# Language Independent Refinement using Partial Modeling

Rick Salay, Michalis Famelis, Marsha Chechik

Department of Computer Science, University of Toronto, Toronto, Canada
{rsalay,famelis,chechik}@cs.toronto.edu

**Abstract.** Models express not only information about their intended domain but also about the way in which the model is incomplete, or "partial". This partiality supports the modeling process because it permits the expression of what is known without premature decisions about what is still unknown, until later refinements can fill in this information. A key observation of this paper is that a number of partiality types can be defined in a modeling language-independent way, and we propose a formal framework for doing so. In particular, we identify four types of partiality and show how to extend a modeling language to support their expression and refinement. This systematic approach provides a basis for reasoning as well as a framework for generic tooling support. We illustrate the framework by enhancing the UML class diagram and sequence diagram languages with partiality support and using Alloy to automate reasoning tasks.

## 1 Introduction

Models are used for expressing two different yet complementary kinds of information. The first is about the *intended domain* for the modeling language. For example, UML class diagrams are used to express information about system structure. The second kind of information is used to express the degree of incompleteness or *partiality* about the first kind. For example, class diagrams allow the type of an attribute to be omitted at an early modeling stage even though the type will ultimately be required for implementation. Being able to express partiality within a model is essential because it permits a modeler to specify the domain information she knows without prematurely committing to information she is still uncertain about, until later refinements can add it.

The motivating observation of this work is that *many types of model partiality are actually domain independent* and thus *support for expressing partiality can be handled in a generic and systematic way in any modeling language!* Furthermore, each type of partiality has its own usage scenarios and comes with its own brand of refinement. Thus, we can define certain model refinements formally yet independently of the language type and semantics. This may be one reason why many practitioners of modeling resist the formalization of the domain semantics for a model: it is possible to do some sound refinements without it!

Current modeling languages incorporate partiality information in ad-hoc ways that do not clearly distinguish it from domain information and leave gaps in

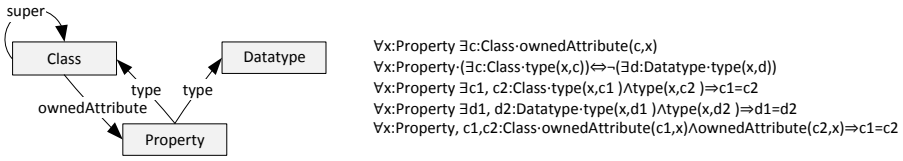



**Fig. 1.** A simplified UML class diagram metamodel.

expressiveness. For example, with a state machine diagram, if the modeler uses multiple transitions on the same event out of a state, it may not be clear (e.g., to another modeler) whether she is specifying a non-deterministic state machine (domain information) or an under-specified deterministic state machine (partiality information). Benefits of explicating partiality in a language-independent manner include generic tool support for checking partiality-reducing refinements, avoiding gaps in expressiveness by providing complete coverage of partiality within a modeling language, and reusing a modeler's knowledge by applying partiality across different modeling languages consistently. Ad-hoc treatments of partiality do not allow the above benefits to be effectively realized. Our approach is to systematically add support for partiality information to any language in the form of annotations with well-defined formal semantics and a refinement relation for reducing partiality.

The use of partiality information has been studied for particular model types (e.g., behavioural models [9,13]) but our position paper [3] was the first to discuss language-independent partiality and its benefits for Model Driven Engineering. In this paper, we build on this work and provide a framework for defining different types of language-independent partiality. Specifically, this paper makes the following contributions: (1) we define the important (and novel) distinction between domain and partiality information in a modeling language; (2) we describe a formal framework for adding support for partiality and its refinement to any modeling language; (3) we use the framework to define four types of language-independent partiality that correspond to typical pragmatic modeling scenarios; (4) we implement the formalization for these using Alloy and show some preliminary results.

The rest of this paper is organized as follows: We begin with a brief introduction to the concept of partiality in Section 2 and give an informal description of four simple language-independent ways of adding partiality to a modeling language. We describe the composition of these partiality types and illustrate them through application to the UML class diagram and sequence diagram languages in Section 3. In Section 4, we describe a formalization of these types of partiality. In Section 5, we describe our tool support based on the use of Alloy [8]. After discussing related work in Section 6, we conclude the paper in Section 7 with the summary of the paper and suggestions for future work.

## 2 Adding Partiality to Modeling Languages

When a model contains partiality information, we call it a *partial* model. Semantically, it represents the set of different possible concrete (i.e., non-partial) models that would resolve the uncertainty represented by the partiality. In this

paper, we focus on adding partiality information to existing modeling languages in a language-independent way, and thus, we must work with arbitrary metamodels. Figure 1 gives an example of a simple metamodel for class diagrams, with boxes for element types and arrows for relations. The well-formedness constraints (on the right) express the fact that every `Property` must have one `type` given by a `Class` or a `Datatype` and must be an `ownedAttribute` of one `Class`. Models consist of a set of *atoms* - i.e., the elements and relation instances of the types defined in its metamodel. In order to remain language-independent, we assume that partiality information is added as annotations to a model.

**Definition 1 (Partial model)** *A partial model $P$ consists of a* base model*, denoted $bs(P)$, and a set of annotations. Let $T$ be the metamodel of $bs(P)$. Then, $[P]$ denotes the set of $T$ models called the* concretizations *of $P$.*

Partiality is used to express uncertainty about the model until it can be resolved using *partiality refinement*. Refining a partial model means removing partiality so that the set of concretizations shrinks until, ultimately, it represents a single concrete model. In general, when a partial model $P'$ refines another one $P$, there is a mapping from $bs(P')$ to $bs(P)$ that expresses the relationship betwen them and thus between their concretizations. We give examples of such mappings later on in this section. In the special case that the base models are equivalent, $P'$ refines $P$ iff $[P'] \subseteq [P]$.

We now informally describe four possible partiality types, each adding support for a different type of uncertainty in a model: *May* partiality – about existence of its atoms; *Abs* partiality – about uniqueness of its atoms; *Var* partiality – about distinctness of its atoms; and *OW* partiality – about its completeness.

