2015-04-30

A Framework for Exploring Finite Models

Salman Saghafi
Worcester Polytechnic Institute

A Framework for Exploring Finite Models

by

Salman Saghafi

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Computer Science

May 2015

APPROVED:

Professor Daniel J. Dougherty, Thesis Advisor, WPI Computer Science

Professor Craig Wills, Head of Department, WPI Computer Science

Professor Joshua Guttman, WPI Computer Science

Professor George Heineman, WPI Computer Science

Doctor John D. Ramsdell, The MITRE Corporation

A Framework for Exploring Finite Models

by

Salman Saghafi

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Computer Science

May 2015

APPROVED:

_____

Professor Daniel J. Dougherty, Thesis Advisor, WPI Computer Science

_____

Professor Craig Wills, Head of Department, WPI Computer Science

_____

Professor Joshua Guttman, WPI Computer Science

_____

Professor George Heineman, WPI Computer Science

_____

Doctor John D. Ramsdell, The MITRE Corporation

Abstract

This thesis presents a framework for understanding first-order theories by investigating their models. A common application is to help users, who are not necessarily experts in formal methods, analyze software artifacts, such as access-control policies, system configurations, protocol specifications, and software designs. The framework suggests a strategy for exploring the space of finite models of a theory via augmentation. Also, it introduces a notion of provenance information for understanding the elements and facts in models with respect to the statements of the theory.

The primary mathematical tool is an information-preserving preorder, induced by the homomorphism on models, defining paths along which models are explored. The central algorithmic ideas consists of a controlled construction of the Herbrand base of the input theory followed by utilizing SMT-solving for generating models that are minimal under the homomorphism preorder. Our framework for model-exploration is realized in Razor, a model-finding assistant that provides the user with a read-eval-print loop for investigating models.

Acknowledgements

I would like to extend several acknowledgements to my learning community, family, and friends. This dissertation would not have been possible with the support and assistance of people with whom I am connected professionally and those with whom I have shared my personal life.

First, I would like to thank my family in Iran and England. Since childhood each of you have cultivated my love for learning and, by extension, computers. Mom and Dad, you provided me with ample opportunities to learn, grow, and prosper academically and personally. My siblings, too, have been instrumental in providing me with a compassionate listening ear and a cheering squad as I pursued my dreams in England and America. Lastly, I would like to extend my thanks and love to Cammy, my partner from day one in the doctoral program. We began our programs together and, rightfully so, we are ending our programs together. I love you.

Secondly, I would like to thank my major advisor, Dan Dougherty. Dan encouraged my interest in integrating my appreciation for logic with my background in software engineering. Also, this dissertation work would not have been possible without the continuous commitment and dedication of my associate advisors, Joshua Guttman, John Ramsdell and George Heineman. I am grateful to Kathi Fisler and Shriram Krishnamurthi for inviting me to WPI from Iran to pursue further study in computer science research. Additionally, Tim Nelson and Ryan Danas are two colleagues who fueled this dissertation project both theoretically and practically. Both Tim and Ryan's input were instrumental in shaping this project. I would be remiss if I didn't extend heartfelt gratitude to the collaborators of Razor, including Hai Hoang Nguyen, Visit Pataranutaporn, Taymon Beal,, Erica Ford, Nicholas Murray, and Henning Günther. I also thank Daniel Jackson for his feedback on Razor.

Next, I am grateful to my mentors at WPI, Stanley Selkow, Gabor Sarkozy, Carolina Ruiz, Bob Kinicki, Jerry Breecher, Gary Pollice, and Glynis Hammel. Additionally, I thank my fellow ALAS lab-mates and friends at WPI including but not limited to: Hang Cai, Michael Ficarra, Theo Giannakopoulos, Marc Green, Sarah Jaffer, Krishna Venkatasubramanian, Douglas MacFarland, Bahador Nooraei, Mohammad Shayganfar, Craig Shue, Doran Smestad, Curtis Taylor, Shubhendu Trivedi, Francis Usher, Jian Xu, and Dan Yoo. Lastly, I owe thanks to Newport Coffee, the primary supplier of coffee to the computer science department!

This work is dedicated to my mother and my father.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It is well-known that model-finding for first-order theories is undecidable. However, finite models of a first-order theory can be found by exhaustive search. Gödel's completeness theorem [1] establishes a connection between semantic truth and syntactic provability of first-order logic: a formula φ is provable in a theory $\mathcal{T}$ if and only if every model $\mathbb{M}$ of $\mathcal{T}$ is also a model of φ. The completeness theorem posits a duality between theorem-proving and model-finding; that is, a formula φ is provable if and only if its negation, $\neg\varphi$, is unsatisfiable. As a consequence, model-finding is often regarded as a technique for demonstrating syntactic provability in automated theorem-proving [2–11]: theorem-provers construct proofs, whereas model-finders construct models.

In a more applied setting, model-finding is employed as a "lightweight formal method" [12] to help users—who are not necessarily expert in formal methods—study declarative specifications of hardware and software systems [12–14]. Tools such as Alloy [15] accept a first-order specification $\mathcal{T}$, such as a software design artifact, a description of a cryptographic protocol, or an access control policy, and present first-order models of $\mathcal{T}$ as *examples* that illustrate the behavior of the specified system. This process enables the user to validate the logical consequences of the specification against her expectations about the system's behavior. A distinguishing feature of this approach from the mainstream theorem-proving is that the user may use the model-finder to *explore* models of $\mathcal{T}$ *without having to articulate logical consequences*. The resulting models might suggest "surprising" scenarios, which could lead to flaws in $\mathcal{T}$.

The goal of *lightweight model-finding* is not only to construct a model that proves a theorem about the user's theory, but also to help the user investigate the spectrum of the (possibly a plethora of) models that satisfy the theory. Ideally, a model-finding tool should facilitate a systematic way of exploring models, which helps the user understand the space of models as a whole, rather than as a series of arbitrary examples. It should provide the user with quality models, enabling the user to accurately pinpoint the implications of her theory. And, the tool should allow the user to incorporate her intuition to direct the model-finding process toward models

that are "interesting" to her. In the context of lightweight model-finding, it is natural to ask questions such as: "which models should the user see?", "in what orders should the models be presented to the user?", "how can the user understand *why* a model satisfies a given specification?", and "how can the user understand the connections between the models of the specification?".

Despite the attempts that address some of these questions for specific applications, a framework for *model-exploration* to facilitate the understanding of the space of models is lacking. The existing model-finding tools (with some exceptions) compute models that are effectively random. A conventional model-finders provides the user with a *sequential* iterator over the models of the input theory; it returns the models in no particular order; and, it does not present any justification for the models that it returns.

Thesis. This work presents a framework for *exploring* finite models, which comprises two core features:

1. Support for systematic *exploration* of the space of models.

2. Support for *provenance* information for individual models.

Exploration is concerned with (i) identifying and presenting models in a particular order, and (ii) facilitating an environment for *controlled* navigation between models. Provenance construction, on the other hand, is about providing dialectic information for individual models, justifying every piece of information in models with respect to the axioms in the user's theory. We argue that the preorder $\preccurlyeq$ induced by the homomorphism on models is a viable guideline for exploring the space of models. The homomorphism preorder reflects the "information content" of models: if $\mathbb{M} \preccurlyeq \mathbb{N}$, then $\mathbb{N}$ contains the information in $\mathbb{M}$ (and perhaps more). If $\mathbb{M}$ and $\mathbb{N}$ are models of the same theory, we prefer $\mathbb{M}$ as it contains less distracting information than $\mathbb{N}$.

We implemented our theoretical framework for model-exploration and provenance-construction into Razor, a *model-finding assistant* that helps the user build and examine models of an input specification. For a given input theory $\mathcal{T}$, Razor's returns a set $\mathcal{M}$ of models of $\mathcal{T}$ that are *minimal* under the homomorphism ordering; that is, every model in $\mathcal{M}$ contains a minimum amount of information which is *necessary* for satisfying $\mathcal{T}$. As a consequence of homomorphism minimality, Razor is able to compute provenance information for the models in $\mathcal{M}$: because every element as well as every fact in a homomorphically minimal model $\mathbb{M}$ is necessary for $\mathbb{M}$ to be a model of $\mathcal{T}$, Razor can point the user to axioms in the specification that entail any given element or fact in $\mathbb{M}$.

Model Construction. Razor's algorithm is a variation of the Chase [16–19], an algorithm developed in the database community, which constructs models of first-order theories written in *geometric form*. In presence of Skolemization, any first-

order theory can be written in geometric form; thus, working with theories in geometric form does not pose a theoretical restriction. The building blocks of theories in geometric form are *positive-existential* formulas (PEFs), whereby negation and universal quantification is prohibited. It is a classical result that PEFs are precisely the formulas that are preserved by the homomorphism on models. Moreover, if α is a PEF true of a tuple $\vec{\mathbf{e}}$ in a model $\mathbb{M}$, then the truth of this fact is witnessed by a finite fragment of $\mathbb{M}$. Razor leverages the former property of PEFs to explore models via homomorphism, and the latter to construct provenance information.

Our preliminary implementations of the Chase revealed that a naïve version of this algorithm in presence of disjunction and equality is inefficient. We found it more efficient to implement a variation of the Chase using Satisfiability Modulo Theory (SMT) to process disjunctions and equations. Given a theory $\mathcal{G}$ in geometric form, Razor generates models of $\mathcal{G}$ in two major phases: (i) Razor executes a variation of the Chase to construct a set of *possible facts* for $\mathcal{G}$, $\mathrm{Pos}(\mathcal{G})$, which is in essence, a refinement of the Herbrand base that is inductively defined by $\mathcal{G}$. The construction of $\mathrm{Pos}(\mathcal{G})$ in such a controlled manner, by an operational reading of the formulas in $\mathcal{G}$, allows Razor to compute provenance information for models. (ii) Razor leverages an SMT-solver to compute homomorphically minimal models of a *ground* instance $\mathcal{G}^*$ of $\mathcal{G}$ over $\mathrm{Pos}(\mathcal{G})$. The *minimization* algorithm, implemented into Razor, recursively reduces an arbitrary model of $\mathcal{G}^*$, returned by the SMT-solver, to a model that is homomorphically minimal.

Indeed, utilizing a satisfiability (SAT) solver is the essence of MACE-style [2] model-finding. But the main distinction between the algorithm implemented into Razor and the conventional MACE-style ones lies in the inductive nature of the set of possible facts. The construction of the Herbrand base for a theory $\mathcal{T}$ in MACE-style algorithms is not "controlled". MACE-style algorithms construct ground instances of $\mathcal{T}$ over an *arbitrary* domain of elements, which is fixed *a priori*. In contrast, Razor computes ground instances of $\mathcal{T}$ over a domain of elements that are named by unique Skolem-terms as witnesses. The witnessing terms justify the elements in the models of $\mathcal{T}$ by relating them to the existential quantifiers in $\mathcal{T}$.

Bounded Model-Finding. Although our model-finding algorithm is guaranteed to terminate on theories with a syntactic property known as *weak acyclicity*, it is often necessary to bound the size of models for arbitrary theories. Unlike the conventional MACE-style and SEM-style [20] model-finding, however, our algorithm is not inherently bounded. In fact, our approach is refutationally complete. A traditional model-finder uses a user-supplied upper bound on the models they construct. Alternatively, Razor uses a rather more systematic notion of bounds, determined by the depth of witnessing Skolem-terms. Even when the search is bounded, an iterative-deepening of the search restores Razor's refutational completeness.

Roadmap. For a reader who is interested in a quick overview of this work, Chapter 3 summarizes our primary contributions, consisting of our framework for model-

exploration and a quick introduction to Razor; and Chapter 6.1 provides case-studies and evaluation results. We review the theoretical foundations of this work in Chapter 2 and Chapter 4, and we present our algorithms and theoretical results in Chapter 5. Chapter 7 and Chapter 8 report on the preliminary work, leading to the current implementation of Razor. Finally, we discuss the related work in Chapter 9.

Chapter 2

Foundations

The theoretical foundation of this work is conventional first-order logic. We review our set-up of first-order logic, used in the rest of this thesis, in Section 2.1; we present key definitions of term-rewrite systems in Section 2.2; and, we look into basic definitions that are commonly used in the SMT community in Section 2.3.

2.1 First-Order Logic

This section presents basic definitions of the first-order syntax and semantics. We then review some convenient forms of first-order formulas and present a basic algebra of first-order models.

2.1.1 Syntax

Definition A *signature* $\Sigma \equiv (\Sigma^R, \Sigma^F)$ consists of a (finite) set of *relation* (or *predicate*) symbols Σ^R, a (finite) set of *function* symbols Σ^F, and an *arity* function from the relation and function symbols of the signature to natural numbers.

A signature Ω is said to be an *expansion* of a signature Σ if and only if $\Sigma^R \subseteq \Omega^R$ and $\Sigma^F \subseteq \Omega^F$. Alternatively, Σ is said to be a sub-signature of Ω.

Notation. Throughout this thesis, capital letters $P, Q, R, \ldots$ are used for relation symbols and lowercase letters $f, g, h, \ldots$ for function symbols.

Definition Let $\mathcal{V}$ be a set of variables and $\Sigma \equiv (\Sigma^R, \Sigma^F)$ be a signature. A Σ-*term* over $\mathcal{V}$—or simply a *term* if $\mathcal{V}$ is irrelevant and Σ is clear from the context—is inductively defined by the following:

- A variable $x \in \mathcal{V}$ is a term.
- A function symbol $f \in \Sigma^F$ of arity k applied to a list of terms $\vec{t} \equiv \langle t_1, \ldots, t_k \rangle$, written as $f(\vec{t})$, is also a term.

Every term t_i $(1 \leq i \leq k)$ is said to be at *position* i of a functional term $f(t_1, \ldots, t_k)$. Nullary functions (of arity zero) will be treated as *constants*. We use the notation $\text{Terms}(\Sigma, \mathcal{V})$ to denote the set of all Σ-terms over $\mathcal{V}$, also, write $\text{Terms}(\Sigma)$ if $\mathcal{V}$ is not relevant.

The set $\text{Subterms}(t)$ of a term t is defined by the following:

- $t \in \text{Subterms}(t)$
- if $t = f(t_1, \ldots, t_k)$, then $t_i \in \text{Subterms}(t)$ $(1 \leq t \leq k)$
- if $u \in \text{Subterms}(s)$ and $s \in \text{Subterms}(t)$, then $u \in \text{Subterms}(t)$

We write $t[s]$ for a term t that contains s as a subterm. Also, the notation $\text{Vars}(t)$ will be used for the set of variable subterms in t. A term t is said to be *ground* (or *closed*) if $\text{Vars}(t) = \emptyset$. A *flat* term is a term that is either a constant (nullary function), a variable or a function applied to only constants and/or variables.

Definition The *depth* of a term t, written $\text{depth}(t)$, is defined as follows:

- For a variable term v, $\text{depth}(v) = 0$.
- For a functional term $t \equiv f(t_1, \ldots, t_k)$ (possibly a constant), $\text{depth}(t) = 1 + d$, where d is the maximum of $\text{depth}(t_i)$ $(1 \leq i \leq k)$.

Definition Let Σ be a signature and $\mathcal{V}$ be a set of variables. A substitution $\sigma : \mathcal{V} \longrightarrow \text{Terms}(\Sigma, \mathcal{V})$ is a function from variables to terms such that $\sigma(x) \neq x$ for only finitely many variables in $\mathcal{V}$. The set of variables that are not mapped to themselves by σ is called the *domain* of σ.

Notation. The notation $\{v_1 \mapsto t_1, \ldots v_n \mapsto t_k\}$ denotes a substitution that sends variables v_i to terms t_i $(1 \leq i \leq k)$.

Notation. We use the following notation for an extension of a substitution σ:

$$\sigma[v \mapsto t](x) = \begin{cases} t & x = v \\ \sigma(x) & x \neq v \end{cases}$$

Definition A *first-order formula* (or simply a formula) over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ is inductively defined by the following:

- $\top$ (*truth*) and $\bot$ (*falsehood*) are formulas.
- A relation $R \in \Sigma^R$ of arity k applied to a list of Σ-terms $\vec{t} \equiv \langle t_1, \ldots, t_k \rangle$, written as $R(\vec{t})$, is an *atomic* formula.
- For two Σ-terms t and s, an equation $t = s$ is also an atomic formula.
- For a formula φ, $\neg\varphi$ is also a formula.

- For two formulas φ and ψ, $\varphi \wedge \psi$ is also a formula.
- For two formulas φ and ψ, $\varphi \vee \psi$ is also a formula.
- For two formulas φ and ψ, $\varphi \rightarrow \psi$ is also a formula.
- For two formulas φ and ψ, $\varphi \leftrightarrow \psi$ is also a formula.
- For a variable x and a formula φ, $\exists x.\varphi$ is also a formula.
- For a variable x and a formula φ, $\forall x.\varphi$ is also a formula.

Similar to functional terms, every term t_i $(1 \leq i \leq k)$ is said to be at *position* i of an atomic formula $R(t_1, \ldots, t_k)$.

Terminology. A *literal* is either an atomic formula or the negation of an atomic formula. A *clauses* is a disjunction of literals.

Definition A variable x is said to be *bound* in $\exists x.\varphi$ or $\forall x.\varphi$. A variable x is *free* in a formula φ if and only if it is not bound in φ.

We write $\text{Vars}(\varphi)$ to denote the set of all variables (whether free or bound) and $\text{FV}(\varphi)$ to denote the set of free variables in φ. Accordingly, $\varphi[\vec{x}]$ denotes a formula φ such that $\text{FV}(\varphi) = \vec{x}$. A formula φ is said to be a *sentence* (or *closed*) if $\text{FV}(\varphi) = \emptyset$.

Definition A first-order *theory* $\mathcal{T}$ over a signature Σ is a (finite) set of first-order formulas over Σ.

2.1.2 Semantics

Definition A Σ-*model* (or simply a model) $\mathbb{M}$ over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ is a structure, comprising the following:

- a set of elements $|\mathbb{M}|$ called the *domain* (or the *universe*) of the $\mathbb{M}$.
- a mapping from every k-ary function symbol $f \in \Sigma^F$ to a function $f^{\mathbb{M}} : |\mathbb{M}|^k \longrightarrow |\mathbb{M}|$.
- a mapping from every k-ary relation symbol $R \in \Sigma^R$ to a function $R^{\mathbb{M}} : |\mathbb{M}|^k \longrightarrow \{\mathbf{true}, \mathbf{false}\}$.

The function $R^{\mathbb{M}}$ may be thought of as a subset of all k-tuples over the elements of $|\mathbb{M}|$. Then, we write $\vec{\mathbf{e}} \in R^{\mathbb{M}}$ and $\vec{\mathbf{e}} \notin R^{\mathbb{M}}$ for $R^{\mathbb{M}}(\vec{\mathbf{e}}) = \mathbf{true}$ and $R^{\mathbb{M}}(\vec{\mathbf{e}}) = \mathbf{false}$ respectively. We say $R^{\mathbb{M}}(\vec{\mathbf{e}})$ is a *fact* in $\mathbb{M}$ if $R^{\mathbb{M}}(\vec{\mathbf{e}}) = \mathbf{true}$ in $\mathbb{M}$.

Notation. Throughout this dissertation, roman boldface letters, such as $\mathbf{d}$ and $\mathbf{e}$, are used to denote elements of a model and roman italic letters, such as a and b, to represent constants of a signature.

Notation. We ambiguously write $|\mathbb{M}|$ to denote the cardinality of $|\mathbb{M}|$.

Remark It turns out to be convenient and flexible for our model-finding algorithms to interpret functions as partial functions. Thus, we assume functions to be partially interpreted in models unless we explicitly state otherwise.

Definition Fix a model $\mathbb{M}$ over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ and a set of variables $\mathcal{V}$. An *environment* $\eta : \mathcal{V} \longrightarrow |\mathbb{M}|$ is a function from $\mathcal{V}$ to the elements in $|\mathbb{M}|$.

