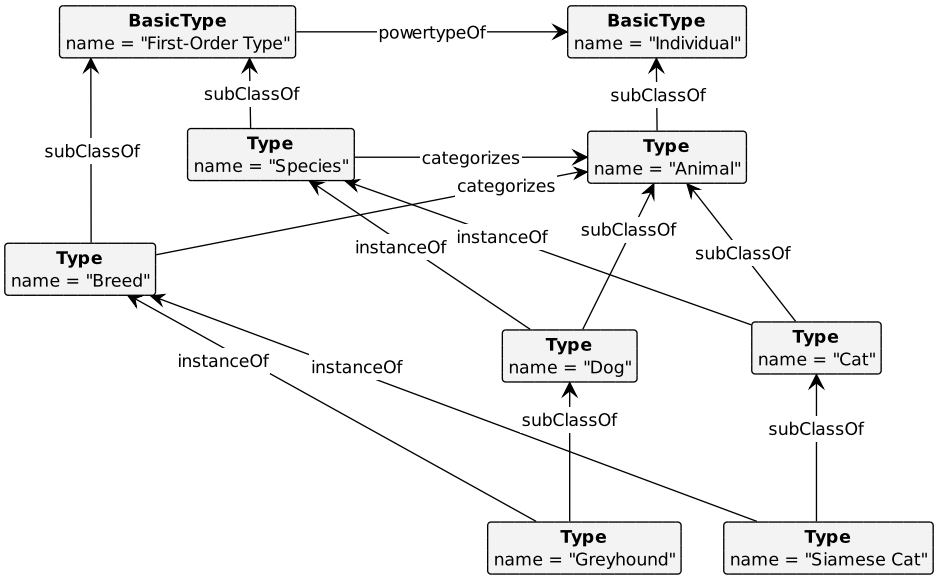


Figure 17: A multi-level model for the biological domain including variants of the powertype pattern.

The consequences of the theory can be used to identify sound multi-level structures, including the following derived rules (these rules are formally proven theorems that follow from the MLT definitions and axioms [10] as summarized in [9]):

- The *instance of* relation in MLT is irreflexive, antisymmetric and anti-transitive. Further, it only relates entities of adjacent levels.
- Every type belongs to a specific order, specializing one and only one basic type.
- *Specialization* (whether proper or not) cannot cross level boundaries (i.e., a first-order type can only be specialized by first-order types, a second-order type can only be specialized by second-order types, and so on).
- Both the relations of *is powertype of* and *categorizes* can only be applied between adjacent levels, with the base type one order lower than the high-order type.
- The *powertype* of a base type (in Cardelli's sense) is unique for that base type;
- Types that *categorize* a base type always specialize the base type's *powertype*.

Violation of these rules and basic definitions of MLT lead to problematic taxonomies, as illustrated in Figure 18 with a multi-level taxonomy found in Wikidata. As discussed in [13], this taxonomy is problematic since 'Mayor' is at the same time an instance of 'Position' and a specialization of 'Position' (through 'Public Office'). Note the logical contradiction: instantiation places 'Position' and 'Mayor' in adjacent levels (e.g., 'Mayor' as a first-order type and 'Position' as a second-order type), while specialization requires them to be at the same level. Further empirical evidence discussed in [11–13] shows that this is a large-scale problem in practice. Our objective here is then to incorporate the basic rules underlying MLT in our graph grammar, and thus rule out problematic multi-level taxonomies by construction.

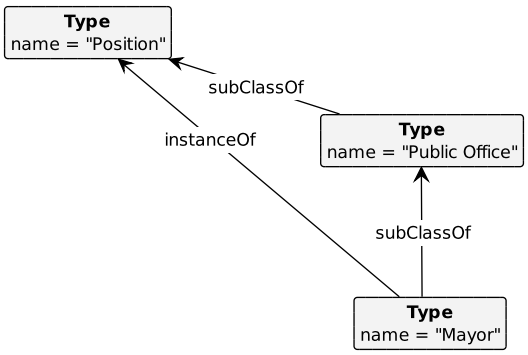


Figure 18: An incorrect multi-level model present in Wikidata and uncovered in [13].

3.2. Graph Transformation Rules to Build Correct Multi-Level Taxonomies

Our construction of multi-level taxonomies starts from a host graph that contains only the basic type representing the class of *individuals*, as shown in Figure 19.

Figure 19: Initial host graph only with one basic type representing the INDIVIDUAL class.

Taxonomies are then further built up with operations that introduce first-order types by specializing this first basic type (Section 3.2.1); or that introduce new basic types for additional levels of classification (or orders – Section 3.2.2). Then, high-order types can be introduced as (Odell) powertypes of existing (lower-order) types (Section 3.2.3). Finally, specialization and instantiation between existing elements can be introduced (Sections 3.2.4 and 3.2.5).

3.2.1. Introducing a New First-Order Type

The order of a particular basic type is inferred by the length of the *powertype relation* chain. The last basic type in this chain—the one that is not a powertype—is always the basic type representing the type of all individuals (and is introduced initially in the host graph as shown in Figure 19). By definition, the specializations of this basic type are first-order types, which are introduced exactly as direct or indirect (proper) specializations of the basic type at the tail of the sequence of basic types. This rule is shown in Figure 20. The forbidden fragment guarantees the selection of the right basic type for specialization (the basic type representing the INDIVIDUAL class).

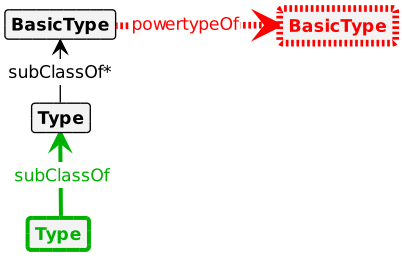


Figure 20: Transformation rule to introduce a FIRST-ORDER TYPE.

3.2.2. Introducing a New Order

The model can be extended to accommodate higher levels of types by creating the *powertype* of the basic type of the highest order present in the model thus far, i.e., the powertype with no existing powertype in the model (see Figure 21).

Figure 21: Transformation rule to introduce a new order.

3.2.3. Introducing a New High-Order Type

New high-order types are then introduced in the model as Odell powertypes categorizing a base type at one order below. As shown in Figure 22, the categorizer specializes the basic type that is the powertype of the basic type at the order below (to which the categorized base type

belongs). Note that, as all specializations of a basic type are instances of its powertype (which is the basic type of the order above), we omit the instantiations of that higher-order basic type, as they are redundant.

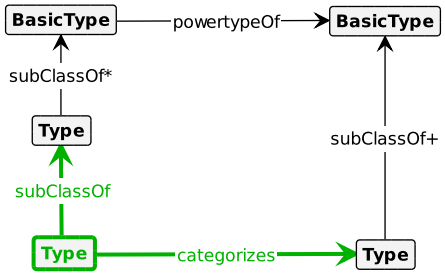


Figure 22: Transformation rule to introduce a new high-order type.