***May* partiality.** Early in the development of a model, we may be unsure whether a particular atom should exist in the model and defer the decision until a later refinement. *May* partiality allows us to express the level of certainty we have about the presence of a particular atom in a model, by annotating it. The annotations come from the lattice $\mathcal{M} = \langle \{\text{E}, \text{M}\}, \preceq \rangle$, where the values correspond to "must exist" (E) and "may exist" (M), respectively, $\prec$ means "less certain than", and $\text{M} \prec \text{E}$.

A *May* model is refined by changing M atoms to E or eliminating them altogether. Thus, refinements result in submodels with more specific annotations. The *ground* refinements of a *May* model $P$ are those that have no M elements and thus, correspond to its set of concretizations $[P]$. Figure 2(a) gives an example of a *May* model (`P`), a refinement (`P'`) and a concretization (`M`). The models are based on the metamodel in Figure 1. Atoms are given as *name:type* with the above annotations, and mappings between models are shown using dashed lines. Model (`P`) says "there is a class `Car` that may have a superclass `Vehicle` and may have a `Length` attribute which may be of type `int`". The refinement (`P'`) and concretization (`M`) resolve the uncertainty.

***Abs* partiality.** Early in the development of a model we may expect to have collections of atoms representing certain kinds of information but not know exactly what those atoms are. For example, in an early state machine diagram for a

text editor, we may know that we have `InputingStates`, `ProcessingStates` and `FormattingStates`, and that `InputingStates` must transition to `ProcessingStates` and then to `FormattingStates`. Later, we refine these to sets of particular concrete states and transitions. *Abs* partiality allows a modeler to express this kind of uncertainty by letting her annotate atoms as representing a "particular", or unique, element (P) or a "set" (S). The annotations come from the lattice $\mathcal{A} = \langle \{\text{P}, \text{S}\}, \preceq \rangle$, where $\preceq$ indicates "less unique than", and $\text{S} \prec \text{P}$.

A refinement of an *Abs* model elaborates the content of "set" atoms S by replacing them with a set of S and P atoms. The ground refinements of an *Abs* model $P$ are those that have no S elements and thus, correspond to its set of concretizations $[P]$. Figure 2(b) illustrates an *Abs* model, a refinement and concretization. Only node mappings are shown to reduce visual clutter. Model (P) says "class `Car` has a set `SizeRelated` of attributes with type `int`". The refinement (P′) refines `SizeRelated` into a particular attribute `Length` and the set `HeightRelated`.

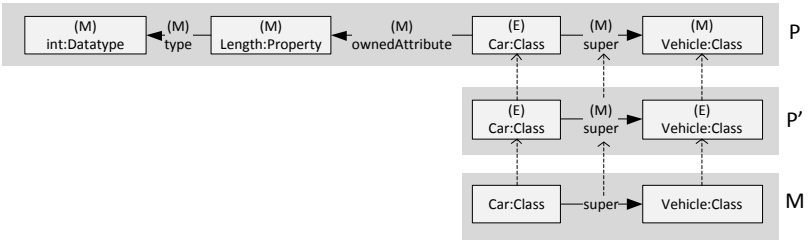
***Var* partiality.** Early in a modeling process, we may not be sure whether two atoms are distinct or should be the same, i.e., we may be uncertain about atom identity. For example, in constructing a class diagram, we may want to introduce an attribute that is needed, without knowing which class it will ultimately be in. To achieve well-formedness, it must be put into *some* class but we want to avoid prematurely putting it in the wrong class. To solve this problem, we could put it temporarily in a "variable" class - i.e., something that is treated like a class but, in a refinement, can be equated (merged) with other variable classes and eventually be assigned to a constant class. *Var* partiality allows a modeler to express uncertainty about distinctness of individual atoms in the model by annotating an atom to indicate whether it is a "constant" (C) or a "variable" (V). The annotations come from the lattice $\mathcal{V} = \langle \{\text{C}, \text{V}\}, \preceq \rangle$, where $\text{V} \prec \text{C}$.

A refinement of a *Var* model involves reducing the set of variables by assigning them to constants or other variables. The ground refinements of a *Var* model $P$ are those that have no V elements and thus, correspond to its set of concretizations $[P]$.

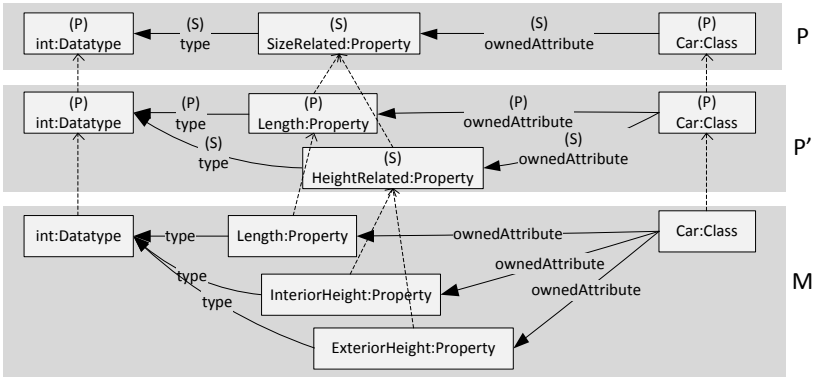
Figure 2(c) illustrates a *Var* model, its refinement and concretization. Model (P) says "class `Car` has superclass `Vehicle` and variable class `SomeVehicle` has attribute Length with variable type `SomeType`". Refinement (P′) resolves some uncertainty by assigning `SomeVehicle` to `Car`.