Definition Fix a set signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ and a set $\mathcal{V}$ of variables. Given an environment $\eta : \mathcal{V} \longrightarrow |\mathbb{M}|$, a Σ-term t over $\mathcal{V}$ *denotes* an element $\mathbf{e} \in |\mathbb{M}|$ under η, written $[\![v]\!]^{\mathbb{M}}_{\eta} = \mathbf{e}$, as follows:

- for a variable $v \in \mathcal{V}$, $[\![v]\!]^{\mathbb{M}}_{\eta} = \eta(v)$.
- for $t = f(t_1, \ldots, t_k)$, where $f \in \Sigma^F$ and t_is are Σ-terms $(1 \leq i \leq k)$ over $\mathcal{V}$, $[\![t]\!]^{\mathbb{M}}_{\eta} = f^{\mathbb{M}}([\![t_1]\!]^{\mathbb{M}}_{\eta}, \ldots, [\![t_k]\!]^{\mathbb{M}}_{\eta})$

Semantics of Formulas. Fix a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ and a model $\mathbb{M}$ over Σ. Let φ be a formula over Σ and $\eta : \mathrm{FV}(\varphi) \longrightarrow |\mathbb{M}|$ be an environment. We say $\mathbb{M}$ satisfies φ in η—or simply $\mathbb{M}$ is a model of φ—written $\mathbb{M} \models_{\eta} \varphi$, if and only if the following induction holds:

- $\mathbb{M} \models_{\eta} R(t_1, \ldots, t_k)$ if and only if $\langle \mathbf{e_1}, \ldots, \mathbf{e_k} \rangle \in R^{\mathbb{M}}$, where $\mathbf{e_i} = [\![t_i]\!]^{\mathbb{M}}_{\eta}$ $(1 \leq i \leq k)$.
- $\mathbb{M} \models_{\eta} t = s$ if and only if $[\![t]\!]^{\mathbb{M}}_{\eta} = [\![s]\!]^{\mathbb{M}}_{\eta}$.
- $\mathbb{M} \models_{\eta} \varphi \wedge \psi$ if and only if $\mathbb{M} \models_{\eta} \varphi$ and $\mathbb{M} \models_{\eta} \psi$.
- $\mathbb{M} \models_{\eta} \varphi \vee \psi$ if and only if $\mathbb{M} \models_{\eta} \varphi$ or $\mathbb{M} \models_{\eta} \psi$.
- $\mathbb{M} \models_{\eta} \varphi \rightarrow \psi$ if and only if $\mathbb{M} \not\models_{\eta} \varphi$ or $\mathbb{M} \models_{\eta} \psi$.
- $\mathbb{M} \models_{\eta} \varphi \leftrightarrow \psi$ if and only if $\mathbb{M} \models_{\eta} \varphi \rightarrow \psi$ and $\mathbb{M} \models_{\eta} \psi \rightarrow \varphi$.
- $\mathbb{M} \models_{\eta} \forall x.\varphi$ if and only if for all $\mathbf{e} \in |\mathbb{M}|$, $\mathbb{M} \models_{\eta[x \mapsto \mathbf{e}]} \varphi$
- $\mathbb{M} \models_{\eta} \exists x.\varphi$ if and only if for some $\mathbf{e} \in |\mathbb{M}|$, $\mathbb{M} \models_{\eta[x \mapsto \mathbf{e}]} \varphi$.

A formula φ is said to be *satisfiable* if and only if a model $\mathbb{M}$ and an environment $\eta : \mathrm{FV}(\varphi) \longrightarrow |\mathbb{M}|$ exist, such that $\mathbb{M} \models_{\eta} \varphi$. φ is said to be *unsatisfiable* (or *inconsistent*) if it is not satisfiable. Two formulas φ and ψ are *equisatisfiable* if φ is satisfiable whenever ψ is satisfiable and *vice versa*.

Semantics of Theories. For a theory $\mathcal{T}$ of formulas, an environment η, and a model $\mathbb{M}$, we write $\mathbb{M} \models_\eta \mathcal{T}$ if and only if $\mathbb{M} \models_\eta \varphi$, for every formula $\varphi \in \mathcal{T}$. A theory $\mathcal{T}$ is satisfiable if and only if $\mathbb{M}$ and η exists such that $\mathbb{M} \models_\eta \mathcal{T}$. $\mathcal{T}$ is unsatisfiable if it is not satisfiable. Two theories $\mathcal{T}$ and $\mathcal{U}$ are equisatisfiable if $\mathcal{T}$ is satisfiable whenever $\mathcal{U}$ is satisfiable and *vice versa*.

2.1.3 Convenient Forms of Formulas

Certain syntactic forms of first-order formulas are often preferable in a variety of applications, including theorem-proving and model-finding. Here, we review some of these convenient forms, also, we state some well-known results about them. In Chapter 4, we present the geometric form as another convenient form of first-order formulas, which supports our mode-finding algorithm.

Definition A first-order formula $\varphi \equiv Q_1 x_1 \ldots Q_n x_n . \alpha$ is in *prenex normal form* (PNF), where every Q_i is either $\exists$ or $\forall$, and α is quantifier-free.

Theorem 2.1.1. Every first-order formula φ is logically equivalent to a formula ψ such that ψ is in PNF.

Definition A formula in *conjunctive normal form* (CNF) is a conjunction of clauses.

Definition A formula $\varphi \equiv \forall \vec{x} \, . \, \alpha$ is said to be in *Skolem normal form* (SNF) if α is a quantifier-free CNF.

Theorem 2.1.2. Every formula φ is equisatisfiable to a formula ψ such that ψ is in SNF. The process of converting a formula to an equisatisfiable SNF is known as *Skolemization*.

Proofs of Theorem 2.1.1 and Theorem 2.1.2 can be found in [21].

2.1.4 Basic Algebra of Models

Common relations on mathematical structures may be extended to first-order models. In this section, we review some basic algebra on models, which will be heavily used in the rest of this document. Specifically, the ordering relation that is determined by the homomorphism on models is the key mathematical tool that supports our framework for exploring models (see Chapter 3).

Definition Let $\mathbb{M}$ and $\mathbb{N}$ be models over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$. $\mathbb{M}$ is a *submodel* of $\mathbb{N}$ if and only if

- $|\mathbb{M}| \subseteq |\mathbb{N}|$.

- $R^{\mathbb{M}}(\vec{\mathbf{e}}) \rightarrow R^{\mathbb{N}}(\vec{\mathbf{e}})$, for every k-ary relation $R \in \Sigma^R$ and every tuple $\vec{\mathbf{e}} \in |\mathbb{M}|^k$.
- $f^{\mathbb{M}}(\vec{\mathbf{e}}) = f^{\mathbb{N}}(\vec{\mathbf{e}})$, for every k-ary function $f \in \Sigma^F$ and every tuple $\vec{\mathbf{e}} \in |\mathbb{M}|^k$.

The model $\mathbb{N}$ is said to be a supermodel of $\mathbb{M}$.

Definition Given a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ and an expansion Ω of Σ, a Σ-model $\mathbb{M}$ is said to be the *reduct* of an Ω-model $\mathbb{N}$ to Σ if and only if

- $|\mathbb{M}| = |\mathbb{N}|$.
- $R^{\mathbb{M}}(\vec{\mathbf{e}}) \leftrightarrow R^{\mathbb{N}}(\vec{\mathbf{e}})$, for every k-ary relation $R \in \Sigma^R$ and every tuple $\vec{\mathbf{e}} \in |\mathbb{M}|^k$.
- $f^{\mathbb{M}}(\vec{\mathbf{e}}) = f^{\mathbb{N}}(\vec{\mathbf{e}})$, for every k-ary function $f \in \Sigma^F$ and every tuple $\vec{\mathbf{e}} \in |\mathbb{M}|^k$.

The model $\mathbb{N}$ is then said to be an expansion of $\mathbb{M}$ to Ω.

Definition Given two models $\mathbb{M}$ and $\mathbb{N}$ over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$, a *homomorphism* $h : |\mathbb{M}| \longrightarrow |\mathbb{N}|$ is a map from the elements of $\mathbb{M}$ to the elements of $\mathbb{N}$ such that

- $h(f^{\mathbb{M}}(\vec{\mathbf{e}})) = f^{\mathbb{N}}(h(\vec{\mathbf{e}}))$, for every k-ary function $f \in \Sigma^F$ and every k-tuple $\vec{\mathbf{e}} \in |\mathbb{M}|^k$.
- $R^{\mathbb{M}}(\vec{\mathbf{e}}) \rightarrow R^{\mathbb{N}}(h(\vec{\mathbf{e}}))$, for every k-ary relation $R \in \Sigma^R$ and every k-tuple $\vec{\mathbf{e}} \in |\mathbb{M}|^k$.

Definition Given two models $\mathbb{M}$ and $\mathbb{N}$ over a signature Σ, an *isomorphism* between $\mathbb{M}$ and $\mathbb{N}$ is a homomorphism $i : |\mathbb{M}| \longrightarrow |\mathbb{N}|$ such that

- i is one to one and onto.
- The inverse $i^{-1} : |\mathbb{N}| \longrightarrow |\mathbb{M}|$ of i is also a homomorphism.

Homomorphism Ordering. The homomorphism over models induces a preorder $\preccurlyeq$ as follows:

- $\mathbb{M} \preccurlyeq_h \mathbb{N}$ if and only if a homomorphism h from $\mathbb{M}$ to $\mathbb{N}$ exists.
- $\mathbb{M} \approx_h \mathbb{N}$ if and only if $\mathbb{M} \preccurlyeq_h \mathbb{N}$ and $\mathbb{N} \preccurlyeq_h \mathbb{M}$.
- $\mathbb{M} \prec_h \mathbb{N}$ if and only if $\mathbb{M} \preccurlyeq_h \mathbb{N}$ but $\mathbb{N} \not\approx_h \mathbb{M}$.

For brevity, we may drop the subscript h when the homomorphism map is irrelevant in the context.

Informally, we say a model $\mathbb{N}$ is in the homomorphism *cone* of a model $\mathbb{M}$ if $\mathbb{M} \preccurlyeq \mathbb{N}$.

2.2 Term-Rewrite Systems

In this section, we review basic definitions of *term-rewrite systems*, specifically, definitions that are commonly used in the context of equality reasoning over terms.

Definition Let l and r be terms. A rewrite rule is an equation $l \to r$ where l is not a variable and $\text{Vars}(l) \supseteq \text{Vars}(r)$. A term-rewrite system (TRS) is a set of rewrite rules. The notation $\to_{\mathcal{R}}$ denotes the rewrite relation induced by a TRS $\mathcal{R}$.

A TRS $\mathcal{G}$ is *ground* if it is defined over ground terms only; *i.e.*, for every $l \to_{\mathcal{G}} r$, $\text{Vars}(l) = \text{Vars}(r) = \emptyset$.

For a rewrite relation $\to$, we respectively write $\leftarrow$, $\leftrightarrow$, $\to^+$ and $\to^*$ for the inverse, symmetric closure, transitive closure and reflexive transitive closure of $\to$.

Terminology. We use the following terminology for term-rewrite systems:

- A term t is *irreducible* or in *normal form* with respect to a TRS $\mathcal{R}$ if and only if there is no u such that $t \to_{\mathcal{R}} u$. A term t is said to be a *normal form of* u if and only if $u \to^*_{\mathcal{R}} t$ and t is in normal form. We write $t \downarrow_{\mathcal{R}}$ to denote the *unique* normal form of t in $\mathcal{R}$ if it exists.
- A TRS $\mathcal{R}$ is *confluent* whenever for every term s, if $s \to^*_{\mathcal{R}} t$ and $s \to^*_{\mathcal{R}} t'$, then a term u exists such that $t \to^*_{\mathcal{R}} u$ and $t' \to^*_{\mathcal{R}} u$, and *vice versa*.
- A TRS $\mathcal{R}$ is *terminating* if there is no infinite chain of reductions in the form of $t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \ldots$.
- A TRS $\mathcal{R}$ is said to be *convergent* if and only if it is terminating and confluent.
- A TRS $\mathcal{R}$ is *left-reduced* if and only if for every $t \to_{\mathcal{R}} u$, t is irreducible by the other rules in $\mathcal{R}$. Similarly, $\mathcal{R}$ is *right-reduced* if and only if for every $t \to_{\mathcal{R}} u$, u is irreducible in $\mathcal{R}$. Finally, $\mathcal{R}$ is said to be *fully-reduced* (or simply reduced) if and only if it is both left and right-reduced.

Definition An equational theory for a set of equations E is the reflexive, symmetric and transitive closure of the rewrite relation $\to_E$, denoted by $\leftrightarrow^*_E$. We may refer to the equations in E as rewrite rules. Then, we may refer to E as a term-rewrite system when we are talking about the rewrite relation $\leftrightarrow^*_E$.

2.3 Satisfiability Modulo Theories

Satisfiability Modulo Theory (SMT) is the problem of checking the satisfiability of a logical formula with respect to background theories [22–25] that fix the interpretations of certain relation or function symbols. By fixing a background theory, SMT-solvers exploit methods that are specific to reasoning about that theory to improve the performance of satisfiability checking. In particular, SMT-solving utilizes

efficient *decision procedures* to check the satisfiability of quantifier-free first-order formulas in background theories, such as equality with uninterpreted functions, integer arithmetic, arrays, bit vectors *etc.*

SMT-LIB. SMT-LIB is an international initiative to encourage and facilitate research and development of SMT [25, 26]. The primary mission of SMT-LIB is to standardize the following aspects of SMT:

- *Background Logic*: includes various fragments of first-order logic, temporal logic, second-order logic *etc.*
- *Background Theory*: is the theory in which the satisfiability of the formula in question is evaluated (*e.g.*, equality with uninterpreted functions or the theory of integer arithmetic).
- *Input Formula*: is the formula whose satisfiability is in question.
- *Interface*: consists of features and standards of interacting with SMT-solvers.

Almost all major SMT-solvers including Boolector [27], UCLID [28], Yices [29] and Z3 [30] recognize the current version of SMT-LIB standard (SMT-LIB 2) and provide a mode of interaction, compatible with this standard.

2.3.1 Syntax

We adopt the syntax and the semantics of SMT-LIB formulas. The underlying logic of SMT-LIB (version 2) is many-sorted first-order logic with equality. However, unlike the conventional formulation of first-order logic, SMT-LIB does not define a syntactic category of formulas distinct from terms. SMT-LIB defines formulas as terms of the built-in **Bool** type [25] [1] . As a consequence, the conversion of standard first-order formulas to SMT-LIB formulas is to coerce relations into **Bool**-valued functions.

Definition An SMT-LIB signature $\Sigma \equiv (\Sigma^S, \Sigma^F, \mu, R)$ consists of the following:

- A set Σ^S of sort symbols including the built-in type **Bool**.
- A set Σ^F of function symbols including $=$, and boolean operators $\wedge$ and $\neg$.
- A partial mapping μ from a set of variables $\mathcal{V}$ to Σ^S.

[1] The logic of SMT-LIB is essentially a variation of many-sorted first-order logic with equality; however, SMT-LIB incorporates ideas from higher-order logic. For example, SMT-LIB recognizes formulas as terms of type **Bool**, or allows sort symbols of arity greater than zero. While these features improve syntactic flexibility, they preserve the underlying many-sorted first-order logic of SMT-LIB. Here, we restrict our definitions to the standard many-sorted first-order logic.

- A left total relation R from Σ^F to Σ^{S+} which includes the following tuples by default:
  - $\langle \neg, \mathbf{Bool} \to \mathbf{Bool} \rangle \in \mathbf{R}$ and $\langle \wedge, \mathbf{Bool} \to \mathbf{Bool} \to \mathbf{Bool} \rangle \in \mathbf{R}$
  - $\langle =, \sigma \to \sigma \to \mathbf{Bool} \rangle \in \mathbf{R}$ for all $\sigma \in \Sigma^S$

Definition Fix a signature $\Sigma \equiv (\Sigma^S, \Sigma^F, \mu, R)$. A term t over Σ is a *well-sorted* term (or simply a term) of type $\sigma \in \Sigma^S$, written as $t : \sigma$, is defined by the following syntax:

- A variable $x \in \mathcal{V}$ such that $\mu(x) = \sigma$ is a term of type σ.
- Given a function symbol f with $\langle f, \sigma_1 \to \cdots \to \sigma_k \to \sigma \rangle \in R$ and a list of well-sorted terms $t_1 : \sigma_1, \ldots, t_k : \sigma_k$ where $\sigma_1, \ldots, \sigma_k \in \Sigma^S$, the term $f(t_1, \ldots, t_k)$ is well-sorted of type σ.
- For a well-sorted term $(t : \mathbf{Bool})$ and a variable $(x : \sigma)$ where $\sigma \in \Sigma^S$, then $\exists (x : \sigma)\ .\ t$ is also a term of type **Bool**.
- For a well-sorted term $(t : \mathbf{Bool})$ and a variable $(x : \sigma)$ where $\sigma \in \Sigma^S$, then $\forall (x : \sigma)\ .\ t$ is also a term of type **Bool**.

Nullary SMT-LIB functions are treated as (SMT-LIB) *constants*.

Definition An SMT-LIB formula is a term of type **Bool**.

Notation. For convenience, we overload the operator : and write $f : \sigma_1 \to \cdots \to \sigma_k \to \sigma$ for $\langle f, \sigma_1 \to \cdots \to \sigma_k \to \sigma \rangle \in R$.

Notice that SMT formulas are often regarded as ground and *quantifier-free* first-order formulas [24]; however, SMT-LIB provides the syntax for existential and universal quantifiers in the general form. Nevertheless, the background fragments of logic defined by SMT-LIB are primarily defined over quantifier-free formulas.

2.3.2 Semantics

The semantics of SMT-LIB formulas are essentially the same as those of many-sorted first-order formulas [25].

Definition Let Σ be an SMT-LIB signature. An SMT-LIB model $\mathbb{M}$ consists of the following:

- a set $|\mathbb{M}|$ of elements as domain, containing $\mathcal{B} = \{\mathbf{true}, \mathbf{false}\}$
- for each sort $\sigma \in \Sigma^S$, a set $\sigma^{\mathbb{M}}$ is a subset of $|\mathbb{M}|$, such that $\mathbf{Bool}^{\mathbb{M}} = \mathcal{B}$
- for each constant $(c : \sigma)$, $c^{\mathbb{M}}$ is an element in $\sigma^{\mathbb{M}}$.

- the interpretation of every (non-nullary) function $(f : \sigma_1 \to \cdots \to \sigma_k \to \sigma)$ in $\mathbb{M}$ is a *total* function $(f : \sigma_1 \to \cdots \to \sigma_k \to \sigma)^{\mathbb{M}}$ from $\sigma_1^{\mathbb{M}} \times \cdots \times \sigma_k^{\mathbb{M}}$ to $\sigma^{\mathbb{M}}$. The function $(= : \ \sigma \to \sigma \to \mathbf{Bool})$ is interpreted as the identity predicate over $\sigma^{\mathbb{M}}$.

Definition Given a list of (typed) SMT-LIB variables $\vec{v}$ and an SMT-LIB model $\mathbb{M}$, an *environment* $\eta : \vec{v} \longrightarrow |\mathbb{M}|$ in SMT-LIB is defined as a function that sends a variable $(v : \sigma)$ to an element $\mathbf{e} \in \sigma^{\mathbb{M}}$.

Fix a signature $\Sigma \equiv (\Sigma^S, \Sigma^F, \mu, R)$ and a model $\mathbb{M}$ over Σ. Let t be an SMT-LIB term (possibly a formula) and η an environment from the variables $\vec{v}$ in t to $|\mathbb{M}|$. A term t denotes an element $\mathbf{e}$ in $\mathbb{M}$, written $[\![t]\!]_\eta^{\mathbb{M}} = \mathbf{e}$, as follows:

- for $t \equiv x$, where x is a variable, $[\![t]\!]_\eta^{\mathbb{M}} = \eta x$.
- for a well-sorted term $t \equiv f(t_1, \ldots, t_k)$, then $[\![t]\!]_\eta^{\mathbb{M}} = f_k^{\mathbb{M}}([\![t_1]\!]_\eta^{\mathbb{M}}, \ldots, [\![t_k]\!]_\eta^{\mathbb{M}})$.
- for a well-sorted term $t \equiv \exists(x : \sigma).s = \mathbf{true}$, then $[\![t]\!]_{\eta[x \mapsto \mathbf{e}]}^{\mathbb{M}} = \mathbf{true}$ for some $\mathbf{e} \in \sigma^{\mathbb{M}}$.
- for a well-sorted term $t \equiv \forall(x : \sigma).s = \mathbf{true}$, then $[\![t]\!]_{\eta[x \mapsto \mathbf{e}]}^{\mathbb{M}} = \mathbf{true}$ for all $\mathbf{e} \in \sigma^{\mathbb{M}}$.

A model $\mathbb{M}$ satisfies an SMT-LIB formula α in an environment $\eta : \mathrm{Vars}(\alpha) \longrightarrow |\mathbb{M}|$, written as $\mathbb{M} \models_\eta \alpha$, if and only if $[\![\alpha]\!]_\eta^{\mathbb{M}} = \mathbf{true}$. Accordingly, $\mathbb{M}$ satisfies a theory $\mathcal{T}$ of SMT-LIB formulas under environment η, denoted by $\mathbb{M} \models_\eta \mathcal{T}$, if and only if for all $\beta \in \mathcal{T}$, $\mathbb{M} \models_\eta \beta$.

The input script of SMT-LIB does not allow free variables in the input formulas but this is theoretically insignificant: in the context of SMT, a formula $\varphi(\vec{x})$ is equivalent to $\exists x_1, \ldots, x_n \ . \ \varphi(\vec{x})$, whereby the free variables are existentially quantified. Again, without loss of generality, the second formula is equisatisfiable to a formula obtained by replacing the existentially quantified variables with fresh constants.

