Exploring Theories with a Model-Finding Assistant[*]

Salman Saghafi, Ryan Danas, and Daniel J. Dougherty

Worcester Polytechnic Institute, Worcester, Massachusetts, USA

Abstract. We present an approach to understanding first-order theories by exploring their models. A typical use case is the analysis of artifacts such as policies, protocols, configurations, and software designs. For the analyses we offer, users are not required to frame formal properties or construct derivations. Rather, they can explore examples of their designs, confirming the expected instances and perhaps recognizing bugs inherent in surprising instances.
Key foundational ideas include: the information preorder on models given by homomorphism, an inductively-defined refinement of the Herbrand base of a theory, and a notion of provenance for elements and facts in models. The implementation makes use of SMT-solving and an algorithm for minimization with respect to the information preorder on models.
Our approach is embodied in a tool, Razor, that is complete for finite satisfiability and provides a read-eval-print loop used to navigate the set of finite models of a theory and to display provenance.

1 Introduction

Suppose $\mathcal{T}$ is a first-order theory. If $\mathcal{T}$ specifies a software artifact written by a user, such as an access-control policy, a description of a protocol, or a software design, our user will want to understand whether or not the logical consequences of $\mathcal{T}$ match her expectations. A standard approach using automated deduction tools offers the following workflow: (i) the user specifies, as a sentence σ, some typical property she hopes will hold about the system, then (ii) checks whether σ is provable from $\mathcal{T}$, using a theorem-prover or a proof-assistant.

An alternative approach is to explore the *models* of $\mathcal{T}$. This is of course logically at least as rich as the deductive approach, since σ will hold iff $\mathcal{T} \cup \neg\sigma$ has no models. But the model-exploring approach offers a wider range of affordances to the user than does deduction. For one thing, if property σ fails of $\mathcal{T}$, it can be instructive to see *example situations,* that is, to see concrete models of $\mathcal{T} \cup \neg\sigma$. This will be especially useful if we can offer tools to help our user *understand* these examples ("what is that element doing there? why is that fact true?").

More radically, our user might use a model-building tool to explore models of $\mathcal{T}$ *without having to articulate logical consequences.* For example, if $\mathcal{T}$ describes

[*]This material is based upon work supported by the National Science Foundation under Grant No. CNS-1116557.

a policy for accessing a building, our user can explore the question, "who can enter after 5 pm?" by expressing "someone enters after 5 pm" as a sentence σ and asking for models of $(\mathcal{T} \cup \{\sigma\})$. The resulting models may capture situations that confirm the user's expectation, but there also may be models with unanticipated settings, allowing surprising accesses, which uncover gaps in the policy.

Model-finding is an active area of investigation [1–8]. But—with some exceptions noted below in Related Work—existing model-finders compute an essentially random set of models, present them to the user in arbitrary order, provide no facility for exploring the space of models in a systematic way, and offer little help to users in understanding a given model. We will clarify below what we mean by "understanding" a model, but the notion has clear intuitive force. For example when a sysadmin is debugging a firewall policy, a typical question at hand is: what rule blocked (or allowed) this packet?

As our main contribution, we initiate a theory of *exploration* of finite models, with two main components: (i) a notion of *provenance* as a way to explain why elements are in the model and why properties are true of them, and (ii) strategies for traversing the models of an input theory by *augmentation*. Our approach is realized in a model-finding assistant, Razor.[1] We call Razor a model-finding *assistant* because users interact with it to build and examine models.

Minimality and the Chase. At the core of our approach is the notion of a homomorphism between models (Section 2) and the preorder $\preccurlyeq$ determined by homomorphism. A homomorphism preserves information, so that if $\mathbb{A}$ and $\mathbb{B}$ are each models of some phenomenon and $\mathbb{A} \preccurlyeq \mathbb{B}$, with $\mathbb{B} \not\preccurlyeq \mathbb{A}$, then we prefer to show $\mathbb{A}$ to the user, at least initially, since it has less "extraneous" information than $\mathbb{B}$. The theoretical foundations of our tool derive from the classical Chase algorithm from database theory (Section 3), and our core algorithm (Section 4) builds models that are minimal in the homomorphism ordering

Provenance. As a direct consequence of the fact that Razor ultimately computes Chase-models, Razor can display provenance information for elements and facts. Any element in a Chase-model is there in response to a sentence in the user's input (Section 3.2), indeed as a witness for a particular existential quantifier in the input theory. Similarly, any atomic fact of a Chase-model is there because of the requirement that a particular input sentence hold. Razor keeps track of these justifications—we call them "naming" and "blaming," respectively—and can answer provenance queries from the user.

Augmentation. Focusing on Chase models promotes conceptual clarity by allowing the user to focus only on models with no inessential aspects. But the user can access other models of the theory by augmenting models by new facts. When a user asks to augment a given model $\mathbb{M}$ of a theory $\mathcal{T}$ by some fact F, other consequences may be entailed by $\mathcal{T}$, perhaps "disjunctive" consequences. Razor thus computes a stream consisting of all the *minimal* extensions of $\mathbb{M}$ by the augmenting fact F (Section 3.1). There may be none: F may be inconsistent with the state of affairs $\mathbb{M}$, which may be of real significance. The (relative) minimality of the resulting models ensures that provenance information can be

[1] http://salmans.github.io/Razor/

computed over them as well. Most important of all, this augmentation will be under the control of the user.

Implementation. We have found it more efficient to implement a variation on the Chase, which leverages an SMT-solver to handle the difficulties arising form disjunctions and equations (Section 4). A key ingredient of this approach is the use of a refinement of the notion of the Herbrand base of a theory, the *possible facts* set defined in Algorithm 3.

The REPL. Since the original input theory need not be a Horn theory, we will not expect unique minimal models. Razor provides a read-eval-print loop in which users can (i) ask for the next model in the current stream, (ii) play "what-if?" by augmenting the currently-displayed model with a new fact or (iii) ask for the provenance of elements or facts in the current model.

1.1 Related Work

Model-Finding. The development of algorithms for the generation of finite models is an active area of research. The prominent method is "MACE-style" [2], embodied in tools such as Paradox [3], Kodkod [6], which reduce the problem to be solved into propositional logic and employ a SAT-solver. "Instance based" methods for proof search can be adapted to compute finite models [5, 7, 9]. Our approach is related to the bottom-up model generation [4] method and the refutationally complete solution presented in [10]. Our techniques for bounding the search are related to those presented in [11]. Closer in spirit to our goals are lightweight formal methods tools such as Alloy [12] and Margrave [13, 14]. The goals of these works differ from ours in that their main concern is usually not the *exploration* of the space of all models of a theory.