3.2.4. Introducing Specialization Relations

As stated by MLT, a (proper) specialization relation can only hold between two types of the same order, i.e., types that specialize the same basic type. In addition, to enforce the semantics of specialization, we require the subtype to have no instances when the *subClassOf* relation is first introduced in the model. This prevents the situation in which an entity is an *instance of* a subtype without being an *instance of* the supertype. All these restrictions are formalized in the graph transformation rule given in Figure 23.

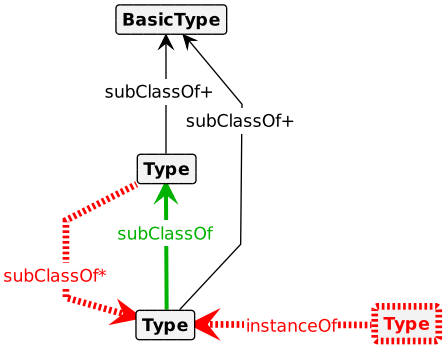


Figure 23: Transformation rule to introduce a new specialization relation.

3.2.5. Introducing Instantiation Relations

A high-order type can be instantiated by a type that specializes the base type it categorizes, as shown by the rule in Figure 24. The new *instance of* the high-order type must be an *instance of* all its categorizer's supertypes, i.e., all its supertypes that are not basic types, to ensure that the semantics of specialization is enforced. The "!instanceOf" red dashed arrow in the rule indicates a *forbidden absence* of an *instance of* relation.

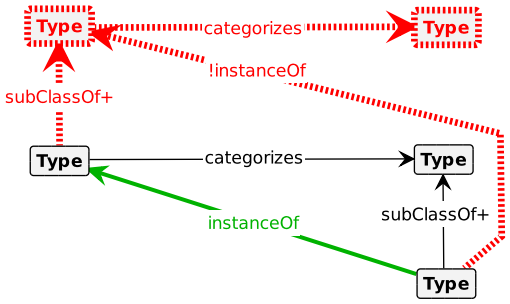


Figure 24: Transformation rule to introduce a new instantiation relation.

3.3. Formal Verification of MLT Constraints

The formal verification of the graph grammar presented in Section 3.2 follows the same process discussed in Section 2.3 for the endurant types grammar. In order to use GROOVE to analyze the new grammar language, we again need verification conditions, this time to formally define the constraints that follow from the MLT axiomatization, as summarized in Section 3.1. These conditions are discussed in the following subsections, with the verification results presented subsequently.

3.3.1. Constraints Related to Basic Types

We start by presenting constraints on how basic types can be related. Figure 25 shows our first graph condition regarding basic types, capturing the restriction that a basic type cannot have more than one powertype.

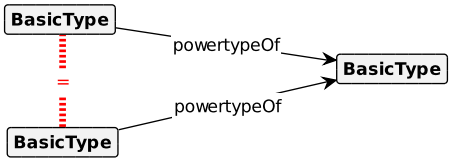


Figure 25: Restrictive condition of a basic type with more than one powertype.

Conversely, the restriction in Figure 26 shows that different basic types cannot have the same powertype.

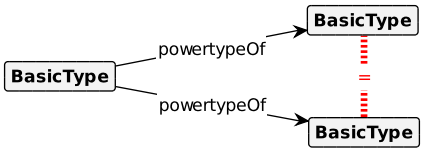


Figure 26: Restrictive condition of different basic types with the same powertype.

The rule presented in Figure 27 captures the restriction that different basic types must be, directly or indirectly, related by the powertype relation, since they have different orders. It is important to note once again that all restrictions in this section are stated *positively* but are checked

negatively. Thus, the condition shown in Figure 27 describes the undesired case where two basic types in the model are not in a transitive powertype relation. Therefore, by checking in the verification phase that such case never occurs, we ensure that all type orders are related.

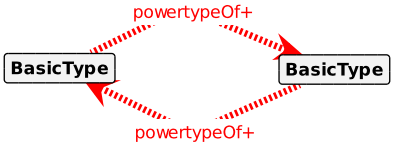


Figure 27: Restrictive condition of unrelated basic types.

Finally, the condition in Figure 28 states that two basic types cannot be mutually connected by the powertype relation. Since a base type is an instance of a Cardelli powertype, basic types thus related would in fact instantiate each other, which would contradict their purpose to establish the basic structure for the stratified level scheme.

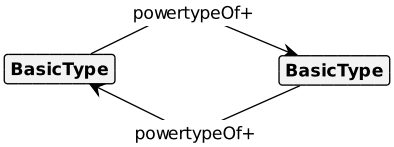


Figure 28: Restrictive condition of mutual powertype relation between basic types.

3.3.2. Constraints Related to Level Stratification

Figure 29 shows a key graph condition related to level stratification, capturing the restriction that every type must be either a basic type or a specialization of a basic type.

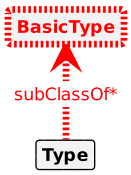


Figure 29: Restrictive condition of a type with no basic type as supertype.

The graph condition depicted in Figure 30 captures the restriction that no type can specialize more than one basic type, otherwise the type would belong to more than one order or level.

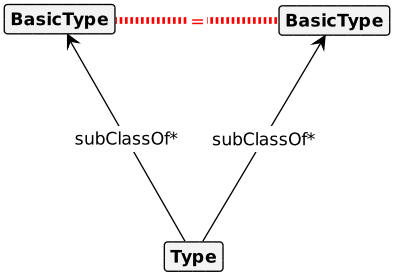
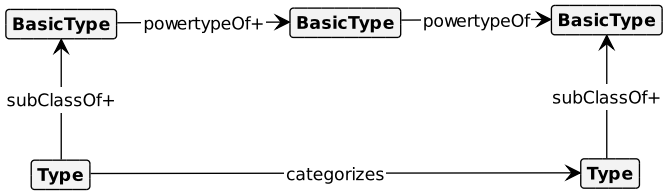
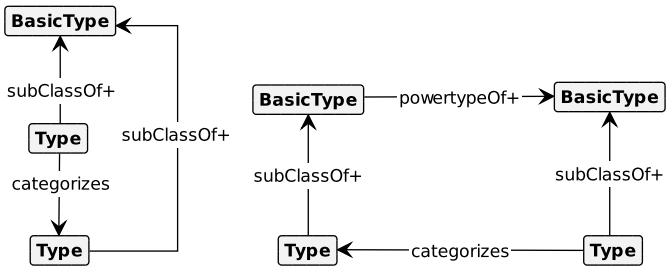


Figure 30: Restrictive condition of a type specializing more than one basic type.

Two more conditions related to level stratification are presented in Figures 31(a, b, c), indicating that a *categorization* relation cannot exist between types that are not in adjacent levels. In other words, a *categorization* cannot cross multiple levels (Figure 31a) or occur between two types in the same level (Figure 31b), and a type cannot categorize a type at a higher order (Figure 31c). Types can only categorize lower-order types one level below.