Retrieving Models. Unfortunately, SMT-LIB's interface for interacting with SMT-solvers does not provide a command for retrieving models. Various implementations of SMT-solvers offer different commands to access models of satisfiable formulas. For instance, the commands `get-model` and `show-model` offered by Z3 and Yices return models of the input formulas after satisfiability checking. Moreover, the structure and the content of the resulting models varies from one implementation to another. Specifically, various implementations do not agree on whether and how uninterpreted elements (*i.e.*, elements that are not denoted by the constants of the input theory) should be included in the resulting models.

SMT-LIB, however, specifies a command `get-value` for querying a model $\mathbb{M}$ of the formula in question. This command accepts a closed term t over the signature of the input signature and returns $[\![t]\!]_\eta^{\mathbb{M}}$.

Chapter 3

Model-Exploration Framework

A standard approach to validate a first-order specification $\mathcal{T}$ of a system is by theorem-proving: the user (i) captures a property about the system by a sentence α, and (ii) utilizes a theorem-prover to see if α is provable from $\mathcal{T}$. Alternatively, the user can study models of $\mathcal{T}$ using a model-finder. By Gödel's completeness theorem, α follows from $\mathcal{T}$ if and only if $\mathcal{T} \cup \{\neg\alpha\}$ is unsatisfiable; that is, the approach based on model-finding is logically equivalent to the one based on theorem-proving.

If α fails in $\mathcal{T}$, a model-finder generates models that can serve as *counterexamples*; the models demonstrate situations where the system specified by $\mathcal{T}$ fails to satisfy α. Also, the user might use a model-finding tool to *explore* models, as examples of the system's execution, without specifying α. These models may correspond to examples that confirm the user's expectation, but there also may be models of unanticipated situations, which reflect flaws in the specification.

In this section, we present a theoretical framework for exploring finite models that facilitates an environment where the user studies his specification in interaction with a model-finder. Our framework advocates for a systematic exploration of models in order to understand the space of models of a theory as a whole, rather than a series of individual incidents. The framework also supports the construction of *provenance* information, which can explain a given model according to the user's theory.

The foundation of our model-exploration framework is twofold:

1. strategies for traversing the models of an input theory by *augmentation*.

2. a notion of *provenance* as a way to explain why elements are in the model and why properties are true of them.

Our approach is realized in a model-finding assistant, Razor [31], described in Section 3.3. We call Razor a model-finding *assistant* because users interact with it to build and examine models.

3.1 Model-Exploration

Our solution to *explore* the space of all models of a theory $\mathcal{T}$ relies on a preorder $\leq$ on the models of $\mathcal{T}$. Accordingly, we define a partial *augmentation* operation, which takes a model $\mathbb{M}$ to a model $\mathbb{N}$ of $\mathcal{T}$ such that $\mathbb{M} \leq \mathbb{N}$. A realization of this exploration strategy is a model-finding tool that

1. constructs models of $\mathcal{T}$ that are *minimal* under the preorder $\leq$ on models.

2. computes a set of models, consisting of all $\leq$-*minimal* extensions of $\mathbb{M}$ by an *augmenting* fact F.

The model-finder starts with a stream of models that are minimal under $\leq$. Starting from an initial minimal model, the user can construct non-minimal models of $\mathcal{T}$ by augmentation. When the user augments a given model $\mathbb{M}$ of $\mathcal{T}$ by some fact F, other consequences, perhaps "disjunctive" ones, may be entailed by $\mathcal{T}$. The result of augmentation is a new stream of models of $\mathcal{T}$ containing the facts in $\mathbb{M}$ and the augmenting facts F. The augmentation operation is partial as the augmenting fact F might be inconsistent with $\mathbb{M}$ in $\mathcal{T}$. An important feature of this framework is that the navigation between the models is in the control of the user. The augmentation operator enables the user to use his intuition to direct the model-finder toward solutions that are potentially *interesting* to him.

Choosing a practically useful preorder $\leq$ is the primary challenge for implementing a model-finder that supports model-exploration. Specifically,

1. different minimal models under $\leq$ should represent "various" solutions to the user's problem, specified as $\mathcal{T}$.

2. every (finite) model $\mathbb{N}$ of $\mathcal{T}$ should be "represented" by some minimal model $\mathbb{M}$ of $\mathcal{T}$, given $\mathbb{M} \leq \mathbb{N}$.

The first property above is necessary for presenting distinctive solutions to the user's problem and filtering the repetitive one. The second property makes every (finite) model of the theory available to the user via augmentation.

Aluminum. Aluminum [32] is our first attempt to develop a tool for model-exploration (see Chapter 7). The ordering relation on the models that Aluminum constructs is the containment relation on models that are defined over the same domain: for two models $\mathbb{M}$ and $\mathbb{N}$, given $|\mathbb{M}| = |\mathbb{N}|$, $\mathbb{M} \leq \mathbb{N}$ if and only if $\mathbb{M}$ is a submodel of $\mathbb{N}$. Accordingly, Aluminum computes a set of models that are minimal with respect to this containment ordering. And, the augmentation operation sends a model $\mathbb{M}$ to a model $\mathbb{N}$ such that $\mathbb{M}$ is a submodel of $\mathbb{N}$.

It is noteworthy that Aluminum requires the user to specify a bound on the models it generates; that is, the domain of models is fixed *a priori*. For that reason,

defining the preorder on models over the same domain is not a restriction for the ordering relation.

Restricting the preorder to models of the same domain, however, limits the augmentation operation. Specifically , Aluminum does not support augmenting models with new elements. Furthermore, Aluminum does not treat signatures with equality; that is, the augmentation cannot equate distinct elements of the model.

Razor. Razor adopts the preorder $\preccurlyeq$, induced by the homomorphism on the models of the input theory $\mathcal{T}$, for the ordering relation; that is, $\mathbb{M} \leq \mathbb{N}$ if $\mathbb{M} \preccurlyeq \mathbb{N}$. Consequently, Razor returns models that are homomorphically minimal.

This homomorphism preorder has a natural "information content" interpretation: when $\mathbb{M} \preccurlyeq \mathbb{N}$, the model $\mathbb{N}$ has all the information in $\mathbb{M}$ (and perhaps more). Suppose a user is wondering whether a certain state of affairs $\{S\}$ is consistent with a theory $\mathcal{T}$, that is, she is looking for examples of $\mathcal{T}$ in which S holds. If $\mathbb{M}$ and $\mathbb{N}$ are models of $\mathcal{T} \cup \{S\}$, with $\mathbb{M} \preccurlyeq \mathbb{N}$, then exhibiting $\mathbb{M}$ conveys a clearer picture to the user: the extra information in $\mathbb{N}$ as compared with $\mathbb{M}$ is not required for S to hold. This extra information is a distraction to the user.

The augmentation operation implemented into Razor allows for adding new facts to the current model, as well as equating distinct elements of the model. Furthermore, unlike Aluminum's augmentation, augmentation may extend the domain of the current model by new elements. For many theories $\mathcal{T}$ we can be sure that every finite model of $\mathcal{T}$ is constructible in this incremental way.

3.2 Provenance Construction

Provenance information for a model allows the user to ask: "why is this element in the model?", "why is this fact true?" or "why are these two elements equal?". The model-finding tool then points the user to an axiom in the specification that can be "blamed". We introduce the two following classes of provenance information:

- *Origin information*: Every element in the model is there for a reason. Origin information justifies the existence of any particular element of a model by pointing to an existential quantifier or a function symbol in the theory.

- *Justification information*: Every fact in the model can be traced back to a specific instance of a particular axiom in the specification.

Provenance information is a direct consequence of computing models that are minimal with respect to information content. Every fact in a minimal model is necessary for satisfying the input theory; thus, it must be justified by an axiom in the theory. An *accidental* fact, which is optionally true in a non-minimal model, cannot be justified.

Razor. Razor computes origin and justification information for the models that it returns to the user. Any element in a homomorphically minimal model, computed by Razor, is there in response to a sentence in the user's input, indeed as a witness for a particular existential quantifier or a function in the input theory. Razor names the element by a term over a *witnessing* signature, which inductively explains the element's origin in terms of other elements' origins. If two names denote the same element in a model, there is a particular equation in the theory which is instantiated with the said two names (see Section 4.3.3). Similarly, any atomic fact of a minimal 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.

The (relative) minimality of the models produced by an augmentation ensures that provenance information can be computed over them as well.

3.3 Razor, a Model-Finding Assistant

We review Razor's key features that enables the user to *explore* the (finite) models of a theory. Razor supplies a Read-Eval-Print Loop (REPL) to the user, which operates in three "modes", illustrated by Figure 3.1:

1. *Theory:* allows the user to load a theory and edit the model-finding options.
2. *Explore:* presents models of the input theory and allows the user to explore (non-minimal) models via augmentation.
3. *Explain:* provides the user with provenance information about an existing model.

The current user interface of Razor is rather primitive. Input theories are presented as text files and models are displayed as text as well. An important area of future work is to improve the visual display of models, to facilitate users' apprehension of them.

In this section, we introduce the primary features of Razor in interaction with the REPL. In Section 6.1, we present concrete examples of such interaction with the tool.

3.3.1 The Theory Mode

The REPL initially starts in the Theory Mode. The user can re-enter this mode by entering the following command:

```
> @theory
```

Figure 3.2 specifies the grammar of Razor's input theory files. The user loads a theory in Razor's input language by the following command:

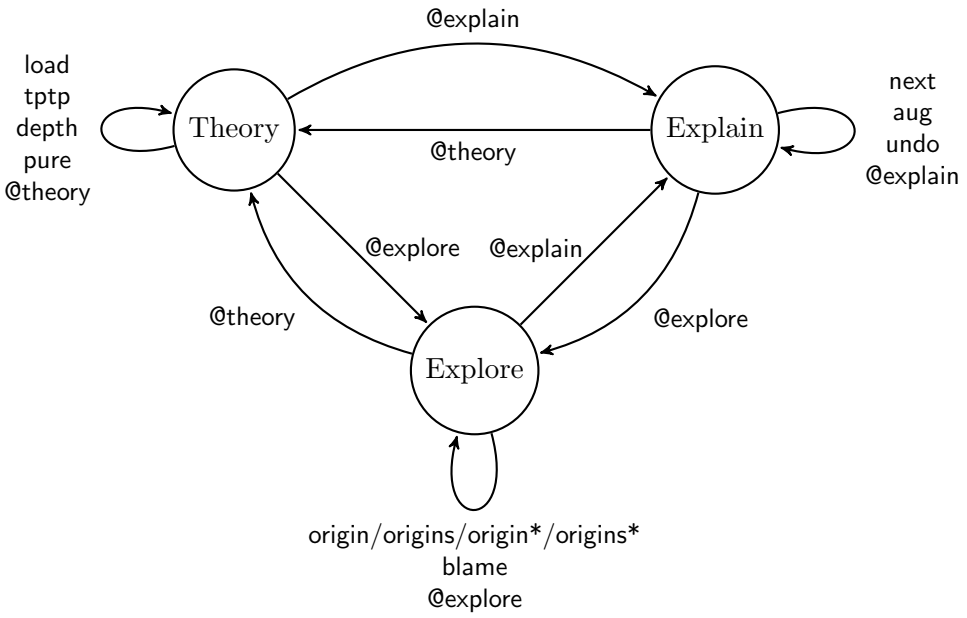

Figure 3.1: State transition in Razor's REPL

```
> load <FILE>
```

If Razor detects a syntax-error in the user's input file, the error will be reported to the user; otherwise, Razor loads the theory and will be ready for constructing models.

Bounding Configuration. The user may edit search options before constructing models. The next command bounds the depth of search (see Section 4.3.5):

```
> depth <DEPTH>
```

The parameter passed to this function is an integer value as the upper-bound for the depth the terms that *witness* the elements of models. The bound restricts the domain of models generated by Razor to elements that are witnessed by terms of a depth less than or equal to the given depth. The default value of this parameter is -1, indicating an unbounded search, which is suitable for *weakly acyclic* (see Section 4.3.4) theories.

If the user runs a bounded search, she can choose between either of the two bounding strategies, *pure* or *reuse*, implemented into Razor. When the maximum bound is reached, the pure mode returns a partial model over the elements that are witnessed by terms of depth within the specified bound. The reuse mode, on the other hand, tries to reuse the existing elements, instead of creating new ones, to create complete models of the input theory. Both strategies are explained in detail in section Section 4.3.5.

```
input       ::= theory depths?
theory      ::= [sequent ";"]*
sequent     ::= body "=>" head | head | "~" body
body        ::= conjunctive
head        ::= [existential "|"]* existential
existential ::= ["exists" [skolemfun? variable]+ "."]* conjunctive
conjunctive ::= [atom "&"]* atom
atom        ::= identifier terms? | term "=" term
                                  | "Truth"
                                  | "Falsehood"
terms       ::= "(" [[term ","]* term]? ")"
term        ::= identifier terms | variable | constant
skolemfun   ::= "<" identifier ">"
constant    ::= "'" identifier
variable    ::= identifier
identifier  ::= [_a-zA-Z][_a-zA-Z0-9]*
depths      ::= [depth ","]* depth
depth       ::= "@DEPTH" skolemfun "=" number

Precedence of operators (from high to low):
=
&
|
=>
```

Figure 3.2: Razor's input grammar

The default bounding strategy is the reuse mode. By running the following command, the user can activate or deactivate the pure strategy:

```
> pure <FILE>
```

TPTP Syntax. The current version of Razor also supports input theories in the syntax of TPTP [33]:

```
> tptp <FILE>
```

Since TPTP inputs in CNF are essentially in the geometric form, they can be processed by the current version Razor. For free-form FOF inputs, Razor currently utilizes a "best-effort" algorithm to convert the input to the geometric form. A more sophisticated conversion may apply Skolemiziation (when needed) to transform any first-order theory to the geometric form.

3.3.2 The Explore Mode

After loading the input theory, the user can *explore* models of the theory in the Explore Mode:

```
> @explore
```

After entering the Explore Mode, the REPL automatically runs the underlying model-finding engine to build models of the input theory, and it presents a first minimal model (if it exists) to the user. The user can enter the next command to see another (minimal) model of the theory if such a model exists:

```
> next
```

Augmentation. The Explore Mode also allows the user to *augment* the current model with additional facts in order to construct non-minimal models of the theory:

```
> aug <FACT>
```

If the augmenting fact is consistent with the current model in the input theory, Razor presents the resulting model, consisting of (i) the facts in the model that was augmented, (ii) the augmenting fact, and (iii) additional facts that are entailed by the augmentation. Otherwise, if the augmenting fact is not consistent with the current model, Razor reports that the augmentation is inconsistent.

Notice that after an augmentation, the command `next` will be available for exploring the possibly multiple resulting models. Moreover, any of the resulting models may be further augmented to construct richer models of the theory. At any moment, the user can undo an augmentation and revisit the previous model (before augmentation) by entering the following command:

```
> undo
```

Conceptually, `next` and `aug` organize the models of the input theory into a two-dimensional space illustrated by Figure 3.3. By contrast, a traditional model-finder delivers an iterator over the models; the iterator visits the models in no particular order; and, the user has no control over the model that will be presented to her next.

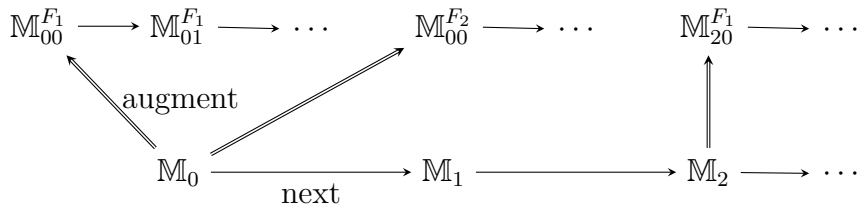

Figure 3.3: Conceptual space of models in Razor

3.3.3 The Explain Mode

The Explain Mode provides the user with explanatory *provenance* information about the models constructed by Razor. The REPL enters the Explain Mode by the following command:

```
> @explain
```

Element Origins. The user can then asks for provenance information about the current model (constructed in the Explore Mode). For obtaining provenance information about an element of the current model, the user can enter the following command:

```
> origin <ELEMENT>
```

where the input parameter to this command is the name of the element.

In response, Razor presents an instance of a sentence in the user's theory, induced by a variable assignment from the free variables of this sentence to the other elements of the current model, which justifies the existence of the element in question (see Section 6.1 for an examples).

An equation in the user's theory may identify distinct elements of the model, resulting in an element that can be explained by multiple origins. The following command displays all possible origins of a given element:

```
> origins <ELEMENT>
```

If the user is interested in a "deep inspection" for the origin of an element, and every element in the model that originates the element in question, she can enter the command below:

```
> origin* <ELEMENT>
```

This command recursively displays the origin of the input element, and the origins of the elements that are mentioned in the origin information about the element in question. Finally, the next command

```
> origins* <ELEMENT>
```

performs a deep inspection recursively while presenting all origins of all elements.

Blaming Facts. The Explain Mode also allows the user to "blame" a given fact in the current model by the following command:

```
> blame <FACT>
```

The input to this command is a fact that is true in the current model. In response, Razor displays an instance of a sentence in the user's theory, which requires the given fact to be true in the current model.

Chapter 4

Geometric Logic

Geometric logic is the logic of *observable* properties. Every statement in a geometric theory is an implication between two *positive-existential formulas* (PEF), in which the use of negation and universal quantification is prohibited. It is a well-known result that PEFs are preserved under the homomorphism on models; that is, once a PEF is "observed" in a model, it will remain true as the the model is extended by further information under homomorphism. Consequently, geometric logic is often understood as the logic of information preserved by the homomorphism on models.

The observable properties of geometric logic comprises the following:

- Only *finitely* many observations are needed to show that an observable property holds in a model.
- Observable properties of a model are preserved as the model is extended by homomorphism.

Because of the positivity of geometric logic, it is always possible to demonstrate the presence of an observable property however, it is not always possible to show its absence [34]. The positivity of geometric logic has profound consequences on the semantics of first-order formulas in geometric form, suggesting them as viable choice for specification languages [34, 35].

In this chapter, we introduce the syntax of formulas in geometric form in Section 4.1 and state well-known results about the semantics of geometric logic in Section 4.2. Next, in Section 4.3, we present the Chase, a well-known algorithm from the database community that can be used to construct models of geometric theories. Finally, in Section 4.4, we discuss how the positivity of geometric logic enables us to augment models of geometric theories with additional information.

4.1 Syntax

Definition A *positive-existential formula* (PEF) over a signature Σ is a formula over Σ with $\wedge$, $\vee$, $\exists$ and $=$ as connectives. Infinitary disjunction $\bigvee_i$ is also permitted.

Remark The use of infinitary disjunction is prohibited in *Coherent logic*, a restricted form of geometric logic.

Definition A *geometric sequent* over a signature Σ is a construct $\varphi \vdash_{\vec{x}} \psi$ where φ and ψ are PEFs over Σ and $\vec{x} = \mathrm{FV}(\varphi) \cup \mathrm{FV}(\psi)$.

A geometric sequent $\varphi \vdash_{\vec{x}} \psi$ may be regarded as a shorthand for $\forall \vec{x} \,.\, (\varphi \to \psi)$. We refer to φ and ψ as the *body* and the *head* of $\varphi \vdash_{\vec{x}} \psi$ respectively.

Definition A *geometric theory* $\mathcal{G}$ over a signature Σ is a set of geometric sequents over Σ.

4.1.1 Standard Forms

The following are well-known results about geometric formulas. Proofs of Lemma 4.1.1 and Lemma 4.1.2 (in a more general setting) may be found in [36].

Lemma 4.1.1. Every PEF $\varphi(\vec{x})$ is logically equivalent to a *standard* PEF of the form

$$\bigvee_i (\exists \vec{y_i} \,.\, \bigwedge_{j=1}^{n_i} P_{ij}(\vec{x}, \vec{y_i}))$$

Lemma 4.1.2. Every geometric sequent $\sigma \equiv \varphi \vdash_{\vec{x}} \psi$ is equivalent to a set of *standard geometric sequent* of the form $\varphi \vdash_{\vec{x}} \psi$ where φ and ψ are standard PEFs, and φ contain no $\exists$ or $\vee$.

Corollary 4.1.3. Every geometric theory is logically equivalent to a geometric theory of standard sequents.

In the rest of this dissertation, we assume geometric sequents in the standard form.

4.2 Semantic Properties

If $\varphi(\vec{x})$ is a PEF true of a tuple $\vec{\mathbf{e}}$ in a model $\mathbb{M}$ then the truth of this fact is witnessed by a finite fragment of $\mathbb{M}$. Thus, if $\mathbb{M}$ satisfies α with $\vec{\mathbf{e}}$ and $\mathbb{M}$ is expanded by new elements or facts, $\alpha(\vec{x})$ still holds of $\vec{\mathbf{e}}$ in the resulting model. For this reason, properties defined by PEFs are sometimes called *observable* properties [37]. It is a classical result that PEFs are precisely the formulas preserved under homomorphisms; Rossman [38] 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 (see Chapter 3). We formally show that

1. PEFs are preserved under homomorphism.

2. every first-order theory is equisatisfiable to a geometric one.

The second claim above suggests geometric logic not as a restricted fragment of first-order logic but a syntactic variation of first-order logic with the same expressive power as first-order formulas.