Minimality. Logic programming languages produce single, *least* models as a consequence of their semantics. In more specialized settings, generation of minimal models usually relies on dedicated techniques, often based on tableaux [15] or hyperresolution [16]. Aluminum [17] supports exploration by returning minimal models: it instruments the model-finding engine of Alloy. It thus inherits the limitation that it requires user-supplied bounds, and it cannot generate provenance information. The Cryptographic Protocol Shapes Analyzer [18] also generates minimal models. However, its application domain and especially algorithms are quite different from ours. The Network Optimized Datalog tool [19], which has been released as a part of Z3 [20], presents limited minimization and provenance construction for reasoning about beliefs in the context of network reachability policies.

Geometric Logic. The case for geometric logic as a logic of observable properties was made clearly by Abramsky [21] and has been explored as a notion of specification by several authors [22, 23] Geometric logic for theorem-proving was introduced in [24] and generalized in [5]. The crucial difference with the current work is of course the fact that we focus on model-finding and exploration.

Chase. Our model-finding is founded on the Chase, an algorithm well-known in the database community [25–27]. Challenges arise for us in managing the complexity that arises due to disjunction, and in treating equality. Our strategy for addressing these challenges comprises Section 4.

2 Preliminaries

We work over a first-order signature with relation symbols (including equality). As syntactic sugar for users, we allow function symbols in the concrete syntax. It turns out to be convenient and flexible to interpret such function symbols as partial functions. What this means formally is that when relation symbols are translated to function symbols in the usual way, theories are augmented with axioms ensuring singled-valued-ness but not necessarily with totality axioms. *Skolemization* will play an important role in the following, especially as regards provenance of elements; we assume familiarity with the basic notions.

Models, and the notion of satisfaction of a formula in a model, are defined in the usual way. If $\mathcal{T}$ is a theory we write $Mod(\mathcal{T})$ for the class of models of $\mathcal{T}$. If $\Sigma \subseteq \Sigma^+$ are signatures and $\mathbb{M}$ is a Σ^+-model then the *reduct* of $\mathbb{M}$ to Σ is obtained in the obvious way by ignoring the relations of Σ^+ not in Σ. A *homomorphism* from $\mathbb{M}$ to $\mathbb{N}$ is a map from the domain of $\mathbb{M}$ to the domain of $\mathbb{N}$, $h : |\mathbb{M}| \to |\mathbb{N}|$, such that for every relational symbol R and tuple $\langle \mathbf{e_1}, \ldots \mathbf{e_n} \rangle$ of elements of $|\mathbb{M}|$, if $\mathbb{M} \models R[\mathbf{e_1}, \ldots, \mathbf{e_n}]$ then $\mathbb{N} \models R[h(\mathbf{e_1}), \ldots, h(\mathbf{e_n})]$. We write $\mathbb{M} \preccurlyeq \mathbb{N}$ for the preorder defined by the existence of a homomorphism from $\mathbb{M}$ to $\mathbb{N}$. We say that $\mathbb{M}$ is a *minimal model* in a class $\mathcal{C}$ of models if $\mathbb{N} \in \mathcal{C}$ and $\mathbb{N} \preccurlyeq \mathbb{M}$ implies $\mathbb{M} \preccurlyeq \mathbb{N}$. A set $\mathcal{M}$ of models is a *set-of-support* for a class $\mathcal{C}$ if for each $\mathbb{N} \in \mathcal{C}$ there is some $\mathbb{M} \in \mathcal{M}$ with $\mathbb{M} \preccurlyeq \mathbb{N}$.

2.1 Logic in Geometric Form

A *positive-existential* formula (PEF) is one built from atomic formulas (including $\top$ and $\bot$) using $\wedge$, $\vee$, and $\exists$. If $\alpha(\vec{x})$ is a PEF true of a tuple $\vec{\mathbf{e}}$ in a model $\mathbb{M}$ then the truth of this fact is supported by a finite fragment of $\mathbb{M}$. Thus if $\mathbb{M}$ satisfies α with $\vec{\mathbf{e}}$ and $\mathbb{M}$ is expanded, by adding new elements and/or new facts, $\alpha(\vec{x})$ still holds of $\vec{\mathbf{e}}$ in the resulting model. For this reason, properties defined by PEF are sometimes called *observable* properties [21].

It is a classical result that PEFs are precisely the formulas preserved under homomorphisms; Rossman [28] has shown that this holds even if we restrict attention to finite models only. 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.

A sentence is *geometric* if it is of the form $\forall \vec{x}\, (\varphi \Rightarrow \psi)$, where φ and ψ are PEFs. It is often convenient to suppress writing the universal quantification explicitly. We sometimes refer to φ and ψ respectively as the *body* and *head* of $\varphi \Rightarrow \psi$. Note that an empty conjunction may be regarded as truth ($\top$) and an empty disjunction as falsehood ($\bot$). So we may view a universally quantified PEF, or a universally quantified negated PEF, as a geometric sentence. A theory is in *geometric form* if it consists of a set of geometric sentences. [2] Thus logic in geometric form is the logic of implications between observable properties.

[2]The term "geometric" arises from the original study of this class of formulas in the nexus between algebraic geometry and logic [29].

By routine logical manipulations we may assume that every geometric sentence $\varphi \Rightarrow \psi$ is in *standard* form

$$\alpha(\vec{x}) \Rightarrow \bigvee_i (\exists y_{i1} \ldots \exists y_{ip} . \beta_i(\vec{x}, y_{i1}, \ldots, y_{ip})),$$

where α and each β_i is a conjunction of atoms.

Transformation to Geometric Form. The sense in which geometric form is—and is not—a restriction is delicate, but interesting. As is well-known, any theory is equisatisfiable with one in conjunctive normal form, by introducing Skolem functions. And modulo trivial equivalences, such a sentence is a geometric one. But Skolemization has consequences for *user-centered* model-finding. For example, traditionally, Skolem functions are *total*, and of course it is easy to achieve this abstractly by making arbitrary choices if necessary. But for reasons connected with computing provenance and keeping models finite, it is much more convenient to work with partial functions, or in other words, at-most-single valued relations.

