Model Finding for Exploration

Daniel J. Dougherty(✉)

Worcester Polytechnic Institute, Worcester, USA
dd@wpi.edu

Abstract. We survey recent results in model finding, focusing on the notion of a model finding assistant to help users, even users not trained in logic, understand their software artifacts. The technical results discussed have all been previously published; the presentation here highlights two themes: (i) geometric logic and homomorphism orders as natural foundations for model finding, and (ii) an implemetation dichotomy between direct model finding and model finding with the aid of SAT- and SMT-solvers. We give generic high-level algorithms for the central problems of programming against such solvers; lower-level details are determined based on the category of homomorphisms being used.

1 Introduction

We are interested in the following situation. A user (a software developer, a protocol designer, a system administrator ...) would like to gain confidence in a certain artifact (a data structure or algorithm design, a protocol, a data center configuration ...). Our user has at hand a specification of their artifact in a logical language, but is not a trained logician. Nor do they have formal assertions to be verified. Our user may even have to work with a specification they didn't compose: many applications compile artifacts such as security policies or UML class diagrams into logic [47,48,55].

What kind of tool might help them?

One possibility is to treat their specification as a logical theory T and use *model finding* (we sometimes use the phrase *scenario finding*) to explore the possibilities inherent in their specification.

Model finding—the process of constructing finite models for first-order theories—is already a well-studied problem. Often model finding is a secondary component of another primary activity; it plays a role in *saturation-based theorem proving* [8–10,33,42,60], *SMT solving* [60], and *property-based testing,* [6,11,12,14,26,40,49,52,56,66], to name a few areas.

But the game is subtler when we want our tool to help our user *understand* their theory T. The model-finding approach we have in mind consists in generating and displaying concrete examples of the abstract specification at hand with an eye towards either reassuring the user when the scenarios match what is expected, or—more interestingly—uncovering surprising scenarios that elicit reactions of the form "whoops, I didn't mean to allow that!"

This work was partially supported by the U.S. National Science Foundation

Dedicated to Joshua Guttman, with appreciation for his insights and with gratitude for his friendship.

D. Dougherty et al. (Eds.): Guttman Festschrift, LNCS 13066, pp. 156–174, 2021.
https://doi.org/10.1007/978-3-030-91631-2_9

Indeed, we have in mind an *interaction* between the tool and the human, in the spirit of proof assistants, but for the purpose of building models rather than proofs. Thus the phrase, implicitly representing a slogan: *model finding assistant.*

The notions above of "reassuring" or "surprising" the user are, as stated, a bit squishy[1]. Can we find *principled* approaches to answering questions like the following?

1. How do we choose a collection of models to show to give a good picture of the space of *all* models?
2. Given the fact that our theory, if consistent, probably has lots of models, what counts as "reassurance" to our user?
3. If we do have a surprise to show the user, what models do the best job of showing them what went wrong?
4. If we decide to present a certain model, can we provide some tools to understand how it works: which parts of the model are necessary for it to satisfy the theory, which parts conspire to satisfy or fail to satisfy certain queries?

After the introductory material of Sects. 2 and 3, Sect. 4 offers some principles to guide our answers. The rest of the paper reports some technical progress on realizing these answers, along two conceptually different lines. The first is to build models "directly:" this is mathematically satisfying but—in the current state of the art—can have performance problems in some domains. The other broad approach is to leverage the amazing recent advances of SAT-solvers and SMT-solvers: here we typically face a tradeoff of expresssive power for efficiency. We shed light on some existing tools in this category by showing that they can be seen as instantiations of some simple abstract building blocks.

Proofs Have Been Omitted Here. Our goal is to create a sense of how the results fit together, and proofs are all available in the originally published papers.

This is a good place to point out that Joshua Guttman has embraced a spirit of "scenarios over deductions" throughout his career. The CPSA project, which he helped found and has guided for years, is an exemplar of domain-specific model-finding, and indeed one that already does quite a good job of addressing our motivating questions (especially the 1st and 3rd questions). This will become clearer as we detail more about CPSA below.

The Human Factor. An important dimension for any user-facing tool for formal methods is the human factor. Tools should provide mathematically sound help, but they are valuable only to the extent that people will use and understand them. Every one of our motivating questions above has a psychological component as well as a logical one.

Formal methods tools must therefore thread a needle between mathematical rigor and accessibility. So the truly "principled" approaches to our informal questions should reflect both mathematical and psychological considerations. Much more user-focused

[1] A technical philosophical term Joshua has been known to employ ....

formal methods research is called for. User studies [17] are one aspect; for a wider discussion see the abstract of Krishnamurthi and Nelson's recent invited talk [43] and the references therein.

In this paper we focus on the mathematical aspects of model finding, while trying to keep in mind the users who are actually using the tools.

2 Foundations

We work in a first-order signature Σ with relations and functions, and we take for granted the standard notions of model for Σ, satisfaction of a formula in a model, and model of a theory. We mostly focus on finite models.

If α is a formula and $a_1, \ldots, a_n$ are elements of a model $\mathbb{A}$ we will sometimes write $\mathbb{A} \models \alpha[a_1, \ldots, a_n]$ as shorthand to mean that $\mathbb{A} \models \alpha(x_1, \ldots, x_n)$ under the environment sending each x_i to a_i.

Definition 1 (Homomorphism). *Let $\mathbb{A}$ and $\mathbb{B}$ be Σ-models. A function $h : |\mathbb{A}| \to |\mathbb{B}|$ is a* homomorphism *if, for functions f and relations R from Σ,*

- $\mathbb{A} \models f[a_1, \ldots, a_n] = a$ *implies* $\mathbb{B} \models f[h(a_1), \ldots, h(a_n)] = h(a)$ *and*
- $\mathbb{A} \models R[a_1, \ldots, a_n]$ *implies* $\mathbb{B} \models R[h(a_1), \ldots, h(a_n)]$

Model $\mathbb{A}$ is a *submodel* of $\mathbb{B}$ if $|\mathbb{A}| \subseteq |\mathbb{B}|$ and the inclusion function is a homomorphism.

Categories of Models. We will consider various *categories* of models: a family of structures with a certain class of homomorphisms between them. For a theory T, the models of T form the objects of a category $\mathcal{T}$ whose arrows are the homomorphisms. We identify three subcategories of interest:

- the subcategory of $\mathcal{T}$ with injective homomorphisms;
- the full subcategory of $\mathcal{T}$ with finite models;
- the subcategory of $\mathcal{T}$ with finite models and injective homomorphisms.