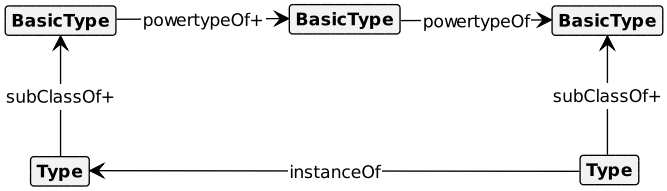
(a) categorization-across-multiple-levels



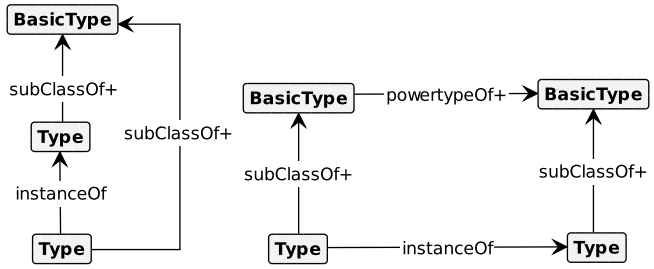
(b) intra-level-categorization (c) reverse-categorization

Figure 31: Restrictive conditions of categorization between nonadjacent levels.

Analogously to categorization, instantiation cannot cross multiple levels or occur between two types in the same level. These restrictions are captured by the graph conditions shown in Figures 32(a, b, c). As can be seen, the conditions in Figures 31 and 32 differ only in the "categorizes" and "instanceOf" relations and their directions.



(a) instantiation-across-multiple-levels



(b) intra-level-instantiation (c) reverse-instantiation

Figure 32: Restrictive conditions of instantiation between nonadjacent levels.

The last condition concerning level stratification is shown in Figure 33, capturing the restriction that a type cannot specialize another type in a different level (whether directly or indirectly).

Figure 33: Restrictive condition of specialization between different levels.

3.3.3. Constraints Related to Categorization and Specialization

The MLT theory summarized in Section 3.1 defines one more condition with respect to categorization, that stems directly from the definition of an Odell powertype. This condition is shown in Figure 34, requiring that all instances of a categorizer must be a proper specialization of the categorized type.

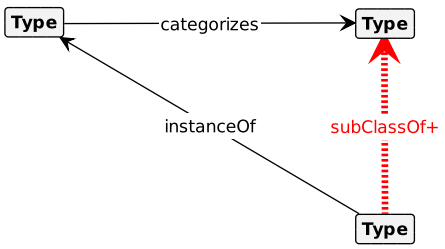


Figure 34: Restrictive condition of a categorizer instance not proper specializing the categorized type.

A similar condition follows straightforwardly from the semantics of specialization. This condition is depicted in Figure 35, where an instance of a type must instantiate all of its (direct or indirect) supertypes.

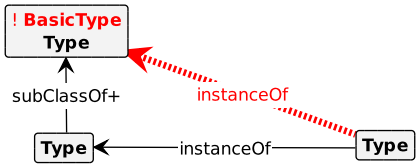


Figure 35: Restrictive condition of an instance violating the definition of specialization.

3.3.4. Verifying Soundness

Similarly to what was reported in Section 2.3.1 for the endurant types grammar, in order to verify that the graph grammar proposed for multi-level taxonomies is sound we need to enumerate its language, constructing all possible taxonomies reachable by any sequence of rule applications. Subsequently, the graph conditions presented for multi-level modeling are checked against these constructed taxonomies. If a model triggers any of the graph conditions, then the taxonomy is incorrect, because it violates some well-formedness rules from the theory. Consequently, the goal of the soundness analysis is to verify that no taxonomy in the language is incorrect.

As done in Section 2.3.1, given that the grammar language is *infinite*, we perform a *bounded* exploration with the GROOVE tool, with bound N being the number of types present in a taxonomy. The tool managed to generate a total of 149,282 taxonomies up to a bound $N = 7$, with a breakdown of this total per bound value shown in Table 3. The table also shows that the soundness goal was validated (at least up to $N = 7$), with no taxonomies being flagged as incorrect by the graph conditions.

# types (N)	Produced taxonomy shapes	Incorrect taxonomy shapes
2	2	0
3	5	0
4	22	0
5	196	0
6	3,685	0
7	145,372	0

Table 3: Results of soundness analysis for the MLT-based graph grammar.

Once more, the exponential growth of the number of possible taxonomies w.r.t. bound N prevented an enumeration for larger values, a common limiting factor when performing an exhaustive explicit enumeration of a grammar language [19]. For the MLT-based grammar, an enumeration for $N = 8$ and beyond was not possible due to time and memory limitations. Nevertheless, to support the significance of the soundness results presented in Table 3, we again rely on the *small scope hypothesis*, as discussed in Section 2.3.3, which requires violations to be detectable in small taxonomy shapes. Concerning this matter, the restrictive conditions on basic types (Figures 25–28) can be violated with taxonomy shapes of size 2 and 3. Conditions on stratification (Figures 29–33) can be violated with taxonomy shapes of size 5 or less. Conditions

on categorization and specialization (Figures 34 and 35) can be violated with taxonomy shapes of size 5 (2 of which are basic types).

3.3.5. Verifying Completeness

For the completeness analysis of the MLT-based grammar we repeat the same procedures described in Section 2.3.2, where we consider both correct and incorrect taxonomies. As before, this requires the elaboration of an additional, permissible graph grammar, that allows the creation of correct and incorrect models. The construction of taxonomies with this grammar starts from a host graph that contains only the basic type representing the class of *individuals*, as shown in Figure 19. This grammar is composed by five rules:

- A rule to create new levels of classification, as in Figure 21;
- A rule to create a new non-basic type as a specialization of any existing type;
- A rule to introduce a categorizer of any non-basic type as a specialization of any existing type;
- A rule to introduce a specialization relation between any two non-basic types, avoiding circularity;
- And a rule to introduce an instantiation relation between any two non-basic types, avoiding circularity.

The exploration results for this permissible grammar are presented in Table 4. The second column lists the number of taxonomy shapes created with the permissible grammar, both correct and incorrect. As we can see, at least up to $N = 5$, the number of correct taxonomy shapes matches those produced by our grammar (second column of Table 3). Even more acutely than in the case of the completeness analysis of the UFO-based grammar presented earlier, there is an exponential growth in the number of shapes, fueled by the additional relations introduced in a multi-level taxonomy (instantiation and characterization). This suggests further work is required to support a completeness claim. Note that given the formation constraints of the grammar, a very large portion of the taxonomy shapes generated by the permissive (control) grammar is incorrect. This shows that much is required from a free-form modeler to produce a correct model on their own by employing what we call – in comparison with a high-level pattern language – 'low-level' primitives; the liberty in an unrestricted setting is so vast that some mistakes are bound to occur, especially in large constructs. Therefore, tool automation and assistance is of the utmost importance for a model designer.

# types (N)	**All taxonomy shapes**	**Incorrect shapes**	**Correct shapes**
2	2	0	2
3	23	18	5
4	4,064	4,042	22
5	6,480,294	6,480,098	196

Table 4: Results of completeness analysis for the MLT-based graph grammar.

4. Joining the Two Foundational Theories

Having designed and assessed two independent graph grammars each corresponding to a foundational theory, we are now ready to combine both grammars into a single rule set, which is presented in Section 4.1. Subsequently, in Section 4.2, we reuse the graph conditions for the grammar of endurant types (Section 2.3) and the grammar of multi-level taxonomies (Section 3.3), to show that the results of soundness still carry over to the combined grammar. In other words, the combined grammar consolidates rules that guarantee the production of correct *ontologically well-founded multi-level taxonomies*.