Theorem 4.2.1 is a well-known result about PEFs. Notice that this theorem is not true for an arbitrary first-order formula as it may contain $\neg$ and $\forall$:

Theorem 4.2.1. Let $\mathbb{M}$ and $\mathbb{N}$ be models such that $\mathbb{M} \preccurlyeq_h \mathbb{N}$. For a PEF φ, if $\mathbb{M} \models_\eta \varphi$ then $\mathbb{N} \models_{h\circ\eta} \varphi$, where $h \circ \eta$ is the functional composition of h and η.

Proof. Observe that this is true for atomic PEFs by the definition of homomorphism on models. We prove the theorem for complex PEFs by induction:

- Assume that for PEFs α and β, $\mathbb{M} \models_\eta \alpha$ implies $\mathbb{N} \models_{h\circ\eta} \alpha$, and $\mathbb{M} \models_\eta \beta$ implies $\mathbb{N} \models_{h\circ\eta} \beta$. For $\varphi \equiv \alpha \wedge \beta$, it is easy to show that $\mathbb{M} \models_\eta \varphi$ implies $\mathbb{N} \models_{h\circ\eta} \varphi$. Likewise, for $\varphi \equiv \alpha \vee \beta$, $\mathbb{M} \models_\eta \varphi$ implies $\mathbb{N} \models_{h\circ\eta} \varphi$.

- For $\varphi \equiv \exists x \ . \ \alpha$, assume that for some environment η, there exists some $\mathbf{e} \in |\mathbb{M}|$ such that $\mathbb{M} \models_{\eta[x\mapsto\mathbf{e}]} \alpha$. By the induction hypothesis, $\mathbb{M} \models_{\eta[x\mapsto\mathbf{e}]} \alpha$ implies $\mathbb{N} \models_{h\circ\eta[x\mapsto\mathbf{e}]} \alpha$; therefore, $\mathbb{N} \models_{h\circ\eta} \varphi$.

□

Preservation of PEFs under homomorphism has been carefully studied in category theory and theory of topos [36, 39–41]. The models of a geometric formula form a category under homomorphism:

$$\mathbb{M}_0 \xrightarrow{h_0} \mathbb{M}_1 \xrightarrow{h_1} \mathbb{M}_2 \xrightarrow{h_2} \dots$$

As we progress along the homomorphism arrows, a model $\mathbb{M}_i$ is augmented with more information. The homomorphism h_i is guaranteed to preserve the information in $\mathbb{M}_i$ while it may introduce new elements to the domain of $\mathbb{M}_i$, add new facts to $\mathbb{M}_i$, or send distinct elements of $\mathbb{M}_i$ to the same element in $\mathbb{M}_{i+1}$.

Expressive Power. By Theorem 4.2.2, any first-order formula is equisatisfiable to a geometric theory. And, because first-order satisfiability is undecidable, satisfiability of theories in geometric form is undecidable as well.

Theorem 4.2.2. A first-order formula is equisatisfiable to a geometric theory.

Proof. Assume $\alpha \equiv (Q_1 x_1 \dots Q_m x_m \,.\, \beta)$ in PNF, where Q_is are either $\forall$ or $\exists$. After Skolemization, α is equisatisfiable to a formula $(\forall \vec{y} \,.\, \gamma)$. Finally, after putting γ in CNF, α is equisatisfiable to $(\forall \vec{y} \,.\, \bigwedge_{i=1}^{k} (\varphi_i \to \psi_i))$ where φ_is are conjunctions and ψ_is are disjunctions of atomic formulas. Every implication $\varphi_i \to \psi_i$ corresponds to a geometric sequent, thus, α is equisatisfiable to a geometric theory. □

Notice that in the previous proof, conversions to PNF and CNF maintains logical equivalency but Skolemization only preserves equisatisfiability.

4.3 The Chase

The *Chase* is a classical model-finding algorithm that constructs a *set-of-support* for a given satisfiable geometric theory. The Chase was developed by the database community, applied on a variety of problems including query equivalence, query optimization and data-dependency [16–19].

In this section, we first introduce the notion of a set-of-support for a theory; we describe the classical Chase algorithm; and, we show that the Chase constructs a set-of-support for a given satisfiable geometric theory.

4.3.1 Universal Models

Definition A model $\mathbb{U}$ is said to be a *universal model* of a theory $\mathcal{T}$ if and only if:

- $\mathbb{U} \models \mathcal{T}$
- For every model $\mathbb{M}$, if $\mathbb{M} \models \mathcal{T}$, then $\mathbb{U} \preccurlyeq \mathbb{M}$.

Observe that an arbitrary theory $\mathcal{T}$ does not necessarily have a universal model. Consider $\varphi \equiv (\exists x, y \,.\, P(x) \vee Q(y))$ for example: φ has two "minimal" models, $\mathbb{M} \equiv \{P^{\mathbb{M}}(\mathbf{e})\}$ and $\mathbb{N} \equiv \{Q^{\mathbb{N}}(\mathbf{e})\}$, which could be candidates for being the universal model of φ. However, neither $\mathbb{M} \not\preccurlyeq \mathbb{N}$ nor $\mathbb{N} \not\preccurlyeq \mathbb{M}$.

The following extends the definition of universal model to a set-of-support for a given theory:

Definition A set $\mathcal{S}$ of models is said to be a *set-of-support* for a theory $\mathcal{T}$ if and only if:

- For every $\mathbb{U} \in \mathcal{S}$ then $\mathbb{U} \models \mathcal{T}$.
- For every model $\mathbb{M} \models \mathcal{T}$, there exists a model $\mathbb{U} \in \mathcal{S}$ such that $\mathbb{U} \preccurlyeq \mathbb{M}$.

Intuitively speaking, every model of $\mathcal{T}$ is reachable from some model in $\mathcal{S}$ by homomorphism. It follows from the definition that any superset of a set-of-support is itself a set-of-support.

4.3.2 The Witnessing Signature

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

Notation. Given a geometric sequent $\alpha \vdash \bigvee_i (\exists y_{i1} \ldots \exists y_{ip} \ . \ \beta_i(\vec{x}, y_{i1}, \ldots, y_{ip}))$*, we assign a unique, fresh,* witnessing (partial) function symbol f_{ik} *to every existential quantifier* $\exists y_{ik}$*, written as* $\alpha \vdash \bigvee_i (\exists^{\mathsf{f}_{\mathsf{i1}}} y_{i1} \ldots \exists^{\mathsf{f}_{\mathsf{ip}}} y_{ip} \ . \ \beta_i(\vec{x}, y_{i1}, \ldots, y_{ip}))$*. This determines an associated sentence* $\alpha \vdash \bigvee_i \beta_i(\vec{x}, \mathsf{f}_{i1}(\vec{x}), \ldots, \mathsf{f}_{ip}(\vec{x}))$ *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 device enables us to reason more conveniently about the models constructed by the Chase. The terms over the witnessing signature allow Razor to report provenance for elements of the models it constructs.

Definition Fix a geometric theory $\mathcal{G}$ over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$, and a set of witness functions $\mathcal{F}$ for the existential quantifiers in $\mathcal{G}$. Let $\mathbb{M}$ be a model over the signature $\Sigma^{\mathrm{w}} \equiv (\Sigma^R, \Sigma^F \cup \mathcal{F})$. The *witness-reduct* of $\mathbb{M}$ is the reduct $\mathbb{M}^-$ of $\mathbb{M}$ to Σ.

4.3.3 The Chase Algorithm

A run of the classical (or standard) *Chase* on a geometric theory $\mathcal{G}$ starts with a given model $\mathbb{M}$ and proceeds by consecutive invocations of a sub-procedure, the *chase-step*. The initial model $\mathbb{M}$ and the subsequent models constructed by the chase-step are models of the witnessing signature of $\mathcal{G}$. The domain of the models that the Chase creates is a partial equivalence relation on the elements witnessed by the witnessing terms. When processing a sequent in $\mathcal{G}$ with disjunctions in its head, the Chase branches to do an exhaustive search for models. It is easiest to present the algorithm as a non-deterministic procedure, where different runs of the Chase induced by the non-deterministic choices lead to various models of the theory.

Definition Fix a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ and a set of witness functions $\mathcal{F}$. Let $\sigma \equiv \varphi \vdash_{\vec{x}} \psi$ be a geometric sequent over Σ whose existential quantifiers are witnessed by functions in $\mathcal{F}$. Let $\mathbb{M}$ be a model over the witnessing signature $\Sigma^{\mathrm{w}} \equiv (\Sigma^R, \Sigma^F \cup \mathcal{F})$, $\mathbb{M}^-$ be the reduct of $\mathbb{M}$ to Σ, and $\eta : \vec{x} \to |\mathbb{M}|$ be an environment. The classical *chase-step*, denoted by $\mathbb{M} \xrightarrow{(\sigma,\eta)} \mathbb{N}$, accepts σ, $\mathbb{M}$, and η as input and returns a model $\mathbb{N}$ over Σ^{w} according to Algorithm 1.

Algorithm 2 demonstrates a non-deterministic version of the Chase. Starting with an empty model $\mathbb{M}$ over the witnessing signature Σ^{w}, for witness-reduct $\mathbb{M}^-$ of $\mathbb{M}$, if $\mathbb{M}^- \models \mathcal{G}$, then $\mathbb{M}$ will be returned. Otherwise, the algorithm non-deterministically

Algorithm 1 Chase-Step

Require: $\mathbb{M}^- \models_\eta \varphi$, $\mathbb{M}^- \not\models_\eta \psi$
Ensure: $\mathbb{N} \models_\eta \varphi$, $\mathbb{N} \models_\eta \psi$

function ChaseStep($\mathbb{M}, \sigma \equiv \varphi \vdash_{\vec{x}} \psi, \eta$)
  if $\psi = \bot$ **then fail**
  $\mathbb{N} \leftarrow \mathbb{M}$
  choose disjunct $(\exists^{\mathsf{f}_1} y_1, \ldots, \exists^{\mathsf{f}_m} y_m \,.\, \bigwedge_{j=1}^{n} P_j(\vec{x}, \vec{y}))$ in ψ
  $\mu \leftarrow \eta[y_1 \mapsto [\![\mathsf{f}_1(\vec{x})]\!]^{\mathbb{M}}_\eta, \ldots, y_m \mapsto [\![\mathsf{f}_m(\vec{x})]\!]^{\mathbb{M}}_\eta]$
  $|\mathbb{N}| \leftarrow |\mathbb{N}| \cup \{[\![\mathsf{f}_i(\vec{x})]\!]^{\mathbb{M}}_\eta \mid 1 \leq i \leq m\}$ ▷ each $[\![\mathsf{f}_i(\vec{x})]\!]^{\mathbb{M}}_\eta$ denotes a fresh element
  $\mathbb{N} \leftarrow \mathbb{N} \cup \{P_1^{\mathbb{N}}[\mu\vec{x}, \mu\vec{y}], \ldots, P_n^{\mathbb{N}}[\mu\vec{x}, \mu\vec{y}]\}$
  return $\mathbb{N}$

chooses a sequent $\sigma \equiv \varphi \vdash_{\vec{x}} \psi$ in $\mathcal{G}$ together with an environment $\eta : \vec{x} \longrightarrow |\mathbb{M}|$ such that $\mathbb{M}^- \not\models_\eta \sigma$. Then, the Chase attempts to "repair" $\mathbb{M}^-$ in σ by making a chase-step.

The process of repairing the sequents of $\mathbb{M}^-$ in $\mathcal{G}$ continues until the Chase terminates with success, resulting in a model $\mathbb{M}$ where the chase-step cannot be further applied. A run of the Chase fails if a chase-step fails on a model $\mathbb{N}$ where the body of a sequent with an empty head is true in the witness-reduct of $\mathbb{N}$. Notice that different runs of the Chase induced by the non-deterministic choices of the algorithm lead to various models of $\mathcal{G}$ in the extended signature Σ^{w}.

Algorithm 2 Chase

function Chase($\mathcal{G}$)
  $\mathbb{M} \leftarrow \emptyset$ ▷ start with an empty model over an empty domain
  while $\mathbb{M}^- \not\models \mathcal{G}$ **do**
    choose $\sigma \in \mathcal{G}$, $\eta : \vec{x} \longrightarrow |\mathbb{M}|$ s.th. $\mathbb{M}^- \not\models_\eta \sigma$
    $\mathbb{M} \leftarrow$ ChaseStep($\mathbb{M}, \sigma, \eta$)
  return $\mathbb{M}$

Terminology. A model $\mathbb{M}$ of a geometric theory $\mathcal{G}$ that is constructed by some run of the classical Chase is said to be a *chase-model* of $\mathcal{G}$.

Fairness

A deterministic implementation of the Chase that utilizes a *scheduler* for selecting the sequent σ and the environment η to repair in the current model is said to be *fair* if the scheduler guarantees to eventually evaluate every pair of possible choices for σ and η. The next definition formally captures this idea:

Definition Let $\mathcal{G}$ be a geometric theory and ρ be an infinite run of the Chase starting from an empty model $\mathbb{M}_0$:

$$\rho = \mathbb{M}_0 \xrightarrow{(\sigma_0,\eta_0)} \mathbb{M}_1 \xrightarrow{(\sigma_1,\eta_1)} \dots \mathbb{M}_i \xrightarrow{(\sigma_i,\eta_i)} \mathbb{M}_{i+1} \xrightarrow{(\sigma_{i+1},\eta_{i+1})} \dots$$

Let $\mathcal{D} \equiv \bigcup_i^j |\mathbb{M}_i|$ be a set of elements, consisting of the domain of the first j models in ρ. Assume that for some environment $\eta : \mathrm{FV}(\sigma) \longrightarrow \mathcal{D}$, and some $\sigma \in \mathcal{G}$, $\mathbb{M}_j \not\models_\eta \sigma$. We say ρ is a *fair* run of the Chase if and only if for all such η, there exists a chase-step k in ρ such that $j \leq k$ and $(\sigma_k, \eta_k) = (\sigma, \eta)$.

Chase Properties

The following records basic properties of the Chase; each result is either well-known or a routine generalization of known facts. We first provide a proof for Lemma 4.3.1, which we refer to as the *Oracle*. The Oracle will then be used to prove a fundamental result about the Chase, stated by Theorem 4.3.2. According to this theorem, the reducts of all models returned by various runs of the Chase form a set-of-support for the given geometric theory.

Lemma 4.3.1. Fix a geometric theory $\mathcal{G}$ over a signature Σ, and a witnessing signature Σ^{w}, obtained by extending Σ with witness functions assigned to the existential quantifiers of $\mathcal{G}$. Let $\mathbb{M}$ be a model over Σ^{w} and $\mathbb{M}^-$ be the witness-reduct of $\mathbb{M}$ to Σ such that $\mathbb{M}^- \models \mathcal{G}$. Let $\mathbb{N}$ be a structure over Σ^{w} given $\mathbb{N} \preccurlyeq \mathbb{M}$ and $\mathbb{N}^-$ be the witness-reduct of $\mathbb{N}$ to Σ. Then for any sequent $\sigma \equiv \varphi \vdash_{\vec{x}} \psi$ in $\mathcal{G}$ and any environment $\eta : \vec{x} \longrightarrow |\mathbb{N}|$, if $\mathbb{N}^- \not\models_\eta \sigma$, then a chase-step $\mathbb{N} \xrightarrow{(\sigma,\eta)} \mathbb{N}'$ exists such that $\mathbb{N}' \preccurlyeq \mathbb{M}$.

Proof. Because $\mathbb{N}^- \not\models_\eta \sigma$, then $\mathbb{N}^- \models_\eta \varphi$ but $\mathbb{N}^- \not\models_\eta \psi$. Letting h be the homomorphism from $\mathbb{N}$ to $\mathbb{M}$, it is given that $\mathbb{M}^- \models_{h\circ\eta} \varphi$. Since $\mathbb{M}$ is a model of $\mathcal{G}$, some disjunct $(\exists^{\mathsf{f}_1} y_1 \dots \exists^{\mathsf{f}_p} y_p \,.\, \beta(\vec{x}, y_1, \dots, y_p))$ of ψ must be true in $\mathbb{M}^-$ in environment $h \circ \eta$. That is, there exist elements $[\![\mathsf{f}_1(\vec{x})]\!]^{\mathbb{M}}_\eta, \dots, [\![\mathsf{f}_\mathsf{p}(\vec{x})]\!]^{\mathbb{M}}_\eta$ in $\mathbb{M}$ such that $\mathbb{M} \models_{h\circ\mu} \beta$, where $\mu = \eta[y_i \mapsto [\![\mathsf{f}_\mathsf{i}(\vec{x})]\!]^{\mathbb{M}}_\eta]$ $(1 \leq i \leq p)$.

There exists a chase-step $\mathbb{N} \xrightarrow{(\sigma,\eta)} \mathbb{N}'$ that chooses the aforementioned disjunct in ψ: it extends $\mathbb{N}$ with fresh elements $[\![\mathsf{f}_1(\vec{x})]\!]^{\mathbb{N}'}_\eta, \dots, [\![\mathsf{f}_\mathsf{p}(\vec{x})]\!]^{\mathbb{N}'}_\eta$ in a way that $\mathbb{N}' \models_\lambda \beta$ where $\lambda = \eta[y_i \mapsto [\![\mathsf{f}_\mathsf{i}(\vec{x})]\!]^{\mathbb{N}'}_\eta]$ $(1 \leq i \leq p)$.
One checks that $h[[\![\mathsf{f}_\mathsf{i}(\vec{x})]\!]^{\mathbb{N}'}_\eta \mapsto [\![\mathsf{f}_\mathsf{i}(\vec{x})]\!]^{\mathbb{M}}_\eta]$ $(1 \leq i \leq p)$ is a homomorphism from $\mathbb{N}'$ to $\mathbb{M}$. □

The following is a well-known result about the Chase: the set of all models generated by the Chase for a given geometric theory $\mathcal{G}$ is a set-of-support for $\mathcal{G}$. Note that the Chase might not halt, in which case, if the Chase is executed in a fair manner, the resulting infinite structure will be a model of $\mathcal{G}$.

Theorem 4.3.2. Let Σ be a signature, $\mathcal{G}$ be a geometric theory over Σ, and Σ^{w} be a witnessing signature obtained by extending Σ with the witness functions assigned to the existential quantifiers in $\mathcal{G}$. Theory $\mathcal{G}$ is satisfiable if and only if there exists a fair run of the Chase on $\mathcal{G}$ that does not fail. Let $\mathcal{S}$ be the set of all models obtained by some run of the Chase on $\mathcal{G}$ (here we include the possibility of infinite runs). For every model $\mathbb{M}$ of $\mathcal{G}$, a model $\mathbb{U} \in \mathcal{S}$ exists such that $\mathbb{U}^- \preccurlyeq \mathbb{M}$, where $\mathbb{U}^-$ is the witness-reduct of $\mathbb{U}$ to Σ.

Proof. We show that for every model $\mathbb{N}$ of $\mathcal{G}$, there exists a run ρ of the Chase that computes a model $\mathbb{M}$ with a homomorphism from the witness-reduct $\mathbb{M}^-$ of $\mathbb{M}$ to $\mathbb{N}$: Let $\mathbb{M}_0$ be the empty model. The empty function $h : |{\mathbb{M}_0}^-| \to |\mathbb{N}|$ is a trivial homomorphism from the witness-reduct ${\mathbb{M}_0}^-$ to $\mathbb{N}$.
Create a fair run ρ by iterating the Oracle (Lemma 4.3.1) on the structures generated by successive chase-steps starting from $\mathbb{M}_0$:

$$\rho = \mathbb{M}_0 \xrightarrow{(\sigma_0,\eta_0)} \mathbb{M}_1 \xrightarrow{(\sigma_1,\eta_1)} \dots \mathbb{M}_i \xrightarrow{(\sigma_i,\eta_i)} \mathbb{M}_{i+1} \xrightarrow{(\sigma_{i+1},\eta_{i+1})} \dots$$

If the Chase terminates with $\mathbb{M}_n$ after n steps, then the witness-reduct ${\mathbb{M}_n}^-$ is a model of $\mathcal{G}$, and ${\mathbb{M}_n}^- \preccurlyeq \mathbb{N}$ by Lemma 4.3.1. Alternatively, if the Chase does not terminate, the resulting structure $\mathbb{M}_\infty$ will be infinite. Let ${\mathbb{M}_\infty}^-$ be the witness-reduct of $\mathbb{M}_\infty$. Since the Oracle guarantees that ${\mathbb{M}_\infty}^- \preccurlyeq \mathbb{N}$, it suffices to show ${\mathbb{M}_\infty}^-$ is a model of $\mathcal{G}$; that is, for every sequent $\sigma \equiv \varphi \vdash_{\vec{x}} \psi$ in $\mathcal{G}$ and every environment $\eta : \vec{x} \longrightarrow |{\mathbb{M}_\infty}^-|$, then ${\mathbb{M}_\infty}^- \models_\eta \sigma$: observe that if ${\mathbb{M}_\infty}^- \models \varphi$, ${\mathbb{M}_i}^- \models \varphi$ for some i. Because ρ is a fair run of the Chase, then for some $j \geq i$, $\mathbb{M}_j \xrightarrow{(\sigma,\eta)} \mathbb{M}_{j+1}$. Finally, because $\mathbb{M}_{j+1}$ is a submodel of $\mathbb{M}_\infty$ and ψ is positive, ${\mathbb{M}_\infty}^- \models_\eta \psi$. □

Provenance

We define the provenance of elements and facts for chase-models. Razor does not directly construct models using the Chase; nevertheless, the Chase-based *grounding* algorithm presented in Section 5.2 computes provenance information for the elements and facts of the set of *possible facts*, which contains the ingredients of the models that Razor computes.