2.1 Homomorphism Orderings

- Fix a category $\mathcal{C}$.
  - write $\mathbb{A} \precsim \mathbb{B}$ if there is a $\mathcal{C}$ map $h : \mathbb{A} \to \mathbb{B}$.
  - write $\mathbb{A} \approx \mathbb{B}$ if $\mathbb{A} \precsim \mathbb{B}$ and $\mathbb{B} \precsim \mathbb{A}$.
  - write $\mathbb{A} \precnsim \mathbb{B}$ if $\mathbb{A} \precsim \mathbb{B}$ and not $\mathbb{B} \precsim \mathbb{A}$.
- A model $\mathbb{A}$ is *a-minimal* for T if it is a minimal element in the homomorphism preorder on models of T.
  A model $\mathbb{A}$ is *i-minimal* for T if it is a minimal element in the injective-homomorphism preorder on models of T.

Well-Foundedness of Homomorphism Orderings. In a category of finite models, the ordering determined by injective homomorphisms is well-founded, by cardinality considerations. But the ordering determined by homomorphisms is not well-founded in general. For example, in the signature with one unary function symbol, define the model $\mathbb{C}_n$ consisting of a chain of n elements $a, f(a), \ldots f^{(n)}(a)$ with f mapping $f^{(n)}(a)$ to itself. Then each $\mathbb{C}_{i+1} \precnsim \mathbb{C}_i$.

But if there happen to be only finitely many $\mathbb{M}_i$ for a theory T (as for example when we uniformly bound the size of models of T), it is easy to see that we cannot have an infinite strictly descending chain of homomorphisms.

Lemma 2. *For any theory T, the injective-homomorphism preorder on models of T is well-founded.*

If T has only finitely many models up to isomorphism, the $\precnsim$ order on models of T is well-founded.

We will often add axioms to a theory to ensure that there is an upper bound on the size of its models. In such a case Lemma 2 will apply, even in the arbitrary-homomorphism situation.

Set of Support The following notion will be crucial for us, as it will supply our primary notion of completeness for model finders.

Definition 3 (Set of Support). *If $\mathcal{C}$ is a category of models and $\mathcal{M}$ is a collection of models in $\mathcal{C}$ we say that $\mathcal{M}_0$ is a* set of support *for $\mathcal{M}$ if for all $\mathbb{M} \in \mathcal{M}$, there exists $\mathbb{M}_0 \in \mathcal{M}_0$ with $\mathbb{M}_0 \precsim_{\mathcal{C}} \mathbb{M}$.*

3 Approaches to Model Finding

In its simplest form, model finding is the following problem. Given a theory T presented as input, either determine that T is unsatisfiable or construct one or more models satisfying T. Typically T is a first-order theory, and typically we ask for finite models; indeed sometimes the input includes a bound on the size of models searched for.

There are many tools that might be termed "standalone" model finders. It is traditional to see them as falling into two categories: *MACE style* and *SEM style*, after McCune's tool MACE [51] and the SEM tool of Zhang and Zhang [73]. These designations correspond to differences in the underlying techniques used to construct models, reduction to propositional logic (MACE-style) *vs.* more-or-less direct searching for a model (SEM-style). There are too many tools to catalog here, but the introductions to either of Claessen et al. [13] or Baumgartner et al. [4] will provide a good list of examples of each kind of tool and a good entry point into the literature.

We should make special mention of Kodkod [69], though. Kodkod is the backend for Alloy [41], is available as a stand-alone Java library and is used in many projects, including several of the tools discussed below: Margrave [55], Aluminum [54], Amalgam [53], and CompSAT [58].

Many model finders resist a simple MACE-vs-SEM taxonomy, especially if we recognize that SMT solving is a blend of propositional and first-order reasoning. For

example Fortress [70] is a tool that reduces problems to the theory EUF of equality with uninterpreted functions, and uses an SMT solver. The method of Reynolds et al. [60] might best be described as SEM-style over SMT. By contrast, Elghazi and Taghdiri [27] translate Alloy theories to (non-propositional) SMT-LIB in order to do search with unbounded scopes.

For our purposes here, surveying model-finding *assistants* it is more useful to distinguish tools based on whether they work directly with first-order theories or incorporate the use of SAT- or SMT-solving: this will guide our organization in the rest of the paper.

4 Three Principles for Model Finding Assistants

The case for building a model finder that can be viewed as a *model finding assistant* was made implicitly in [54] and explicitly in [63]. We offer the following pre-theoretic intuitions about such a tool.

1. It should specify and respect a notion of "fitness" for the models it returns
2. It should provide for exploration of individual models
3. Most important: it should satisfy reasonable soundness and completeness properties

4.1 Fitness

"Fitness" is intentionally less-than-specific. One would like to see a commitment to fitness *for a given purpose.* In this general discussion we don't want to argue for a notion of fitness applying to all model-finding activities.

If one wants to check a safety property of a specification, one may hope that no counterexample models will be found, and if such models exist it is probably best to show the user models that have no extraneous information: minimal models. This principle is manifest in the universal injunction to "provide a minimal bug report."

On the other hand if our user is truly exploring a system in the early stages of a design, they may be interested in a robust selection of *consistent* phenomena: lots of things that *could* happen, even if they don't always happen. If a model finder only produces minimal models it will not be of any help in detecting *underconstraint* in a specification.

The main force of this principle is simply that the output of a model finder should not be determined by accidents of the underlying SAT- or SMT-technology.

By the way, as we will see, there are purely technical benefits of having a tool produce minimal models automatically, see the discussions of *provenance*, *augmentation*, and *set of support* below, each of which is easier to implement when we start with minimal models.

We have elsewhere [21] discussed the relative pros and cons of minimality in the respective categories of arbitrary and injective homomorphisms.

4.2 Exploration of Individual Models

Provenance. Suppose $\mathbb{A}$ is a model of a theory T (think of T as comprising a background theory perhaps together with some conjecture). If a certain fact F holds in $\mathbb{A}$ the

user might very well wonder, "does that F have to be there in order that $\mathbb{A}$ satisfies T? Or is it an accident?" If F is not an accident, the user will be interested to know *why* F is there: what is it in T that makes F hold? The analogous questions can be asked about a given *element* a of the model: is a present in order to satisfy some existence requirement imposed by T, or is a superfluous?.

We call these questions of *provenance*. Several recent tools ([53,58,63,64]) afford the user the capability of asking provenance questions.

Augmentation. A model finder with augmentation will allow the user to (attempt to) add a fact F to a given scenario. If the model finder reports that F is inconsistent with the model under consideration, this may call attention to an *overconstraint* in the specification. More subtly, it might happen that a commitment to adding F means that another, surprising, ancillary fact F' must necessarily hold. Augmentation is typically a straightforward functionality to add to a model finder ([54,63,64].)

4.3 Completeness

We propose the following (ambitious) criterion for completeness: for each input theory T, every finite model of T should be reachable in principle.

A natural strategy to achieve this in a category $\mathcal{C}$ of models is the following:

1. ensure that the tool automatically produces a stream of models (perhaps on-demand by the user) comprising a **set of support** (Definition 3).
2. ensure that whenever $\mathbb{A} \precsim \mathbb{B}$ then user-directed **augmentation** of $\mathbb{A}$ can reach $\mathbb{B}$.