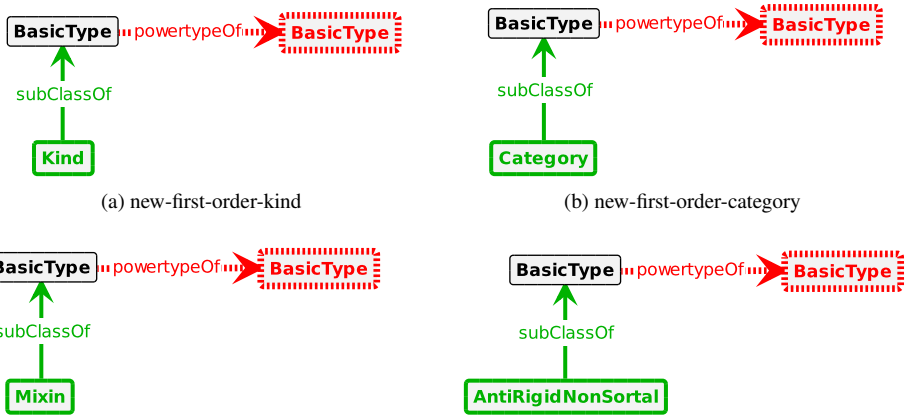
As discussed in [35], the ontological distinctions among endurant types discussed in section 2.1 can also apply to higher-order types. For example, *Bird Species* can be conceived as a second-order KIND, whose instances are individual bird species such as *American Eagle* or *Emperor Penguin*, which instantiate that type necessarily. In their turn, *Endangered Bird Species* or *Extinct Bird Species* are anti-rigid second-order types (phases) that can contingently classify instances of *Bird Species*, and *Recognized Bird Species* is an anti-rigid second-order type (a role) played by instances of *Bird Species* when officially recognized. Hence, a combined theory is required to take into account these ontological distinctions for higher-order types in multi-level taxonomies.

4.1. Graph Transformation Rules to Build Ontologically Correct Multi-Level Taxonomies

Our construction of ontology-based multi-level taxonomies starts from the same host graph of Section 3.2 (Figure 19). This initial host graph contains only a single basic type representing the class of *individuals*.

4.1.1. Introducing First-Order New Types

The endurant independent types originally presented in Section 2.2.1, i.e., KINDS, CATEGORIES, MIXINS and ANTI-RIGID NON-SORTALS, are now introduced in the first level as a direct specialization of the basic type representing the INDIVIDUAL class. The amalgamation of the rules given in Figures 5(a-d) and 20 lead to the new rules shown in Figures 36(a-d).



(a) new-first-order-kind

(b) new-first-order-category

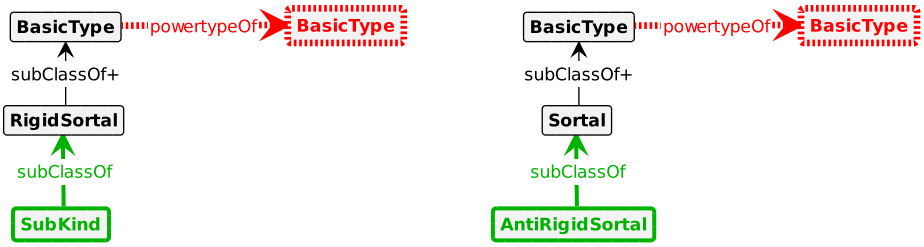
(c) new-first-order-mixin

(d) new-first-order-antirigid-nonsortal

Figure 36: Transformation rules to introduce a new first-order independent type.

4.1.2. Introducing First-Order Dependent Types

Similarly to the independent ones, the endurant dependent types from Section 2.2.2, i.e., SUBKINDS and ANTI-RIGID SORTALS, are introduced in the first level as a specialization of some SORTAL type from which they inherit a principle of identity. The new rules depicted in Figures 37(a, b) result from the combination of the rules originally given in Figures 6, 7 and 20. In case of a SUBKIND, its principle of identity is inherited from a RIGID-SORTAL. In case of an ANTI-RIGID SORTAL, it is inherited from any SORTAL.



(a) new-first-order-subkind (b) new-first-order-antirigid-sortal

Figure 37: Transformation rules to introduce a new first-order dependent type.

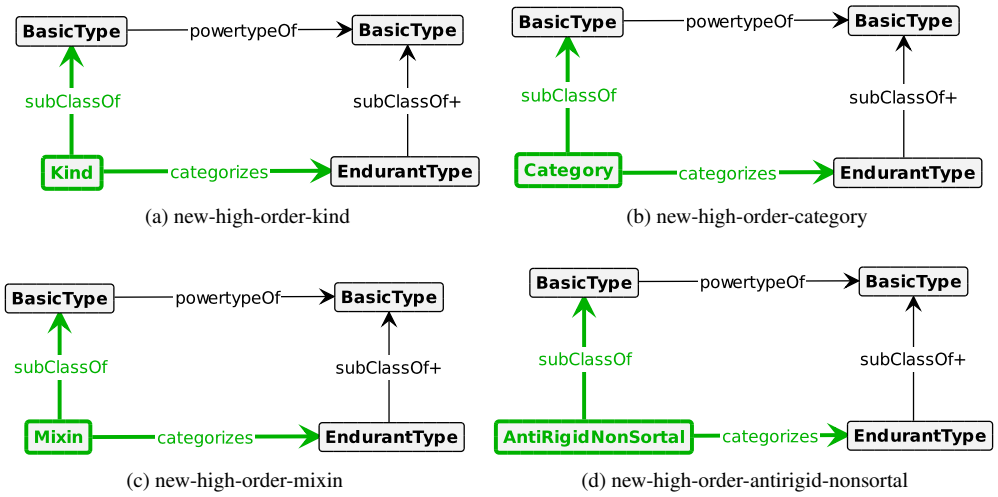
4.1.3. Introducing a New Order

A new order in the taxonomy is introduced in the same way as discussed in Section 3.2.2, via the creation of a *powertype* of a basic type in the highest order so far. This is accomplished by the rule shown in Figure 38, which is the same as the original rule in Figure 21, being repeated here just for convenience.

Figure 38: Transformation rule to introduce a new order.

4.1.4. Introducing High-Order New Types

Endurant independent types are introduced in high levels as a direct specialization of a basic type representing some high-order, categorizing any ENDURANT TYPE at one order below, as shown in Figures 39(a-d). These new rules result from the combination of the ones presented in Figures 5(a-d) and 22. Once more the instantiations of basic types remain implicit as they can be inferred from the powertype declaration: all subtypes of the base type are, by definition, instances of the powertype.