***OW* partiality.** It is common, during model development, to make the assumption that the model is still incomplete, i.e., that other elements are yet to be added to it. This status typically changes to "complete" (if only temporarily) once some milestone, such as the release of software based on the model, is reached. In this paper, we view a model as a "database" consisting of a set of syntactic facts (e.g., "a class $C_1$ is a superclass of a class $C_2$", etc.). Thus, incompleteness corresponds to an Open World assumption on this database (the list of atoms is not closed), whereas completeness – to a Closed World. *OW* partiality allows a modeler to explicitly state whether her model is incomplete

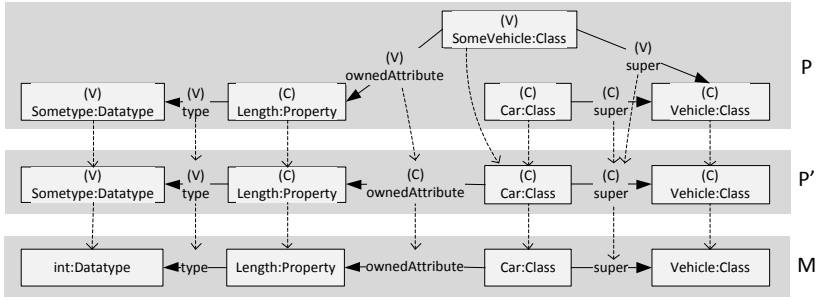
(a)




(b)




(c)




(d)

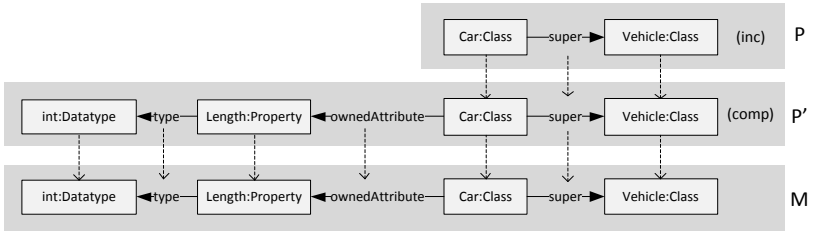



**Fig. 2.** Examples of different partiality types: (a) *May*; (b) *Abs*; (c) *Var*; (d) *OW*. In each example, model M concretizes both P′ and P, and P′ refines P.

(i.e., can be extended) (INC ) or not (COMP ). In contrast to the other types of partiality discussed in this paper, here the annotation is at the level of the entire model rather than at the level of individual atoms. The annotations come from the lattice $\mathcal{O} = \langle\{\text{COMP}, \text{INC}\}, \preceq\rangle$, where $\text{INC} \prec \text{COMP}$.

A refinement of an *OW* model means making it "more complete". The ground refinements of an *OW* model $P$, corresponding to its set of concretizations $[P]$, are its "complete" extensions. Figure 2(d) illustrates an *OW* model, refinement and concretization.

## 3 Combining and Applying Partiality Types

In this section, we show how to combine the four partiality types defined in Section 2 and then apply them to UML class diagrams and sequence diagrams, showing the language-independence of partiality-reducing refinements.

**Combining Partiality Types.** The four partiality types described above have distinctly different pragmatic uses for expressing partiality and can be combined within a single model to express more situations. We refer to the combination as the *MAVO* partiality, which allows model atoms to be annotated with *May*, *Abs* and *Var* partiality by using elements from the product lattice $\mathcal{M} \times \mathcal{A} \times \mathcal{V}$ defined as $\mathcal{MAV} = \langle\{\text{E}, \text{M}\} \times \{\text{P}, \text{S}\} \times \{\text{C}, \text{V}\}, \preceq\rangle$. For example, marking a class as $(\text{M}, \text{S}, \text{C})$ means that it represents a set of classes that may be empty, while marking it as $(\text{E}, \text{S}, \text{V})$ indicates that it is a non-empty set of classes but may become a different set of classes in a refinement. *OW* partiality is also used, but only at the model level, to indicate completeness.

*MAVO* refinement combines the refinement from the four types component-wise. If *MAVO* model $P_1$ is refined by model $P_2$, then there is a mapping from the atoms of $P_1$ to those of $P_2$, and the annotation in $P_2$ has a value that is no less than any of its corresponding atoms in $P_1$. Thus, the class marked $(\text{M}, \text{S}, \text{C})$ can be refined to a set of classes that have annotations such as $(\text{M}, \text{P}, \text{C})$ or $(\text{E}, \text{S}, \text{C})$ but not $(\text{M}, \text{S}, \text{V})$. Examples of applying the *MAVO* partiality are given below.

**Application: *MAVO* Class Diagrams.** One of the benefits of the fact that a partiality type extends the base language is that we can build on the existing concrete syntax of the languages. For example, consider the *MAVO* partial class diagram `P1` shown in the top of Figure 3. We do not show ground annotations (i.e., C for Var, P for Abs, etc.) and use the same symbols as in the abstract syntax for non-ground annotations. While there may be more intuitive ways to visualize some of these types of partiality (e.g., dashed outlines for "maybe" elements), we consider this issue to be beyond the scope of this paper.

In `P1`, the modeler uses *May* partiality to express uncertainty about whether a `TimeMachine` should be a `Vehicle` or not. *May* partiality is also used with `Hovercraft` to express that the modeler is uncertain whether or not to include it and which class should be its superclass. *Var* partiality is used with "variable" class `C1` to introduce the attribute `numOfDoors : Integer` since the modeler is uncertain about which class it belongs in. *Abs* and *Var* partiality are used together to model sets of `Vehicle` attributes with unknown types with