Observe that every element added to a chase-model (line 6 of Algorithm 1) is "named" by a closed term in the witnessing signature. This term is a provenance for that element. When the chase-step identifies two (distinct) elements as it processes an equation in the head of a sequent, the provenance of the resulting element is the set union of the two elements being equated.

Moreover, notice that every fact F added by a chase-step $\mathbb{M} \xrightarrow{(\sigma,\eta)} \mathbb{N}$ (line 7 of Algorithm 1) can be "blamed" in $\mathbb{N}$ on a pair (σ, η, i), written as $\mathrm{blame}_{\mathbb{N}}(F) = (\sigma, \eta, i)$, where i is the ordinal of the disjunct chosen by the chase-step (line 4 of Algorithm 1).

4.3.4 Termination and Decidability

In general, termination of the Chase on an arbitrary geometric theory is undecidable. [42]. However, Fagin *et al.* [43] suggest a syntactic condition on geometric theories, known as *weak acyclicity*, under which the Chase is guaranteed to terminate. Briefly, one constructs a directed graph, namely a *dependency graph* whose vertices are positions in relations and whose edges capture the possible "information flow" among positions. A theory is weakly acyclic if there are no cycles of a certain form in this graph. The notion of weakly acyclicity in [43] is defined for theories without disjunction, equality, and function symbols. However, the obvious extension of the definition to the general case supports the argument for termination in the general case.

Definition Let $\mathcal{G}$ be a geometric theory over a relational signature $\Sigma = (\Sigma^R, \emptyset)$. A dependency graph $D^{\mathcal{G}} = (\mathcal{V}, \mathcal{E}, \mathcal{E}^*)$ for $\mathcal{G}$—consisting of a set of vertices $\mathcal{V}$, a set of regular edges $\mathcal{E}$, and a set of *special* edges $\mathcal{E}^*$—is defined by the following:

- For each relation R of arity k in Σ^R, the pair (R, i) $(1 \leq i \leq k)$ is a vertex in $\mathcal{V}$.
- For every sequent $(\varphi \vdash_{\vec{x}} \bigvee_i \exists \vec{y} \,.\, \psi_i)$ in $\mathcal{G}$
  - for every occurrence of a variable $x \in \vec{x}$ at a position p of a relation R in φ, for every occurrence of x at a position q of a relation S in some ψ_i, $(R, p) \to (S, q)$ is an edge in $\mathcal{E}$.
  - for every occurrence of a variable $y \in \vec{y}$ at position q of a relation S in some ψ_i, for every occurrence of a variable $x \in \vec{x}$ at position p of a relation R in φ, $(R, p) \xrightarrow{*} (S, q)$ is a special edge in $\mathcal{E}^*$.

Weak Acyclicity

The theory $\mathcal{G}$ is said to be weakly acyclic if and only if $D^{\mathcal{G}}$ contains no cycle involving a special edge. It is a well-known result that if $D^{\mathcal{G}}$ is weakly acyclic, then every run of the Chase on $\mathcal{G}$ is terminating. Observe that if $\mathcal{G}$ 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 provides a set-of-support for all models of $\mathcal{G}$ 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; that is, a weakly acyclic theory is satisfiable if and only if it has a finite model. Furthermore, entailment of positive-existential sentences from a weakly acyclic theory is decidable, as follows. Suppose $\mathcal{G}$ is weakly acyclic and α is a positive-existential sentence. Let $\mathbb{M}_1, \ldots, \mathbb{M}_n$ be the models of $\mathcal{G}$. To check that α holds in all models of $\mathcal{G}$ it suffices to test α in each of the (finite) models $\mathbb{M}_i$, since if $\mathbb{N}$ were a counter-model for α, and $\mathbb{M}_i$ the chase-model such that $\mathbb{M}_i \preccurlyeq \mathbb{N}$, then $\mathbb{M}_i$ would be a counter-model for α, recalling

that positive-existential sentences are preserved by homomorphisms. This proof technique was used recently in [44] to show decidability for the theory of a class of Diffie-Hellman key-establishment protocols.

4.3.5 The Bounded Chase

For theories that do not guarantee termination of the Chase, we must resort to bounding the search. We describe three variations of a bounding strategy based on the notion of *Skolem depth* or simply the *depth* of search, which is inspired by the inductive nature of the Chase. The depth of search is a natural number d that bounds the depth of the witness terms constructed by the Chase in line 6 of Algorithm 1. We demonstrate the behavior of the three bounding strategies by an example in Section 4.3.6.

Partial Model Building

A straight-forward strategy is to simply refuse introducing *fresh* elements for witnessing terms at a depth greater than the bound d. That is, we return a submodel of a (possibly infinite) chase-model defined over the witnessing terms of depth less than or equal to d. Notice that the resulting *partial* model is not always a model of the input theory; however, every fact or element in this model is supported by a provenance. This is an optional strategy, namely the *pure* mode, which can be selected by the user to bound the search in Razor.

Uncontrolled Reuse

If we insist on constructing a (complete) model of the input theory within the given depth d—provided such a model exists—we can *reuse* an existing element witnessed by s ($\mathrm{depth}(s) \leq d$) instead of creating a fresh element for a witnessing term t where $\mathrm{depth}(t) > d$. Given n existing elements within the depth d, the "uncontrolled reuse" strategy induces n disjunctive choices for all possible reuses; thus, this method is inefficient for a naïve implementation of the Chase (see Section 8) that branches on disjunctions. Also, observe that the equations between witnessing terms, induced by reusing existing terms, cannot be justified with respect to the user's input theory.

A more radical "loop check" strategy is suggested by Baumgartner [45], which freely reuses elements without considering a depth d. Unlike the reuse strategy described above, loop check guarantees to return all existing models within depth d. However, we are interested in models that are as close to the homomorphically minimal ones as possible; thus, we avoid introducing "accidental" equations in order to preserve the "purity" of the models inside the given bound. It is noteworthy that by Theorem 4.3.2, for every model $\mathbb{N}$ that could be constructed by introducing arbitrary equations, the Chase guarantees to deliver at least a model $\mathbb{M}$ such that $\mathbb{M} \preccurlyeq \mathbb{N}$ if the bound d is sufficiently large.

Controlled Reuse

A more subtle reusing strategy equates elements witnessed by terms that are equivalent up to depth d. Say that two ground terms $s \equiv f(s_1, \ldots, s_m)$ and $t \equiv g(t_1, \ldots, t_n)$ are *d-equivalent,* written $s \approx_d t$, if they agree, as trees, up to depth d. There is a natural way to view the set of closed terms as a metric space, defining the distance between distinct terms s and t is 2^{-d} where d is the least depth such that $s \not\approx_d t$. If we fix a constant symbol c then every closed term has a canonical representative for its $\approx_d$-equivalence class: we simply truncate the term at depth d if necessary by replacing the level-d subterm by c.

When we apply the above ideas to the closed terms that serve as names for elements of a model built by the Chase, terms that are "close" in the metric above are those that share much of their name, and intuitively *behave similarly in the model*. The user can specify the maximum depth d, and when the Chase is about to create a new element with name t whose name-depth is greater than d, we instead reuse the element named by the depth-d term that is $\approx_d$-equivalent to t. Note that the two terms thus equated are at distance less than 2^{-d}.

The net result is that we construct a model in which terms have been identified that may not have been equal in a pure chase-model. But in a precise sense, we only identify elements that are close in the sense of the metric above. This is the default bounding strategy, namely the *reuse* mode, implemented into Razor.

4.3.6 Chase Examples

In this section, we provide examples of runs of the Chase on various geometric theories.

Weak Acyclicity

$$\begin{aligned} \vdash\ & \exists^{\mathsf{a},\mathsf{b}} x\ y\ .\ R(x,y) \\ R(x,w) \vdash\ & \exists^{\mathsf{f}} y\ .\ Q(x,y) \\ Q(u,v) \vdash\ & \exists^{\mathsf{g}} z\ .\ R(u,z) \end{aligned}$$

Figure 4.1 displays the dependency graph of the theory above. Since the dependency graph does not contain a cycle with a special edge (labelled with asterisks), the theory is clearly weakly acyclic. The Chase proceeds on this theory as follows: starting with an empty model, the Chase treats the first sequent. The Chase creates new element $\mathbf{e_1}$ and $\mathbf{e_2}$, respectively witnessed by a and b, and adds the fact $R(\mathbf{e_1}, \mathbf{e_2})$. Then the second sequent fails, so the Chase adds an element $\mathbf{e_3}$ to instantiate $\exists y$, witnessed by $\mathsf{f}(\mathsf{a}, \mathsf{b})$, and adds the fact $Q(\mathbf{e_1}, \mathbf{e_3})$. The third sequent is satisfied in the resulting model

$$\{R(\mathbf{e_1}, \mathbf{e_2}), Q(\mathbf{e_1}, \mathbf{e_3}), \mathsf{a} = \mathbf{e_1}, \mathsf{b} = \mathbf{e_2}, \mathsf{f}(\mathsf{a}, \mathsf{b}) = \mathbf{e_3}\}$$

thus, the procedure terminates with this model.

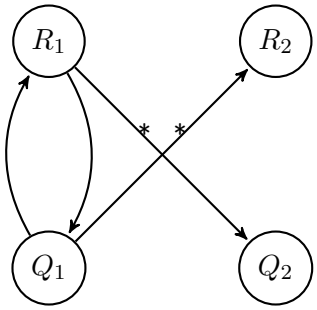

Figure 4.1: Dependency graph of the weakly acyclic example

Non-Terminating Chase This example is a very slight syntactic modification of the previous one (in the head of the third sequent). The dependency graph of this theory, illustrate by Figure 4.2, contains a cycle with special edges; thus, the theory is not weakly-acyclic.

$$\begin{aligned} &\vdash \exists^{\mathsf{a},\mathsf{b}} x\ y\ .\ R(x,y) \\ R(x,w) &\vdash \exists^{\mathsf{f}} y\ .\ Q(x,y) \\ Q(u,v) &\vdash \exists^{\mathsf{g}} z\ .\ R(z,v) \end{aligned}$$

In this example, the first two steps of the Chase are the same as before, yielding the model below:

$$\{R(\mathbf{e_1},\mathbf{e_2}), Q(\mathbf{e_1},\mathbf{e_3}), \mathsf{a}=\mathbf{e_1}, \mathsf{b}=\mathbf{e_2}, \mathsf{f}(\mathsf{a},\mathsf{b})=\mathbf{e_3}\}$$

But now the third sequent requires creation of a new element $\mathbf{e_4}$, witnessed by $\mathsf{g}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{b}))$, and the fact $R(\mathbf{e_4},\mathbf{e_3})$. Then the second sequent fails in the model

$$\{R(\mathbf{e_1},\mathbf{e_2}), Q(\mathbf{e_1},\mathbf{e_3}), R(\mathbf{e_4},\mathbf{e_3}), \mathsf{a}=\mathbf{e_1}, \mathsf{b}=\mathbf{e_2}, \mathsf{f}(\mathsf{a},\mathsf{b})=\mathbf{e_3}, \mathsf{g}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{b}))=\mathbf{e_4}\}$$

and so we add another element ... and it is easy to see that the Chase will never terminate on this theory.

Note, however, that the infinite run of the Chase that is induced does create an (infinite) model of the theory. As noted above, this is a general fact about fair Chase runs. The fact that the process creates models "in the limit" is a necessary component of the completeness proof: *unsatisfiable* theories will necessarily lead to finitely-failing computations.

Fairness This example demonstrates the importance of fairness for correctness of Theorem 4.3.2:

$$\begin{aligned} &\vdash \exists^{\mathsf{a},\mathsf{b}} x\ y\ .\ R(x,y) \\ R(x,y) &\vdash \exists^{\mathsf{f}} z\ .\ R(y,z) \\ R(x,y) &\vdash \bot \end{aligned}$$

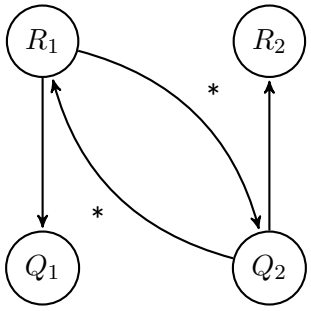

Figure 4.2: Dependency graph of the non-terminating example

Consider an "unfair" run of the Chase, which starts with the first sequent, then, continuously favors the second sequent over the last one: it repeatedly creates new facts $R(\mathbf{e_1}, \mathbf{e_2})$, $R(\mathbf{e_2}, \mathbf{e_3})$, $R(\mathbf{e_3}, \mathbf{e_4})$, ..., thus, never terminates. However, a fair run of the Chase will eventually treat the last sequent and terminate with failure.

The Bounded Chase Consider the theory from the previous example in the absence of the its last sequent:

$$\vdash \exists^{\mathsf{a},\mathsf{b}} x\ y\ .\ R(x, y)$$
$$R(x, y) \vdash \exists^{\mathsf{f}} z\ .\ R(y, z)$$

An unbounded run of the Chase on this theory is non-terminating. Let us explore the behavior the bounding strategies presented in Section 4.3.5 up to depth 2:
The following is a partial model over witnessing terms of less than or equal to 2:

$$\mathbb{M}_0 = \{R(\mathbf{e_1}, \mathbf{e_2}), R(\mathbf{e_2}, \mathbf{e_3}), R(\mathbf{e_3}, \mathbf{e_4}), \mathsf{a} = \mathbf{e_1}, \mathsf{b} = \mathbf{e_2}, \mathsf{f}(\mathsf{a}, \mathsf{b}) = \mathbf{e_3}, \mathsf{f}(\mathsf{b}, \mathsf{f}(\mathsf{a}, \mathsf{b})) = \mathbf{e_4}\}$$

The elements $\mathbf{e_1}$, $\mathbf{e_2}$, $\mathbf{e_3}$, and $\mathbf{e_4}$ are named by terms of depth 0, 0, 1, and 2 respectively. Clearly, $\mathbb{M}_0$ does not satisfy the given theory as the second sequent is not satisfied by the environment $\{x \mapsto \mathbf{e_3}, y \mapsto \mathbf{e_4}\}$. The uncontrolled reuse strategy results in four distinct models of this theory, induced by the four reusing options. Each model is an extension of the previous partial model, $\mathbb{M}_0$:

$$\mathbb{M}_1 \equiv \mathbb{M}_0 \cup \{R(\mathbf{e_4}, \mathbf{e_1}), \mathsf{f}(\mathsf{f}(\mathsf{a}, \mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{f}(\mathsf{a}, \mathsf{b}))) = \mathbf{e_1}\}$$
$$\mathbb{M}_2 \equiv \mathbb{M}_0 \cup \{R(\mathbf{e_4}, \mathbf{e_2}), \mathsf{f}(\mathsf{f}(\mathsf{a}, \mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{f}(\mathsf{a}, \mathsf{b}))) = \mathbf{e_2}\}$$
$$\mathbb{M}_3 \equiv \mathbb{M}_0 \cup \{R(\mathbf{e_4}, \mathbf{e_3}), \mathsf{f}(\mathsf{f}(\mathsf{a}, \mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{f}(\mathsf{a}, \mathsf{b}))) = \mathbf{e_3}\}$$
$$\mathbb{M}_4 \equiv \mathbb{M}_0 \cup \{R(\mathbf{e_4}, \mathbf{e_4}), \mathsf{f}(\mathsf{f}(\mathsf{a}, \mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{f}(\mathsf{a}, \mathsf{b}))) = \mathbf{e_4}\}$$

Observe that the reused element in each model is denoted by two distinct witnessing terms, evidencing an (accidental) equation among models. The resulting models are models of the input theory; however, they are not homomorphically minimal due to the reuse. The controlled reuse strategy only results in the

model $\mathbb{M}_4$ above: instead of introducing a new element for the witnessing term $t \equiv \mathsf{f}(\mathsf{f}(\mathsf{a},\mathsf{b}),\mathsf{f}(\mathsf{b},\mathsf{f}(\mathsf{a},\mathsf{b})))$, the element $\mathbf{e_4}$, which is named by $s \equiv \mathsf{f}(\mathsf{b},\mathsf{f}(\mathsf{a},\mathsf{b}))$ is reused because $\mathsf{f}(\mathsf{f}(\mathsf{a},\mathsf{b}),\mathsf{f}(\mathsf{b},\mathsf{f}(\mathsf{a},\mathsf{b}))) \approx_2 \mathsf{f}(\mathsf{b},\mathsf{f}(\mathsf{a},\mathsf{b}))$.

A Failure of Minimality To gain some intuition about minimality and disjunction, consider the (propositional) theory

$$\begin{aligned} &\vdash (A \vee B) \\ A &\vdash B \end{aligned}$$

The Chase can generate two models here, one of which fails to be minimal. What has gone wrong is that the disjunction is spurious, in the sense that one disjunct is subsumed by the other in the context of the rest of the theory. This little example makes it clear that any implementation of the Chase that explores disjuncts without doing global checks for the relationships among the models being constructed can end up creating models that are not minimal, essentially because the theory the user presented is not as "tight" as it might be.

This is an interesting phenomenon, since it is still the case that, even in a non-minimal model, no element or fact is created except when it is required: every element constructed by the Chase will be witnessed by a term in the witnessing signature.

Nevertheless, Razor's model-finding algorithm presented in Chapter 5 never returns non-minimal models as it generates models by a minimization algorithm that checks for minimal models globally.

Conference Management To see how disjunctions induce various branches of the Chase, consider the following theory from a conference management system:

$$\top \vdash \exists^{\mathsf{a}} x \,.\, \exists^{\mathsf{p}} y \,.\, \mathit{Author}(x) \wedge \mathit{Paper}(y) \wedge \mathit{Assigned}(x,y) \tag{4.1}$$

$$\mathit{Author}(x) \wedge \mathit{Paper}(y) \vdash \mathit{ReadScore}(x,y) \vee \mathit{Conflicted}(x,y) \tag{4.2}$$

$$\mathit{Assigned}(x,y) \wedge \mathit{Author}(x) \wedge \mathit{Paper}(y) \vdash \mathit{ReadScore}(x,y) \tag{4.3}$$

$$\mathit{Assigned}(x,y) \wedge \mathit{Conflicted}(x,y) \vdash \bot \tag{4.4}$$

The Chase initially starts with an empty model $\mathbb{M}_0$ (see Figure 4.3). Equation (4.1) introduces two elements $\mathbf{a_1}$ and $\mathbf{p_1}$ and adds $\mathit{Author}(\mathbf{a_1})$, $\mathit{Paper}(\mathbf{p_1})$, and $\mathit{Assigned}(\mathbf{a_1},\mathbf{p_1})$ to $\mathbb{M}_0$, resulting in a (partial) model $\mathbb{M}_1$. The witness constant a, assigned to the first existential quantifier of Equation (4.1), denotes $\mathbf{a_1}$ in $\mathbb{M}_1$. Also, the witness constant p, assigned to the second existential quantifier of Equation (4.1), denotes $\mathbf{p_1}$ in $\mathbb{M}_1$. Equation (4.2) states that for any pair of an author and a paper, either the author can read the paper's score, or there the author and the paper are conflicting. Therefore, two models $\mathbb{M}_{2,1}$ and $\mathbb{M}_{2,2}$ extend $\mathbb{M}_1$ in two separate chase-branches; $\mathit{ReadScore}(\mathbf{a_1},\mathbf{p_1})$ is true in $\mathbb{M}_{2,1}$ and $\mathit{Conflicted}(\mathbf{a_1},\mathbf{p_1})$ is true in $\mathbb{M}_{2,2}$. According to Equation (4.3), authors can read the scores of the papers to which they are assigned. This sequent does not impact $\mathbb{M}_{2,1}$ (since $\mathit{ReadScore}(\mathbf{a_1},\mathbf{p_1})$ is

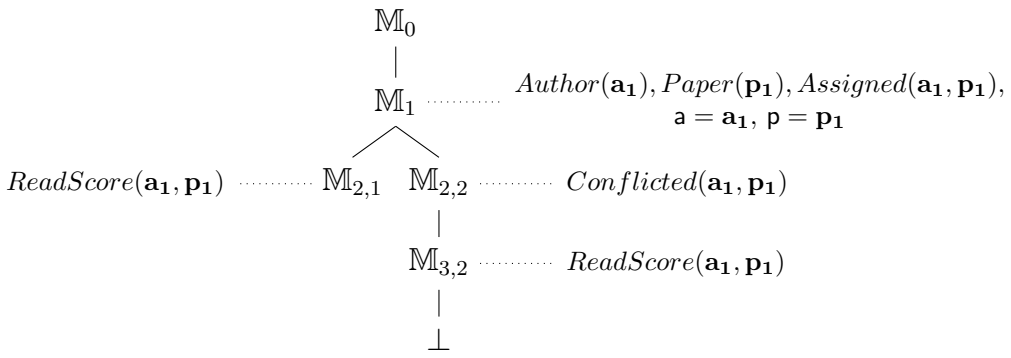

Figure 4.3: Runs of the Chase on the conference management example

already true) but it adds the fact $ReadScore(\mathbf{a_1}, \mathbf{p_1})$ to $\mathbb{M}_{2,2}$, resulting in a model $\mathbb{M}_{3,2}$. Finally, Equation (4.4) states that authors cannot be assigned to conflicting papers; thus, the second branch of the Chase (which contains $\mathbb{M}_{3,2}$) fails. That is, $\mathbb{M}_{2,1}$ is the only model of this theory that the Chase returns.