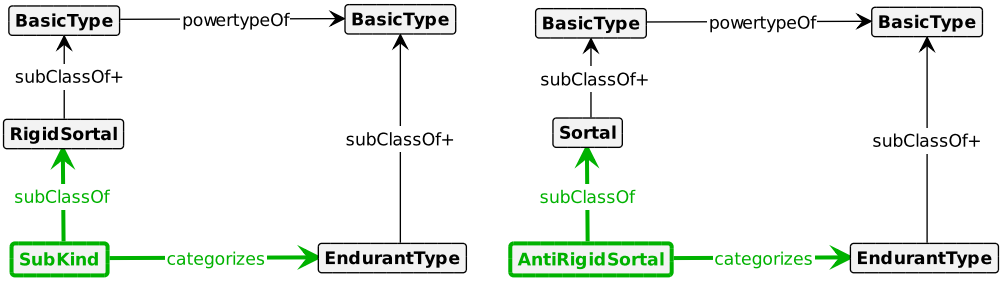
(a) new-high-order-kind (b) new-high-order-category

(c) new-high-order-mixin (d) new-high-order-antirigid-nonsortal

Figure 39: Transformation rules to introduce a new high-order independent type.

4.1.5. Introducing High-Order Dependent Types

Endurant dependent types are introduced in high levels as a specialization of some SORTAL type from which they inherit a principle of identity, respecting the *rigidity* ontological principle, and categorizing any ENDURANT TYPE at one order below. This is performed by the rules shown in Figures 40(a, b), which are the product of the amalgamation of the original rules presented in Figures 6, 7 and 22.

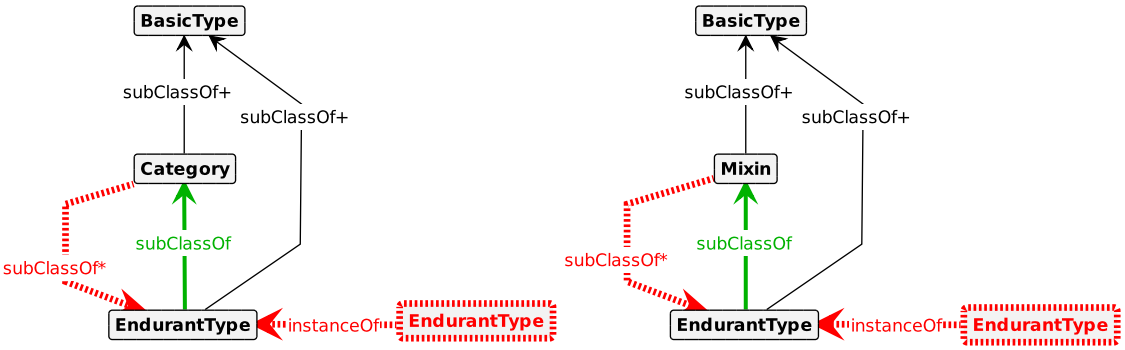


(a) new-high-order-subkind (b) new-high-order-antirigid-sortal

Figure 40: Transformation rules to introduce a new high-order dependent type.

4.1.6. Introducing Specializations for Existing Non-Sortal Types

Proceeding with the merging of the rules in Sections 2.2.3 and 3.2.4, CATEGORIES and MIXINS can be specialized in any ENDURANT TYPE without instances at the same level, and ANTI-RIGID NON-SORTALS can be specialized in an ANTI-RIGID TYPE without instances at the same level. Two new rules are shown in Figures 41(a, b), resulting from combining the ones from Figures 8(a, b) and 23. Additionally, the new rule depicted in Figure 42 stems from the merging of original rules from Figures 9 and 23.



(a) category-specialization (b) mixin-specialization

Figure 41: Transformation rules to specialize a CATEGORY or a MIXIN.

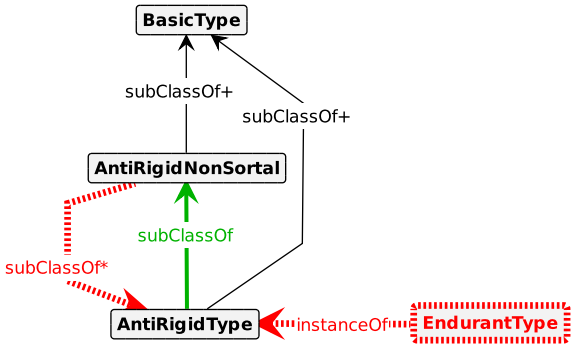
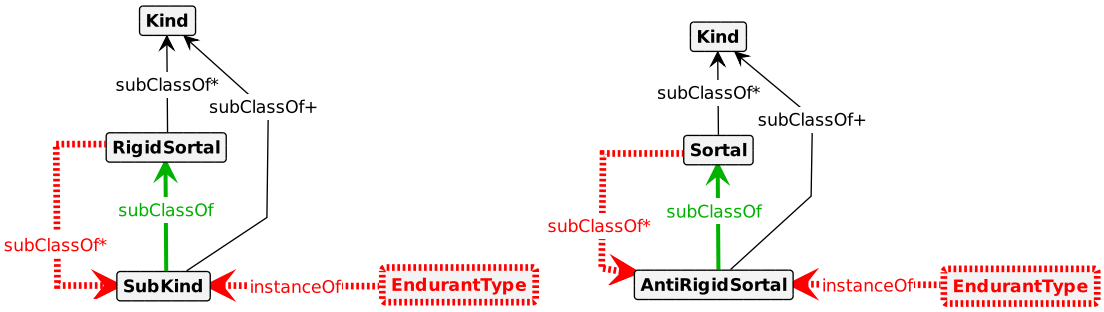


Figure 42: Transformation rule to specialize an ANTI-RIGID NON-SORTAL.

4.1.7. Introducing Generalizations for Existing Sortal Types

In line with the discussion from Sections 2.2.4 and 3.2.4, SORTAL types without instances can be generalized accordingly to the rules in Figures 43(a, b), which are formed from the original ones in Figures 10(a, b) and 23.



(a) subkind-generalization (b) antirigid-sortal-generalization

Figure 43: Transformation rules to generalize a SORTAL.

4.1.8. Classifying Types

Finally, to introduce a new instantiation relation similar to the one described in Section 3.2.5, we adapt the original rule from Figure 24 to consider only the instantiation of ENDURANT TYPES, preventing the instantiation of two different KINDS by the same type, yielding the new rule shown in Figure 44.

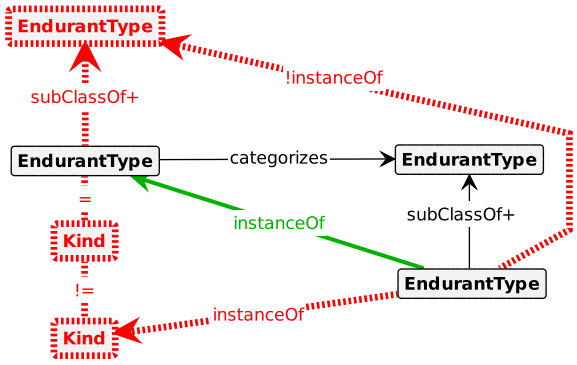


Figure 44: Transformation rule to classify an ENDURANT TYPE.

4.2. Formal Verification

We perform the same verification process as previously described for each individual micro-theory grammar. Luckily, given that the ontological restrictions from each micro-theory are orthogonal, our new set of well-formedness restrictions can be composed by just taking the graph conditions presented for the grammar of endurant types (Section 2.3) and the grammar of multi-level taxonomies (Section 3.3) in tandem, except for one restriction, which is of a type instantiating simultaneously two different KINDS. This last restriction, that only appears when both micro-theories are considered in tandem, is presented in Figure 45.

