Constructive Techniques for Meta- and Model-Level Reasoning

Ethan K. Jackson and Janos Sztipanovits

Institute for Software Integrated Systems,
Vanderbilt University, Nashville, TN 37235, USA
ejackson@isis.vanderbilt.edu
janos.sztipanovits@vanderbilt.edu

Abstract. The structural semantics of UML-based metamodeling were recently explored\[^1\], providing a characterization of the models adhering to a metamodel. In particular, metamodels can be converted to a set of constraints expressed in a decidable subset of first-order logic, an extended Horn logic. We augment the constructive techniques found in logic programming, which are also based on an extended Horn logic, to produce constructive techniques for reasoning about models and metamodels. These methods have a number of practical applications: At the meta-level, it can be decided if a (composite) metamodel characterizes a non-empty set of models, and a member can be automatically constructed. At the model-level, it can be decided if a submodel has an embedding in a well-formed model, and the larger model can be constructed. This amounts to automatic model construction from an incomplete model. We describe the concrete algorithms for constructively solving these problems, and provide concrete examples.

1 Preliminaries - Metamodels, Domains, and Logic

This paper describes constructive techniques, similar to those found in logic programming, for reasoning about domain-specific modeling languages (DSMLs) defined with metamodels. Before we proceed, we must describe how a metamodel can be viewed as a formal object that characterizes the well-formed models adhering to that metamodel. We will refer to the models that adhere to metamodel $X$ as the models of metamodel $X$. In order to build some intuition for this view, consider the simple $DIGRAPH$ metamodel of Figure 1. The models of $DIGRAPH$ consist of instances of the $Vertex$ and $Edge$ classes such that $Edge$ instances “connect” $Vertex$ instances. In other words, $DIGRAPH$ characterizes

![Fig. 1. DIGRAPH: A simple metamodel for labeled directed graphs](image-url)
a class of labeled directed graphs. Thus, a model might be formalized as a pair 
\[ G = (V \subseteq \Sigma, E \subseteq V \times V) \], where \( \Sigma \) is an alphabet of vertex labels. If \( \Sigma \) is fixed, then the set \( G \) of all models of DIGRAPH is: 
\[ G = \{ (V, E) | V \subseteq \Sigma, E \subseteq V^2 \} \].
This is the classic description of labeled digraphs, and at first glance it might 
appear possible to extend this description to characterize the models of arbitrary metamodels. Unfortunately, UML-like metamodels\[^2\][^3\] contain a number of constructs that deny a simple extension of graph-based descriptions. The UNSAT metamodel of Figure 2 illustrates some of these constructs. First, classes

![Class Diagram](image)

**Fig. 2.** UNSAT: A complex metamodel with no finite non-trivial models

may have non-trivial internal structure. For example, classes of UNSAT have typed member fields (called attributes). An instance of ClassA has a boolean field named bAttribute. Classes also inherit this structure, e.g. an instance of ClassC has two attributes, bAttribute and zAttribute, via inheritance. Instances may contain other instances with constraints on the type and number of contained instances. An instance of ClassA must contain between 1 and 3 instances of ClassB. Second, internal instance structure can be “projected” onto the outside of an instance as ports. The containment relation from ClassA to RootClass

![Model Diagram](image)

**Fig. 3.** Model that (partially) adheres to the UNSAT metamodel