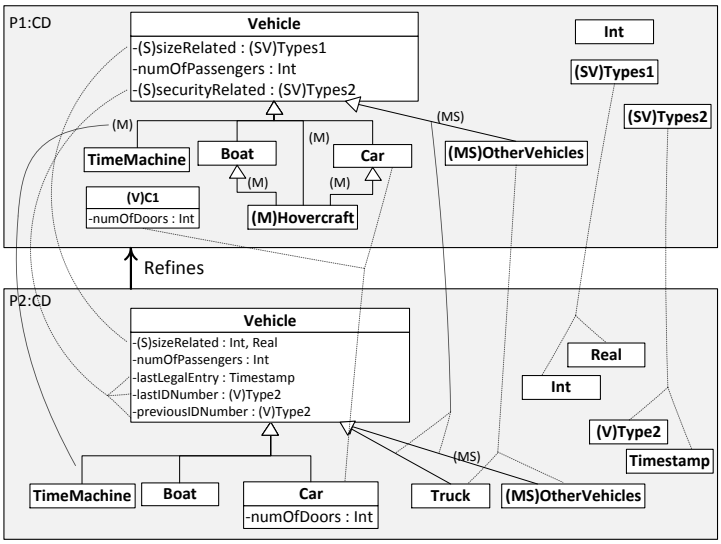



**Fig. 3.** Example of *MAVO* class diagrams with refinement.

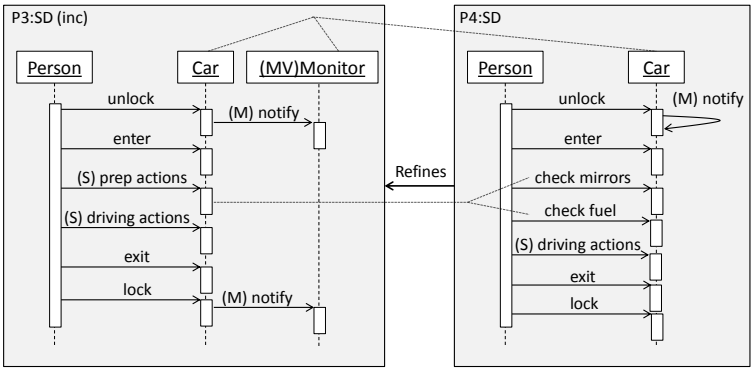



**Fig. 4.** Example sequence diagram with *MAVO* partiality.

`sizeRelated` : `Types1` and `securityRelated` : `Types2`. Finally, *May* and *Abs* partiality are used with `OtherVehicles` and `super`(`OtherVehicles`, `Vehicle`) to indicate that the modeler thinks that there may be other, not yet known, vehicle classes.

Model `P2`, on the bottom of Figure 3, is a refinement of `P1`. Refinement mappings are shown as dashed lines and, to avoid visual clutter, we omit the identity mappings between ground atoms. In `P2`, the modeler refines `super`(`TimeMachine`, `Vehicle`) from "may exist" to "exists"; however, the decision on `Hovercraft` is to omit it. The refinement puts attribute `numOfDoors` : `Integer` into `Car` by setting `C1` = `Car`. Also, the types of `sizeRelated` attributes are refined to `Int` or `Real`, and the `securityRelated` attributes are refined as well; however, the types of `LastIDNumber` and `PreviousIDNumber` are still unknown, although they are now known to be the same `SType2`. Finally, `OtherVehicles` is refined to expose `Truck`

as one of these but still leaves the possibility for more `Vehicle` subclasses. The omitted *OW* annotation indicates that the models are "complete", and thus, new elements can only be added by refining an *Abs* set such as `OtherVehicles`.

**Application: *MAVO* Sequence Diagrams.** The left model in Figure 4, `P3`, shows a *MAVO* sequence diagram specifying how a `Person` interacts with a `Car`. We follow the same concrete syntactic conventions for annotations as for the class diagrams in Figure 4. While some interactions are known in `P3`, at this stage of the design process, it is known only that there will be a set of `prepActions` and `drivingActions`, and *Abs* partiality is used to express this. In addition, there is a possibility of there being a monitoring function for security. *May* partiality is used to indicate that this portion may be omitted in a refinement, and *Var* partiality is used to indicate that it is not yet clear which object will perform the `Monitor` role. Finally, `P3` uses the *OW* partiality since we expect more objects to be added in a refinement.

In the model `P4`, on the right of Figure 4, the modeler has refined `prepActions` to a particular set of actions. In addition, she has assigned the `Monitor` role to `Car` itself (i.e., `Monitor`=`Car`) and retained only the first `Notify` message. Finally, she has decided that the model will not be extended further and it is set as "complete".

**Discussion.** While class diagrams and sequence diagrams are different syntactically and in their domains of applicability (i.e., structure vs. behaviour), the *MAVO* partiality provides the same capabilities for expressing and refining uncertainty in both languages. In particular, it adds the ability to treat atoms as removable (*May*), as sets (*Abs*), and as variables (*Var*), and to treat the entire model as extensible (*OW*). Furthermore, we were able to use the same concrete syntactic conventions in both languages — this is significant because modeler knowledge can be reused across languages. Note that while our examples come from UML, *MAVO* annotations are not UML-specific and can be applied to any metamodel-based language, regardless of the degree of formality of the language. The reason is that the semantics of partiality is expressed in terms of sets of models (i.e., possible concretizations) and does not depend on the native semantics of the underlying modeling language.

Most of the expressions of partiality in these examples required the added partiality mechanisms. The exceptions, which could have been expressed natively, are: (1) that types of attributes are unknown (as with the `sizeRelated` attributes), in class diagram `P1`, and (2) the choice between the `Monitor` and its `Notify` messages (using an *Alt* operator, e.g., based on the STAIRS semantics [6]), in sequence diagram `P3`. This suggests that language-independent partiality types can add significant value to modeling languages.

## 4 Formalizing Partiality

In this section, we define an approach for formalizing the semantics of a partial model and apply it to *MAVO* partiality. Specifically, given a partial model $P$, we specify the set of concretizations $[P]$ using First Order Logic (FOL). Our

approach has the following benefits: (1) it provides a general methodology for defining the semantics of a partial modeling language; (2) it provides a mechanism for defining refinement, even between partial models of different types; (3) it provides the basis for tool support for reasoning with partial models using off-the-shelf tools; and (4) it provides a sound way to compose partial modeling languages.

We begin by noting that a metamodel represents a set of models and can be expressed as an FOL theory.

**Definition 2 (Metamodel)** *A* metamodel *is a First Order Logic (FOL) theory* $T = \langle \Sigma, \Phi \rangle$*, where* $\Sigma$ *is th e* signature *and* $\Phi$ *is a set of sentences representing the well-formedness constraints.* $\Sigma = \langle \sigma, \rho \rangle$ *consists of the set of sorts* $\sigma$ *defining the element types and the set* $\rho$ *of predicates defining the types of relations between elements. The models that conform to* $T$ *are the finite FO* $\Sigma$*-structures that satisfy* $\Phi$ *according to the usual FO satisfaction relation. We denote the set of models with metamodel* $T$ *by* $Mod(T)$*.*

The class diagram metamodel in Figure 1 fits this definition if we interpret boxes as sorts and edges as predicates.