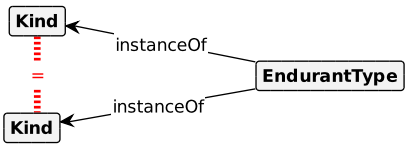


Figure 45: Restrictive condition of a type instantiating multiple KINDS.

We once more run a language enumeration, this time to analyze the combined graph grammar. The GROOVE tool managed to generate a total of 2,389,713 correct taxonomy shapes of up to 7 types. The results obtained with this experiment are summarized in Table 5. Once again, we can see that the soundness property holds at least up to 7 types. As discussed in Section 3.3.5 for the MLT-based grammar, a completeness analysis beyond $N = 5$ with the proposed strategy is not feasible, and further work is required to support a completeness claim for the joint grammar, possibly involving partial exploration of the state space or proof assistants.

# types (N)	Produced taxonomy shapes	Incorrect taxonomy shapes
2	5	0
3	26	0
4	202	0
5	2,514	0
6	55,581	0
7	2,330,512	0

Table 5: Results of soundness analysis for the combined grammar.

5. Related Work

Graph grammars and graph transformation (GT) have long been advocated as a suitable formalism for the specification, analysis and verification of models from many distinct domains [36]. One of the reasons for this proposition is the smaller syntactic gap between some modeling languages and graph rewriting. Since several languages are graphical, GT rules fit naturally within the model syntax; much more so, for instance, than when compared with text-based formalisms. For instance, Triple Graph Grammars (TGGs), a specialized form of graph grammar, have been applied successfully for several years in a range of application scenarios including: model generation, conformance testing, bidirectional model transformation, and incremental model synchronization [37]. In fact, model transformation is a very active field of research where the modeling and GT communities intersect. As a particular example, one case study in [19] describes how GT rules can be used to transform from BPMN to BPEL models. This formalization identified inconsistencies in the original transformation semantics, which was only informally specified.

Several other GT tools have been developed to perform some kind of model transformation or analysis. For instance, Fujaba [38] is a tool with support for model-based software engineering and re-engineering, employing UML class diagrams and specialized activity diagrams (called story diagrams in the tool), that are based on graph transformations. Viatra2 [39] and Henshin [40] are model transformation tools that are part of the Eclipse Modeling Framework (EMF), and thus can handle Ecore models.

Closer to conventional programming languages and methods, GrGen.NET [41] is a GT-based tool designed for compiler optimization and code refactoring. Similarly, the domain-specific language Chart [42] adds GT-based functionality to existing Java programs. The approach relies on a set of annotations to identify the intended graph structure, as well as on user methods to manipulate that structure, within the user's own Java class declarations. The advantage of the approach is that it allows any Java program to be enhanced, non-invasively, with declarative graph rules that can later be used for program analysis. Along the lines of programming language semantics, a type graph (a sort of graph grammar metamodel) for Java was defined in [43] and later refined in [18]. Subsequently, the GROOVE tool was used to define a complete formalization for the control flow semantics of Java [44].

As a final note, it is worth mentioning that the GT rules used in GROOVE have the same expressive power than First-Order Logic (FOL) [45]. Thus, any ontological theory based on FOL can, in principle, be properly supported/formalized by a GROOVE graph grammar.

The work discussed here is also related to other efforts that employ foundational theories in conceptual modeling tasks. As discussed in [4], UFO's typology of endurant types was originally inspired by the typology of types on which the OntoClean methodology is based. A pioneering and among the most important methodologies for the construction of ontologically correct taxonomies, OntoClean is extended in important ways by the graph grammar proposed here: (i) firstly, because OntoClean does not provide any concrete modeling mechanism for building taxonomies, but only a set of methodological guidelines and only for taxonomy *evaluation*. Our proposal instead formalizes a grammar with concrete ontological patterns as modeling constructs; (ii) secondly, OntoClean is focused (even if not explicitly) on object types (also called substantial types) as opposed to more generally addressing taxonomies formed by other endurant types (e.g., types whose instances are dependent entities such as symptoms, marriages, enrollments, etc. [5]): (iii) finally, OntoClean does not address the construction of higher-order types. In summary, our proposal provides for a concrete modeling mechanism that supports the construction of taxonomic structures that go much beyond what is covered by OntoClean.

The aforementioned points (ii-iii) but also (i) (regarding the pattern-based representation support) are also true for theories of type structures that extend OntoClean (e.g [46–48]). In particular, they also apply to a series of proposals of Order-Sorted Logics extended with ontological predicate meta-hierarchies. These logics play an important role in knowledge representation and, more generally, in symbolic artificial intelligence by supporting the construction and formal verification of ontologically-informed taxonomies, as well as computationally tractable automated reasoning with them. These include: [49, 50], which propose new order-sorted modal logics and tableau calculus to check the (un)satisfiability and validity of sorted modal formulas; [51, 52], which propose order-sorted horn-calculus combining ontologies and logic programming.

6. Final Considerations

We developed a formally specified, ontologically well-founded, metamodel-independent and sound graph grammar for the elaboration of multi-level taxonomies. This expands on the current state-of-the-art modeling methods, by proposing a novel technique that leads to the development of (multi-level) taxonomies that are correct by construction. We have accomplished that by leveraging on a *typology of endurant types* and a *theory of high-order types*. The typology of endurant types is part of the Unified Foundational Ontology (UFO) [15], and which underlies the Ontology-Driven Conceptual Modeling language OntoUML [5]. The theory of high-order types was conceived as a foundation for multi-level models [9, 10] and was also applied successfully in

a number of initiatives, including the definition of a well-founded multi-level modeling language [11].

The original theory of UFO puts forth a number of ontological distinctions based on formal meta-properties. As a result of the logical characterization of these meta-properties, we have that certain structures (patterns) are imposed on the language primitives representing these distinctions [6]. We have identified a set of primitive operations on taxonomic structures that incorporates the ontological distinctions of UFO and multi-level modeling concepts from MLT, and that guarantees the correctness of the generated taxonomies. This forms the basis for the systematic design of such structures at a higher level of abstraction.

Given the limitations of metamodels as a mechanism for representing the abstract syntax of a language, these structures were not treated as first-class citizens before and have remained hidden in the abstract syntax of the original OntoUML proposal [4]. This paper addresses this exact problem. By leveraging on that theory, we propose a *pattern grammar* (formalized as a graph grammar) that embeds these distinctions and multi-level modeling concepts, ensuring *by design* the construction of taxonomic structures that abide by the formal constraints governing their relations. The work proposed here is inspired by the work in [7] and advances the work initiated in [8]. For example, by employing the state exploration mechanism supported by GROOVE, we managed to detect important omissions in the rule set proposed in that first work.

