Geometric Logic for Policy Analysis*

Salman Saghafi, Tim Nelson and Daniel J. Dougherty

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609, USA

Abstract

We describe a new computational engine for model-finding and its application to security policy analysis. We evaluate a preliminary implementation of our algorithm by comparing with a mature tool, the Margrave Policy Analyzer, with respect to performance and quality of output.

1 Introduction

Modern software systems are complex hives of policy configuration. There are access-control policies that govern high-level data access; filesystem and operating system-level policies that govern access to protected resources; firewall policies that govern the flow of packets; and many more. Somehow, all these policies must coordinate to achieve the high-level security goals of an organization. Not surprisingly, it is difficult for administrators to obtain a global view of these pieces; harder still to combine these into a coherent whole; and even more problematic to reason about their composition.

Of course, policy authors make mistakes: rules can have unintended consequences and programs and policies can interact in ways that an author didn't intend. The problem of reasoning about policies is exacerbated by the means of their construction. They are usually authored not through some formal process but by manual edits, usually in response to some pressing need. As a result, well-intentioned actions often lead to configuration errors.

For some years now, colleagues at WPI and Brown University have been developing a policy-analysis tool called Margrave. Two aspects of the tool are central to the work reported here:

- Margrave is based on *first-order logic*, which provides an expressive foundation for capturing both policies and queries, and is well-suited to reasoning across multiple interacting policies, and
- its fundamental analysis technique is *model finding*, which presents users with concrete witnesses to queries. We like to refer to models as *scenarios*, to be somewhat more evocative for our users (whom we do not assume to be logicians!)

The emphasis on building models/scenarios rather than proofs is a central tenet of our overall project.

Our first-order approach to the foundation but our current implementation is based on SAT-solving and has certain limitations. In this paper we describe a new approach to analysis based on *geometric logic*. The key expected benefit of this approach is that it will allow us to produce *a better selection of models* to present to the user, and permit better organized *exploration* of the space of models of a theory. Our sense of "better selection of models" will be described formally below but for now we can say that the goal is to present models that are "minimal" in the sense that they do not realize any atomic facts that are unnecessary in order that the model satisfy the user's query.

We do not expect that staying within first-order logic (as opposed to reducing to SAT-solving) will be without a performance penalty. For us the interesting research question is to explore and balance the tradeoff between (i) the expressive power gained by working in the first-order geometric fragment and (ii) the efficiency of SAT-solving.

This paper constitutes a (very) preliminary report on the state of our project.

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

1.1 Related Work

Finite model-finding for first-order logic is a powerful technique that has been applied to various domains such as software modeling [Jac06], policy analysis [NBD+10] and protocol analysis. Rather than attempt a comprehensive survey of the state of the art of model finding we simply describe the main approaches to the problem by way of giving context for our contrasting approach.

The two primary classes of finite model-finding methods are *MACE-style* model-finding and *SEM-style* model-finding. MACE-style model-finding, named after the model-finder MACE, was first suggested by William McCune [McC01]. The idea of SEM-style model-finding was first developed by Jian Zhang and Hantao Zhang and was implemented as Falcon and its successor SEM [ZZ95]. The key idea behind MACE-style model-finding is to translate the input first-order formula to a set of ground propositional clauses followed by SAT-solving on the propositional clause set. SEM-style model-finders, on the other hand, directly build models for the first-order formula based on a backtracking search strategy [Tam03].

Kodkod is a well-known example of MACE-style model-finders, which accepts *multi-sorted* first-order formulas with *t*ransitive closure [TJ07]. The primary features of Kodkod are: (1) support for partial models, (2) a mechanism for detecting and breaking symmetric models (isomorphism elimination), and (3) efficient translation from first-order logic to propositional logic. Margrave currently uses Kodkod as its underlying model-finding engine.