4.4 Augmentation in Geometric Logic

Let $\mathcal{G}$ be a geometric theory and $\mathbb{M}$ be a model of $\mathcal{G}$. The positivity of geometric logic allows us to conveniently *augment* $\mathbb{M}$ with an additional PEF α resulting in an extension $\mathbb{N}$ of $\mathbb{M}$ such that $\mathbb{N} \models \mathcal{G}$ and $\alpha \in \mathbb{N}$. A key point is that if α entails other observations given $\mathcal{G}$ and the facts already in $\mathbb{M}$, those observations will be added to the resulting model. The observation α can in principle be an arbitrary PEF referring to the elements of $\mathbb{M}$.

The model $\mathbb{N}$ can be computed by a run of the Chase starting with a model $\mathbb{M}' \equiv \mathbb{M} \cup \{\alpha\}$. The augmentation may *fail* if $\mathbb{M}$ is inconsistent with α according to $\mathcal{T}$.

Theorem 4.4.1. Let $\mathbb{N}$ be a finite model of the theory $\mathcal{G}$. 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 $\mathcal{G}$ 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 $\mathbb{N}$.

Proof. The homomorphism $h : \mathbb{M} \to \mathbb{N}$ can be factored into a surjection $h_0 : \mathbb{M} \to \mathbb{N}_0$ followed by a injection: $h_1 : \mathbb{N}_0 \to \mathbb{N}$. Without loss of generality we may assume that $\mathbb{N}_0$ is a quotient of $\mathbb{M}$ and h_1 is simple inclusion. The effect of h_0 can be captured by a sequence of equality augmentations between the elements of $\mathbb{M}$, yielding $\mathbb{N}_0$. A further sequence of primitive relation-augmentations will yield a model $\mathbb{N}_1$ isomorphic to the image of $\mathbb{N}_0$ in $\mathbb{N}$. A final sequence of augmentations

adding new elements and facts corresponding to $\mathbb{N} \setminus \mathbb{N}_0$ yields $\mathbb{N}$.

The second claim follows from the fact that when $\mathcal{G}$ is weakly acyclic there is a (finite) set-of-support for $\mathcal{G}$ consisting of finite models. $\square$

Chapter 5

Algorithm and Implementation

Our preliminary implementations of Razor (see Chapter 8) revealed to us that the major cost of a model-finding algorithm that is purely based on the standard Chase corresponds to branching on disjunctions in the head of sequents. In a naïve implementation of the Chase, the disjunctions in the head of sequents give rise to distinct branches of the Chase sharing the partial models that are constructed up to the branching points. In the worst case, the total number of branches induced by the disjunctions grows exponentially with the number of ground instances of sequents in the resulting model. In practice, however, a large portion of these branches fail, terminating with no models. Therefore, from a model-finding perspective, the major time of computation will be wasted on these inconsistent branches.

The Magic of SMT-Solving. We present an algorithm that takes advantage of SAT-solving technology to navigate the disjunctions. Using 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 $\mathcal{G}$ over a fixed set of constants. Rather we compute a ground theory $\mathcal{G}^*$ consisting of a sufficiently large set of instantiations of $\mathcal{G}$ by closed terms of the witness signature for $\mathcal{G}$.

Since we handle theories with equality, we want to construct models of $\mathcal{G}^*$ modulo equality reasoning and the uninterpreted theory of functions, so we actually pass to using an *SMT-solver.*

In Section 5.2, we present a *grounding* algorithm that constructs a *set of possible facts*, $\mathrm{Pos}(\mathcal{G})$, and a set $\mathcal{G}^*$ of ground instances of $\mathcal{G}$ over $\mathrm{Pos}(\mathcal{G})$. This algorithm is essentially a variation of the Chase, where disjunctions in the head of sequents are treated as conjunctions. In this way we represent all branches that could be taken in a chase-step over the original $\mathcal{G}$. Intuitively, the grounding algorithm creates a refined Skolem-Herbrand base, containing a set of *possible facts* $\mathrm{Pos}(\mathcal{G})$, which could be true in any Chase model of $\mathcal{G}$. Consequently, we construct models of $\mathcal{G}$ which are built only from the facts in $\mathrm{Pos}(\mathcal{G})$.

Minimization. But we need to do more than simply use a refined set of domain elements. A naive use of SMT-solving would result in losing control over the model-building: even though the elements of a model 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 for the model to satisfy $\mathcal{G}^*$, and even "un-collapse" elements when possible.

The minimization algorithm utilizes an SMT-solver in a way that is comparable to that of Aluminum (see Chapter 7). Unlike Aluminum's minimization procedure, Razor's algorithm works in the presence of equality; that is, it eliminates the unnecessary identification of elements. As a consequence, Razor constructs models that are minimal under the homomorphism ordering.

The chapter is organized as follows: in Section 5.1, we present a number of transformations to convert an arbitrary theory in geometric form into one that can be processed by the grounding algorithm. Next, in Section 5.2, we present the grounding algorithm, and we state a soundness and a completeness theorem about it. Section 5.3 explains our model-finding and minimization algorithms based on SMT-solving. Finally, in Section 5.4, we connect the grounding and the minimization algorithms to show how Razor generates models, and in Section 5.5, we discuss various aspects of our implementation of Razor in Haskell.

5.1 Transformations

The grounding algorithms described in Section 5.2 works with relational (function-free) signatures. Nevertheless, we allow function symbols in the concrete signature as syntactic sugar for users. In a preprocessing phase, the user's input theory over a standard first-order theory is transformed to an equisatisfiable theory over a relational signature by standard relationalization.

Moreover, the grounding algorithm requires the geometric sequents over the relational signature to be

1. *range-restricted:* every free variable in the head of a sequent must appear in the body of the sequent as well.

2. *body-linear:* the body of a sequent is *linear* (see Section 5.1.2) with respect to the free variables in the sequent.

Furthermore, the minimization algorithm presented in Section 5.3 operates on theories that are ground and flat; the input to the minimization algorithm may contain function symbols but the use of complex terms is prohibited.

In this section, we present transformations for converting formulas to the forms that can be processed by the grounding and the minimization algorithms. The relationalization (Section 5.1.1), linearization, and range-restriction (Section 5.1.2)

transformations put the user's theory into a form that can be fed to the grounding algorithm. The term-hashing conversion (Section 5.1.3) prepares the ground theory, computed by the grounding algorithm, for minimization.

5.1.1 Relationalization

By relationalization, an input first-order formula $\mathcal{T}$ over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ is transformed to an equisatisfiable formula $\mathcal{T}_{\mathsf{Rel}}$ over a signature $\Sigma_{\mathsf{Rel}} \equiv (\Sigma^R_{\mathsf{Rel}}, \emptyset)$. Relationalization is done in two steps:

- $\mathcal{T}$ over Σ is transformed to a *flat* theory $\mathcal{T}_{\mathsf{Flat}}$ over Σ, which is logically equivalent to $\mathcal{T}$.
- $\mathcal{T}_{\mathsf{Flat}}$ over Σ is transformed to a relational theory $\mathcal{T}_{\mathsf{Rel}}$ over Σ_{Rel}, equisatisfiable to $\mathcal{T}_{\mathsf{Flat}}$.

Flattening Transformations. *Flattening* consists of repeated applications of the following transformations on a first-order formula over a signature Σ until no more transformations is applicable:

- $R(\ldots, f(\vec{t}), \ldots) \rightsquigarrow \exists x \,.\, R(\ldots, x, \ldots) \wedge f(\vec{t}) = x$
  where R is a Σ-relation other than $=$ and $\vec{t}$ is a list of Σ-terms.
- $f(\vec{t}) = t \rightsquigarrow \exists x \,.\, x = t \wedge f(\vec{t}) = x$
  where t is a closed Σ-term and $\vec{t}$ is a list of Σ-terms.
- $u = f(\vec{t}) \rightsquigarrow \exists x \,.\, u = x \wedge f(\vec{t}) = x$
  where u is a closed Σ-term and $\vec{t}$ is a list of Σ-terms.
- $x = f(\vec{t}) \rightsquigarrow f(\vec{t}) = x$
  where $\vec{t}$ is a list of Σ-terms and x is a variable.

The last transformation orients equations so that function symbols (including constants) only appear on the left of equations. Also, notice that every relation (except equality) and every function symbol is only applied on variable terms in the resulting theory.

The flattening transformation is a standard procedure, widely employed by theorem-provers and model-finders to convert formulas into a form that is more convenient for equality reasoning [21, 46–49]. It is a well-known result that the resulting flattened formula is logically equivalent to the input formula.

Relationalization Transformation. The *Relationalization* transformation below converts a flattened formula (obtained by the previous transformations) over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ to a relational formula over a signature $\Sigma_{\mathsf{Rel}} \equiv (\Sigma^R \cup \mathcal{F}, \emptyset)$:

$$f(\vec{x}) = y \rightsquigarrow F(\vec{x}, y)$$

Here, $F \in \mathcal{F}$ is a fresh relation symbol of arity $k+1$ assigned to the k-ary function symbol $f \in \Sigma^R$. Observe that this converts a constant $c \in \Sigma^F$ to a unary relation symbol $C \in \mathcal{F}$.

It can be shown that the previous transformation converts a first-order theory $\mathcal{T}$ over Σ to an equisatisfiable theory $\mathcal{T}_{\mathsf{Rel}}$ over Σ_{Rel} in presence of additional congruence axioms.

5.1.2 Linearization and Range Restriction

The linearization and range-restriction transformations convert the sequents of a relational theory into a form that can be processed by the grounding algorithm.

Linearization

A first-order formula φ is said to be *linear* if every variable in $\mathrm{Vars}(\varphi)$ appears in exactly one non-equational position in φ.

The following transformations converts a first-order formula to a logically equivalent linear formula:

- $P(\dots, x, \dots, x, \dots) \rightsquigarrow P(\dots, x, \dots, y, \dots) \wedge (x = y)$
- $\dots P(\dots, x, \dots) \dots Q(\dots, x, \dots) \dots \rightsquigarrow$
  $\dots P(\dots, x, \dots) \dots Q(\dots, y, \dots) \wedge (x = y) \dots$

where P and Q are relation symbols other than $=$ and y is a fresh variable such that $y \notin \mathrm{Vars}(\varphi)$.

Range-Restriction Transformations

A sequent $\varphi \vdash_{\vec{x}} \psi$ as an implication between φ and ψ is said to be *range-restricted* if every variable in $\vec{x}$ appears in $\mathrm{FV}(\varphi)$. It is often convenient to work with range-restricted transformations. In a variety of applications, where implications are read as rules for forward reasoning, range-restriction transformations are used to convert implications to range-restricted ones over an expanded signature [50, 51].

Here, we introduce a set of range-restriction transformations on geometric sequents, suitable for the grounding algorithm in Section 5.2: let $\Sigma \equiv (\Sigma^R, \emptyset)$ be a relational signature and Dom be a fresh unary relation symbol not in Σ^R. The result of applying the following transformations on a sequent $\sigma \equiv \varphi \vdash_{\vec{x}} \psi$ over Σ ($\vec{x} \equiv \langle x_1, \dots, x_n \rangle$), is a *range-restricted* sequent τ over $\Omega \equiv (\Sigma^R \cup \{Dom\}, \emptyset)$:

1. $\varphi \vdash \psi \rightsquigarrow \varphi \wedge Dom(x_1) \cdots \wedge Dom(x_n) \vdash \psi$
2. $\varphi \vdash (\exists x \,.\, \alpha) \vee \psi \rightsquigarrow \varphi \vdash (\exists x \,.\, Dom(x) \wedge \alpha) \vee \psi$

The first transformation turns the input into a range-restricted sequent. The second transformation makes the predicate *Dom* true for every element of a model of the resulting sequent. These transformations are sound and complete: the sequent σ is satisfiable if and only if τ is. Moreover, the reduct of every model of τ to Σ will be a model of σ [50, 52].

5.1.3 Term-Hashing

Term-hashing is a process by which complex terms in an input formula are replaced by (simple) constant terms.

Definition Fix a set of function symbols $\mathcal{F}$ and a set of constants $\mathcal{K}$. A *term-hash* function $h : \text{Terms}(\mathcal{F}) \longrightarrow \mathcal{K}$ is an injective function that maps every term over $\mathcal{F}$ to a unique constant from $\mathcal{K}$.

We may lift a term-hash function to act on first-order formulas in the standard way. We also lift a term-hash function $h : \Sigma^F \longrightarrow \mathcal{K}$ to send a model $\mathbb{M}^\#$ over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$ and domain $|\mathbb{M}|$ to a model $\mathbb{M}^\#$ over $\Sigma^\# \equiv (\Sigma^R, \mathcal{K})$ and domain $|\mathbb{M}|$ such that

- $\mathbb{M} \models_\eta P(\vec{t})$ if and only if $\mathbb{M}^\# \models_\eta P(h(\vec{t}))$.
- $\mathbb{M} \models_\eta t = s$ if and only if $\mathbb{M}^\# \models_\eta h(t) = h(s)$.

Theorem 5.1.1. Fix a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$, a set of constants $\mathcal{K}$ disjoint from Σ^F, and a hash-term function $h : \text{Terms}(\Sigma^R) \longrightarrow \mathcal{K}$. Given a first-order formula φ and a model $\mathbb{M}$ over Σ, $\mathbb{M} \models_\eta \varphi$ if and only if $h(\mathbb{M}) \models_\eta h(\varphi)$

The proof of Theorem 5.1.1 is straight forward by induction on first-order formulas.

5.2 Grounding

In this section, we present a *grounding* algorithm for converting a geometric theory $\mathcal{G}$ to a ground geometric theory $\mathcal{G}^*$. We assume that the sequents in $\mathcal{G}$ are relational and range-restricted. We also assume the bodies of the sequents in $\mathcal{G}$ to be linear.

5.2.1 Grounding Algorithm

Fix a relational signature $\Sigma \equiv (\Sigma^R, \emptyset)$. Let $\mathcal{G}$ be a theory over Σ with witness functions from a set $\mathcal{F}$ associated to its existential quantifiers. Algorithm 3 constructs a set of *possible facts* $\text{Pos}(\mathcal{G})$ together with a corresponding "ground" instance $\mathcal{G}^*$ of $\mathcal{G}$ over the witnessing signature $\Sigma^{\text{w}} \equiv (\Sigma^R, \mathcal{F})$.

Algorithm 3 Grounding

function GROUND($\mathcal{G}$)
  $\text{Pos}(\mathcal{G}) \leftarrow \emptyset$ ▷ $\text{Pos}(\mathcal{G})$ is initially the empty model
  $\mathcal{G}^* \leftarrow \emptyset$ ▷ $\mathcal{G}^*$ is initially an empty theory
  do
    choose $\sigma \equiv \varphi \vdash_{\vec{x}} \psi \in \mathcal{G}$
    for each λ where $\varphi[\lambda\vec{x}] \in \text{Pos}(\mathcal{G})$ **do** ▷ λ sends $\vec{x}$ to terms in $\text{Pos}(\mathcal{G})$
      $\text{Pos}(\mathcal{G}) \leftarrow$ EXTEND($\text{Pos}(\mathcal{G})$, σ, λ)
      $\mathcal{G}^* \leftarrow \mathcal{G}^* \cup \{\text{INSTANTIATE}(\sigma, \lambda)\}$
  while $\mathcal{G}^*$ and $\text{Pos}(\mathcal{G})$ are changing
  return ($\mathcal{G}^*$, $\text{Pos}(\mathcal{G})$)

function EXTEND($\mathcal{F}, \varphi \vdash_{\vec{x}} \psi, \eta$) ▷ $\mathcal{F}$ is a set of facts
  if $\psi = \bot$ **then** $\mathcal{F}$
  $\mathcal{H} \leftarrow \mathcal{F}$
  for each disjunct $\exists^{\mathsf{f}_1} y_1, \ldots, \exists^{\mathsf{f}_m} y_m \,.\, \bigwedge_{j=1}^{n} P_j(\vec{x}, \vec{y})$ in ψ **do**
    $\mu \leftarrow \eta[y_1 \mapsto \mathsf{f}_1(\eta\vec{x}), \ldots, y_m \mapsto \mathsf{f}_m(\eta\vec{x})]$
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{P_1[\mu\vec{x}, \mu\vec{y}], \ldots, P_n[\mu\vec{x}, \mu\vec{y}]\}$
    return $\mathcal{H}$

function INSTANTIATE($(\varphi \vdash_{\vec{x}} \bigvee_i \exists^{\mathsf{f}_{i1}} y_{i1} \ldots \exists^{\mathsf{f}_{im}} y_{im}.\psi_i)$, η)
  $\mu \leftarrow \eta[y_{ij} \mapsto \mathsf{f}_{ij}(\eta\vec{x})]$ $(1 \leq j \leq m)$
  return $\mu\sigma$

Example Consider the conference management example from Section 4.3.6. The grounding algorithm computes the set of possible facts ($\text{Pos}(\mathcal{G})$ in Algorithm 3) below (notice that a and p are witness constants, assigned to the set of existential quantifiers of the theory:

$$\{Author(\mathsf{a}), Paper(\mathsf{p}), Assigned(\mathsf{a}, \mathsf{p}), ReadScore(\mathsf{a}, \mathsf{p}), Conflicted(\mathsf{a}, \mathsf{p})\}$$

This set consists of all facts that could be true in a chase-model of this theory. Notice that this set even contains the facts that were computed in the failing branch of the Chase during its execution. Accordingly, the following set of ground sequents ($\mathcal{G}^*$ in Algorithm 3) is computed over the aforementioned set of possible facts:

$$\vdash Author(\mathsf{a}) \wedge Paper(\mathsf{p}) \wedge Assigned(\mathsf{a}, \mathsf{p})$$
$$Author(\mathsf{a}) \wedge Paper(\mathsf{p}) \vdash ReadScore(\mathsf{a}, \mathsf{p}) \vee Conflicted(\mathsf{a}, \mathsf{p})$$
$$Assigned(\mathsf{a}, \mathsf{p}) \wedge Author(\mathsf{a}) \wedge Paper(\mathsf{p}) \vdash ReadScore(\mathsf{a}, \mathsf{p})$$
$$Assigned(\mathsf{a}, \mathsf{p}) \wedge Conflicted(\mathsf{a}, \mathsf{p}) \vdash \bot$$

We present a soundness and a completeness theorem by which it is possible to compute models of a theory $\mathcal{G}$ by computing models for the ground theory $\mathcal{G}^*$,

computed by the grounding algorithm. By the soundness theorem (Theorem 5.2.2), the witness-reducts of models of $\mathcal{G}^*$ to the original signature of $\mathcal{G}$ are models of $\mathcal{G}$. The completeness theorem (Theorem 5.2.3) states that every chase-model of $\mathcal{G}$ is a model of $\mathcal{G}^*$ in the expanded witnessing signature.

These theorems may be thought of as a "controlled" variation of Herbrand's theorem for constructing models of a geometric theory $\mathcal{G}$. The Herbrand base of a theory is computed over a set of constants that is fixed *a priori*. In contrast, the grounding algorithm presented here constructs a *Skolem-Herbrand base* (*i.e.*, the set of possible facts), which is inductively defined by an operational reading of the sequents in $\mathcal{G}$. In other words, the construction of the set of possible facts is controlled, allowing only for facts that are necessary for some model of $\mathcal{G}$. In fact, the controlled construction of the set of possible facts is what enables us to compute provenance information.

Before stating the soundness and the completeness theorems, we define the notion of *image*, a map from a set of facts to a model, used to restrict the information in the model to the set of facts.

Definition Fix a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$. Let $\mathcal{F}$ be a set of facts closed under the equality axioms, and $\mathbb{M}$ be a model over Σ. We say $\mathbb{M}$ is an *image* of $\mathcal{F}$ under a map $i : \text{Terms}(\Sigma^F) \longrightarrow |\mathbb{M}|$ if the following conditions hold:

- For every element $\mathbf{e} \in |\mathbb{M}|$, a term $t \in \text{Terms}(\Sigma^F)$ exists such that $\mathbf{e} = i(t)$.
- If $R^{\mathbb{M}}(\vec{\mathbf{e}})$ is true in $\mathbb{M}$ then $\vec{t} \in \text{Terms}(\Sigma^F)$ exist, such that $\vec{\mathbf{e}} = i(\vec{t})$ and $R[\vec{t}] \in \mathcal{F}$.
- If $i(t) = i(s)$ in $\mathbb{M}$, then $t = s \in \mathcal{F}$.