As will be explained below, this strategy relies on a commitment to having the tool produce *minimal* models by default.

5 Geometric Logic

Geometric logic is a variant of first-order logic that makes a rich specification language and supports a view of model finding that is congenial to the analysis goals introduced earlier. In this section we describe the sense in which model finding for a geometric theory is in a sense "syntax directed,"

Positive-Existential Formulas. A formula is *positive-existential* if it is built from atomic formulas (including $\top$ and $\bot$) using finitary $\wedge$, infinitary $\vee$ and $\exists$; with the requirement that each subformula has only finitely many free variables. (Positive existential formulas are referred to in the database community as *unions of conjunctive queries*.)

Suppose α is a positive-existential formula. Let $\mathbb{M}$ be a model and η an environment such that $\mathbb{M} \models_\eta \alpha$. Then there is a *finite* submodel $\mathbb{M}_0$ of $\mathbb{M}$ such that $\mathbb{M}_0 \models_\eta \alpha$ (in particular $\mathbb{M}_0$ encompasses the range of η). For this reason, properties defined by positive-existential formulas are sometimes called *observable* properties [1]. It is important to note that infinite disjunctions—but not conjunctions—are accommodated easily here.

It is also worth noting that if we come to learn (or come to ensure) that a positive-existential fact $\alpha[\vec{a}]$ is true in a model $\mathbb{A}$, perhaps based on the fact that $\mathbb{A}$ has not been fully constructed, then further information added to $\mathbb{A}$ cannot make $\alpha[\vec{a}]$ false. So we might also call positive-existential formulas "imperturbable".

It is a classical result that positive-existential formulas are precisely those preserved under homomorphisms; Rossman [61] has shown that this holds even if we restrict attention to finite models only.

Theorem 4. *The following are equivalent, for a formula* $\alpha(\vec{x})$*:*

1. α *is preserved by homomorphism: if* $h : \mathbb{A} \to \mathbb{B}$ *is a homomorphism, and* $\vec{a}$ *is a vector of elements from* $\mathbb{A}$ *such that* $\mathbb{A} \models \alpha[\vec{a}]$*, then* $\mathbb{B} \models \alpha[\vec{ha}]$.
2. α *is logically equivalent to a positive-existential formula.*
3. α *is equivalent to a positive-existential formula in the category of finite models.*

Thus the homomorphism preorder captures the observable properties of models: this is the sense in which we view this preorder as an "information-preserving" one.

Geometric Theories. A theory T is a *geometric theory* if has an axiomatization by sentences of the form

$$\forall \vec{x}. \quad \alpha(\vec{x}) \to \beta(\vec{x}) \tag{1}$$

where α and β are positive-existential. [2] A geometric formula is *coherent* if all of its disjunctions are finite.

To gain intuition for why geometric logic is well-adapted to model finding, consider that a geometric formula $\alpha(\vec{x}) \to \beta(\vec{x})$ is true of a tuple $\vec{a}$ in a model $\mathbb{A}$ whenever $\mathbb{A}$ passes the test: "whenever $\alpha[\vec{a}]$ is observed, $\beta[\vec{a}]$ must also be observed."

The case for geometric logic as a logic of observable properties was made clearly by Abramsky [1]. Geometric logic plays a role in categorical logic [46] and topos theory [44]; and geometric logic is a natural formalism for specification [22,65,71,72]. Independently, Guttman observed [36,38] that a robust class of security goals for protocols are naturally expressed in the form (1).

There have been several investigations into deductive calculi for coherent logic [5,15,16,18,31,68]. Geometric sentences make a *Glivenko class*. That is to say, if a geometric formula is classically derivable from a geometric theory then it is in fact derivable intuitionistically.

Expressivity of Geometric Logic. Many theories are naturally geometric: any algebraic theory, any Horn theory, the theories that arise in disjunctive logic programming, etc. The fact that infinite disjunctions are permitted means that common inductive notions are geometric: transitive closure, an abelian group having torsion, the notion of a model of successor being standard, etc.

More interestingly, *any* first-order theory has a conservative geometric extension. There are a variety of approaches to this result; Dyckhoff and Negri [25] have given a particularly illuminating treatment of the issues.

[2] Confusingly, some authors use "geometric" to refer to what is more broadly called "positive existential". For them a "geometric theory" is a collection of quantified implications between "geometric formulas" (thus the axioms themselves are not "geometric formulas").

Set of Support and Completeness. A set of support for a class of models provides a complete "testbed" for entailment of geometric sentences.

Theorem 5. *Suppose T is a geometric theory and $\forall\vec{x}\,.\,\alpha(\vec{x}) \to \beta(\vec{x})$ is a geometric sentence. Let $\vec{a}$ be a sequence of fresh constants appropriate for $\vec{x}$ and suppose $\{\mathbb{A}_1, \mathbb{A}_2 \ldots\}$ is an arbitrary-homomorphisms set of support for $T \cup \{\alpha[\vec{a}]\}$. Then*

$$T \models \forall\vec{x}\,.\,\alpha(\vec{x}) \to \beta(\vec{x})$$

if and only if, each $\mathbb{A}_i \models \beta[\vec{a}]$.

Notice that when we can compute a *finite* set of support of finite models $\{\mathbb{A}_1, \mathbb{A}_2 \ldots \mathbb{A}_k\}$ for $T \cup \{\alpha[\vec{a}]\}$, Theorem 5 yields a decision procedure. Indeed it suffices that the $\{\mathbb{A}_1, \mathbb{A}_2 \ldots \mathbb{A}_k\}$ be, even if infinite, presented in such a way that $T \cup \{\alpha[\vec{a}]\}$ is decidable. This is the key to the decidability result in [23].

6 Direct Model Finding Methods

6.1 Chase-Based Approaches

The Chase is a method for building a model of a geometric theory T, or detecting that T is unsatisfiable. In fact the Chase as we present it here is a natural adaptation of the well-studied [19,29,30,34,34,45] Chase algorithm in the database community, used for checking implications between constraints and computing solutions to data exchange problems.

So let T be a geometric theory. We need to assume that T is written without function symbols other than constants. This is no real constraint, though, since the axioms required when replacing function symbols by relation symbols are themselves geometric.

By standard manipulations we can bring any geometric formula into the form

$$\forall\vec{x}\,.\,P(\vec{x}) \to \bigvee\{\exists\vec{y}_j\,.\,Q_j(\vec{x},\vec{y}_j) \mid j \in J\} \tag{2}$$

where P and each Q_j are conjunctions of atomic formulas. As usual, we view an empty disjunction as representing Falsehood, so that a formula as above with $J = \emptyset$ encodes $\forall\vec{x}\,.\,\neg P(\vec{x})$.

Assume now that each axiom of T is in this standard form.

Let C be an infinite set of fresh constants, to be used to name elements of the model we construct. Say that a *fact* over C is a closed atomic sentence over C. We build our model by starting with the empty set of facts, and gradually adding to it until it represents a model of T. Our set of facts is enlarged by doing *Chase steps* in a fair manner.