2 Policy Analysis

Margrave is an interactive tool that produces concrete scenarios in response to user queries. Queries cover not only conventional verification-style properties, but also "what if" style questions about the semantic impact of edits. Margrave allows users to specify policies and, optionally, properties of the environment. It then processes user-defined queries about the policy and presents scenarios as output. Scenarios are, informally, snapshots of the system—as governed by the policy—in which the query holds. User interaction takes place via a read-eval-print loop, with an associated command language, that mediates the display of scenarios.

Margrave has been in active development for six years. It originally targeted XACML-style access control policies, and was implemented using BDDs. The current version designed for richer configuration policies, using first-order predicate logic, has been successful in analyzing firewalls [NBD+10]. The chief novelty of Margrave's current approach lies in embracing the richness of full first-order predicate logic for specifying policies, systems, and queries. Here we outline the current state of Margrave, emphasizing what we have learned about the benefits and challenges associated with working in such an powerfully expressive setting.

2.1 Exploring a Conference Manager Policy

Representing Policies The fragment in figure 1 shows two rules: the first permits those assigned to a paper to read the paper; the first denies reviewers from reading reviews for papers with which they are conflicted. Each rule bears a name (such as PaperAssigned), a decision (permit or deny) over a set of request variables (s, a, r denoting subject, action, and resource, respectively), and a set of conditions (action a is ReadPaper and subject s is conflicted with resource r). Conditions may reference both the request variables and additional variables as needed. Concepts such as conflicted are specific to conference management. Margrave captures domain ontologies through *vocabularies* that are separate from policies. In this example, ConfVocab (first line) references the relevant vocabulary (not shown here). The vocabulary declares the sorts in the policy (subject, action, resource, paper, reviewer, *etc.*), provides

```
(Policy uses ConfVocab
    (Rules
     (PaperAssigned = (permit s a r) :- (and (assigned s r) (ReadPaper a)))
     (PaperConflict = (deny s a r) :- (and (conflicted s r) (ReadPaper a)))
     (PaperNoConflict = (permit s a r) :- (and (not (conflicted s r)) (ReadPaper a)))
     ...
    ))
```

Figure 1: Conference policy (ConfPol) in Margrave's language.

signatures for relations in this domain (such as conflicted and assigned), and captures constraints on the sorts (such that every subject must lie in one of the defined roles). Vocabularies also define the decisions allowed in a policy; a user could, for example, use separate decisions for "deny" and "deny and log", trusting the conference-manager software to treat these differently. We revisit the conference example in section 6.

Generating and Exploring Scenarios Given that combinators may override individual rules and that rules may interact through overlapping roles, policy effects are not always as straightforward as the individual rules might suggest. As a result, policy authors should sanity-check their policies. Imagine that our conference-policy author wants to check on the permissions granted to reviewers who are conflicted with papers. The following Margrave command achieves this:

>EXPLORE conflicted(s, r) **and** permit(s, a, r) UNDER ConfPol;

Margrave computes a set of scenarios that witness the given query; this set is exhaustive when possible. The policy author then issues further commands to display the results; options include asking whether there are any scenarios, asking for a single scenario, or asking for all generated scenarios. In this example, a request for a single scenario would yield a scenario that shows that the rules permit a reviewer to submit a review for a paper with which s/he is conflicted. This is disconcerting: the policy author can issue a subsequent query to find out which policy rules were responsible for generating this scenario. In this case, that command would report that the PaperAssigned rule rendered the decision. Had multiple rules applied, Margrave would report which rules applied, and which rule took precedence under the decision combinators. Note that instead of requiring its users to provide a formal property, as conventional verification tools would, Margrave supports querying to explore scenarios of a specification interactively. In addition, Margrave supports other property-free forms of analysis.

3 Policy Foundations

Here we give an outline of our notion of "policy", in particular the representation of policies as certain first-order theories. A fuller account can be found in [DFK06] and [Gia12].

Here, "policy" means a monitoring policy, for example as in access control. Such a policy, when deployed, constitutes a function that consumes a request and some information about the environment and returns a decision. Let Dec be the set of decisions (just some set of identifiers like permit, deny, etc), Req the set of requests (it doesn't matter for now what the requests look like), and Env the type of information about the environment that the policy needs.

