Infinitary Model Finding in Support of Scientific Discovery

Andrew Shilliday
Rensselaer AI & Reasoning Lab
Rennselaer Polytechnic Institute, Troy, NY
http://www.cs.rpi.edu/~shilla
shilla@cs.rpi.edu

Introduction

Model finding technology is highly useful in nearly every stage of logico-mathematical discovery. In the initial formalization of domain knowledge, model finders may be employed to ensure that formulae intended to capture some real domain knowledge are not contradictory; in conjecture generation and verification, the presence or absence of certain models can indicate whether or not the conjecture is actually a logical consequence of its premises. The task of a model finder is straightforward: given a set of formulae in a logical system, the finder seeks a single *interpretation* which satisfies every formula. In other words, for every function symbol, relation symbol, and constant symbol present in the logical system, the desired result is a mapping of those otherwise semantically meaningless symbols to real functions, real relations, and real constants ranging over one or more non-empty sets (or domains).

For example, consider the following two axioms in a standard first-order logic:

$$\forall_x \exists_y : x < y \tag{1}$$

$$\forall_{x,y,z} : [(x < y) \wedge (y < z)] \rightarrow (x < z) \tag{2}$$

Here, (1) and (2) are satisfiable — there exists an interpretation on which both are satisfied according to the semantics of the logic. If (as the language may already suggest) an interpretation mapped the symbol $<$ to the `less-than` relation over, say, the natural numbers, the above sentences evaluate to true according to that interpretation (i.e., (1) every natural number is less than another natural number, and (2) the less-than relation is transitive). Of course, that obvious interpretation is not the only one which *models* the two formulae: a domain consisting of every person currently residing in Troy, NY with the $<$ relation symbol mapped to the "is same person" relation (where $x < y$ only if x and y are the same person) would also satisfy the given formulae.

Model finders can be employed to evaluate the veracity of a conjecture, particularly in an attempt to *falsify* the claim. If a model can be found which satisfies the premises, but in which the conjecture is false, then the conjecture is not logically entailed by the premises. Such an interpretation is called a *counter-model*. Suppose we conjectured that the following sentence follows from (1) and (2) above:

$$\forall_x : (x < y) \rightarrow \neg(y < x) \tag{3}$$

To falsify this conjecture, we seek a counter-model. Consider the first interpretation for (1) and (2): if $<$ is the `less-than` relation, then (3) remains true — this interpretation is not a counter-model for the conjecture. The second interpretation, however, is a counter-model. If x is the same person as y, then clearly y is the same person as x; (3) is false in this interpretation and therefore we know that the sentence is not logically entailed by (1) and (2).

If the model finder was unable to produce a counter-model, then, depending on the way in which it failed, the lack of a model may support that the conjecture is entailed. Formally, if no interpretation exists in which the premises are true and the conclusion false, the conclusion *is* a logical consequence of the premises. That being said, the non-existence of a model and the inability of a system to produce one do not necessarily go hand in hand.

Finitary Model Finding

Several systems have been developed for producing models from a set of premises. By and large, the current successful systems are based on one of two strategies, one developed by McCune (1994), the other by Zhang (1995a). The latter resulted in the SEM system (Zhang & Zhang 1995b; 1996) and employed backtracking search and powerful constraint propagation methods to search for interpretations of the given sentences. The system also used heuristics to reduce the number of isomorphic models it had to search through (Claessen & Sörensson 2003). The former approach yielded the MACE model-finder (McCune 1994), which first converted the first-order sentences into an propositional logic clause set assuming a fixed and finite number of domain elements. Returning to our example in the previous section, if we temporarily limit our search to only models containing two elements (call them a and b), we can construct a set of propositional sentences new propositional variables assigned to the each possible instantiation of the existing relation symbols. In other words, if the only objects are a and b, the following new propositional variables are introduced: $\{<_{aa}, <_{ab}, <_{ba}, <_{bb}\}$ where $<_{xy}$ is understood to be the propositional equivalent of the first-order formula $x < y$. Instead of the formula $\forall_x : \neg(x < x)$, for example, we would have the clause $(\neg <_{aa} \wedge \neg <_{bb})$. The premises and negated conclusion can be rewritten into propositional clauses accordingly and the resulting problem is then at-

tacked by a SAT solver for propositional logic. The MACE system repeats this process for incrementally larger domain sizes until it finds a satisfying model.Building on the MACE tradition, the Paradox model-finder (Claessen & Sörensson 2003) implements a number of additional heuristics, including sort-inference and static symmetry reduction, to gain a substantial boost in efficiency and has stood up well against other such systems in the annual CADE ATP System Competition (CASC) for automated theorem provers and model finders (Sutcliffe & Suttner 2006).