It is easy to check that $\mathcal{T}$ can be put in geometric form—without Skolemization —whenever each axiom is an $\forall\exists$ sentence, with the caveat that no existential quantifier has within its scope both an atom with negative polarity and one of positive polarity. This circumstance arises infrequently in practice. We prefer to avoid Skolemization if possible. Any Skolemization necessary for putting a theory in this form is considered to happen "off stage."

3 Model-Finding via the Chase

In this section we outline the essential features of the Chase, since it is the most natural setting for understanding the way that minimality and provenance drive a general model-finding framework based on geometric form. It turns out that a straightforward implementation of the Chase algorithm is too inefficient. In Section 4 we describe the strategy we use in Razor to build the same models the Chase would construct but using SMT-solving technology for efficiency.

It is easiest to present the standard Chase as a non-deterministic procedure. We assume given an infinite set K of symbols used to construct elements of the model: at any stage of the process we will have identified a finite subset K' of K and (if the theory $\mathcal{T}$ involves equality) a congruence relation over K'. The congruence classes are the elements of the model.

Assume that the input theory $\mathcal{T}$ is presented in standard geometric form. At a given stage, if the current model $\mathbb{M}$ is not yet a model of $\mathcal{T}$ then there is some sentence σ of $\mathcal{T}$ false in $\mathbb{M}$:

$$\sigma \equiv \alpha(x_1, \ldots, x_k) \Rightarrow \bigvee_i (\exists y_{i1} \ldots \exists y_{ip} . \beta_i(x_1, \ldots, x_k, y_{i1}, \ldots, y_{ip})). \tag{1}$$

That is, there is an environment (a mapping from variables to elements of $\mathbb{M}$) $\eta \equiv \{x_1 \mapsto e_1, \ldots, x_k \mapsto e_k\}$ such that $\alpha[\vec{e}]$ holds in $\mathbb{M}$ yet for no i does $\beta_i[\vec{e}]$ hold. The data (σ, η) determines a *chase-step*. We may execute such a chase-step and return a new model $\mathbb{N}$; this process proceeds as follows:

1. Choose some disjunct $\beta_i(x_1, \dots, x_k, y_{i1}, \dots, y_{ip})$
2. Choose new elements $k_1, \dots, k_{ip}$ from K and add them to the domain,
3. Add facts, that is, enrich the relations of $\mathbb{M}$, to ensure the truth of $\beta_i(e_1, \dots, e_k, k_1, \dots, k_{ip})$. Here we have slightly abused notation, since some atom in β_i may be an equality, in which case we must enrich the congruence relation to identify the appropriate elements of $|\mathbb{M}| \cup \{k_1, \dots k_{ip}\}$

A chase-step can be viewed as a database repair of the failure of the current model to satisfy the dependency expressed by σ.

There are three possible outcomes of a run of the Chase. (i) It may halt with success if we reach model $\mathbb{M}$ where we cannot apply a step, *i.e.* when $\mathbb{M} \models \mathcal{T}$. (ii) It may halt with failure, if there is a sentence $\alpha \Rightarrow \bot$ of $\mathcal{T}$ and we reach a model $\mathbb{M}$ in which some instance of α holds, (iii) It may fail to terminate.

Properties of the Chase Theorem 1 records the basic properties of the Chase. These results are adaptations of well-known [27, 30] results in database theory. A run of the Chase is said to be *fair* if—in the notation above—every pair of possible choices for σ and η will be eventually evaluated.

Theorem 1. *Let $\mathcal{T}$ be a geometric theory. Then $\mathcal{T}$ is satisfiable if and only if there is a fair run of the Chase, starting with the empty model, that does not fail. Let $\mathcal{U}$ be the set (possibly infinite) of models obtained by fair runs of the Chase. Then $\mathcal{U}$ is a set-of-support for Mod($\mathcal{T}$): for any model $\mathbb{M}$ of $\mathcal{T}$, there is a $\mathbb{U} \in \mathcal{U}$ and a homomorphism from $\mathbb{U}$ to $\mathbb{M}$.*

Note that the Theorem implies that the fair Chase is refutationally complete. If $\mathcal{T}$ has no models, then in any fair run of the Chase, each set of non-deterministic choices will eventually yield failure. By König's Lemma, then, the Chase process will halt.

Termination and Decidability In general, termination of the Chase for an arbitrary theory is undecidable [31]. However, Fagin *et al.* [30] define a syntactic condition on theories, known as *weak acyclicity*, by which the Chase is guaranteed to terminate. Briefly, one constructs a directed graph whose nodes are positions in relations and whose edges capture possible "information flow" between positions; a theory is weakly acyclic if there are no cycles of a certain form in this graph. (The notion of weakly acyclicity in [30] is defined for theories without disjunction, but the obvious extension of the definition to the general case supports the argument for termination in the general case.)

Observe that if $\mathcal{T}$ is such that all runs of the Chase terminate, then—by König's Lemma—there is a finite set of models returned by the Chase. Thus we can compute a finite set that jointly provides a set-of-support for all models of $\mathcal{T}$ relative to the homomorphism order $\preccurlyeq$.

Since weak acyclicity implies termination of the Chase we may conclude that weakly acyclic theories have the finite model property. Furthermore, entailment of positive-existential sentences from a weakly acyclic theory is decidable, as follows. Suppose $\mathcal{T}$ is weakly acyclic and α is a positive-existential sentence. Let $\mathbb{A}_1, \ldots, \mathbb{A}_n$ be the models of $\mathcal{T}$. To check that α holds in all models of $\mathcal{T}$ it suffices to test α in each of the (finite) models $\mathbb{A}_i$, since if $\mathbb{B}$ were a counter-model for α, and $\mathbb{A}_i$ the chase-model such that $\mathbb{A}_i \preccurlyeq \mathbb{B}$, then $\mathbb{A}_i$ would be a counter-model for α, recalling that positive-existential sentences are preserved by homomorphisms. This proof technique was used recently [32] to show decidability for the theory of a class of Diffie-Hellman key-establishment protocols.

The Bounded Chase For theories that do not enjoy termination of the Chase, we must resort to bounding our search. A traditional way to do so, used by tools such as Alloy, Margrave, and Aluminum, is to use user-supplied upper bounds on the domain of the model. Razor uses a somewhat more subtle device, which is outlined in [33], but which we cannot detail here for lack of space.