A document $\mathcal{P}$ intended to implement a policy might not be well-behaved enough to define a *function* from requests and environments to decisions: (i) it may not have enough information in itself to decide each request (*e.g.* because it is part of a suite of documents cooperating to make a policy), so it is

"partial" in general, and (ii) and it might not be consistent, *i.e.* it might compute more than one decision for a given request in a given environment. So we should view it as defining a relation:

$$\mathcal{P} : (\mathsf{Req} \times \mathsf{Env}) \to 2^{\mathsf{Dec}}$$

Note that via a natural isomorphism with the above we can equivalently take the perspective

$$\mathcal{P} : \mathsf{Env} \to (\mathsf{Dec} \to 2^{\mathsf{Req}})$$

in which a policy is a way to, given an environment, determine which requests are assigned to each decision. In this reading a decision is a predicate over requests, determined by a policy and an environment. In other words we think of each decision as being a unary predicate over requests. This leads to writing $\mathsf{dec}(\mathsf{q})$ to say that request q gets decision dec.

We take the environment E to be a first-order model over a certain signature Σ, and the decisions be predicate symbols Dec distinct from those in Σ. Viewed in this light, policies are maps that when given a model of Σ, produce an expansion of that model to a signature augmented with predicates Dec.

So when a user is—informally—exploring which "scenarios," if any, are consistent with a certain configuration of policy, environment, and decisions, he/she is indeed asking about the existence of certain first-order models. This is the essential connection between model-finding and policy analysis.

4 Geometric Logic

SAT solving is a well-established technique for model finding, especially for theories that are known to have the finite-model property. That said, however, SAT solving allows only limited possibilities for controlling the space of models to be constructed. In our setting, where the collection of models/scenarios satisfying a query can be too large for a user to conveniently grasp as a whole, it is crucial to have a principled criterion for which models to display and to have a model-finding technique consistent with this criterion.

The essence of our new foundational engine for scenario generation comprise the notions of *observable property* and an ordering $\preceq$ on models that captures the idea of one model satisfying all the observable properties of another. While this engine is not specific to policy reasoning, we adapt it for a logical fragment that covers policies as represented in Margrave.

Observable properties First-order formulas built up from atomic formulas using only $\wedge$, $\vee$ and $\exists$ are called *geometric* [Abr91]. It is easy to see that if $\alpha(\vec{x})$ is a geometric formula true of a tuple $\vec{a}$ in a model $\mathcal{A}$ then (i) the truth of this fact is witnessed by a finite fragment of $\mathcal{A}$, and (ii) if $\mathcal{A}$ is expanded, by adding new elements and/or new facts, $\alpha(\vec{x})$ still holds of $\vec{a}$ in the resulting model. For this reason, properties defined by positive-existential formulas are sometimes called *observable* properties.

Geometric formulas are known to be preserved under homomorphisms. This induces an ordering on models based on their observable properties; in particular, minimal models under this ordering omit facts that are not necessary to witness the query that yielded the models. A "complete" set of answers for a query σ is a set $\mathcal{S}$ of models of σ that cover the space of answers in the sense that for any model $\mathcal{B}$ of σ there is a model $\mathcal{A} \in \mathcal{S}$ with $\mathcal{A} \preceq \mathcal{B}$. It is natural to call such a set $\mathcal{S}$ a *set of support* for the models of σ.

Combining these intuitions, minimality and set-of-support, we adopt the slogan

> *For a given query σ the ideal set of outputs is a class of models which is a minimal set of support for the class of all models of σ.*

This is a theoretical ideal in the sense that for an arbitrary query there is no reason to think that there is a finite set of finite models comprising such a set of support, not to mention the problem of computing such a set. But it is good to have ideals. And indeed, in many cases there is such a nice set, attainable in theory and approximable in practice. Here is where *geometric logic* plays a role.

Summarizing:

Definition 1.