Like a metamodel, a partial model also represents a set of models and thus can also be expressed as an FOL theory. Specifically, for a partial model $P$, we construct a theory $FO(P)$ s.t. $Mod(FO(P)) = [P]$. Furthermore, since $P$ represents a subset of $T$ models, we require that $Mod(FO(P)) \subseteq Mod(T)$. We guarantee this by defining $FO(P)$ to be an extension of $T$ that adds constraints.

Let $M = bs(P)$ be the base model of a partial model $P$ and let $P_M$ be the *ground* partial model which has $M$ as its base model and its sole concretization – i.e., $bs(P_M) = M$ and $[P_M] = \{M\}$. We first describe the construction of $FO(P_M)$ and then define $FO(P)$ in terms of $FO(P_M)$. To construct $FO(P_M)$, we extend $T$ to include a unary predicate for each element in $M$ and a binary predicate for each relation instance between elements in $M$. Then, we add constraints to ensure that the only first order structure that satisfies the resulting theory is $M$ itself.

We illustrate the above construction using the class diagram `M` in Figure 2(a). Interpreting it as a partial model $P_{\texttt{M}}$, we have:

$$FO(P_{\texttt{M}}) = \langle \langle \sigma_{\texttt{CD}}, \rho_{\texttt{CD}} \cup \rho_{\texttt{M}} \rangle, \Phi_{\texttt{CD}} \cup \Phi_{\texttt{M}} \rangle$$

(see Definition 2), where $\sigma_{\texttt{CD}}$, $\rho_{\texttt{CD}}$ and $\Phi_{\texttt{CD}}$ are the sorts, predicates and well-formedness constraints, respectively, for class diagrams, as described in Figure 1. $\rho_{\texttt{M}}$ and $\Phi_{\texttt{M}}$ are model `M`-specific predicates and constraints, defined in Figure 5. Since $FO(P_{\texttt{M}})$ extends `CD`, the FO structures that satisfy $FO(P_{\texttt{M}})$ are the class diagrams that satisfy the constraint set $\Phi_{\texttt{M}}$ in Figure 5. Assume $N$ is such a class diagram. The constraint *Complete* ensures that $N$ contains no more elements or relation instances than `M`. Now consider the class `Car` in `M`. *Exists* says that $N$ contains at least one class called `Car`, *Unique* – that it contains no more than one class called `Car`, and *Distinct* – that the class called `Car` is different from the class called `Vehicle`. Similar sentences are given for class `Vehicle` and `super`

instance `CsuperV`. The constraint *Type* ensures that `CsuperV` has correctly typed endpoints. These constraints ensure that $FO(P_{\mathtt{M}})$ has exactly one concretization and thus $N = \mathtt{M}$.

Relaxing the constraints $\Phi_{\mathtt{M}}$ allows additional concretizations and represents a type of uncertainty. For example, if we are uncertain about whether `M` is complete, we can express this by removing the *Complete* clause from $\Phi_{\mathtt{M}}$ and thereby allow concretizations to be class diagrams that extend `M`. Note that keeping or removing the *Complete* clause corresponds exactly to the semantics of the annotations COMP and INC in *OW* partiality, as defined in Section 2. Similarly, expressing each of *May*, *Abs* and *Var* partiality corresponds to relaxing $\Phi_{\mathtt{M}}$ by removing *Exists*, *Unique* and *Distinct* clauses, respectively, for particular atoms. For example, removing the *Exists* clause $\exists x : \mathtt{Class} \cdot \mathtt{Car}(x)$ is equivalent to marking the class `Car` with M(i.e., `Car` may or may not exist), while removing the *Distinct* clause $\forall x : \mathtt{Class} \cdot \mathtt{Car}(x) \Rightarrow \neg \mathtt{Vehicle}(x)$ is equivalent to marking the class *Car* with V(i.e., `Car` is a variable that can merge with `Vehicle`).

Figure 6 generalizes the construction in Figure 5 to an arbitrary ground theory $FO(P_M)$. $\rho_M$ contains a unary predicate $\mathsf{E}$ for each element $E$ in $M$ and a binary predicate $\mathsf{R_{ij}}$ for instance $R(E_i, E_j)$ of relation $R$ in $M$. Each of the atom-specific clauses is indexed by an atom in model $M$ to which it applies (e.g., $Exists_{\mathsf{E}}$ applies to element $E$). For simplicity, we do not show the element types of the quantified variables.

We now formalize our earlier observation about relaxing $\Phi_M$:

**Observation 3** *Given a ground partial model $P_M$ with $FO(P_M) = \langle\langle \sigma_T, \rho_T \cup \rho_M\rangle, \Phi_T \cup \Phi_M\rangle$ constructed as in Figure 5, any relaxation of the constraint $\Phi_M$ introduces additional concretizations into $Mod(FO(P_M))$ and represents a case of uncertainty about* `M`.

This observation gives us a general and sound approach for defining the semantics of a partial model. For partial model $P$ with base model $M$, we define $FO(P)$ as $FO(P_M)$ with $\Phi_M$ replaced by a sentence $\Phi_P$, where $\Phi_M \Rightarrow \Phi_P$.

**Application to *MAVO*.** Table 1 applies the general construction in Figure 6 to the individual *MAVO* partiality annotations by identifying which clauses to remove from $\Phi_M$ for each annotation. For example, the annotation (S)$\mathsf{E}$ corresponds to removing the clause $Unique_{\mathsf{E}}$. *Note that nothing in the construction of $FO(P_M)$ or in Table 1 is dependent on any specific features of the metamodel and hence the semantics of MAVO is language-independent.*

The semantics for combined annotations is obtained by removing the clauses for each annotation – e.g., the annotation (SV)$\mathsf{E}$ removes the clause $Unique_{\mathsf{E}}$ and the clauses $Distinct_{\mathsf{EE'}}$ and $Distinct_{\mathsf{EE'}}$ for all elements $\mathsf{E'}$.