A Chase Step. Suppose F is a set of facts over C. Let σ be a formula in the form (2).

Suppose $\theta \equiv \{x_1 \mapsto c_1, \ldots, x_k \mapsto c_k\}$ is a substitution making P false in F, *i.e.*, $P(\theta\vec{x})$ holds in F yet for no j do we have $Q_j(\theta\vec{x})$ true in F. A Chase-step on F, σ, and θ is the result of

1. choosing some disjunct $E_j \equiv \exists y_{j1} \ldots y_{jk}\,.\,Q_j(\vec{x}, y_{j1} \ldots y_{jk})$

2. adding new elements $d_1, \ldots, d_k$ to C ;
3. adding each of the facts in $\theta' Q_j$ to F, where θ' is the substitution obtained by adding to θ each of the bindings $y_{ji} \mapsto d_i$.

The *Chase* consists of starting with the empty set of facts and iterating the above process.

We halt with success if we reach a finite set F of facts where we cannot apply a step, *i.e.*, when F is a model of T. We halt with failure if we reach a set F of facts where a formula with empty right-hand-side fails in F (which is to say, its left-hand-side is true): we cannot "repair" F to make such a formula true. It is posible that the Chase may not halt: conditions under which termination is guaranteed are an active area of study.

When the Chase is done in a "fair" manner and does not terminate, the resulting infinite set of facts will be a model of T: this is the essential content of the claim that The Chase is a complete deduction method for geometric logic.

Theorem 6 (Deductive Completeness). *Let T be geometric. Then T is satisfiable if and only if there is a fair run of the Chase which does not fail.*

A crucial thing about Chase calculations is that they make no commitments to facts that are not required by the theory. This leads to the following key theorem for model finding.

Theorem 7 (Set of Support). *Let T be geometric. For any model $\mathbb{M}$ of T there is an $\mathbb{M}_i$ obtained by some execution of the Chase and a homomorphism from $\mathbb{M}_i$ to $\mathbb{M}$.*

Provenance in Chase Models. Another aspect of the fact noted above that models built by the Chase are built "by need" is that provenance is easy to compute.

Referring to the general form of axioms (2) above, the only elements in a Chase model are those added to instantiate existential quantifiers $\exists \vec{y}$, and the only atomic facts in the model are those added to make the $Q_j(\vec{x}, \vec{y})$ true. It is easy enough to keep track of these justifications by introducing Skolem functions in the implementation to name elements, so that any fact in the model (Skolemized under the hood) can be traced back through the line of Chase steps that led to its addition. In this way we can answer provenance queries from the user.

The Chase in Practice. The Chase algorithm is described above as a nondeterministic procedure. The choice of rule to fire is one source of nondeterminism, but the important source is the disjunctions on the right-hand sides. Different choices of disjuncts to satisfy will (usually) lead to different output models. An implementation must negotiate this tree of models-in-progress. The original version of Razor managed this structure directly, but this was seen to be too slow for *general* use. (Specifically, performance on the TPTP suite of problems [67] was unacceptable).

Subsequent versions of Razor [63] take a hybrid approach. Roughly speaking, an SMT solver is used to manage the disjunctions, while Chase steps are used to construct elements and facts.

The Chase is used by Rowe, Ramsdell, and Kretz [62] to discover adversary behaviors that thwart layered attestation specifications. This implementation has good performance [59] in its domain without resorting to mechanisms external to the Chase.

6.2 CPSA

The CPSA protocol analyzer [20] is an example of a direct model finder. Actually CPSA is—superficially—of a different character from the other model finders discussed in this paper, since it is does not explicitly take a first-order theory as input. But the theory of strand spaces can be axiomatized, as a geometric theory [22] and the skeletons it returns are readily seen as first-order structures. And CPSA has always been founded on a notion of homomorphism as organizing principle [36]. The theory of strand spaces is not first-order, since a well-foundedness assumption about causality is imposed. But this well-foundedness can be enforced using the infinite disjunctions available in geometric logic.

The CPSA algorithm is not Chase-based, rather it relies instead on the Authentication Test mechanism [39]. (There is however, at least one project, an undergraduate thesis, implementing the strand space approach using the Chase [57].)

CPSA satisfies [37] the set of support criterion expressed by Theorem 5. As such it represents a sound and complete method for establishing security goals (for an unbounded number of sessions) that are represented as geometric sentences. As a technical remark, the category of models that CPSA works with has homomorphisms that are injective on nodes representing message events and arbitrary otherwise.

7 Programming Against a Solver: Theory

We present generic algorithms for model finding that rely on the primitive operation of asking a SAT or SMT solver, or an algorithm like the Chase, for a single finite model of a given theory.

To see that there is some work to do beyond simply invoking a solver, note the following.

1. Once the solver has determined that a theory T is satisfiable, and computed—internally—a model for T, the application must extract the model from the solver. But the API for doing this—in the solvers we are familiar with—is quite restricted. In any event, SMT-Lib compliant solvers are not *required* to make this process particularly convenient. Quoting from the SMT-Lib Standard (v.2.6) [2]:
   > The internal representation of the model A is not exposed by the solver. Similarly to an abstract data type, the model can be inspected only through ... [certain commands] ... As a consequence, it can even be partial internally and extended as needed in response to successive invocations of some of these commands.
2. A typical solver makes no promises about the models it returns other that they satisfy the input theory. So *fitness* requires some attention.
3. Repeated requests to a solver for the same theory will often return the same model. So *completeness* must be explicitly managed. (The Alloy tool does priority of excluding—in a best-effort way—models isomorphic to those previously seen.)

The generic algorithms in this section assume that they are working in categories of finite models that have a well-founded homomorphism ordering. In practice for us this

means that our categories have finite models as objects and either (i) there is a uniform bound on the domain size of models, or (ii) homomorphisms are injective or inclusions.

Space doesn't permit a review of solver-based augmentation or provenance here, even though tools such as Aluminum, Amalgam, and CompoSAT pursue ambitious goals and solve interesting problems along the way.

7.1 Building Blocks

There are two key sentences we can construct about a model that serve as the essential API with the solver. We'll show how to construct our models using these two building blocks, then make some remarks about how to compose these sentences. In each case there is no mystery about writing *some* sentence that works—the interesting task is to write sentences that behave well in practice.

Fix a theory T and a category of models of T.

1. $homTo_{\mathbb{A}}$: a sentence defining the models $\mathbb{P} \models T$ such that there is a homomorphism $h : \mathbb{P} \to \mathbb{A}$.
2. $homFrom_{\mathbb{A}}$: a sentence defining the models $\mathbb{P} \models T$ such that there is a homomorphism $h : \mathbb{A} \to \mathbb{P}$.