As we have shown, the ratio between incorrect and correct taxonomies is enormous, and worse still, grows exponentially as the number of elements in the model grows. This provides support for our claim that the graph grammar patterns identified here are indeed 'higher-level' constructs when compared to the direct creation of modeling elements and relations in conventional (free-form) modeling environments. Moreover, the results presented here can be leveraged in the engineering of computational tools that guarantee the correctness of taxonomies by design. In contrast, given the significance of the gap between what one can do and one intends to do, in an unrestricted modeling setting, mistakes are bound to occur, especially in the case of large taxonomies.

Another important aspect is that our proposal captures the representation consequences of ontology theories in a way that is metamodel-independent. For this reason, these results can be carried out to other languages and platforms. In particular, we are currently developing a plugin for Protégé that, among other things, implements the primitive operations proposed in this paper. This plugin is intended to be used in tandem with the gUFO ontology (a lightweight implementation of UFO) [53]. In that implementation, these operations take the form of ontology patterns to be applied, to support its users in modeling consistent Semantic Web ontologies.

All the graph grammars produced in this paper can be obtained at `https://github.com/nemo-ufes/ufo-mlt-taxonomy-graph-grammar`.

Acknowledgments

We are grateful to Ricardo A. Falbo (*in memoriam*) for the spark that led to this work. This research is partly funded by Brazilian funding agencies CNPq (grants numbers 313687/2020-0 and 407235/2017-5), CAPES (Finance Code 001 and grant number 23038.028816/2016-41) and FAPES (grant number 281/2021).

References

[1] N. Guarino, C. A. Welty, An overview of OntoClean, in: S. Staab, R. Studer (Eds.), Handbook on Ontologies, International Handbooks on Information Systems, Springer, 2004, pp. 151–172.

[2] A. Oltramari, A. Gangemi, N. Guarino, C. Masolo, Restructuring WordNet's top-level: The OntoClean approach, Vol. 49, 2002.
[3] G. Guizzardi, G. Wagner, N. Guarino, M. van Sinderen, An ontologically well-founded profile for UML conceptual models, in: International Conference on Advanced Information Systems Engineering, Vol. 3084 of Lecture Notes in Computer Science, Springer, 2004, pp. 112–126. doi:10.1007/978-3-540-25975-6_10.
[4] G. Guizzardi, Ontological foundations for structural conceptual models, no. 15 in Telematica Institute Fundamental Research Series, University of Twente, 2005.
[5] G. Guizzardi, C. Fonseca, A. B. Benevides, J. Almeida, D. Porello, T. Sales, Endurant Types in Ontology-Driven Conceptual Modeling: Towards OntoUML 2.0, in: Conceptual Modeling - 37th International Conference, ER 2018, Vol. 11157 of Lecture Notes in Computer Science, Springer, 2018, pp. 136–150. doi:10.1007/978-3-030-00847-5_12.
[6] F. B. Ruy, G. Guizzardi, R. A. Falbo, C. C. Reginato, V. A. Santos, From reference ontologies to ontology patterns and back, Data & Knowledge Engineering 109 (2017) 41–69. doi:10.1016/j.datak.2017.03.004.
[7] E. Zambon, G. Guizzardi, Formal definition of a general ontology pattern language using a graph grammar, in: 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), Vol. 11 of Annals of Computer Science and Information Systems, 2017, pp. 1–10. doi:10.15439/2017F001.
[8] J. O. Batista, J. P. A. Almeida, E. Zambon, G. Guizzardi, Building correct taxonomies with a well-founded graph grammar, in: 15th International Conference on Research Challenges in Information Science (RCIS 2021), Vol. 415 of Lecture Notes in Business Information Processing, Springer, 2021, pp. 506–522. doi:10.1007/978-3-030-75018-3_33.
[9] J. P. A. Almeida, V. Carvalho, F. Brasileiro, C. Fonseca, G. Guizzardi, Multi-Level Conceptual Modeling: Theory and Applications, in: Proc. XI Seminar on Ontology Research in Brazil and II Doctoral and Masters Consortium on Ontologies (Ontobras 2018), Vol. 2228 of CEUR Workshop Proceedings, CEUR-WS.org, 2018, pp. 26–41.
[10] V. A. Carvalho, J. P. A. Almeida, Toward a well-founded theory for multi-level conceptual modeling, Software and Systems Modeling 17 (2018) 205–231. doi:10.1007/s10270-016-0538-9.
[11] C. Fonseca, J. P. A. Almeida, G. Guizzardi, V. Carvalho, Multi-level conceptual modeling: Theory, language and application, Data & Knowledge Engineering 134. doi:10.1016/j.datak.2021.101894.
URL https://doi.org/10.1016/j.datak.2021.101894
[12] F. Brasileiro, J. P. A. Almeida, V. Carvalho, G. Guizzardi, Applying a Multi-Level Modeling Theory to Assess Taxonomic Hierarchies in Wikidata, in: Proceedings of the 25th International Conference Companion on World Wide Web, International World Wide Web Conferences Steering Committee, 2016, pp. 975–980. doi:10.1145/2872518.2891117.
[13] A. A. Dadalto, J. P. A. Almeida, C. M. Fonseca, G. Guizzardi, Type or Individual? Evidence of Large-Scale Conceptual Disarray in Wikidata, in: 40th International Conference on Conceptual Modeling (ER 2021), Vol. 13011 of Lecture Notes in Computer Science, Springer, 2021, pp. 367–377. doi:10.1007/978-3-030-89022-3_29.
[14] T. Halpin, T. Morgan, Information modeling and relational databases, Morgan Kaufmann, 2010.
[15] G. Guizzardi, G. Wagner, J. P. A. Almeida, R. S. Guizzardi, Towards ontological foundations for conceptual modeling: The unified foundational ontology (UFO) story, Applied ontology 10 (3-4) (2015) 259–271.
[16] E. Hirsch, The concept of identity, Oxford University Press, 1992.
[17] R. Heckel, Graph transformation in a nutshell, Electronic notes in theoretical computer science 148 (1) (2006) 187–198. doi:10.1016/j.entcs.2005.12.018.
[18] E. Zambon, Abstract Graph Transformation – Theory and Practice, Centre for Telematics and Information Technology, University of Twente, 2013.
[19] A. H. Ghamarian, M. de Mol, A. Rensink, E. Zambon, M. Zimakova, Modelling and analysis using GROOVE, International journal on software tools for technology transfer 14 (1) (2012) 15–40. doi:10.1007/s10009-011-0186-x.
[20] L. Gammaitoni, P. Kelsen, Q. Ma, Agile validation of model transformations using compound F-Alloy specifications, Science of Computer Programming 162 (2018) 55–75. doi:10.1016/j.scico.2017.07.001.
[21] M. Roberson, M. Harries, P. T. Darga, C. Boyapati, Efficient software model checking of soundness of type systems, in: Proc. 23rd Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2008, ACM, 2008, pp. 493–504. doi:10.1145/1449764.1449803.
[22] D. Jackson, Alloy: a language and tool for exploring software designs, Communications of the ACM 62 (9) (2019) 66–76. doi:10.1145/3338843.
[23] V. A. Carvalho, J. P. A. Almeida, A Semantic Foundation for Organizational Structures: A Multi-level Approach, in: 2015 IEEE 19th International Enterprise Distributed Object Computing Conference, IEEE, 2015, pp. 50–10. doi:10.1109/EDOC.2015.18.
[24] C. Gonzalez-Perez, B. Henderson-Sellers, A powertype-based metamodelling framework, Software & Systems Modeling 5 (1) (2006) 72–90. doi:10.1007/s10270-005-0099-9.
[25] B. Neumayr, K. Grün, M. Schrefl, Multi-level domain modeling with m-objects and m-relationships, in: Proceed-