The *MAVO* partiality types represent special cases of relaxing the ground sentence $\Phi_M$ by removing clauses but, as noted in Observation 3, any sentence weaker than $\Phi_M$ could be used to express partiality of $M$as well. This suggests a natural way to enrich *MAVO* to express more complex types: augment the basic annotations with sentences that express additional constraints. We illustrate this using examples based on model `P1` in Figure 3. The statement "if `TimeMachine` is

$\rho_{\mathtt{M}}$ contains the unary predicates $\mathtt{Car}(\mathtt{Class})$, $\mathtt{Vehicle}(\mathtt{Class})$ and the binary predicate $\mathtt{CsuperV}(\mathtt{Class}, \mathtt{Class})$.

$\Phi_{\mathtt{M}}$ contains the following sentences:

*(Complete)* $(\forall x : \mathtt{Class} \cdot \mathtt{Car}(x) \vee \mathtt{Vehicle}(x)) \wedge$
$(\forall x, y : \mathtt{Class} \cdot \mathtt{super}(x, y) \Rightarrow \mathtt{CsuperV}(x, y)) \wedge \neg\exists x \cdot \mathtt{Datatype}(x) \wedge \ldots$

`Car`:

*(Exists)* $\exists x : \mathtt{Class} \cdot \mathtt{Car}(x)$

*(Unique)* $\forall x, x' : \mathtt{Class} \cdot \mathtt{Car}(x) \wedge \mathtt{Car}(x') \Rightarrow x = x'$

*(Distinct)* $\forall x : \mathtt{Class} \cdot \mathtt{Car}(x) \Rightarrow \neg\mathtt{Vehicle}(x)$

similarly for `Vehicle`

`CsuperV`:

*(Type)* $\forall x, y : \mathtt{Class} \cdot \mathtt{CsuperV}(x, y) \Rightarrow \mathtt{Car}(x) \wedge \mathtt{Vehicle}(y)$

*(Exists)* $\forall x, y : \mathtt{Class} \cdot \mathtt{Car}(x) \wedge \mathtt{Vehicle}(y) \Rightarrow \mathtt{CsuperV}(x, y)$

*(Unique)* $\forall x, y, x', y' : \mathtt{Class} \cdot \mathtt{CsuperV}(x, y) \wedge \mathtt{CsuperV}(x', y') \Rightarrow x = x' \wedge y' = y$

**Fig. 5.** Example constraints for class diagram `M` in Figure 2(a).

**Table 1.** Semantics of *MAVO* Partiality Annotations.

| MAVO annotation | Clause(s) to remove from $\Phi_M$ |
|---|---|
| INC | *Complete* |
| (M)E | $Exists_{\mathsf{E}}$ |
| (S)E | $Unique_{\mathsf{E}}$ |
| (V)E | $Distinct_{\mathsf{EE}'}$ and $Distinct_{\mathsf{E}'\mathsf{E}}$ for all $\mathsf{E}'$, $\mathsf{E}' \neq \mathsf{E}$ |
| (M)$\mathsf{R}_{\mathsf{ij}}$ | $Exists_{\mathsf{R}_{\mathsf{ij}}}$ |
| (S)$\mathsf{R}_{\mathsf{ij}}$ | $Unique_{\mathsf{R}_{\mathsf{ij}}}$ |
| (V)$\mathsf{R}_{\mathsf{ij}}$ | $Distinct_{\mathsf{R}_{\mathsf{ij}}\mathsf{R}'_{\mathsf{kl}}}$ and $Distinct_{\mathsf{R}'_{\mathsf{kl}}\mathsf{R}_{\mathsf{ij}}}$ for all $\mathsf{R}'_{\mathsf{kl}}$, $i \neq k, j \neq l$ |

a `Vehicle`, then `Hovercraft` must be one as well" imposes a further constraint on the concretizations of `P1`. Using $FO(\mathtt{P1})$, we can express this in terms of the *Exists* constraints for individual atoms: $Exists_{\mathtt{TimeMachine}} \Rightarrow Exists_{\mathtt{Hovercraft}} \wedge Exists_{\mathtt{HsuperV}}$. Thus, propositional combinations of *Exists* sentences allow richer forms of the *May* partiality to be expressed.

Richer forms of the *Abs* partiality can be expressed by putting additional constraints on "S"-annotated atoms to further constrain the kinds of sets to which they can be concretized. For example, we can express the multiplicity constraint that there can be at most two `sizeRelated` attributes by replacing the constraint $Unique_{\mathtt{sizeRelated}}$ with the following weaker one:

$$\forall x, x', x'' \cdot \mathtt{sizeRelated}(x) \wedge \mathtt{sizeRelated}(x') \wedge \mathtt{sizeRelated}(x'') \Rightarrow (x = x' \vee x = x'' \vee x' = x'')$$

Of course, this can be easily expressed in a language with sets and counting, like OCL. Similar enrichments of the *Var* and the *OW* partialities can be produced by an appropriate relaxation of the *Distinct* and *Complete* constraints, respectively. These enrichments of *MAVO* remain language-independent because they do not make reference to the metamodel-specific features.

**Refinement of *MAVO* partiality.** We have defined partial model semantics in terms of relaxations to $\Phi_M$. Below, we define refinement in terms of these as well. Specifically, assume we have relaxations $\Phi_{P'}$ and $\Phi_P$ for partial models $P'$

**Input**: model $M$ of type $T = \langle\langle \sigma_T, \rho_T\rangle, \Phi_T\rangle$
**Output**: $FO(P_M)$
$FO(P_M) = \langle\langle \sigma_T, \rho_T \cup \rho_M\rangle, \Phi_T \cup \Phi_M\rangle$
$\rho_M = \rho^e \cup \rho^r$, where $\rho^e = \{\mathsf{E}(\cdot) | E$ is an element of $M\}$
and $\rho^r = \{\mathsf{R_{ij}}(\cdot,\cdot) | \mathsf{R_{ij}}$ is an instance of relation $R \in \rho_T$ in $M\}$
$\Phi_M$ contains the following sentences:

*(Complete)* $(\forall x \cdot \bigvee_{\mathsf{E}\in\rho^e} \mathsf{E}(x)) \wedge (\bigwedge_{R\in\rho_T} \forall x, y \cdot R(x,y) \Rightarrow \bigvee_{\mathsf{R_{ij}}\in\rho^r} \mathsf{R_{ij}}(x,y))$

for each element $\mathsf{E}$ in $M$:

*(Exists*$_{\mathsf{E}}$*)* $\exists x \cdot \mathsf{E}(x)$

*(Unique*$_{\mathsf{E}}$*)* $\forall x, y \cdot \mathsf{E}(x) \wedge \mathsf{E}(y) \Rightarrow x = y$

$\bigwedge_{\mathsf{E'}\in\rho^e, \mathsf{E'}\neq\mathsf{E}}$ *(Distinct*$_{\mathsf{EE'}}$*)* $\forall x \cdot \mathsf{E}(x) \Rightarrow \neg\mathsf{E'}(x)$

for each relation instance $\mathsf{R_{ij}}$ in $M$:

*(Type*$_{\mathsf{R_{ij}}}$*)* $\forall x, y \cdot \mathsf{R_{ij}}(x,y) \Rightarrow \mathsf{E_i}(x) \wedge \mathsf{E_j}(y)$

*(Exists*$_{\mathsf{R_{ij}}}$*)* $\forall x, y \cdot \mathsf{E_i}(x) \wedge \mathsf{E_j}(y) \Rightarrow \mathsf{R_{ij}}(x,y)$

*(Unique*$_{\mathsf{R_{ij}}}$*)* $\forall x, y, x', y' \cdot \mathsf{R_{ij}}(x,y) \wedge \mathsf{R_{ij}}(x',y') \Rightarrow x = x' \wedge y = y'$

$\bigwedge_{\mathsf{R'_{kl}}\in\rho^r, i\neq k, j\neq l}$ *(Distinct*$_{\mathsf{R_{ij}R'_{kl}}}$*)* $\forall x, y \cdot \mathsf{R_{ij}}(x,y) \Rightarrow \neg\mathsf{R'_{kl}}(x,y)$

**Fig. 6.** Construction of $FO(P_M)$.

and $P$, respectively. In the special case that their base models are equivalent, we have $P'$ refines $P$ iff $[P'] \subseteq [P]$ and this holds iff $\Phi_{P'} \Rightarrow \Phi_P$. However, when the base models are different, the sentences are incomparable because they are based on different signatures. The classic solution to this kind of problem (e.g., in algebraic specification) is to first translate them into the same signature and then check whether the implication holds in this common language (e.g., see [5]). In our case, we can use a refinement mapping $\mathsf{R}$ between the base models, such as the one in Figures 3 and 4, to define a function that translates $\Phi_P$ to a semantically equivalent sentence $\mathsf{R}(\Phi_P)$ over the signature $\Sigma_{P'}$. Then, $P'$ refines $P$ iff $\Phi_{P'} \Rightarrow \mathsf{R}(\Phi_P)$. We omit the details of this construction due to space limitations; however, interested readers can look at the Alloy model for Experiment 6 in Section 5 for an example of this construction.

## 5 Tool Support and Preliminary Evaluation

In order to show the feasibility of using the formalization in Section 4 for automated reasoning, we developed an Alloy [8] implementation for *MAVO* partiality. We used a Python script to generate the Alloy encoding of the clauses (as defined in Figure 6) for the models `P1` and `P2`, shown in Figure 3. The Alloy models are available online at `http://www.cs.toronto.edu/se-research/fase12.htm`. We then used this encoding for *property checking*. More specifically, we attempted to address questions such as "does any concretization of $P$ have the property $Q$?" and "do all concretizations of $P$ have the property $Q$?", where $Q$ is expressed in FOL. The answer to the former is affirmative iff $\Phi_P \wedge Q$ is satisfiable, and to the latter iff $\Phi_P \wedge \neg Q$ is not satisfiable. We also used the tooling to check correctness of *refinement*, cast as a special case of property checking. As discussed in Section 4, $P'$ refines $P$ iff $\Phi_{P'} \Rightarrow \mathsf{R}(\Phi_P)$ where $\mathsf{R}$ translates $\Phi_P$ according to

| Exp. # | Question | Answer | Scope | Time (ms) |
|---|---|---|---|---|
| 1 | Does the ground case for P1 have a single instance? | **Yes** | 7 | 453 |
| 2 | Does the ground case for P2 have a single instance? | **Yes** | 6 | 366 |
| 3 | Is P1 extended with Q1 consistent? | **Yes** | 4 | 63 |
| 4 | Is P1 extended with Q1 and Q2 consistent? | **No** | 20 | 64677 |
| 5a | Is P1 extended with Q1 and Q3 consistent? | **Yes** | 4 | 64 |
| 5b | Is P1 extended with Q1 and ¬Q3 consistent? | **Yes** | 5 | 151 |
| 6 | Is P2 a correct refinement of P1? | **Yes** | 10 | 9158 |

**Table 2.** Results of experiments using Alloy.

the refinement mapping. Thus, the refinement is correct iff $\Phi_{P'} \wedge \neg \mathsf{R}(\Phi_P)$ is not satisfiable.

Table 2 lists the experiments we performed, using the following properties:

Q1 : Vehicle has at most two direct subclasses.
Q2 : Every class, except C1 is a direct subclass of C1.
Q3 : There is no multiple inheritance.

Experiments (1) and (2) verify our assumption that the encoding described in Figure 6 admits only a single concretization. Although any pure MAVO model is consistent by construction, Experiments (3) and (4) illustrate that this is not necessarily the case when additional constraints are added. First, P1 is extended with Q1 and shown to be consistent. However, extending P1 with both Q1 and Q2 leads to an inconsistency. This happens because Q2 forces (a) C1 to be merged with Vehicle, and (b) TimeMachine to be its subclass, raising its number of direct subclasses to 3. This contradicts Q1, and therefore, $\mathtt{P1} \wedge \mathtt{Q1} \wedge \mathtt{Q2}$ is inconsistent. Note that Experiment (4) takes longer than the others because showing inconsistency requires that the SAT solver enumerate all possible models within the scope bounds. In Experiment (5), we asked whether the version of P1 extended with Q1 satisfies property Q3 and found that this is the case in some (Experiment 5a) but not all (Experiment 5b) concretizations. Finally, in Experiment (6) we verified the refinement described in Figure 3, using the mapping in the figure to construct a translation of $\Phi_{\mathtt{P1}}$, as discussed in Section 4.