One might think of $\mathcal{F}$ as a set of facts that may possibly be true. The image $\mathbb{M}$ of $\mathcal{F}$ is restricted to the information in $\mathcal{F}$; it at most contains the information in $\mathcal{F}$. Notice that every identification among the elements of $\mathbb{M}$ also must respect the equational facts in $\mathcal{F}$.

Lemma 5.2.1. Fix a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$. Let $\mathcal{F}$ be a set of facts over Σ and $\mathbb{M}$ be an image of $\mathcal{F}$ under a map i. Let $\varphi(\vec{x})$ be a linear relational PEF over Σ such that $\mathbb{M} \models_\eta \varphi$. A map $\lambda : \vec{x} \longrightarrow \text{Terms}(\Sigma^F)$ exists such that $\varphi[\lambda\vec{x}] \in \mathcal{F}$ and $\eta = i \circ \lambda$.

Proof. Proof by induction on φ:

- Assume $\varphi(\vec{x})$ to be a non-equational atomic formula $R(\vec{x})$. Let $\vec{\mathbf{e}} = \eta\vec{x}$. Because $\mathbb{M}$ is an image of $\mathcal{F}$ under i, for a list $\vec{t}$ of terms such that $\vec{\mathbf{e}} = i(\vec{t})$, $R[\vec{t}] \in \mathcal{F}$. Let λ send $\vec{x}$ to $\vec{t}$. Such λ is well-defined: because φ is linear, cardinality of $\vec{t}$ is at most equal to cardinality of $\vec{x}$. Notice that $\eta = i \circ \lambda$.

- Assume $\varphi(\vec{x})$ be an equation $x = y$. Let $\mathbf{e_1} = \eta x$, $\mathbf{e_2} = \eta y$, $\mathbf{e_1} = i(t)$ and $\mathbf{e_2} = i(s)$. By the definition of image $t = s \in \mathcal{F}$. Construct λ such that it sends x to t and y to s. Clearly $\eta = i \circ \lambda$.

The lemma is preserved by the connectives in φ:

- Let $\varphi(\vec{x})$ be $\alpha(\vec{x}) \wedge \beta(\vec{x})$. Assume that for a map λ where $\eta = i \circ \lambda$, $\alpha[\lambda\vec{x}] \in \mathcal{F}$ and $\beta[\lambda\vec{x}] \in \mathcal{F}$. It follows that $\varphi[\lambda\vec{x}] \in \mathcal{F}$.
- Let $\varphi(\vec{x})$ be $\alpha(\vec{x}) \vee \beta(\vec{x})$. Assume that for a map λ where $\eta = i \circ \lambda$, $\alpha[\lambda\vec{x}] \in \mathcal{F}$ and $\beta[\lambda\vec{x}] \in \mathcal{F}$. It follows that $\varphi[\lambda\vec{x}] \in \mathcal{F}$.
- Let $\varphi(\vec{x})$ be $\exists y \,.\, \alpha(\vec{x})$. Assume that a map λ and a term $t \in \text{Terms}(\Sigma^F)$ exist such that $\eta = i \circ \lambda$ and $\alpha[\lambda[y \mapsto t]] \in \mathcal{F}$. It is easy to see that $\varphi(\lambda\vec{x}) \in \mathcal{F}$.

□

Theorem 5.2.2 (Soundness)**.** Fix a relational signature Σ and its corresponding witnessing signature Σ^{w}. Let $\mathcal{G}$ be a relational theory Σ. Let $\text{Pos}(\mathcal{G})$ be the set of possible facts and $\mathcal{G}^*$ be the ground theory, constructed by Algorithm 3, for $\mathcal{G}$. Let $\mathbb{M}$ be a model of $\mathcal{G}^*$ over Σ^{w}, and $\mathbb{M}^-$ be the witness-reduct of $\mathbb{M}$. If $\mathbb{M}$ is an image of $\text{Pos}(\mathcal{G})$ then $\mathbb{M}^-$ is a model of $\mathcal{G}$.

Proof. Let $\sigma \equiv \varphi \vdash_{\vec{x}} \bigvee_i \exists^{\mathsf{f}_{i1}} y_{i1} \ldots \exists^{\mathsf{f}_{im}} y_{im}.\psi_i$ be a sequent in $\mathcal{G}$. We show that if $\mathbb{M}^- \models_\eta \varphi$, then $\mathbb{M}^- \models_\eta (\bigvee_i \exists^{\mathsf{f}_{i1}} y_{i1} \ldots \exists^{\mathsf{f}_{im}} y_{im}.\psi_i)$: letting g be the image map from $\text{Pos}(\mathcal{G})$ to $\mathbb{M}$, because φ is a linear PEF, by Lemma 5.2.1 a map λ exists such that $\eta = g \circ \lambda$ and $\varphi[\lambda\vec{x}] \in \text{Pos}(\mathcal{G})$.
Because $\varphi[\lambda\vec{x}] \in \text{Pos}(\mathcal{G})$, a ground sequent $\varphi[\vec{t}] \vdash \bigvee_i \psi_i[\vec{t}, \vec{u}_i]$ exists in $\mathcal{G}^*$ (line 8 of Algorithm 3) where $\vec{t} = \lambda\vec{x}$, and for each u_{ij} in $\vec{u}_i$, $u_{ij} = \mathsf{f}_{\mathsf{ij}}(\vec{t})$ $(1 \leq j \leq m)$. Let $\vec{\mathbf{e}} \equiv \eta\vec{x}$ be a tuple in $\mathbb{M}^-$. Because $\eta = g \circ \lambda$, it checks that $\vec{\mathbf{e}} = g\vec{t}$; therefore, $\mathbb{M} \models \varphi[\vec{t}]$. Finally, since $\mathbb{M}$ is a model of $\mathcal{G}^*$, for some disjunct j, $\mathbb{M} \models \psi_j[\vec{t}, \vec{u}_j]$. It follows that $\mathbb{M}^- \models_\eta (\exists y_{j1}, \ldots, \exists y_{jm}.\psi_j)$, therefore, $\mathbb{M}^- \models_\eta (\bigvee_i \exists y_{i1}, \ldots, \exists y_{im}.\psi_i)$. □

Theorem 5.2.3 (Completeness)**.** Fix a relational theory $\mathcal{G}$ and a ground theory $\mathcal{G}^*$ for $\mathcal{G}$, computed by Algorithm 3. If $\mathbb{M}$ is a chase-model of $\mathcal{G}$ (over the witnessing signature for $\mathcal{G}$) then it is a model of $\mathcal{G}^*$.

Proof. Let $\sigma^* \equiv \varphi[\vec{t}] \vdash \bigvee_i \psi_i[\vec{t}, \vec{u}_i]$ be a sequent in $\mathcal{G}^*$. By definition, σ^* is an instance of a sequent $\sigma \equiv \bigvee_i \varphi \vdash_{\vec{x}} \exists^{\mathsf{f}_{i1}} y_{i1} \ldots \exists^{\mathsf{f}_{im}} y_{im} \,.\, \psi_i$ by a substitution that sends $\vec{x}$ to $\vec{t}$ and $\vec{y}_i$ to $\vec{u}_i$ $(\vec{y}_i \equiv \langle y_{i1}, \ldots, y_{im} \rangle)$. Notice that according to Algorithm 3, for each u_{ij} in $\vec{u}_i$ $(1 \leq i \leq m)$, $u_{ij} = \mathsf{f}_{\mathsf{ij}}(\vec{t})$.
Assume $\mathbb{M} \models \varphi[\vec{t}]$. Then, for the reduct $\mathbb{M}^-$ of $\mathbb{M}$ to the signature of $\mathcal{G}$, $\mathbb{M}^- \models_\eta \varphi(\vec{x})$; the elements $\vec{\mathbf{e}}$ are denoted by $\vec{t}$ in $\mathbb{M}$ (*i.e.*, $[\![\vec{t}]\!]_\eta^{\mathbb{M}} = \vec{\mathbf{e}}$). Because $\mathbb{M}$ is a chase-model for $\mathcal{G}$, then for some i, $\mathbb{M}^- \models_\lambda (\exists y_{i1} \ldots \exists y_{im} \,.\, \psi_i)$ where $\lambda = \eta[y_{ij} \mapsto \mathbf{d_j}]$ $(1 \leq j \leq m)$. Let u_{ji} denote every element $\mathbf{d_j}$ in $\mathbb{M}$ $([\![u_{ij}]\!]_\eta^{\mathbb{M}} = \mathbf{d_j})$. Therefore, $\mathbb{M} \models \psi_i[\vec{t}, \vec{u}_i]$. □

5.3 Model-Finding and Minimization

This section presents an algorithm for computing models of a first-order theory that are minimal with respect to the homomorphism ordering. The minimization algorithm described here has been inspired by that of Aluminum, extending Aluminum's algorithm by allowing equality in the input theory. As a result of this extension, the minimization algorithm constructs models that are minimal under the homomorphism ordering.

We assume the input to the minimization algorithm to be ground and flat. A ground theory $\mathcal{G}^*$ that is computed by by Algorithm 3 for an input theory $\mathcal{G}$, after term-hashing (see Section 5.1.3) satisfies these conditions.

Definition Let $\mathbb{M}$ be a model over a signature $\Sigma \equiv (\Sigma^R, \mathcal{K})$, where $\mathcal{K}$ is a set of constants. The *negation preserving axiom* $\text{Neg}(\mathbb{M})$ for $\mathbb{M}$ is a conjunction of literals as follows (Algorithm 4):

- For all k-ary relation symbols $R \in \Sigma^R$ and all constants $c_i \in \mathcal{K}$ $(1 \leq i \leq k)$, $\neg R(c_1, \ldots, c_k)$ is a conjunct in $\text{Neg}(\mathbb{M})$ if and only if $R^{\mathbb{M}}(c_1^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}})$ is not true in $\mathbb{M}$.
- For all constants $c, d \in \mathcal{K}$, $c \neq d$ is a conjunct in $\text{Neg}(\mathbb{M})$ if and only if $c^{\mathbb{M}} \neq d^{\mathbb{M}}$ in $\mathbb{M}$.
- For all k-ary function symbols $f \in \Sigma^F$, all constants $c_i \in \mathcal{K}(1 \leq i \leq k)$, and all constants $d \in \mathcal{K}$, $f(c_1, \ldots, c_k) \neq d$ is a conjunct in $\text{Neg}(\mathbb{M})$ if and only if $f^{\mathbb{M}}(c_1^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}}) \neq d^{\mathbb{M}}$ in $\mathbb{M}$.

The negation preserving axiom about a model $\mathbb{M}$ is a way of recording the negative information in $\mathbb{M}$ syntactically.

Algorithm 4 Negation Preserving Axioms

```
1: function NegPreserve(𝕄)
2:     α ← ⋀{¬R(c_1, ..., c_k) | R^𝕄(c_1^𝕄, ..., c_k^𝕄) ∉ 𝕄}
3:     β ← ⋀{c ≠ d | c^𝕄 ≠ d^𝕄}
4:     γ ← ⋀{f(c_1, ..., c_k) ≠ d | f^𝕄(c_1^𝕄, ..., c_k^𝕄) ≠ d^𝕄}
5:     return α ∧ β ∧ γ
```

Definition Let $\mathbb{M}$ be a model over signature $\Sigma = (\Sigma^R, \mathcal{K})$, where $\mathcal{K}$ is a set of constants. A *chip axiom* $\text{Chip}(\mathbb{M})$ about $\mathbb{M}$ is a *disjunction* of literals as follows (Algorithm 5):

- For all k-ary relation symbols $R \in \Sigma^R$ and all constants $c_i \in \mathcal{K}(1 \leq i \leq k)$, $\neg R(c_1, \ldots, c_k)$ is a disjunct in $\text{Chip}(\mathbb{M})$ if and only if $R^{\mathbb{M}}(c_1^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}})$ is true in $\mathbb{M}$.

- For all constants $c, d \in \Sigma^F$, $c \neq d$ is a disjunct in Chip($\mathbb{M}$) if and only if $c^{\mathbb{M}} = d^{\mathbb{M}}$ in $\mathbb{M}$.

- For all k-ary function symbols $f \in$, all constants $c_i \in \mathcal{K}(1 \leq i \leq k)$, and all constants $d \in \mathcal{K}$, $f(c_1, \ldots, c_k) \neq d$ is a disjunct in Chip($\mathbb{M}$) if and only if $f^{\mathbb{M}}(c_1^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}}) = d^{\mathbb{M}}$ in $\mathbb{M}$,

The chip axiom about $\mathbb{M}$ characterizes a model in which some positive fact in $\mathbb{M}$ is not true. A model that resides below $\mathbb{M}$ under the homomorphism ordering on models must satisfy both the negation preserving and the chip axioms about $\mathbb{M}$. This constitutes the core idea for the minimization algorithm, described in Section 5.3.1.

Algorithm 5 Chip Axioms

1: **function** CHIP($\mathbb{M}$)
2: $\quad \alpha \leftarrow \bigvee\{\neg R(c_1, \ldots, c_k) \mid R^{\mathbb{M}}(c_1^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}}) \in \mathbb{M}\}$
3: $\quad \beta \leftarrow \bigvee\{c \neq d \mid c^{\mathbb{M}} = d^{\mathbb{M}}\}$
4: $\quad \gamma \leftarrow \bigvee\{f(c_1, \ldots, c_k) \neq d \mid f^{\mathbb{M}}(c_1^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}}) = d^{\mathbb{M}}\}$
5: $\quad$ **return** $\alpha \vee \beta \vee \gamma$

5.3.1 Minimization

Algorithm 6 illustrates a *reduction* step on a model $\mathbb{M}$ of a theory $\mathcal{T}$. The essence of the reduction step is to extend the theory $\mathcal{T}$ with the negation preserving and chip axioms of $\mathbb{M}$, followed by SMT-solving, to construct a model $\mathbb{N}$ of $\mathcal{T}$ in a way that $\mathbb{N} \prec \mathbb{M}$ under the homomorphism ordering. If such a model exists, it will be returned; otherwise, the reduction step fails with **unsat**, informing that $\mathbb{M}$ is in fact homomorphically minimal.

The *minimization* algorithm (Algorithm 7) on a model $\mathbb{M}$ consists of repeated applications of the reduction step until the resulting model $\mathbb{N}$ cannot be further reduced.

Before we present Theorem 5.3.2 about the reduction step, we give a definition for the *Skolem-hulls* of a first-order models, and we state Lemma 5.3.1 that will be used in the proof of Theorem 5.3.2.

Definition Let $\mathbb{M}$ be a first-order model over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$. The Skolem-hull of $\mathbb{M}$ is the sub-model $\hat{\mathbb{M}}$ of $\mathbb{M}$ where every element is named by a term in Terms(Σ). More precisely, $\hat{\mathbb{M}}$ is the least structure such that for every constant $c \in \Sigma^F$, $c^{\mathbb{M}} \in |\hat{\mathbb{M}}|$. Also, $\hat{\mathbb{M}}$ is closed under the interpretation of the functions in Σ^F and the relations in Σ^R.

Intuitively, $\hat{\mathbb{M}}$ is the sub-model of $\mathbb{M}$ defined over those elements in $|\mathbb{M}|$ that are named by the constants in the signature. From a model-finding perspective, $\hat{\mathbb{M}}$

Algorithm 6 Reduce

Require: $\mathbb{M} \models \mathcal{T}$
Ensure: $\mathbb{N} \prec \mathbb{M}$

```
1: function Reduce(𝒯, 𝕄)
2:     ν ← NegPreserve(𝕄 )
3:     φ ← Chip(𝕄 )
4:     if exists ℕ such that ℕ ⊨ 𝒯 ∪ {ν, φ} then        ▷ Ask the solver for ℕ
5:         return ℕ
6:     else
7:         return unsat                                  ▷ 𝕄 is minimal.
```

Algorithm 7 Minimize

Require: $\mathbb{M} \models \mathcal{T}$
Ensure: $\mathbb{N} \preccurlyeq \mathbb{M}$, $\mathbb{N}$ is homomorphically minimal

```
1: function Minimize(𝒯, 𝕄)
2:     repeat
3:         ℕ ← 𝕄
4:         𝕄 ← Reduce(𝕄)
5:     until 𝕄 = unsat                                  ▷ Cannot reduce
6:     return ℕ                                         ▷ ℕ is a minimal model for 𝒯
```

captures the information in $\mathbb{M}$ that is "relevant" as an answer to the user's question, specified as a first-order theory. Put differently, $\hat{\mathbb{M}}$ restricts $\mathbb{M}$ to the elements that are denoted by the constants mentioned in the user's specification.

Lemma 5.3.1. Fix a ground and flat first-order theory $\mathcal{T}$ over a signature $\Sigma \equiv (\Sigma^R, \Sigma^F)$. A structure $\mathbb{M}$ is a model of $\mathcal{T}$ if and only if its Skolem-hull $\hat{\mathbb{M}}$ is a model of $\mathcal{T}$.

Proof. We give a proof for Lemma 5.3.1 by showing that for every atomic formula α in $\mathcal{T}$, $\mathbb{M} \models \alpha$ if and only if $\hat{\mathbb{M}} \models \alpha$. The proof can be easily extended to any formula in $\mathcal{T}$ by induction on the structure of first-order formulas:

- Let α be $R(c_1, \ldots, c_k)$ where $R \in \Sigma^R$ is a relation symbol of arity k and $c_i \in \Sigma^F (1 \leq i \leq k)$ are constants. A tuple $\langle c_i^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}} \rangle \in R^{\hat{\mathbb{M}}}$ if and only if $\langle c_i^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}} \rangle \in R^{\mathbb{M}}$.
- Let α be $c = d$ where $c, d \in \Sigma^F$ are constants. For elements $c^{\mathbb{M}}$ and $d^{\mathbb{M}}$, $c^{\mathbb{M}} = d^{\mathbb{M}}$ if and only if $c^{\hat{\mathbb{M}}} = d^{\hat{\mathbb{M}}}$.
- Let α be $f(c_1, \ldots, c_k) = d$ where $f \in \sigma^F$ is a function symbol of arity k and $c_i \in \Sigma^F (1 \leq i \leq k)$ and $d \in \Sigma^F$ are constants. Because every $c_i (1 \leq i \leq k)$ and d denote elements in $|\hat{\mathbb{M}}|$, the domain of $f^{\mathbb{M}}$ is defined on $\langle c_i^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}} \rangle$ and its range is defined on $d^{\hat{\mathbb{M}}}$. Therefore, by definition of $\hat{\mathbb{M}}$, $f^{\hat{\mathbb{M}}}(c_1^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}}) = d^{\hat{\mathbb{M}}}$ if and only if $f^{\mathbb{M}}(c_1^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}}) = d^{\mathbb{M}}$.