- A *geometric formula* is a formula with only $\wedge$, $\vee$, $\exists$ and $=$ as connectives. Infinitary disjunction $\bigvee_i$ is permitted.
- A *geometric sequent* is a construct $\varphi \vdash_{\vec{x}} \psi$ where φ and ψ are geometric formulas that contain free variables from a set of variables $\vec{x}$. A geometric sequent $\varphi \vdash_{\vec{x}} \psi$ can be viewed as a shorthand for the formula $\forall x_1.x_2,\ldots,x_n.(\varphi \rightarrow \psi)$ where $\vec{x} = \{x_1, x_2, \ldots, x_n\}$. We refer to φ and ψ respectively as *left* and *right* of $\varphi \vdash_{\vec{x}} \psi$.
- A *geometric theory* is a *vocabulary* $\mathcal{V}$ together with a finite set of *geometric sequents* $\mathcal{T}$ over $\mathcal{V}$.
- A set $\mathcal{U}$ of models is a set of *jointly universal models* for a theory $\mathcal{T}$ if and only if:
  - For every $\mathbb{M} \in \mathcal{U}$ then $\mathbb{M} \models \mathcal{T}$.
  - For every model $\mathbb{N} \models \mathcal{T}$, there exists a model $\mathbb{M} \in \mathcal{U}$ such that $\mathbb{M} \preceq \mathbb{N}$.

Thus if $\mathcal{U}$ is jointly universal for $\mathcal{T}$, every model of theory $\mathcal{T}$ is in the homomorphism cone of at least one model $\mathcal{U}$.

4.1 Policies, Queries, and Geometric Logic

The following is an empirical observation, not surprising in light of the discussion above about observable properties and finite evidence.

> *Security policies tend to be geometric theories; security goals such as safety and authentication tend to be geometric formulas. Axiomatizations of environment constraints tend naturally to be geometric theories. As such, verifying security goals in the context of polices can be often be reduced to searching for models for geometric theories.*

5 The Chase: A New Model-Finding Engine

The chase is a model-finding algorithm for finding a set of jointly universal models for geometric theories. The chase was introduced by Fagin and his colleagues for solving *data-exchange* problems [FKMP05]. Within the context of a data-exchange problem, dependencies between a source schema and a target schema together with additional constraints on the target schema are described as a geometric theory. The chase is then employed to extend an instance over the source to an instance over the target such that the dependencies are satisfied. The algorithm suggested by Fagin and his colleagues assumes that the geometric sequents describing data-exchange dependencies do not contain disjunctions; therefore, the chase will never return more than a single model if it succeeds. Later on, Deutch and his colleagues introduced a new version of the chase that allowed disjunctions on the right of sequents; consequently, their algorithm was capable of finding a set of jointly universal instances for a data-exchange problem [DNR08].

The Algorithm Given a geometric theory $\mathcal{T}$ as input, the chase repeatedly uses a sub-procedure, the *chase step* to build a model for $\mathcal{T}$ incrementally. We first introduce the chase step in algorithm 1; then, we describe the classical chase in algorithm 2.

The *chase* consists of starting with the empty set of facts and iterating the above process. For each iteration, the chase chooses a sequent that is not true in the current model and updates the model in a chase step (lines 4 and 5 of Algorithm 2). Given an input sequent $\varphi \vdash_x \psi$ and a model $\mathbb{M}$ where φ is true in $\mathbb{M}$ but ψ is not, a chase step makes one of the disjuncts in ψ true by adds new facts to $\mathbb{M}$ (lines 4 to 6 of Algorithm 1), then returns the updated model (lines 7 and 8 of Algorithm 1). We halt with success if we reach a 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 sequent 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 sequent true. It is possible that the chase may not halt. In this case, if the chase is done in a "fair" manner the resulting infinite set of facts will be a model of T.

Termination In general, termination of the chase for an arbitrary geometric theory is undecidable. However, it is guaranteed that for a class of geometric theories, known as *weakly acyclic*, the chase will always terminate [DNR08].