3.1 Augmentation: Exploring the Set of Models

Let $\mathcal{T}$ be a geometric theory and $\mathbb{M}$ be a model of $\mathcal{T}$. Razor allows the user to *augment* $\mathbb{M}$ with an additional positive-existential formula α resulting in an extension model $\mathbb{N}$ of $\mathcal{T}$ such that α is true in $\mathbb{N}$.

$\mathbb{N}$ can be computed by a run of the Chase starting with a model $\mathbb{M}' \equiv \mathbb{M} \cup \{\alpha\}$. A key point is that if α entails other observations given $\mathcal{T}$ and the facts already in $\mathbb{M}$, those observations will be added to the resulting model. And the augmentation may *fail* if adding α to $\mathbb{M}$ is inconsistent with $\mathcal{T}$.

Theorem 2. *Let $\mathbb{N}$ be a finite model of the theory $\mathcal{T}$. Suppose that $\mathbb{M}$ is a finite model returned by the Chase with $\mathbb{M} \preccurlyeq \mathbb{N}$. Then there is a finite sequence of augmentations on $\mathbb{M}$ resulting in a model isomorphic to $\mathbb{N}$.*

In particular, if $\mathcal{T}$ is weakly acyclic, then for every $\mathbb{N}$ there is a Chase model $\mathbb{M}$ and a finite sequence of augments of $\mathbb{M}$ yielding $\mathbb{N}$.

3.2 Provenance and the Witnessing Signature

A crucial aspect of our approach to constructing and reasoning about models is a notation for *witnessing* an existential quantifier.

Notation 3. *Given a sentence $\alpha \Rightarrow \bigvee_i (\exists y_{i1} \ldots \exists y_{ip} . \beta_i(\vec{x}, y_{i1}, \ldots, y_{ip}))$, we assign a unique, fresh,* witnessing (partial) function symbol $\mathsf{f}^{\sigma}_{\mathsf{ik}}$ *to each quantifier $\exists y_{ik}$. This determines an associated sentence $\alpha \Rightarrow \bigvee_i \beta_i(\vec{x}, \mathsf{f}^{\sigma}_{i1}(\vec{x}), \ldots, \mathsf{f}^{\sigma}_{ip}(\vec{x}))$ in an expanded signature, the* witnessing signature.

This is closely related to Skolemization of course, but with the important difference that our witnessing functions are partial functions, and this witnessing is not a source-transformation of the input theory. This alternate representation of geometric sentences allows us to define a refined version of the Chase, that maintains bookkeeping information about elements and facts. Specifically: *each element of a model built using the Chase will have a closed term of the witnessing signature associated with it: this is that element's "provenance."*

To illustrate, consider a chase-step as presented earlier, using a formula such as Formula 1, whose associated sentence over the witnessing signature is

$$\sigma^{\mathsf{w}} \equiv \alpha(x_1, \ldots, x_k) \Rightarrow \bigvee_i \beta_i(x_1, \ldots, x_k, \mathsf{f}^{\sigma}_{i1}(\vec{x}), \ldots, \mathsf{f}^{\sigma}_{ip}(\vec{x})) \tag{2}$$

In the chase-step, when $\vec{x}$ is instantiated by $\vec{e}$, the elements $k_1, \ldots, k_{ip}$ added in line 2 are naturally "named" by $\mathsf{f}^{\sigma}_{i1}(\vec{e}), \ldots, \mathsf{f}^{\sigma}_{ip}(\vec{e})$. Proceeding inductively, each of the e_j will be named by a closed term t_j, so that the elements $k_1, \ldots, k_{ip}$ added in line 2 have, respectively, the provenance $\mathsf{f}^{\sigma}_{i1}(\vec{t}), \ldots, \mathsf{f}^{\sigma}_{ip}(\vec{t})$.

It is possible that an element enjoys more than one provenance, in the case when a chase-step equates two elements.

Also observe that every fact added (in line 3) to the model being constructed can be "blamed" on the pair (σ, η), that is, the sentence and binding that fired the rule. *This is that fact's provenance.*

4 Implementation

A naive implementation of the Chase in our setting can be computationally prohibitive, due to the need to fork different branches of the model-construction in the presence of disjunctions. Instead, we take advantage of SAT-solving technology to navigate the disjunctions. The use of SAT-solving is of course the essence of MACE-style model-finding, but the difference here is that we do not simply work with the ground instances of the input theory, $\mathcal{T}$, over a fixed set of constants. Rather we compute a ground theory $\mathcal{T}^*$ consisting of a sufficiently large set of instantiations of $\mathcal{T}$ by closed terms of the witness signature for $\mathcal{T}$.

Since we want to handle theories with equality, we want to construct models of $\mathcal{T}^*$ modulo equality reasoning and the theory of uninterpreted functions, so we use an *SMT-solver*. We utilize Z3 (QF_UFBV) as the backend SMT-solver.

A straightforward use of SMT-solving would result in losing control over the model-building: even though the elements of a model of $\mathcal{T}^*$ returned would have appropriate provenance, the solver may make unnecessary relations hold between elements, and may collapse elements unnecessarily. So we follow the SMT-solving with a *minimization* phase, in which we eliminate relational facts that are not necessary, and even "un-collapse" elements when possible.

BuildModel (Algorithm 1) presents the overall process by which models of an input theory $\mathcal{T}$ are generated. The Ground procedure (line 2, given as Algorithm 3) consists of construction of ground instances of $\mathcal{T}$ by a run of a variation of the Chase, where every disjunction in the head of geometric sentences is replaced by a conjunction. In this way we represent all repair-branches that could be taken in a Chase step over the original $\mathcal{T}$. Such a computation creates a refined Skolem-Herbrand base, containing a set of *possible facts* $\mathbb{P}^{\mathcal{T}}$ that could be true in any any Chase model of $\mathcal{T}$. (Some care is required to handle contingent equalities; space does not permit a detailed explanation here.)

The *anonymization* procedure (line 3, not detailed here) constructs a flat theory $\mathcal{T}^{\mathsf{K}}$ by replacing every term in $\mathcal{T}^*$ over the witness signature with constants from a signature Σ^{K}. The theory $\mathcal{T}^{\mathsf{K}}$ is in a form that can be fed to the underlying model-finding and minimization algorithms by a call to NEXT (Algorithm 2). Finally, Razor returns the set models $\mathcal{U}$ produced by model-finding and minimization, reduced to the signature of the original input $\mathcal{T}$.