So

$$\neg homFrom_{\mathbb{A}}$$

defines the models $\mathbb{P} \models T$ that are not in the homomorphism cone of $\mathbb{A}$. Thus

$$below_{\mathbb{A}} := (homTo_{\mathbb{A}} \wedge \neg homFrom_{\mathbb{A}})$$

defines the models $\mathbb{P} \models T$ strictly below $\mathbb{A}$ in the homomorphism order.

Note that the above notions make sense whether we consider arbitrary homomorphisms or restrict to injective homomorphisms.

7.2 Minimization

If we bound the size of the domain(s) of our models then a-minimal models exist for any satisfiable theory: the $\precsim$ preorder is well-founded, so the set of minimal elements with respect to this order is non-empty. The question is, how do we compute a-minimal models?

The idea is that, given a model $\mathbb{A}$, we can use the sentences $homTo_{\mathbb{A}}$ and $homFrom_{\mathbb{A}}$ to iterate the process of constructing a model that is strictly below $\mathbb{A}$ in the $\precsim$ ordering.

Algorithm 8 (Minimize)

// A relevant category $\mathcal{C}$ of models of T is part of the context
input: *theory* T *and model* $\mathbb{A} \models T$
output: *model* $\mathbb{M} \models T$ *such that* $\mathbb{M}$ *is minimal for* $\mathcal{C}$ *and* $\mathbb{M} \precsim \mathbb{A}$
initialize: *set* $\mathbb{M}$ *to be* $\mathbb{A}$
while $T' \stackrel{def}{=} T \cup \{below_{\mathbb{M}}\}$ *is satisfiable, set* $\mathbb{M}$ *to be a model of* T'
return $\mathbb{M}$

Lemma 9. *Algorithm 8 is correct: if the category* $\mathcal{C}$ *has a well-founded homomorphism ordering and* $\mathbb{A}$ *is a finite model of* T *then Algorithm 8 terminates on* $\mathbb{A}$*, and the output* $\mathbb{M}$ *is an a-minimal model of* T *with* $\mathbb{M} \precsim \mathbb{A}$

7.3 Set of Support

We take the ability to generate a set-of-support for the class of all models of a theory T to be a natural notion of "completeness" in model-finding. Theorem 5 makes a precise claim of completness with respect to reasoning about geometric consequences of T.

Computing sets-of-support is another application of $homFrom_{\mathbb{A}}$, or more to the point, $\neg homFrom_{\mathbb{A}}$. Given theory T and model $\mathbb{A}$, if we construct the theory $T' \stackrel{\text{def}}{=} T \cup \{\neg homFrom_{\mathbb{A}}\}$ then calls to the SMT solver on theory T' are guaranteed to return models of T outside the hom-cone of $\mathbb{A}$ if any exist. So a set-of-support for T can be generated by iterating this process.

Recall that we can instrument any theory with a set of extra sentences uniformly bounding the size of the models of the enriched theory.

Algorithm 10 (SetOfSupport)

// A relevant category $\mathcal{C}$ of models of T is part of the context
input: *theory T with a uniformly bounded model size.*
output: *a stream $\mathbb{M}_1, \mathbb{M}_2, \ldots$ of minimal models of T such that for any $\mathbb{P} \models T$, there is some i such that $\mathbb{M}_i \precsim \mathbb{P}$.*
initialize: *set theory T^* to be T*
while *T^* is satisfiable*
    let $\mathbb{M}$ be a (minimal)[3] model of T^*
    output *$\mathbb{M}$*
    set T^* to be $T^* \cup \{\neg homFrom_{\mathbb{M}}\}$

If desired, of course, we can generate a potentially infinite set of support for a theory with unbounded domain sizes by "iterative deepening," letting the bounds increase indefinitely.

8 Programming Against a Solver: Practice

The generic model-finding algorithms above rely on two crucial building blocks: the sentences $homFrom_{\mathbb{A}}$ and $homTo_{\mathbb{A}}$, characterizing $\{\mathbb{P} \mid \mathbb{A} \precsim \mathbb{P}\}$ and $\{\mathbb{P} \mid \mathbb{P} \precsim \mathbb{A}\}$ respectively. It turns out that writing versions of these sentences to behave efficiently depends on the category of models we work in.

8.1 Constructing HomTo and HomFrom for Arbitrary Homomorphisms

The "Homomorphism Problem," deciding whether $\mathbb{M} \precsim \mathbb{N}$ for arbitrary homomorphisms, is NP-complete [32]. The more general problems of characterizing, for fixed $\mathbb{A}$, the sets $\{\mathbb{P} \mid \mathbb{A} \precsim \mathbb{P}\}$ and $\{\mathbb{P} \mid \mathbb{P} \precsim \mathbb{A}\}$ are deep and well-studied [3,7,35,50]. Here we focus, not on the computational complexity of the problem but the practical problem of asking SAT- and SMT-solvers to produce appropriate models, using *homFrom* and *homTo* sentences.

[3] Completeness of this algorithm does not require that the models $\mathbb{M}$ we work with are minimal. But if we do work with minimal models there will be fewer iterations.

homTo. Constructing a $homTo_{\mathbb{A}}$ sentence is straightforward ([21]) and even a naive such sentence seems to not cause our tools any trouble. We simply add a function symbol to the signature and write axioms saying that it is a homomorphism.

Algorithm 11 (Constructing homTo)

input: *model* $\mathbb{A}$ *over signature* Σ.
output: *sentence* $homTo_{\mathbb{A}}$ *in an expanded signature* Σ^+, *such that for any model* $\mathbb{P} \models \Sigma$, $\mathbb{P} \precsim \mathbb{A}$ *iff there is an expansion* $\mathbb{P}^+$ *of* $\mathbb{P}$ *to* Σ^+ *with* $\mathbb{P}^+ \models homTo_{\mathbb{A}}$.

define Σ^+ *to be the extension of* Σ *obtained by*
- *adding a set of fresh constants naming elements of the domain of* $\mathbb{A}$
- *adding a function symbol* $h_S : S \to S$ *at each sort* S

return $homTo_{\mathbb{A}}$ *as the conjunction of the following sentences, one for each function symbol* f *and predicate* R *in* Σ. *Here* $\vec{e}$ *and* e' *range over the names for elements of* $\mathbb{A}$.

$$\forall \vec{x}, y \,.\, f\vec{x} = y \;\rightarrow\; \bigvee \{(\vec{hx} = \vec{e} \wedge y = e') \mid \mathbb{A} \models f\vec{e} = e'\}$$
$$\forall \vec{x} \,.\, R\vec{x} = true \;\rightarrow\; \bigvee \{(\vec{hx} = \vec{e}) \mid \mathbb{A} \models R\vec{e} = true\}$$

If we are working in a category of injective maps, simply add a sentence to say that h is injective.

Lemma 12. *There is a homomorphism from* $\mathbb{B}$ *to* $\mathbb{A}$ *iff there is a model* $\mathbb{B}^+ \models homTo_{\mathbb{A}}$ *such that* $\mathbb{B}$ *is the reduction to* Σ *of* $\mathbb{B}^+$.