Theorem 2. *Let T be geometric.*

- *T is satisfiable if and only if there is a fair run of the chase which does not fail.*
- *Let $\mathcal{U} = \mathcal{M}_1, \mathcal{M}_2, \ldots$ be the set of models obtained by some execution of the chase. The $\mathcal{U}$ is jointly universal for $\mathcal{T}$.*
- *If $\mathcal{T}$ is weakly acyclic then $\mathcal{T}$ has a finite jointly universal set of models.*

Algorithm 1 Chase Step

Require: $\mathbb{M} \models_\eta \varphi$, $\mathbb{M} \not\models_\eta \psi$

1: **function** CHASESTEP($\mathbb{M}, \varphi \vdash_x \psi, \eta$)
2: **if** $\psi = \bot$ **then** fail
3: $\mathbb{M}' \leftarrow \mathbb{M}$
4: choose disjunct $E \wedge \exists y_1, \ldots, y_m. \bigwedge_{j=1}^{n} P_j \in \psi$
5: $|\mathbb{M}'| \leftarrow (|\mathbb{M}'| \cup \{d_1, \ldots, d_m\})$ ▷ each d_i is a fresh element
6: $\eta' \leftarrow \eta_{[y_1 \mapsto d_1, \ldots, y_m \mapsto d_m]}$
7: $\mathbb{M}' \leftarrow (\mathbb{M}' \cup \{P_1(\eta'(\vec{x} \cup \vec{y})), \ldots, P_n(\eta'(\vec{x} \cup \vec{y}))\})$
8: **return** $\mathbb{M}'$

Algorithm 2 Chase

1: **function** CHASE($\mathcal{T}$)
2: $\mathbb{M} \leftarrow \emptyset$ ▷ start with an empty model over an empty domain
3: **while** $\mathbb{M} \not\models \mathcal{T}$ **do**
4: choose $\varphi \vdash_{\vec{x}} \psi \in \mathcal{T}, \eta : \vec{x} \longrightarrow |\mathbb{M}|$ s.th. $\mathbb{M} \not\models \varphi \vdash_{\eta\vec{x}} \psi$
5: $\mathbb{M} \leftarrow$ CHASESTEP($\mathbb{M}, \varphi \vdash_{\vec{x}} \psi, \eta$)
6: **return** $\mathbb{M}$

```
PaperAssigned = assigned(s,r) & ReadPaper(a) & Paper(r) => permit(s,a,r)
PaperConflict = conflicted(s,r) & ReadPaper(a) & Paper(r) => deny(s,a,r)
PaperNoConflict = ReadPaper(a) & Paper(r) & Author(s) => permit(s,a,r) | conflicted(s,r)
```

Figure 2: Conference policy in geometric logic.

6 Preliminary Results

We have built a prototype implementation of the chase in Haskell. Our current emphasis is not on run-time efficiency but rather on generating jointly universal sets of models in order to compare this output with output from standard model-finders such as Kodkod. We did this by running queries on some sample policies, both in Margrave (which uses Kodkod as its model-finder) and under our chase-based implementation.

We used three examples from Margrave's distribution, *phone*, *conference* and a modified version of *firewall* as specifications for our experiments and converted the examples from Margrave's language to equisatisfiable geometric theories manually. During the translation process, we removed the constraints in Margrave's specifications and felt free to introduce additional relations in the translated geometric theories in order to maintain consistency between the two sets of specifications.

Conference Example For the first experiment, we used a restricted version of the *conference* policy introduced in Figure 1. Figure 2 is a (partial) translation of the conference policy in geometric logic.