Algorithm 1 Razor

```
1: function BuildModel($\mathcal{T}$)                         ▷ $\mathcal{T}$ over signature $\Sigma$
2:     $(\mathcal{T}^*, \mathbb{P}^{\mathcal{T}}) \leftarrow$ Ground($\mathcal{T}$)            ▷ $\mathcal{T}^*$ over the witness signature $\Sigma^{\mathsf{w}}$
3:     $\mathcal{T}^{\mathsf{K}} \leftarrow$ Anonymize($\mathcal{T}^*$)            ▷ $\mathcal{T}^{\mathsf{K}}$ over the anonymized signature $\Sigma^{\mathsf{K}}$
4:     $\mathcal{U} \leftarrow \emptyset$
5:     $\mathbb{M} \leftarrow$ Next($\mathcal{T}^{\mathsf{K}}$, $\mathcal{U}$)
6:     while $\mathbb{M} \neq$ unsat do
7:         $\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathbb{M}\}$
8:         $\mathbb{M} \leftarrow$ Next($\mathcal{T}^{\mathsf{K}}$, $\mathcal{U}$)
9:     return Reduct($\mathcal{U}$)                    ▷ Reduct of models in $\mathcal{U}$ to $\Sigma$
```

Algorithm 2 Next Model

```
Require:
    $\mathcal{T}$ is ground and flat
    for all $\mathbb{U} \in \mathcal{U}$, $\mathbb{U} \models \mathcal{T}$ and $\mathbb{U}$ is homomorphically minimal

1: function Next($\mathcal{T}$, $\mathcal{U}$)
2:     $\Phi \leftarrow \bigcup_i \{$Flip$(\mathbb{U}_i)\}$ for all $\mathbb{U}_i \in \mathcal{U}$      ▷ Flip axioms about existing models.
3:     if exists $\mathbb{M}$ such that $\mathbb{M} \models (\mathcal{T} \cup \Phi)$ then        ▷ Ask the SMT-solver for $\mathbb{M}$.
4:         $\mathbb{N} \leftarrow$ Minimize($\mathcal{T}, \mathbb{M}$)
5:         return $\mathbb{N}$
6:     else
7:         return unsat                                  ▷ No more models.
```

The NEXT algorithm (Algorithm 2) accepts a set $\mathcal{U}$ of minimal models under the homomorphism ordering and returns a minimal model $\mathbb{M}$ for $\mathcal{T}$ that is not reachable from any of the models in $\mathcal{U}$ via homomorphism. The FLIP procedure (line 2, not detailed here) on an existing model $\mathbb{U} \in \mathcal{U}$ records the disjunction of the negation of all facts (including equational facts) true in $\mathbb{U}$: this guarantees that the next model returned by the solver will not be reachable from any of the models in $\mathcal{U}$ via homomorphism. The call to MINIMIZE (Algorithm 4) on line 4 reduces the next model returned by the solver to a homomorphically minimal one by repeated invocations of the solver. In every reduction step i, the solver is asked for a model $\mathbb{M}_i$ that satisfies

– the input theory $\mathcal{T}$.
– the *negation preserving* axiom of $\mathbb{M}_{i-1}$, which is the conjunction of all facts (including equational facts) that are false in $\mathbb{M}_{i-1}$.
– the *flip axioms* about the model $\mathbb{M}_{i-1}$ from the previous step.

Algorithm 3 Grounding

```
 1: function GROUND($\mathcal{G}$)
 2:     $\mathbb{P}^{\mathcal{G}} \leftarrow \emptyset$                          ▷ $\mathbb{P}^{\mathcal{G}}$ is initially the empty model
 3:     $\mathcal{G}^* \leftarrow \emptyset$                          ▷ $\mathcal{G}^*$ is initially an empty theory
 4:     repeat
 5:         choose $\sigma \equiv \varphi \Rightarrow \psi \in \mathcal{G}$
 6:         for each $\lambda$ where $\lambda\varphi \in \mathbb{P}^{\mathcal{G}}$ do
 7:             $\mathbb{P}^{\mathcal{G}} \leftarrow$ EXTEND($\mathbb{P}^{\mathcal{G}}$, $\sigma$, $\lambda$)
 8:             $\mathcal{G}^* \leftarrow \mathcal{G}^* \cup \{$INSTANTIATE$(\mathbb{P}^{\mathcal{G}}, \sigma, \lambda)\}$
 9:     until $\mathcal{G}^*$ and $\mathbb{P}^{\mathcal{G}}$ are changing
10:     return ($\mathcal{G}^*$, $\mathbb{P}^{\mathcal{G}}$)

11: function EXTEND($\mathbb{M}, \varphi \Rightarrow_{\vec{x}} \psi, \eta$)
12:     if $\psi = \perp$ then fail
13:     $\mathbb{N} \leftarrow \mathbb{M}$
14:     for each disjunct $\exists^{\mathsf{f_1}} y_1, \ldots, \exists^{\mathsf{f_m}} y_m. \bigwedge_{j=1}^{n} P_j$ in $\psi$ do
15:         $|\mathbb{N}| \leftarrow |\mathbb{N}| \cup \{[\![\mathsf{f_i}(\vec{x})]\!]^{\mathbb{M}}_{\eta} \mid 1 \leq i \leq m\}$
16:         $\mu \leftarrow \eta[y_1 \mapsto [\![\mathsf{f_1}(\vec{x})]\!]^{\mathbb{M}}_{\eta}, \ldots, y_m \mapsto [\![\mathsf{f_m}(\vec{x})]\!]^{\mathbb{M}}_{\eta}]$
17:         $\mathbb{N} \leftarrow \mathbb{N} \cup \{P_1[\mu(\vec{x}, \vec{y})], \ldots, P_n[\mu(\vec{x}, \vec{y})]\}$
18:         return $\mathbb{N}$

19: function INSTANTIATE($\mathbb{P}^{\mathcal{G}}$, $\varphi \Rightarrow_{\vec{x}} \bigvee_i \exists^{\mathsf{f_{i1}}} y_{i1} \ldots \exists^{\mathsf{f_{im}}} y_{im}.\psi_i$, $\eta$)
20:     $\mu \leftarrow \eta[y_{ij} \mapsto [\![\mathsf{f_{ij}}(\vec{x})]\!]^{\mathbb{P}^{\mathcal{G}}}_{\eta}]$ $(1 \leq j \leq m)$
21:     return $\mu\sigma$
```