Constructing HomFrom. Let $\mathbb{A}$ be a finite model over signature Σ. Define an expanded signature Σ^+ by adding, for each element e of the domain of $\mathbb{A}$, a constant c_e, and let $\mathbb{A}^+$ be the corresponding expansion of $\mathbb{A}$. The *diagram* $\Delta_{\mathbb{A}}$ of $\mathbb{A}$ is the set of atomic sentences and negations of atomic sentences true in $\mathbb{A}^+$. If we take only the atomic sentences, the result is the *positive diagram* $\Delta^+_{\mathbb{A}}$ of $\mathbb{A}$.

The sentence—in the original signature Σ—obtained by converting the new constants in the positive diagram to variables and existentially quantifying them is called the *characteristic sentence* $ch_{\mathbb{A}}$ of $\mathbb{A}$.

It is easy to see that the models of $ch_{\mathbb{A}}$ are precisely those models $\mathbb{B}$ with $\mathbb{A} \precsim \mathbb{B}$.

But observe that for our purposes we are interested in characterizing (by a first-order sentence) the *complement* of the set of $\mathbb{B}$ such that $\mathbb{A} \precsim \mathbb{B}$. Certainly the negation $\neg ch_{\mathbb{A}}$ suffices. But simply negating $ch_{\mathbb{A}}$ leads to computationally unwieldy formulas, since universal quantifiers are bottlenecks for SMT-solvers.

The ideal outcome would be to construct an existential sentence capturing the complement of the hom cone of $\mathbb{A}$. Equivalently we might look for a structure $\mathbb{D}$ such that for any $\mathbb{X}$, $\mathbb{X} \precsim \mathbb{D}$ iff $\mathbb{A} \not\precsim \mathbb{X}$. This is called "homomorphism duality" in the literature. Such a structure doesn't always exist, and even if it does, it can be exponentially large in the size of $\mathbb{A}$ [28]. So we must turn to heuristic methods.

The model finders Razor and LPA work with arbitrary homomorphisms. The former tool uses, essentially, $homTo_{\mathbb{A}}$ as defined in Algorithm 11, and uses the straightforward $ch_{\mathbb{A}}$ in building $homFrom_{\mathbb{A}}$. LPA inherits this approach from Razor. Elsewhere we have written [21,24] about some best-effort techniques for constructing $homFrom_{\mathbb{A}}$ sentences.

8.2 Constructing HomTo and HomFrom for Submodel Morphisms

"Bounded model finding" as represented by (for example) Alloy imposes a uniform bound on the sizes of models for the duration of each analysis session. Indeed it introduces a set of *names* for model elements[3]. Let us choose to respect distinctness of names in our homomorphisms. We arrive at the notion of a "submodel" category $\mathcal{B}$ of models, with the following properties.

- There is a fixed finite set C of constants, and every element of a model in $\mathcal{B}$ is named by a constant in C (we do not assume unique names per element).
- The ordering $\mathbb{A} \sqsubseteq \mathbb{B}$ on models is the submodel ordering

It is not obvious that ordering by submodel, which requires that relationships between models respect names, is the best choice for model exploration. The next section takes up this question.

In these submodel categories it turns out that $homTo_{\mathbb{A}}$ and $homFrom_{\mathbb{A}}$ can be expressed compactly, with propositional formulas.

Constructing homTo. Given model $\mathbb{A}$ we take $homTo_{\mathbb{A}}$ to be the propositional sentence

$$\bigwedge\{\neg\alpha \mid \alpha \text{ is atomic, } \mathbb{A} \models \neg\alpha\}$$

Note in particular that if c and c' are constants naming distinct elements of $\mathbb{A}$, then $c \neq c'$ is one of the conjuncts of $homTo_{\mathbb{A}}$. This sentence works: if $\mathbb{B}$ satisfies this then the identity map on names is a homomorphism, since there are no facts of $\mathbb{B}$ that make an obstacle.

Constructing homFrom. Given model $\mathbb{A}$ we take $homFrom_{\mathbb{A}}$ to be the propositional sentence

$$\bigwedge\{\beta \mid \beta \text{ is atomic, } \mathbb{A} \models \beta\}$$

so that $\neg homFrom_{\mathbb{A}}$, used to avoid the homomorphism-cone of $\mathbb{A}$, is

$$\bigvee\{\neg\beta \mid \beta \text{ is atomic, } \mathbb{A} \models \beta\}$$

This sentence works: it is just a version of the naive $ch_{\mathbb{A}}$ but we do not have to introduce existential quantifiers since our homomorphisms are so constrained.

Minimization and Set of Support are now easily computed. One might wonder whether the SAT-solving iteration required, in Algorithm 8, to reduce a model to a minimal one might be expensive. But experimental evaluation [54] has shown that in fact minimal models are returned quite quickly.

Examples. The model finders Aluminum, and Amalgam work with the inclusion ordering on models. (They inherit this aspect from Kodkod, their core model-finding engine.) They use the versions of $homTo_{\mathbb{A}}$ and $homFrom_{\mathbb{A}}$ developed in this section.

[3] somewhat confusingly, names are called "bounds" in the Alloy community.

8.3 Constructing homTo and homFrom for Injective Morphisms

As a final note, we observe the somewhat surprising fact that if we relax the notion of homomorphism in bounded categories to require only injectivity, as opposed to preservation of names, then analysis is almost as easy as working with submodels.

For minimization: to say that there is an injective homomorphism from $\mathbb{A}$ to $\mathbb{B}$ is to say that $\mathbb{A}$ is isomorphic to a submodel of $\mathbb{B}$. Since we only care about models up to isomorphism, the problem of minimization with respect to injective maps is the same problem as minimization with respect to the submodel ordering.

For set of support: this is not quite "the same problem" in the two orderings. We require one more idea. The key fact is this: if $\mathbb{B}$ is a *minimal* model of a theory and there is an injective homomorphism from $\mathbb{A}$ to $\mathbb{B}$ then in fact $\mathbb{A}$ and $\mathbb{B}$ are isomorphic. Since: our assumption implies that $\mathbb{A}$ is isomorphic to a submodel of $\mathbb{B}$. If this submodel is not $\mathbb{B}$ itself, it would provide a counterexample to the minimality of $\mathbb{B}$.

The upshot of this is that the only obstacle to the correctness of the Set of Support algorithm for injective homomorphism when we use the submodel-based *homTo* and *homFrom* constructions is that we might construct isomorphic copies of previously-derived models. So all we have to do is add an isomorphism check at the end of the next-model algorithm. Such a check is not known to be asymptotically efficient, but is a standard operation and behaves well in practice.

9 Conclusion

Motivated by the idea of using model finding to explore theories as opposed to simply checking satisfiability, we have argued for development of *model-finding assistants* to aid users in understanding software artifacts. We have (i) suggested some core design principles for a model finding tool, (ii) argued for geometric logic as a convenient formalism for a specifications supporting a computational interpretation, and (iii) outlined some fundamental building blocks that supply core functionality in generating a set of support for the finite models of a theory.