To take a more robust example from the TPTP (Thousands of Problems for Theorem Provers) Problem Library (Sutcliffe & Suttner 1998) (`GRA013+1`), suppose we are concerned with the domain of Graph Theory and conjecture that there is no 2-colored complete graph of size 5 that does not contain a subgraph of size three in which all of the edges have the same color.[1] The domain and problem is encoded in a first-order logical system as follows.

$$\mathtt{e}(n_1, n_2) \wedge \mathtt{e}(n_2, n_3) \wedge \mathtt{e}(n_3, n_4) \wedge \mathtt{e}(n_4, n_5) \quad (4)$$

$$\forall_{x,y,z} : (\mathtt{e}(x, y) \wedge \mathtt{e}(y, z)) \rightarrow \mathtt{e}(x, z) \quad (5)$$

$$\forall_{x,y} : \mathtt{e}(x, y) \rightarrow (\mathtt{r}(x, y) \vee \mathtt{g}(x, y)) \quad (6)$$

$$\neg\exists_{x,y,z} : \mathtt{r}(x, y) \wedge \mathtt{r}(y, z) \wedge \mathtt{r}(x, z) \quad (7)$$

$$\neg\exists_{x,y,z} : \mathtt{g}(x, y) \wedge \mathtt{g}(y, z) \wedge \mathtt{g}(x, z) \quad (8)$$

Formulas (4) and (5) set that there are five vertices and that an edge connects each vertex to the remaining four. (6) fixes that every edge must either be green or red, and (7) and (8) establish that there does not exists any complete subgraph of size three whose edges are all the same color. An interpretation will specify exactly which edges are red and green. When fed to Paradox, a model of size 5 is constructed almost immediately. The five members of the interpretation's domain correspond to the five vertices of the graph, with the edges and colorings indicated as shown in Figure 1.

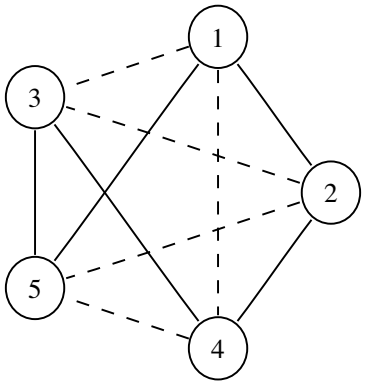

Figure 1: The output from Paradox, represented as a visual diagram.

[1] Contrary to the standard definition, a 2-coloring, here, is a coloring of the *edges* using only two colors.

Infinitary Model Finding

While such systems are particularly effective on examples such as the one given, they all share a common problem: these model finders are only capable of finding *finite* setback models. While these systems could, theoretically, incrementally search arbitrarily large domains, they will only ever search finite domains. It is, however, exceedingly simple to produce a set of satisfiable formulae for which these model-finder are completely impotent; e.g.,

$$\forall_x : s(x) \neq a \quad (9)$$

$$\forall_{x,y} : (s(x) = s(y)) \rightarrow (x = y) \quad (10)$$

Any interpretation which satisfies (9) and (10) must contain mutually distinct elements for $\{a, s(a), s(s(a)), \dots\}$; (9) and (10) are satisfied only by infinite models.

One such interpretation is the natural numbers, where the function symbol, s, is interpreted as the "successor" function and the constant a is the zero. How can we be sure that this interpretation satisfies the given universally-quantified formulae? When dealing with finite models, such verification was simple: check that the formula holds for every object in the domain. When an infinite domain is involved, the task becomes less straightforward.

When the domain in question is well-founded (as the above interpretation is), we can use well-founded induction to prove that each formula would hold no mater what element (or number, in this case) we replaced the quantified variables with.

In this way, it is possible to raise the bar, so to speak, by building a new model-finder which is able to construct *some* infinitary models. Such a system requires automated inductive theorem proving but should be useful for several domains relevant to scientific discovery where reasoning over infinite domains is a matter of convenience or necessity.

References

Claessen, K., and Sörensson, N. 2003. New techniques that improve MACE-style finite model finding. In *CADE-19, Workshop W4. Model Computation — Principles, Algorithms, Applications*.

McCune, W. 1994. A Davis-Putnam program and its application to finite first-order model search: quasigroup existence problems. Technical Report ANL/MCS-TM-194, Argonne National Laboratory.

Sutcliffe, G., and Suttner, C. 1998. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning* 21(2):177–203.

Sutcliffe, G., and Suttner, C. 2006. The State of CASC. *AI Communications* 19(1):35–48.

Zhang, J., and Zhang, H. 1995a. Constraint propagation in model generation. *Principles and Practice of Constraint Programming* 398–414.

Zhang, J., and Zhang, H. 1995b. Sem: a system for enumerating models. In *IJCAI*, 298–303.

Zhang, J., and Zhang, H. 1996. Generating models by sem. In *Proceedings of International Conference on Automated Deduction*, Lecture Notes in Computer Science, 308–312.