Algorithm 4 Minimize

Require: $\mathbb{M} \models \mathcal{T}$

```
1: function MINIMIZE($\mathcal{T}$, $\mathbb{M}$)
2:     repeat
3:         $\mathbb{N} \leftarrow \mathbb{M}$
4:         $\mathbb{M} \leftarrow$ REDUCE($\mathbb{M}$)
5:     until $\mathbb{M} =$ unsat                          ▷ Cannot reduce
6:     return $\mathbb{N}$                          ▷ $\mathbb{N}$ is a minimal model for $\mathcal{T}$
```

Algorithm 5 Reduce

Require: $\mathbb{M} \models \mathcal{T}$

```
1: function REDUCE($\mathcal{T}$, $\mathbb{M}$)
2:     $\nu \leftarrow$ NEGPRESERVE($\mathcal{T}$ ,$\mathbb{M}$ )
3:     $\varphi \leftarrow$ FLIP($\mathcal{T}$ ,$\mathbb{M}$ )
4:     if exists $\mathbb{N}$ such that $\mathbb{N} \models \mathcal{T} \cup \{\nu \wedge \varphi\}$ then          ▷ Ask the SMT solver for $\mathbb{N}$
5:         return $\mathbb{N}$
6:     else
7:         return unsat                          ▷ $\mathbb{M}$ is minimal.
```

It can be shown that for every model $\mathbb{M}$, $\textsc{Reduce}(\mathcal{T}, \mathbb{M}) \prec \mathbb{M}$. The reduction process continues until the solver returns "unsatisfiable".

Theorem 4. *Fix a relational theory in geometric form $\mathcal{T}$. Let $\mathbb{P}^{\mathcal{T}}$ and $\mathcal{T}^*$ be a set of possible facts and its corresponding ground theory for $\mathcal{T}$, constructed by the Chase-based grounding algorithm. Let $\mathbb{M}$ be a model in the witness signature for $\mathcal{T}^*$ and $\mathbb{M}^-$ the reduct of $\mathbb{M}$ to the signature of $\mathcal{T}$.*

1. *(Soundness.) If $\mathbb{M} \models \mathcal{T}^*$ and $\mathbb{M}$ is homomorphically minimal, then $\mathbb{M}^-$ is a model of $\mathcal{T}$.*
2. *(Completeness.) If $\mathbb{M}$ is constructed by the Chase and $\mathbb{M}^- \models \mathcal{T}$ then $\mathbb{M}$ is a model of $\mathcal{T}^*$.*

Proof (Sketch). For (1): Let $\sigma \equiv \varphi \Rightarrow (\bigvee_i \exists^{\mathsf{f_{i1}}} y_{i1} \ldots \exists^{\mathsf{f_{ip}}} y_{ip} \,.\, \psi_i)$ be a sentence in $\mathcal{T}$. Let $\vec{x}$ be the free variables of σ. We show that if $\mathbb{M}^- \models_\eta \varphi$ for environment η, then $\mathbb{M}^- \models_\eta (\bigvee_i \exists^{\mathsf{f_{i1}}} y_{i1} \ldots \exists^{\mathsf{f_{ip}}} y_{ip} \,.\, \psi_i)$: because $\mathbb{M}$ is minimal, the facts in $\mathbb{M}$ are *contained* in $\mathbb{P}^{\mathcal{T}}$, and since φ is positive, $\eta\varphi \in \mathbb{P}^{\mathcal{T}}$. Therefore, by the construction of $\mathcal{T}^*$, a sentence $\varphi[\vec{t}] \Rightarrow \bigvee_i \psi_i[\vec{t}, \vec{u}_i]$ exists in $\mathcal{T}^*$ where $\vec{t} = \eta\vec{x}$, and for each u_{ij} in $\vec{u}_i$, $u_{ij} = \mathsf{f_{ij}}(\vec{t})$ $(1 \leq j \leq p)$. Observe that because $\mathbb{M}^- \models_\eta \varphi$ then $\mathbb{M} \models \varphi[\vec{t}]$ as $\vec{t}$ are witnesses for the elements that are images of $\vec{x}$ in η. Finally, since $\mathbb{M}$ is a model of $\mathcal{T}^*$, then $\mathbb{M} \models \psi_i[\vec{t}, \vec{u}_i]$ for some i. Therefore, it follows that $\mathbb{M}^- \models_\eta (\bigvee_i \exists y_{i1}, \ldots, \exists y_{ip} \,.\, \psi_i)$.

For (2): Let $\sigma^* \equiv \varphi[\vec{t}] \Rightarrow \bigvee_i \psi_i[\vec{t}, \vec{u}_i]$ be a sentence in $\mathcal{T}^*$. By definition, σ^* is an instance of a sentence $\sigma \equiv \bigvee_i \varphi \Rightarrow (\exists^{\mathsf{f_{i1}}} y_{i1} \ldots \exists^{\mathsf{f_{ip}}} y_{ip} \,.\, \psi_i)$ by a substitution that sends the free variables $\vec{x}$ of σ to $\vec{t}$ and $\vec{y}_i$ to $\vec{u}_i$. Moreover, for each u_{ij} in $\vec{u}_i$ $(1 \leq i \leq p)$, $u_{ij} = \mathsf{f_{ij}}(\vec{t})$.
Assume $\mathbb{M} \models \varphi[\vec{t}]$. Then, $\mathbb{M}^- \models_\eta \varphi$ where the environment η sends the variables in $\vec{x}$ to the elements $\vec{\mathbf{e}}$ in $\mathbb{M}^-$ that are denoted by $\vec{t}$ in $\mathbb{M}$. Because $\mathbb{M}^-$ is a chase-model for $\mathcal{T}$, then for some i, $\mathbb{M}^- \models_\lambda \exists y_{i1} \ldots \exists y_{ip} \,.\, \psi_i$ where $\lambda = \eta[y_{ij} \mapsto \mathbf{d_j}]$ $(1 \leq j \leq p)$. Let u_{ij} denote $\mathbf{d_j}$ in $\mathbb{M}$ under λ. Therefore, $\mathbb{M} \models \psi_i[\vec{t}, \vec{u}_i]$ follows.

It remains to show that a set-of-support computed by the minimization algorithm for $\mathcal{T}^*$ (modulo anonymization) is in fact a set-of-support for $\mathcal{T}$.