Note and Acknowledgements. This paper surveys some recent work—foundational and applied—by a variety of authors in model finding. All of it has been previously published. My purpose in gathering this material into one place is to point out the shared foundations for a number of different model finders and to identify some differences in their aspirations and in their functionalities.

As the technical content of this paper draws so heavily on previous work with coauthors, my feeling of gratitude to my colleagues is stronger than usual. I want to particularly thank *Natasha Danas, Joshua Guttman, Kathi Fisler, Shriram Krishnamurthi, Timothy Nelson, John Ramsdell, and Salman Saghafi* for their insights and contributions.

References

1. Abramsky, S.: Domain theory in logical form. Ann. Pure Appl. Logic **51**(1–2), 1–77 (1991)
2. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB) (2016). www.SMT-LIB.org

3. Barto, L., DeMeo, W.J., Mottet, A.: The complexity of the homomorphism problem for boolean structures (2020). CoRR **abs/2010.04958**, https://arxiv.org/abs/2010.04958
4. Baumgartner, P., Fuchs, A., Nivelle, H.D., Tinelli, C.: Computing finite models by reduction to function-free clause logic. J. Appl. Logic **7**(1), 58–74 (2009)
5. Bezem, M., Coquand, T.: Automating coherent logic. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 246–260. Springer, Heidelberg (2005). https://doi.org/10.1007/11591191_18
6. Blanchette, J.C., Nipkow, T.: Nitpick: a counterexample generator for higher-order logic based on a relational model finder. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 131–146. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14052-5_11
7. Bodirsky, M., Feller, T., Knäuer, S., Rudolph, S.: On logics and homomorphism closure (2021). CoRR abs/2104.11955, https://arxiv.org/abs/2104.11955
8. Bouajjani, A., Fernandez, J.-C., Halbwachs, N.: Minimal model generation. In: Clarke, E.M., Kurshan, R.P. (eds.) CAV 1990. LNCS, vol. 531, pp. 197–203. Springer, Heidelberg (1991). https://doi.org/10.1007/BFb0023733
9. Bry, F., Yahya, A.: Minimal model generation with positive unit hyper-resolution tableaux. In: Miglioli, P., Moscato, U., Mundici, D., Ornaghi, M. (eds.) TABLEAUX 1996. LNCS, vol. 1071, pp. 143–159. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61208-4_10
10. Bry, F., Yahya, A.: Positive unit hyperresolution tableaux and their application to minimal model generation. J. Autom. Reas **25**, 35–82 (2000)
11. Bulwahn, L.: The new quickcheck for isabelle. In: Hawblitzel, C., Miller, D. (eds.) CPP 2012. LNCS, vol. 7679, pp. 92–108. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35308-6_10
12. Chamarthi, H.R., Dillinger, P.C., Kaufmann, M., Manolios, P.: Integrating testing and interactive theorem proving. In: Hardin, D., Schmaltz, J. (eds.) Proceedings 10th International Workshop on the ACL2 Theorem Prover and its Applications, ACL2 2011, Austin, Texas, USA, 3–4 November 2011. EPTCS, vol. 70, pp. 4–19 (2011)
13. Claessen, K., Sorensson, N.: New techniques that improve MACE-style finite model finding. In: Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications. Citeseer (2003)
14. Claessen, K., Hughes, J.: QuickCheck. In: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming - ICFP '00. ACM Press (2000)
15. Coquand, T.: A completeness proof for geometric logic. In: Logic, Methodology and Philosophy of Science. Proceedings of the Twelfth International Congress, pp. 79–90 (2010)
16. Coste, M., Lombardi, H., Roy, M.F.: Dynamical method in algebra: effective nullstellensätze. Ann. Pure Appl. Logic **111**(3), 203–256 (2001)
17. Danas, N., Nelson, T., Harrison, L., Krishnamurthi, S., Dougherty, D.J.: User studies of principled model finder output. In: Cimatti, A., Sirjani, M. (eds.) SEFM 2017. LNCS, vol. 10469, pp. 168–184. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66197-1_11
18. de Nivelle, H., Meng, J.: Geometric resolution: a proof procedure based on finite model search. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 303–317. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_28
19. Deutsch, A., Nash, A., Remmel, J.: The chase revisited. In: ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 149–158 (2008)
20. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Searching for shapes in cryptographic protocols. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 523–537. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_41
21. Dougherty, D.J., Guttman, J.D., Ramsdell, J.D.: Homomorphisms and Minimality for Enrich-by-Need Security Analysis. ArXiv e-prints (2018)

22. Dougherty, D.J., Guttman, J.: Geometric logic and strand spaces. In: 5th International Workshop on Security and Rewriting Techniques (2010)
23. Dougherty, D.J., Guttman, J.D.: Decidability for lightweight Diffie-Hellman protocols. In: IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19–22 July 2014, pp. 217–231 (2014)
24. Dougherty, D.J., Guttman, J.D., Ramsdell, J.D.: Security protocol analysis in context: computing minimal executions using SMT and CPSA. In: Furia, C.A., Winter, K. (eds.) IFM 2018. LNCS, vol. 11023, pp. 130–150. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98938-9_8
25. Dyckhoff, R., Negri, S.: Geometrisation of first-order logic. Bull. Symb. Logic **21**, 123–163 (2015)
26. Eastlund, C.: Doublecheck your theorems. In: Proceedings of the Eighth International Workshop on the ACL2 Theorem Prover and its Applications, pp. 42–46 (2009)
27. El Ghazi, A.A., Taghdiri, M.: Analyzing alloy constraints using an SMT solver: a case study. In: 5th International Workshop on Automated Formal Methods (AFM) (2010)
28. Erdős, P.L., Pálvölgyi, D., Tardif, C., Tardos, G.: Regular families of forests, antichains and duality pairs of relational structures. Combinatorica **37**(4), 651–672 (2017). https://doi.org/10.1007/s00493-015-3003-4
29. Fagin, R., Kolaitis, P.G., Popa, L.: Data exchange: getting to the core. ACM Trans. Database Syst. (TODS) **30**(1), 174–210 (2005)
30. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theor. Comput. Sci. **336**(1), 89–124 (2005)
31. Fisher, J., Bezem, M.: Skolem machines. Fundamenta Informaticae **91**(1), 79–103 (2009)
32. Garey, M.R., Johnson, D.S.: Computers and intractability. w. h (1979)
33. Geisler, T., Panne, S., Schütz, H.: Satchmo - the compiling and functional variants. J. Autom. Reas. **18**(2), 227–236 (1997)
34. Gottlob, G.: Computing cores for data exchange: new algorithms and practical solutions. In: ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 148–159 (2005)
35. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. J. ACM (JACM) **54**(1), 1–24 (2007)
36. Guttman, J.D.: Security theorems via model theory. EXPRESS Express. Conc. (EPTCS) **8**, 51 (2009). https://doi.org/10.4204/EPTCS.8.5
37. Guttman, J.D.: Shapes: surveying crypto protocol runs. In: Cortier, V., Kremer, S. (eds.) Formal Models and Techniques for Analyzing Security Protocols. IOS Press, Cryptology and Information Security Series (2011)
38. Guttman, J.D.: Establishing and preserving protocol security goals. J. Comput. Secur. **22**(2), 203–267 (2014)
39. Guttman, J.D., Thayer, F.J.: Authentication tests and the structure of bundles. Theor. Comput. Sci. **283**(2), 333–380 (2002)
40. Hughes, J.: QuickCheck testing for fun and profit. In: Hanus, M. (ed.) PADL 2007. LNCS, vol. 4354, pp. 1–32. Springer, Heidelberg (2006). https://doi.org/10.1007/978-3-540-69611-7_1
41. Jackson, D.: Alloy: a language and tool for exploring software designs. Commun. ACM **62**(9), 66–76 (2019)
42. Koshimura, M., Nabeshima, H., Fujita, H., Hasegawa, R.: Minimal model generation with respect to an atom set. In: International Workshop on First-Order Theorem Proving (2009)
43. Krishnamurthi, S., Nelson, T.: The human in formal methods. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 3–10. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_1