ings of the Sixth Asia-Pacific Conference on Conceptual Modeling-Volume 96, Australian Computer Society, Inc., 2009, pp. 107–116.
[26] C. Atkinson, T. Kühne, The essence of multilevel metamodeling, in: International Conference on the Unified Modeling Language, Springer, 2001, pp. 19–33. doi:10.1007/3-540-45441-1_3.
[27] J. P. A. Almeida, U. Frank, T. Kühne, Multi-Level Modelling (Dagstuhl Seminar 17492), Dagstuhl Reports 7 (2018) 18–49. doi:10.4230/DagRep.7.12.18.
[28] V. Carvalho, J. P. A. Almeida, C. Fonseca, G. Guizzardi, Multi-level ontology-based conceptual modeling, Data & Knowledge Engineering 109 (2017) 3–24. doi:10.1016/j.datak.2017.03.002.
[29] V. Carvalho, J. P. A. Almeida, G. Guizzardi, Using a Well-Founded Multi-Level Theory to Support the Analysis and Representation of the Powertype Pattern in Conceptual Modeling, in: Proc. 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016), Springer, 2016, pp. 309–324. doi:10.1007/978-3-319-39696-5_19.
[30] F. Brasileiro, J. P. A. Almeida, V. A. Carvalho, G. Guizzardi, Expressive Multi-level Modeling for the Semantic Web, in: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I, Springer, 2016, pp. 53–69. doi:10.1007/978-3-319-46523-4_4.
[31] L. Cardelli, Structural subtyping and the notion of power type, in: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1988, pp. 70–79. doi:10.1145/73560.73566.
[32] C. Atkinson, T. Kühne, Meta-level independent modelling, in: International Workshop on Model Engineering at 14th European Conference on Object-Oriented Programming, Vol. 12, 2000, p. 16.
[33] J. Odell, Power types, J. Object Oriented Program. 7 (2) (1994) 8–12.
[34] B. Henderson-Sellers, On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages, Springer Briefs in Computer Science, Springer, 2012. doi:10.1007/978-3-642-29825-7.
[35] G. Guizzardi, J. Almeida, N. Guarino, V. Carvalho, Towards an Ontological Analysis of Powertypes, in: Proceedings of the Joint Ontology Workshops 2015 Episode 1: The Argentine Winter of Ontology, Vol. 1517 of CEUR Workshop Proceedings, CEUR-WS.org, 2015.
URL http://ceur-ws.org/Vol-1517/JOWO-15_FOfAI_paper_7.pdf
[36] R. Heckel, G. Taentzer, Graph Transformation for Software Engineers: With Applications to Model-Based Development and Domain-Specific Language Engineering, Springer, 2020. doi:10.1007/978-3-030-43916-3.
[37] A. Anjorin, E. Leblebici, A. Schürr, 20 years of triple graph grammars: A roadmap for future research, Electron. Commun. Eur. Assoc. Softw. Sci. Technol. 73. doi:10.14279/tuj.eceasst.73.1031.
[38] U. Nickel, J. Niere, A. Zündorf, The FUJABA environment, Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium (2000) 742–745doi:10.1145/337180.337620.
[39] D. Varró, A. Balogh, The model transformation language of the VIATRA2 framework, Sci. Comput. Program. 68 (3) (2007) 214–234. doi:10.1016/j.scico.2007.05.004.
[40] T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer, Henshin: advanced concepts and tools for in-place EMF model transformations, in: MODELS'10, Vol. 6394 of Lecture Notes in Computer Science, Springer, 2010, pp. 121–135. doi:10.1007/978-3-642-16145-2_9.
[41] E. Jakumeit, S. Buchwald, M. Kroll, GrGen.NET - the expressive, convenient and fast graph rewrite system, International Journal on Software Tools for Technology Transfer 12 (3-4) (2010) 263–271. doi:10.1007/s10009-010-0148-8.
[42] M. de Mol, A. Rensink, J. J. Hunt, Graph transforming Java data, in: FASE, Vol. 7212 of Lecture Notes in Computer Science, Springer, 2012, pp. 209–223. doi:10.1007/978-3-642-28872-2_15.
[43] A. Rensink, E. Zambon, A type graph model for Java programs, in: FMOODS/FORTE, Vol. 5522 of Lecture Notes in Computer Science, Springer, 2009, pp. 237–242. doi:10.1007/978-3-642-02138-1_18.
[44] E. Zambon, A. Rensink, Recipes for coffee: compositional construction of Java control flow graphs in GROOVE, in: Principled Software Development, 2018, pp. 305–323. doi:10.1007/978-3-319-98047-8_19.
[45] A. Rensink, Representing first-order logic using graphs, in: ICGT, Vol. 3256 of Lecture Notes in Computer Science, Springer, 2004, pp. 319–335. doi:10.1007/978-3-540-30203-2_23.
[46] C. Welty, W. Andersen, Towards OntoClean 2.0: A framework for rigidity, Applied Ontology 1 (1) (2005) 107–116.
[47] P. Seyed, A Method for Evaluating Ontologies - Introducing the BFO-Rigidity Decision Tree Wizard, in: Formal Ontology in Information Systems - Proceedings of the Seventh International Conference, FOIS 2012, Vol. 239 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2012, pp. 191–204. doi:10.3233/978-1-61499-084-0-191.
[48] A. P. Seyed, Integrating ontoclean's notion of unity and identity with a theory of classes and types, in: Formal Ontology in Information Systems: Proceedings of the Seventh International Conference (FOIS 2012), Vol. 239 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2012, pp. 205–218. doi:10.3233/978-1-61499-084-0-205.
[49] K. Kaneiwa, R. Mizoguchi, An order-sorted quantified modal logic for meta-ontology, in: International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Vol. 3702 of Lecture Notes in Computer

Science, Springer, 2005, pp. 169–184. doi:10.1007/11554554_14.
[50] K. Kaneiwa, Existential rigidity and many modalities in order-sorted logic, Knowledge-Based Systems 24 (5) (2011) 629–641. doi:10.1016/j.knosys.2011.02.001.
[51] K. Kaneiwa, P. H. Nguyen, Decidable order-sorted logic programming for ontologies and rules with argument restructuring, in: International Semantic Web Conference, Springer, 2009, pp. 328–343. doi:10.1007/978-3-642-04930-9_21.
[52] K. Kaneiwa, Order-sorted logic programming with predicate hierarchy, Artificial Intelligence 158 (2) (2004) 155–188. doi:10.1016/j.artint.2004.05.001.
[53] J. P. A. Almeida, G. Guizzardi, R. A. Falbo, T. P. Sales, gUFO: a lightweight implementation of the Unified Foundational Ontology (UFO), http://purl.org/nemo/doc/gufo (2019).