Theorem 5. *Fix a theory $\mathcal{T}$ in geometric form over a signature Σ. Let $\mathcal{T}^*$ be computed by a run of the grounding algorithm on $\mathcal{T}$.*

1. *The set $\mathcal{U}$ of models computed during Algorithm 1 is a set-of-support for $\mathcal{T}^*$.*
2. *The reducts $\mathcal{U}^-$ of $\mathcal{U}$ to Σ, returned by Algorithm 1, is a set-of-support for $\mathcal{T}$.*

Proof (Sketch). For (1): In every call of $\textsc{Next}$ for a set of models $\mathcal{U}_i$, the flip axioms about the models in $\mathcal{U}_i$ ensure that every model $\mathbb{M}$ returned satisfies $\mathbb{M} \not\preccurlyeq \mathbb{U}$ for each $\mathbb{U} \in \mathcal{U}$. If an $\mathbb{M}$ is returned by the solver, a model $\mathbb{U}$ with $\mathbb{U} \preccurlyeq \mathbb{M}$ will be added to $\mathcal{U}_i$.

For (2): Let $\mathbb{A}$ be a model of $\mathcal{T}$. By Theorem 1 a chase-model $\mathbb{M}$ over the witness signature exists such $\mathbb{M}^- \preccurlyeq \mathbb{A}$. By Theorem 4, part (2) $M \models \mathcal{T}^*$. By part (1) of this theorem there exists a model $\mathbb{U} \in \mathcal{U}$ such that $\mathbb{U} \preccurlyeq \mathbb{M}$. Therefore, for the reduct $\mathbb{U}^-$ of $\mathbb{U}$ to Σ, $\mathbb{U}^- \preccurlyeq \mathbb{A}$.

5 Examples

From the Alloy Repository Our main focus is on theories developed by hand. A natural source of such theories is the Alloy repository. We ran Razor on 11 theories from the Alloy book, suitably translated to Razor's input language. The following summarizes the experience; space precludes a detailed report.

For 6 theories Razor returned a complete set-of-support in unbounded mode; the time to return the first model was less than a second. For the remaining 5, we had to run in bounded mode in order to return models within 5 minutes. For 2 of these, iterative deepening succeed in finding a bound that was sufficient for finding a complete set of support for all finite models: the times-to-first-model were 375 msec and 17.2 sec, respectively. For the other 3, with respective bounds 1,2, and 3, we computed models quickly at the respective bounds but incrementing the bound led to a 5-minute timeout. In all cases, once the first model was found, subsequent models were completed in negligible time.

From TPTP We performed several experiments running Razor on the satisfiable problems in the TPTP problem repository [34] Razor's current performance on these problems is not satisfactory: it frequently fails to terminate within a five-minute bound. Razor tends to perform better on problems that are developed by hand, have a limited number of predicates, and don't include relations with high arity. Future developments in Razor's implementation will improve performance; a long-term research question is exploring the tradeoffs between efficiency and the kind of enhanced expressivity we offer.

Extended Example: Lab Door Security Here is an introductory example, demonstrating a specification in Razor of a simple policy for access to a our local lab, and typical queries about the policy. The sentences below capture the following policy specification.

Logic and Systems are research groups in lab (1-2). Research group members must be able to enter the lab (3). Key or card access allows a person to enter (4-5). To enter a lab, a member must have a key or card (6). Only members have cards (7). Employees grant keys to people (8-9). Systems members are not allowed to have keys (10).

```
 1. LabOf('Logic,'TheLab);
 2. LabOf('Systems, 'TheLab);
 3. MemberOf(p,r) & LabOf(r,l) => Enter(p,l);
 4. HasKey(p,k) & KOpens(k,l) => Enter(p,l);
 5. COpens(cardOf(p),l) => Enter(p,l);
 6. Enter(p,l) => COpens(cardOf(p),l)
               |  exists k. HasKey(p,k) & KOpens(k,l);
 7. COpens(cardOf(p), l) => exists r. MemberOf(p,r) & LabOf(r,l);
 8. HasKey(p,k) => exists e. Grant(e,p,k) & Employee(e);
 9. Grant(e,p,k)  => HasKey(p,k)
10. MemberOf(p,'Systems) & HasKey(p,k)
         & KOpens(k,'TheLab) => Falsehood;
```

The user can ask if a thief can access the lab without being Logic or Systems member:

```
Enter('Thief, 'TheLab);
```

First Model: Granted a Key. The first model exhibits that there is no policy restriction on employee key-granting capabilities:

```
Enter    = {(p1, l1)}          'TheLab  = l1
Employee = {(e1)}              'Systems = r2
Grant    = {(e1,p1,k1)}        'Logic   = r1
HasKey   = {(p1,k1)}           'Thief   = p1
KOpens   = {(k1,l1)}
LabOf    = {(r1,l1), (r2,l1)}
```

The user can investigate if the thief p1 and the employee e1 can be the same person in this example? This may be done by augmenting the model with aug p1 = e1. The augmentation results in one model (not shown).

Second Model: Third research group. The second example is more curious:

```
Enter    = {(p1, l1)}          'TheLab  = l1
COpens   = {(c1, l1)}          'Systems = r2
MemberOf = {(e1, r3)}          'Logic   = r1
cardOf   = {(p1, c1)}          'Thief   = p1
LabOf    = {(r1, l1), (r2, l1), (r3, l1)}
```

The user may ask “where did this third research group come from?”, then user look at the provenance information about r3 by running origin r3. Razor pinpoints an instance of the causal sentence (sentence 7):

```
7: COpens(c1, l1) => MemberOf(e1, r3) & LabOf(r3, l1)
```

This policy rule does not restrict which research groups live in the lab. Such a restriction would force the mystery group r3 to be the Systems or Logic group. The user confirms this policy fix by applying aug r3 = r2. The augmentation produces no counter examples; the fix is valid. The research group r3 exists because the thief has a card. By asking blame COpens(c1, l1), the user sees why:

```
6: Enter(p1, l1) => COpens(c1, l1)
                 |  HasKey(p1, k1) & KOpens(k1, l1)
```

The thief has a card because the user's query said he could enter the lab. He could also have a key, which is evident in the first model. Why does the thief belong to a research group in this scenario, but not in the previous? Being a research group member is a consequence of having a card; not for having a key. Belonging to a research group when having a key is extraneous information. Razor does not include this scenario in the minimal model returned.