44. Mac Lane, S., Moerdijk, I.: Sheaves in Geometry and Logic: A First Introduction to Topos Theory. Universitext, Springer, New York (1992). https://doi.org/10.1007/978-1-4612-0927-0
45. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Trans. Database Syst. (TODS) **4**(4), 455–469 (1979)
46. Makkai, M., Reyes, G.E.: First Order Categorical Logic. LNM, vol. 611. Springer, Heidelberg (1977). https://doi.org/10.1007/BFb0066201
47. Maldonado-Lopez, F.A., Chavarriaga, J., Donoso, Y.: Detecting network policy conflicts using Alloy. In: Ameur, Y.A., Schewe, K. (eds.) Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014, Toulouse, France, 2–6 June 2014. Proceedings. Lecture Notes in Computer Science, vol. 8477, pp. 314–317. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43652-3_31
48. Maoz, S., Ringert, J.O., Rumpe, B.: CD2Alloy: class diagrams analysis using alloy revisited. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 592–607. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24485-8_44
49. Marinov, D., Khurshid, S.: Testera: a novel framework for automated testing of java programs. In: Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001), pp. 22–31. IEEE (2001)
50. Marx, D.: Tractable hypergraph properties for constraint satisfaction and conjunctive queries. J. ACM (JACM) **60**(6), 1–51 (2013)
51. McCune, W.: Mace4 reference manual and guide (2003). arXiv preprint cs/0310055
52. Milicevic, A., Misailovic, S., Marinov, D., Khurshid, S.: Korat: a tool for generating structurally complex test inputs. In: 29th International Conference on Software Engineering (ICSE'07), pp. 771–774. IEEE (2007)
53. Nelson, T., Danas, N., Dougherty, D.J., Krishnamurthi, S.: The power of Why and Why Not: enriching scenario exploration with provenance. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, 4–8 September 2017, pp. 106–116 (2017)
54. Nelson, T., Saghafi, S., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Aluminum: Principled scenario exploration through minimality. In: 35th International Conference on Software Engineering (ICSE), pp. 232–241 (2013)
55. Nelson, T., Barratt, C., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: The Margrave tool for firewall analysis. In: Proceedings of the 24th USENIX Large Installation System Administration Conference (LISA 2010) (2010)
56. Paraskevopoulou, Z., Hriţcu, C., Dénès, M., Lampropoulos, L., Pierce, B.C.: Foundational property-based testing. In: Urban, C., Zhang, X. (eds.) ITP 2015. LNCS, vol. 9236, pp. 325–343. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22102-1_22
57. Pombrio, J.L.: Protocol analysis via the chase. Technical report, Worcester Polytechnic Institute (2011)
58. Porncharoenwase, S., Nelson, T., Krishnamurthi, S.: CompoSAT: specification-guided coverage for model finding. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) FM 2018. LNCS, vol. 10951, pp. 568–587. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_34
59. Ramsdell, J.: Personal communication (2021)
60. Reynolds, A., Tinelli, C., Goel, A., Krstić, S.: Finite model finding in SMT. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 640–655. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_42
61. Rossman, B.: Homomorphism preservation theorems. J. ACM (JACM) **55**(3), 15 (2008)
62. Rowe, P.D., Ramsdell, J.D., Kretz, I.D.: Automated trust analysis for layered attestations. Submitted for publication (2021)

63. Saghafi, S., Danas, R., Dougherty, D.J.: Exploring theories with a model-finding assistant. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 434–449. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_30
64. Saghafi, S., Dougherty, D.J.: Razor: provenance and exploration in model-finding. In: 4th Workshop on Practical Aspects of Automated Reasoning (PAAR) (2014)
65. Saghafi, S., Nelson, T., Dougherty, D.J.: Geometric logic for policy analysis. In: International Workshop on Automated Reasoning in Security and Software Verification (ARSEC 2013), pp. 12–20 (2013)
66. Shao, D., Khurshid, S., Perry, D.E.: Whispec: white-box testing of libraries using declarative specifications. In: Proceedings of the 2007 Symposium on Library-Centric Software Design, pp. 11–20 (2007)
67. Sutcliffe, G.: The TPTP problem library and associated infrastructure. From CNF to TH0, TPTP v6.4.0. J. Autom. Reas. **59**(4), 483–502 (2017)
68. Thorstensen, E.: Instance-Based Hyper-Tableaux for Coherent Logic. Master's thesis, University of Oslo (2009)
69. Torlak, E., Jackson, D.: Kodkod: a relational model finder. In: Conference on Tools and Algorithms for the Construction and Analysis of Systems (2007)
70. Vakili, A., Day, N.A.: Finite model finding using the logic of equality with uninterpreted functions. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) FM 2016. LNCS, vol. 9995, pp. 677–693. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48989-6_41
71. Vickers, S.: Geometric logic in computer science. In: Burn, G.L., Gay, S.J., Ryan, M. (eds.) Theory and Formal Methods 1993, Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods, Isle of Thorns Conference Centre, Chelwood Gate, Sussex, UK, 29–31 March 1993, pp. 37–54. Workshops in Computing, Springer, Heideleberg (1993). https://doi.org/10.1007/978-1-4471-3503-6_4
72. Vickers, S.: Geometric logic as a specification language. In: Hankin, C., Mackie, I., Hankin, R.N., Mackie, I., Nagarajan, R. (eds.) Proceedings for the Second Imperial College Department of Computing Workshop on Theory and Formal Methods, pp. 321–340 (1995)
73. Zhang, J., Zhang, H.: SEM: a system for enumerating models. In: IJCAI, vol. 95, pp. 298–303 (1995)