Our experiments have validated the feasibility of using our formalization for reasoning tasks. In our earlier work [4], we have done a scalability study for property checking using a SAT solver for *May* partiality (with propositional extensions). The study showed that, compared to explicitly handling the set of concretizations, our approach offers significant speedups for large sets of concretizations. We intend to do similar scalability studies for all *MAVO* partialities in the future.

## 6 Related Work

In this section, we briefly discuss other work related to the types of partiality introduced in this paper.

A number of partial *behavioural* modeling formalisms have been studied in the context of abstraction (e.g., for verification purposes) or for capturing early design models [12]. The goal of the former is to represent property-preserving

abstractions of underlying concrete models, to facilitate model-checking. For example, Modal Transition Systems (MTSs) [9] allow introduction of uncertainty about transitions on a given event, whereas Disjunctive Modal Transition Systems (DMTSs) [10] add a constraint that at least one of the possible transitions must be taken in the refinement. Concretizations of these models are Labelled Transition Systems (LTSs). MTSs and DMTSs are results of a limited application of `May` partiality. Yet, the MTS and DMTS refinement mechanism allows resulting LTS models to have an arbitrary number of states which is different from the treatment provided in this paper, where we concentrated only on "structural" partiality and thus state duplication was not applicable.

In another direction, Herrmann [7] studied the value of being able to express *vagueness* within design models. His modeling language SeeMe has notational mechanisms similar to *OW* and *May* partiality; however, there is no formal foundation for these mechanisms.

Since models are like databases capturing facts about the models' domain, work on representing incomplete databases is relevant. *Var* partiality is traditionally expressed in databases by using null values to represent missing information. In fact, our ideas in this area are inspired by the work on data exchange between databases (e.g., [2]) which explicitly uses the terminology of "variables" for nulls and "constants" for known values. An approach to the *OW* partiality is the use of the Local Closed World Assumption [1] to formally express the places where a database is complete.

Finally, our heavy reliance on the use of FOL as the means to formalize metamodels and partial models gives our work a strong algebraic specification flavor and we benefit from this connection. In particular, partial model refinement is a kind of specification refinement [11]. Although our application is different – dealing with syntactical uncertainty in models rather than program semantics – we hope to exploit this connection further in the future.

## 7 Conclusion and Future Work

The key observation of our work is that many types of partiality information and their corresponding types of refinement are actually language-independent and thus can be added to any modeling language in a uniform way. In this paper, we defined a formal approach for doing so in any metamodel-based language by using model annotations with well-defined semantics. This allows us to incorporate partiality across different languages in a consistent and complete way, as well as to develop language-independent tools for expressing, reasoning with, and refining partiality within a model. We then used this approach to define four types of partiality, each addressing a distinctly different pragmatic situation in which uncertainty needs to be expressed within a model. We combined all four and illustrated their language independence by showing how they can be applied to class diagrams and to sequence diagrams. Finally, we demonstrated the feasibility of tool support for our partiality extensions by describing an Alloy-based implementation of our formalism and various reasoning tasks using it.

The investigation in this paper suggests several interesting directions for further research. First, since adding support for partiality lifts modeling lan-

guages to partial modeling languages, it is natural to consider whether a similar approach could be used to lift *model transformations* to *partial model transformations*. This would allow partiality to propagate through a transformation chain during model-driven development and provide a principled way of applying transformations to models earlier in the development process, when they are incomplete or partial in other ways. Second, it would be natural to want to interleave the partiality-reducing refinements we discussed in this paper with other, language-specific, refinement mechanisms during a development process. We need to investigate how these two types of refinements interact and how they can be soundly combined. Third, since modelers often have uncertainty about entire model fragments, it is natural to ask how to extend *MAVO* annotation to this case. Applying *May* partiality to express a design alternative is straightforward – a fragment with annotation M may or may not be present; however, the use of the other *MAVO* types is less obvious and deserves further exploration. Finally, although we have suggested scenarios in which particular *MAVO* annotations would be useful, we recognize that the methodological principles for applying (and refining) partial models require a more thorough treatment. We are currently developing such a methodology.

## References


1. A. Cortés-Calabuig, M. Denecker, and O. Arieli. "On the Local Closed-World Assumption of Data-Sources". *J. Logic Programming*, 2005.
2. R. Fagin, P. Kolaitis, R. Miller, and L. Popa. "Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 336(1):89–124, May 2005.
3. M. Famelis, S. Ben-David, M. Chechik, and R. Salay. "Partial Models: A Position Paper". In *Proc. of MoDeVVa'11*, pages 1–6, 2011.
4. M. Famelis, R. Salay, and M. Chechik. "Partial Models: Towards Modeling and Reasoning with Uncertainty", Submitted, 2011.
5. J.A. Goguen and R.M. Burstall. "Institutions: Abstract model theory for specification and programming". *Journal of the ACM (JACM)*, 39(1):95–146, 1992.
6. O. Haugen, K.E. Husa, R.K. Runde, and K. Stolen. "STAIRS: Towards Formal Design with Sequence Diagrams". *SoSyM*, 4(4):355–357, October 2005.
7. T. Herrmann. *Hndbk of Research on Socio-Technical Design and Social Networking Systems*, chapter "Systems Design with the Socio-Technical Walkthrough", pages 336–351. 2009.
8. D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.
9. K. G. Larsen and B. Thomsen. "A Modal Process Logic". In *Proc. of LICS'88*, pages 203–210, 1988.
10. P. Larsen. "The Expressive Power of Implicit Specifications". In *Proc. of ICALP'91*, volume 510 of *LNCS*, pages 204–216, 1991.
11. D. Sannella and A. Tarlecki. "Essential Concepts of Algebraic Specification and Program Development". *Formal Aspects of Computing*, 9(3):229–269, 1997.
12. S. Uchitel and M. Chechik. "Merging Partial Behavioural Models". In *Proc. of FSE'04*, pages 43–52, 2004.
13. O. Wei, A. Gurfinkel, and M. Chechik. "On the Consistency, Expressiveness, and Precision of Partial Modeling Formalisms". *J. Inf. Comput.*, 209(1):20–47, 2011.