Software-Defined Networks At Razor's web page `http://salmans.github.io/Razor/` one can find a more advanced extended example, showing how Razor can reason about controller programs for Software-Defined Networks. For a program P in the declarative networking program Flowlog [35], we show how to define a theory $\mathcal{T}_P$ such that a model $\mathbb{M}$ of $\mathcal{T}_P$ is a snapshot of the state of the system at a moment in time. The user can augment $\mathbb{M}$ by a fact capturing a network event, and the resulting models correspondingly capture the next state of the system. In this way, *augmentation acts as a stepper in a debugger.*

6 Future Work

Highlights of the ongoing work on this project include (i) work on efficiency of the model-building, (ii) taking real advantage of the fact that we incorporate an SMT-solver, to work more effectively when part of a user's input theory has a known decision procedure, and (iii) an improved user interface for the tool, including a more sophisticated GUI for presenting models, and parsers to allow input in native formats such as Description Logic, firewall specifications, XAMCL, and cryptographic protocols.

Acknowledgements We benefitted from discussions with Henning Günther, Joshua Guttman, Daniel Jackson, Shriram Krishnaturthi, Tim Nelson, and John Ramsdell. The name of our tool is homage to Ockham's Razor (William of Occam 1285-1349): "Pluralitas non est ponenda sine neccesitate."

References

1. Zhang, J., Zhang, H.: SEM: a system for enumerating models. In: International Joint Conference On Artificial Intelligence. (1995)
2. McCune, W.: MACE 2.0 Reference Manual and Guide. CoRR (2001)
3. Claessen, K., Sörensson, N.: New Techniques that Improve MACE-Style Finite Model Finding. In: CADE Workshop on Model Computation-Principles, Algorithms, Applications. (2003)
4. Baumgartner, P., Schmidt, R.A.: Blocking and Other Enhancements for Bottom-Up Model Generation Methods. In: IJCAR. (2006)
5. Nivelle, H.D., Meng, J.: Geometric Resolution: A Proof Procedure Based on Finite Model Search. In: IJCAR. (2006)
6. Torlak, E., Jackson, D.: Kodkod: A Relational Model Finder. In: Tools and Algorithms for the Construction and Analysis of Systems. (2007)
7. Baumgartner, P., Fuchs, A., De Nivelle, H., Tinelli, C.: Computing Finite Models by Reduction to Function-Free Clause Logic. Journal of Applied Logic (2009)
8. Reynolds, A., Tinelli, C., Goel, A., Krstic, S.: Finite Model Finding in SMT. In: Int. Conf. Computer Aided Verification. (2013)
9. Korovin, K., Sticksel, C.: iProver-Eq: An Instantiation-Based Theorem Prover with Equality. In: IJCAR. (2010)
10. Bry, F., Torge, S.: A deduction method complete for refutation and finite satisfiability. In Dix, J., del Cerro, L.F., Furbach, U., eds.: Logics in Artificial Intelligence, European Workshop, JELIA '98, Dagstuhl, Proceedings. Volume 1489 of Lecture Notes in Computer Science., Springer (1998) 122–138

11. Baumgartner, P., Suchanek, F.M.: Automated reasoning support for first-order ontologies. Principles and Practice of Semantic Web Reasoning (2006) 18
12. Jackson, D.: Software Abstractions. 2 edn. MIT Press (2012)
13. Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and Change-Impact Analysis of Access-Control Policies. In: Int. Conf. Soft. Eng. (2005)
14. Nelson, T., Barratt, C., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: The Margrave Tool for Firewall Analysis. In: USENIX Large Installation System Administration Conference. (2010)
15. Niemelä, I.: A Tableau Calculus for Minimal Model Reasoning. In: Workshop on Theorem Proving with Analytic Tableaux and Related Methods. (1996)
16. Bry, F., Yahya, A.: Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation. J. Automated Reasoning (2000)
17. Nelson, T., Saghafi, S., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Aluminum: Principled Scenario Exploration Through Minimality. In: Int. Conf. Soft. Eng. (2013)
18. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Searching for shapes in cryptographic protocols. In: Tools and Algorithms for the Construction and Analysis of Systems. (2007)
19. Lopes, N., Bjorner, N., Godefroid, P., Jayaraman, K., Varghese, G.: Checking beliefs in dynamic networks. Technical report, Microsoft Research (2014)
20. De Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Tools and Algorithms for the Construction and Analysis of Systems. (2008)
21. Abramsky, S.: Domain Theory in Logical Form. Ann. Pure Applied Logic (1991)
22. Vickers, S.: Geometric Logic as a Specification Language. In: Imperial College Department of Computing Workshop on Theory and Formal Methods. (1995)
23. Sofronie-Stokkermans, V.: Sheaves and Geometric Logic and Applications to Modular Verification of Complex Systems. Electronic Notes on Theoretical Computer Science (2009) 161–187
24. Bezem, M., Coquand, T.: Automating Coherent Logic. In: Logic for Programming, Artificial Intelligence, and Reasoning. (2005)
25. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing Implications of Data Dependencies. ACM Trans. Database Systems (1979)
26. Beeri, C., Vardi, M.Y.: A Proof Procedure for Data Dependencies. J. ACM (1984)
27. Deutsch, A., Tannen, V.: XML Queries and Constraints, Containment and Reformulation. ACM Symposium on Theory Computer Science (2005)
28. Rossman, B.: Existential Positive Types and Preservation under Homomorphisms. In: IEEE Logic in Computer Science, IEEE (2005)
29. Makkai, M., Reyes, G.E.: First Order Categorical Logic. Springer (1977)
30. Fagin, R., Kolaitis, P., Miller, R., Popa, L.: Data Exchange: Semantics and Query Answering. Database Theory ICDT (2002)
31. Deutsch, A., Nash, A., Remmel, J.: The Chase Revisited. In: ACM Symposium on Principles of Database Systems. (2008)
32. Dougherty, D.J., Guttman, J.D.: Decidability for Lightweight Diffie-Hellman Protocols. In: IEEE Symposium on Computer Security Foundations. (2014) 217–231
33. Saghafi, S., Dougherty, D.J.: Razor: Provenance and exploration in model-finding. In: 4th Workshop on Practical Aspects of Automated Reasoning (PAAR). (2014)
34. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. J. Automated Reasoning (2009)
35. Nelson, T., Ferguson, A.D., Scheer, M., Krishnamurthi, S.: Tierless programming and reasoning for software-defined networks. NSDI, Apr (2014)