The translation of the PaperAssigned and PaperConflicted rules is trivial. However, because of the positivity of geometric logic, the sequent corresponding to PaperNoConflict rule introduces a disjunction on right. Intuitively, the geometric version of PaperNoConflict states that either an author s is permitted to read paper r or the author and the paper are conflicted.
Here, we are interested in scenarios in which at least a subject is permitted to perform any action on any resource:
>EXPLORE permit(s, a, r) UNDER ConfPol;

The query translates to the following geometric sequent:
>exists s.exists a.exists r.permit(s,a,r)

The chase-based model finder may simply answer this query with a model in which only one fact, permit(Author#1, ReadPaper#1, Paper#1) for some subject, action and resource is true. In order to prevent the chase-based model finder from returning such a trivial model, the preconditions of the PaperAssigned and PaperNoConflict rules must be forced in the model. This can be done by adding extra sequents to the theory, which turn these two rules into a bi-implication:
>permit(s,a,r) =>(ReadPaper(a) & Paper(r) & Author(s) & notConflicted(s,r))
|(assigned(s,r) & ReadPaper(a) & Paper(r))

>(conflicted(s,r) & notConflicted(s,r)) =>⊥

Notice that we introduce a helper relation notConflicted for to denote the negation of conflicted.

Results Figure 3 compares the outputs of the chase-based model finder and those of Margrave. The chase-based implementation computed precise models in running times that were comparable to those of Margrave and produced far fewer models for *phone* and *firewall*. The restriction to a set of jointly

Spec.	Margrave			Chase		
	# models	min.size	max. size	# models	min.size	max. size
Phone	66	3	4	4	4	4
Conference	3	3	3	3	3	3
Firewall	≥ 1000	2	≥ 3	4	4	4

Figure 3: Models constructed by Margrave and the chase-based model finder.

universal models has the effect of pruning the space of answers returned dramatically. Further investigation, however, showed that Margrave and the chase-based implementation constructed identical models for *conference* due to the constraints in the specification, which forced Margrave to produce only minimal models.

We also observed that the models produced by the chase-based model finder are homomorphically minimal; that is, they do not contain extra facts that are not necessary for satisfying the input theory. However, those models are not minimal with respect to the size of their domains. For instance, Margrave finds models of size 3 for the *Phone* example and models of size 2 and 3 for the *Firewall* example whereas the models produced by the chase-based implementation are of sizes 4 and 4 respectively. Margrave finds models of the smaller sizes by collapsing the elements of the model; however, the chase creates a new element of every existential quantifier.

7 Conclusion and Future Work

We presented a chase-based model-finding strategy in geometric logic as a new approach to model-finding. Our preliminary studies show that our method can be applied in access control policy analysis, where both access control policy rules and the queries are specified as geometric theories. The emphasis of our new strategy is on producing minimal models, models that only contain what is necessary for satisfying the input specification. We are currently working on implementing more efficient data-structures and algorithms to improve the performance of our current implementation.

References

[Abr91] Samson Abramsky. Domain theory in logical form. *Ann. Pure Appl. Logic*, 51(1-2):1–77, 1991. Second Annual IEEE Symposium on Logic in Computer Science (Ithaca, NY, 1987).

[DFK06] Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In *3rd International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *Lecture Notes in Computer Science*, pages 632–646, 2006.

[DNR08] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

[FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[Gia12] Theophilos Giannakopoulos. Multi-decision policy and policy combinator specifications. Master's thesis, Worcester Polytechnic Institute, 2012.

[Jac06] Daniel Jackson. *Software Abstractions - Logic, Language, and Analysis*. MIT Press, 2006.

[McC01] William McCune. Mace 2.0 reference manual and guide. *arXiv preprint cs/0106042*, 2001.

[NBD+10] Timothy Nelson, Christopher Barratt, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. The Margrave tool for firewall analysis. In *Proceedings of the 24th USENIX Large Installation System Administration Conference (LISA 2010)*, 2010.

[Tam03] T. Tammet. Finite model building: improvements and comparisons. In *CADE-19, Workshop W*, volume 4, 2003.

[TJ07] E. Torlak and D. Jackson. Kodkod: A relational model finder. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 632–647, 2007.

[ZZ95] Jian Zhang and Hantao Zhang. SEM: a system for enumerating models. In *IJCAI*, pages 298–303. Morgan Kaufmann, 1995.