□

Below is the main result about the reduction steps of our minimization algorithm:

Theorem 5.3.2. Fix a ground and flat first-order $\mathcal{T}$ over signature Σ and a model $\mathbb{M}$ of $\mathcal{T}$. Let $\mathbb{N}$ be the model returned by $\mathit{Reduce}(\mathcal{T}, \mathbb{M})$. Take the Skolem-hull $\hat{\mathbb{N}}$ of $\mathbb{N}$ with respect to Σ. Then,

1. $\hat{\mathbb{N}} \models \mathcal{T}$
2. $\hat{\mathbb{N}} \preccurlyeq_h \mathbb{M}$

Proof. It is easy to show that $\hat{\mathbb{N}} \models \mathcal{T}$: line 4 of Algorithm 6 requires the returning model of *Reduce* to satisfy the input theory, that is, $\mathbb{N} \models \mathcal{T}$. And, by Lemma 5.3.1, $\hat{\mathbb{N}} \models \mathcal{T}$.
For the second part of the proof, construct h such that for every constant c, $c^{\hat{\mathbb{N}}}$ is sent to $c^{\mathbb{M}}$ by h. Observe that h is well-defined: given two constants c and d where $c^{\mathbb{M}} \neq d^{\mathbb{M}}$, a conjunct $c \neq d$ is in $\mathrm{Neg}(\mathbb{M})$, which must be satisfied by $\hat{\mathbb{N}}$ (line 4 of Algorithm 6). Therefore, c and d name distinct elements in $\hat{\mathbb{N}}$, *i.e.*, $c^{\hat{\mathbb{N}}} \neq$

$d^{\hat{\mathbb{N}}}$. Furthermore, h is a homomorphism: suppose that $R^{\hat{\mathbb{N}}}(c_1^{\hat{\mathbb{N}}}, \ldots, c_k^{\hat{\mathbb{N}}})$ is true in $\hat{\mathbb{N}}$. Because $\hat{\mathbb{N}}$ satisfies $\mathrm{Neg}(\mathbb{M})$, generated in the reduction step on line 2 of Algorithm 6, a conjunct $\neg R(c_1, \ldots, c_k)$ cannot be in $\mathrm{Neg}(\mathbb{M})$. Therefore, by definition of negation axiom about $\mathbb{M}$, $R^{\mathbb{M}}(c_1^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}})$ is true in $\mathbb{M}$.
Next, consider two constants c and d where $c^{\hat{\mathbb{N}}} = d^{\hat{\mathbb{N}}}$: because $\hat{\mathbb{N}} \models \mathrm{Neg}(\mathbb{M})$, a conjunct $c \neq d$ cannot be in $\mathrm{Neg}(\mathbb{M})$. By definition of negation axiom, $c^{\mathbb{M}} = d^{\mathbb{M}}$.
Finally, consider a function application $f^{\hat{\mathbb{N}}}(c_1^{\hat{\mathbb{N}}}, \ldots, c_k^{\hat{\mathbb{N}}})$ in $\hat{\mathbb{N}}$: because $\hat{\mathbb{N}}$ is defined over elements that are named by constants, $f^{\hat{\mathbb{N}}}(c_1^{\hat{\mathbb{N}}}, \ldots, c_k^{\hat{\mathbb{N}}}) = d^{\hat{\mathbb{N}}}$ for some element $d^{\hat{\mathbb{N}}}$ in $\hat{\mathbb{N}}$, denoted by a constant d. Since $\mathbb{N} \models \mathrm{Neg}(\mathbb{M})$, a conjunct $f(c_1, \ldots, c_k) \neq d$ is not in $\mathrm{Neg}(\mathbb{M})$; that is, $f^{\mathbb{M}}(c_1^{\mathbb{M}}, \ldots, c_k^{\mathbb{M}}) = d^{\mathbb{M}}$. Because h sends every $c_i^{\hat{\mathbb{N}}}$ to $c_i^{\mathbb{M}}$ $(1 \leq i \leq k)$, and sends $d^{\hat{\mathbb{N}}}$ to $d^{\mathbb{M}}$, it follows that $h(f^{\hat{\mathbb{N}}}(c_1^{\hat{\mathbb{N}}}, \ldots, c_k^{\hat{\mathbb{N}}})) = f^{\hat{\mathbb{N}}}(h(c_1^{\hat{\mathbb{N}}}), \ldots, h(c_k^{\hat{\mathbb{N}}}))$. □

5.3.2 Termination

Theorem 5.3.2 confirms that the presented reduction algorithm reduces an arbitrary model of a ground and flat theory in the homomorphism ordering. Yet, it remains to show that the minimization algorithm, consisting of repeated applications of the reduction step, (i) is terminating, (ii) and, returns a minimal model under the homomorphism ordering.

It is easy to show that the minimization algorithm is terminating: let the size of a negation preserving axiom be the number of conjuncts in the axiom. Suppose the following

$$\mathbb{M}_0 \to \mathbb{M}_1 \to \ldots \mathbb{M}_n \to \ldots$$

to be a chain of models reduced by Algorithm 7. Every arrow denote the reduction steps according to Algorithm 6. As we progress along the direction of the arrows, the size of the negation preserving axioms of models increases monotonically: in a step from a model $\mathbb{M}_i$ to $\mathbb{M}_{i+1}$, every conjunct in $\mathrm{Neg}(\mathbb{M}_i)$ is also a conjunct in $\mathrm{Neg}(\mathbb{M}_{i+1})$. Moreover, some disjunct in $\mathrm{Chip}(\mathbb{M}_i)$ must be satisfied by $\mathbb{M}_{i+1}$ by the definition of the reduction step. The corresponding chip disjunct will be present in $\mathrm{Neg}(\mathbb{M}_{i+1})$.
Considering that the set of constants in the signature is finite, the set of ground literals in a negation preserving axiom is finite. Therefore, the size of negation preserving axioms increases monotonically toward a finite maximum size during minimization; thus, the minimization chain is finite.

Before we show that the minimization algorithm terminates with a minimal model under the homomorphism ordering, we state a lemma that will be used in the proof of Theorem 5.3.4.

Lemma 5.3.3. Let $\mathcal{T}$ be a ground and flat theory over signature Σ and $\hat{\mathbb{M}}$ be a Skolem-hull of a model $\mathbb{M}$ of $\mathcal{T}$. Suppose that $\mathbb{N} \preccurlyeq \hat{\mathbb{M}}$. Then, $\mathbb{N}$ satisfies the negation preserving axiom $\mathrm{Neg}(\hat{\mathbb{M}})$ about $\hat{\mathbb{M}}$.

Proof. We show that the theorem holds for the different categories of the conjuncts in $\text{Neg}(\hat{\mathbb{M}})$:

- A conjunct $\neg R(c_1, \ldots, c_k)$ is in $\text{Neg}(\hat{\mathbb{M}})$ if and only if $R^{\hat{\mathbb{M}}}(c_1^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}})$ is not true in $\hat{\mathbb{M}}$. Because of the homomorphism from $\mathbb{N}$ to $\hat{\mathbb{M}}$, the fact $R^{\mathbb{N}}(c_1^{\mathbb{N}}, \ldots, c_k^{\mathbb{N}})$ is not true in $\mathbb{N}$; therefore, $\mathbb{N} \models \neg R(c_1, \ldots, c_k)$.

- A conjunct $c \neq d$ is in $\text{Neg}(\hat{\mathbb{M}})$ if and only if $c^{\hat{\mathbb{M}}} \neq d^{\hat{\mathbb{M}}}$ in $\hat{\mathbb{M}}$. Also, by the homomorphism from $\mathbb{N}$ to $\hat{\mathbb{M}}$, $c^{\mathbb{N}} \neq d^{\mathbb{N}}$ in $\mathbb{N}$ and $\mathbb{N} \models c \neq d$ follows.

- A conjunct $f(c_1, \ldots, c_k) \neq d$ is in $\text{Neg}(\hat{\mathbb{M}})$ if and only if $f^{\hat{\mathbb{M}}}(c_1^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}}) \neq d^{\hat{\mathbb{M}}}$ in $\hat{\mathbb{M}}$. Again, by the homomorphism from $\mathbb{N}$ to $\hat{\mathbb{M}}$, $f^{\mathbb{N}}(c_1^{\mathbb{N}}, \ldots, c_k^{\mathbb{N}}) \neq d^{\mathbb{N}}$ in $\mathbb{N}$, thus $\mathbb{N} \models (f(c_1, \ldots, c_k) \neq d)$ follows.

□

Finally, the next theorem confirms that when Algorithm 7 terminates, the resulting model is a minimal model under the homomorphism ordering.

Theorem 5.3.4. Fix a ground and flat first-order theory $\mathcal{T}$. Let $\mathbb{M} \equiv \mathsf{Minimize}(\mathcal{T}, \mathbb{A})$ be a model of $\mathcal{T}$, returned by a run of Algorithm 7, where $\mathbb{A}$ is an arbitrary model of $\mathcal{T}$. Let $\hat{\mathbb{M}}$ be the Skolem-hull of $\mathbb{M}$. For a model $\mathbb{N}$ of $\mathcal{T}$, if $\mathbb{N} \preccurlyeq \hat{\mathbb{M}}$, then $\hat{\mathbb{M}} \preccurlyeq \mathbb{N}$.

Proof. Let $\text{Neg}(\hat{\mathbb{M}})$ and $\text{Chip}(\hat{\mathbb{M}})$ respectively be the negation preserving and chip axioms about $\hat{\mathbb{M}}$. Notice that because $\mathsf{Reduce}(\mathcal{T}, \mathbb{A}) = \textbf{unsat}$ (line 5 of Algorithm 7), then $\mathcal{T} \cup \{\text{Neg}(\hat{\mathbb{M}})\} \vdash \neg\text{Chip}(\hat{\mathbb{M}})$. Therefore, because $\mathbb{N} \models \mathcal{T}$ and $\mathbb{N} \models \text{Neg}(\hat{\mathbb{M}})$ by Lemma 5.3.3, then $\mathbb{N} \not\models \text{Chip}(\hat{\mathbb{M}})$.
Construct a map $h : |\hat{\mathbb{M}}| \longrightarrow |\mathbb{N}|$ that sends the interpretation $c^{\hat{\mathbb{M}}} \in |\hat{\mathbb{M}}|$ of every constant c to its corresponding element $c^{\mathbb{N}} \in |\mathbb{N}|$. Observe that h is well-defined: assume that for two distinct constants c and d, $c^{\hat{\mathbb{M}}} = d^{\hat{\mathbb{M}}}$ in $\hat{\mathbb{M}}$. Consequently, $c \neq d$ is a disjunct in $\text{Chip}(\mathbb{M})$. Because $\mathbb{N} \not\models \text{Chip}(\mathbb{M})$, then $c^{\mathbb{N}} = d^{\mathbb{N}}$ in $\mathbb{N}$.
By a similar argument, it is possible to show that h is a homomorphism, thus $\hat{\mathbb{M}} \preccurlyeq \mathbb{N}$:

- Suppose a fact $R^{\hat{\mathbb{M}}}(c_1^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}})$ is true in $\hat{\mathbb{M}}$. A disjunct $\neg R(c_1, \ldots, c_k)$ is then in $\text{Chip}(\hat{\mathbb{M}})$. But because $\mathbb{N} \not\models \text{Chip}(\hat{\mathbb{M}})$, $R^{\mathbb{N}}(c_1^{\mathbb{N}}, \ldots, c_k^{\mathbb{N}})$ is true in $\mathbb{N}$.

- Suppose that $f^{\hat{\mathbb{M}}}(c_1^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}}) = d^{\hat{\mathbb{M}}}$ is true in $\hat{\mathbb{M}}$; a disjunct $f(c_1, \ldots, c_k) \neq d$ is in $\text{Chip}(\hat{\mathbb{M}})$. And since $\mathbb{N} \not\models \text{Chip}(\hat{\mathbb{M}})$, $f^{\mathbb{N}}(c_1^{\mathbb{N}}, \ldots, c_k^{\mathbb{N}}) = d^{\mathbb{N}}$ is true in $\mathbb{N}$.

□

5.3.3 Iterating over Minimal Models

In Section 5.3.1 and Section 5.3.2, we presented an algorithm for minimizing arbitrary models of ground and flat first-order theories. In this section, we utilize the

minimization algorithm to iterate over the minimal models of $\mathcal{T}$ in order to construct a set-of-support $\mathcal{S}$ for $\mathcal{T}$; every model of $\mathcal{T}$ is accessible from some model in $\mathcal{S}$ via homomorphism. We also show that the set $\mathcal{S}$, constructed by Algorithm 9 of this section, is a "minimal" set-of-support for $\mathcal{T}$; there does not exist a homomorphism between any two models in $\mathcal{S}$.

Algorithm 8 illustrates a procedure Next for iterating over models of $\mathcal{T}$. Next accepts $\mathcal{T}$ and a set $\mathcal{S}$ of models for $\mathcal{T}$, as input, and returns a new model (if exists) that is (i) homomorphically minimal (Theorem 5.3.6), and (ii) not in the homomorphism cone of any models of $\mathcal{S}$ (Lemma 5.3.5).

Algorithm 9 constructs a set-of-support for the input theory $\mathcal{T}$ (Theorem 5.3.6): starting with an empty set of models, repeated invocations of Next produce models that are homomorphically minimal but disjoint from the ones that were previously generated.

Algorithm 8 Next Model

Require: for all $\mathbb{U} \in \mathcal{S}$, $\mathbb{U} \models \mathcal{T}$

```
1: function NEXT(T, S)
2:     Φ ← ⋃_i {CHIP(U_i)} for all U_i ∈ S      ▷ Chip axioms about existing models
3:     if exists M such that M ⊨ (T ∪ Φ) then      ▷ Ask the SMT-solver for M
4:         N ← MINIMIZE(T, M)
5:         return SKOLEMHULL(N)
6:     else
7:         return unsat                          ▷ No more models
```

Algorithm 9 Set-of-Support

```
1: function SUPPORTSET(T)
2:     S ← ∅
3:     M ← NEXT(T, S)
4:     while M ≠ unsat do
5:         S ← S ∪ {M}
6:         M ← NEXT(T, S)
7:     return S                                  ▷ S is a set-of-support
```

Lemma 5.3.5. Let $\mathcal{T}$ be a ground and flat theory over signature Σ and $\hat{\mathbb{M}}$ be the Skolem-hull of a model $\mathbb{M}$ of $\mathcal{T}$. Let Chip($\hat{\mathbb{M}}$) be the chip axiom about $\hat{\mathbb{M}}$. Given a model $\mathbb{N}$ over Σ, $\mathbb{N} \not\models$ Chip($\hat{\mathbb{M}}$) if and only if $\hat{\mathbb{M}} \preccurlyeq \mathbb{N}$.

Proof. We first show if $\mathbb{N} \not\models$ Chip($\hat{\mathbb{M}}$) then $\hat{\mathbb{M}} \preccurlyeq \mathbb{N}$:
Construct a map h such that for every constant c in Σ, it maps $c^{\hat{\mathbb{M}}}$ to $c^{\mathbb{N}}$. Observe

that h is well-defined: suppose that for two constants c and d, $c^{\hat{\mathbb{M}}} = d^{\hat{\mathbb{M}}}$ in $\hat{\mathbb{M}}$. Consequently, a disjunct $c \neq d$ is in Chip($\hat{\mathbb{M}}$), but because $\mathbb{N} \not\models \text{Chip}(\hat{\mathbb{M}})$, the corresponding disjunct is not true in $\mathbb{N}$; that is, $c^{\mathbb{N}} = d^{\mathbb{N}}$.

By a similar argument, it is possible to show that h is a homomorphism: consider the various categories of disjuncts in Chip($\hat{\mathbb{M}}$):

- Suppose a fact $R^{\hat{\mathbb{M}}}(c_1^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}})$ is true in $\hat{\mathbb{M}}$. Therefore, a disjunct $\neg R(c_1, \ldots, c_k)$ is in Chip($\hat{\mathbb{M}}$), but because $\mathbb{N} \not\models \text{Chip}(\hat{\mathbb{M}})$, the corresponding disjunct is not true in $\mathbb{N}$. Consequently, $R^{\mathbb{N}}(c_1^{\mathbb{N}}, \ldots, c_k^{\mathbb{N}})$ is in $\mathbb{N}$.

- Suppose that $f^{\hat{\mathbb{M}}}(c_1^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}}) = d^{\hat{\mathbb{M}}}$ is true in $\hat{\mathbb{M}}$. The corresponding disjunct in Chip($\hat{\mathbb{M}}$), $f(c_1, \ldots, c_k) \neq d$ cannot be satisfied by $\mathbb{N}$. Therefore, $f^{\mathbb{N}}(c_1^{\mathbb{N}}, \ldots, c_k^{\mathbb{N}}) = d^{\mathbb{N}}$ is true in $\mathbb{N}$.

It remains to show if $\hat{\mathbb{M}} \preccurlyeq \mathbb{N}$ then $\mathbb{N} \not\models \text{Chip}(\hat{\mathbb{M}})$:

We show that $\mathbb{N} \not\models \alpha$ for every disjunct α in Chip($\hat{\mathbb{M}}$):

- If α is a formula $\neg R(c_1, \ldots, c_k)$ for a k-ary relation symbol R, then by definition, $R^{\hat{\mathbb{M}}}(c_1^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}})$ is true in $\hat{\mathbb{M}}$. Because $\hat{\mathbb{M}} \preccurlyeq \mathbb{N}$, $R^{\mathbb{N}}(c_1^{\mathbb{N}}, \ldots, c_k^{\mathbb{N}})$ is true in $\mathbb{N}$. Therefore, $\mathbb{N} \not\models \alpha$.

- If α is in form of $c \neq d$ for a two constants c and d, then $c^{\hat{\mathbb{M}}} = d^{\hat{\mathbb{M}}}$ in $\hat{\mathbb{M}}$. Since $\hat{\mathbb{M}} \preccurlyeq \mathbb{N}$, $c^{\mathbb{N}} = d^{\mathbb{N}}$ in $\mathbb{N}$, thus, $\mathbb{N} \not\models \alpha$.

- If α is in form of $f(c_1, \ldots, c_k) \neq d$ for a k-ary function symbol f, then $f^{\hat{\mathbb{M}}}(c_1^{\hat{\mathbb{M}}}, \ldots, c_k^{\hat{\mathbb{M}}}) = d^{\hat{\mathbb{M}}}$ in $\hat{\mathbb{M}}$ Because $\hat{\mathbb{M}} \preccurlyeq \mathbb{N}$, $f^{\mathbb{N}}(c_1^{\mathbb{N}}, \ldots, c_k^{\mathbb{N}}) = d^{\mathbb{N}}$ in $\mathbb{N}$, thus, $\mathbb{N} \not\models \alpha$.

□

Theorem 5.3.6. Fix a ground and flat first-order theory $\mathcal{T}$. Let $\mathcal{S}$ be a set of models where for every model $\mathbb{U} \in \mathcal{S}$

- $\mathbb{U} \models \mathcal{T}$
- for a model $\mathbb{N}$ of $\mathcal{T}$, $\mathbb{N} \preccurlyeq \mathbb{U}$ if and only if $\mathbb{U} \preccurlyeq \mathbb{N}$

For a call of $\mathsf{Next}(\mathcal{T}, \mathcal{S})$,

(i) if a model $\mathbb{M}$ is returned, then for every model $\mathbb{U} \in \mathcal{S}$, $\mathbb{M} \not\preccurlyeq \mathbb{U}$ and $\mathbb{U} \not\preccurlyeq \mathbb{M}$.

(ii) if the result is **unsat**, then $\mathcal{S}$ is a set-of-support for $\mathcal{T}$.

Proof. For a proof of i: because $\mathbb{M} \preccurlyeq \mathbb{U}$ if and only if $\mathbb{U} \preccurlyeq \mathbb{N}$, it suffices to show that $\mathbb{U} \not\preccurlyeq \mathbb{M}$. Let Chip($\mathbb{U}$) be the chip axiom about $\mathbb{U}$. According to Algorithm 8 for Next, $\mathbb{M} \models \text{Chip}(\mathbb{U})$. Therefore, by Lemma 5.3.5, $\mathbb{U} \not\preccurlyeq \mathbb{M}$.
For a proof of ii: let Φ be a theory, containing chip axioms for the models in $\mathcal{S}$. By Gödel's completeness theorem, when the result of Algorithm 8 is **unsat**, $\mathcal{T} \vdash \neg\Phi$:

- If $\mathcal{T}$ is unsatisfiable, $\mathcal{S}$ is the empty set, a vacuous jointly universal set for the models of $\mathcal{T}$.
- If $\mathcal{T}$ is satisfiable, every model $\mathbb{N}$ of $\mathcal{T}$ must not satisfy some chip axiom $\varphi \in \Phi$. Assuming that φ is a chip axiom for some model $\mathbb{U} \in \mathcal{S}$, then by Lemma 5.3.5, there exists a homomorphism from $\mathbb{U}$ to $\mathbb{N}$.

□

As a result of Theorem 5.3.6, Algorithm 9 computes a set-of-support for the input ground and flat first-order theory $\mathcal{T}$ by repeated invocations of Algorithm 8. Observe that the set of models $\mathcal{S}$ that is computed during Algorithm 9 satisfies the criteria of Theorem 5.3.6: every model in this set (i) is a model of $\mathcal{T}$, and (ii) is homomorphically minimal (by Theorem 5.3.4).

5.4 Model Construction in Razor

Model-construction in Razor is performed in three phases:

1. *Preprocessing*: the user's theory in geometric form $\mathcal{G}$ is transformed to a relational theory $\mathcal{G}_{\mathsf{Rel}}$ by the transformations in Section 5.1.1; every sequent in $\mathcal{G}_{\mathsf{Rel}}$ is range-restricted with a linear body by the transformations in Section 5.1.2.
2. *Grounding*: a run of the grounding algorithm (Algorithm 3) computes a ground theory $\mathcal{G}^*$ for $\mathcal{G}_{\mathsf{Rel}}$ over a set of possible facts $\mathrm{Pos}(\mathcal{G})$.
3. *Minimal Model-Finding*: a run of Algorithm 9 constructs a set-of-support for the ground and flat theory $\mathcal{G}^\#$, obtained by hashing the terms of $\mathcal{G}^*$ according to Section 5.1.3.

Figure 5.1 illustrates the model-construction process implemented into Razor. This process is sound and complete: observe that the theory $\mathcal{G}_{\mathsf{Rel}}$ is equisatisfiable to the user's input $\mathcal{G}$. Moreover, every model $\mathbb{M}_{\mathsf{Rel}}$ can be converted to a model $\mathbb{M}$ of $\mathcal{G}$ by undoing relationalization, whereby the relations in $\mathcal{G}_{\mathsf{Rel}}$ that had substituted the functions in $\mathcal{G}$ are converted back to function symbols by the obvious transformation. It remains to show that the set-of-support that is computed for $\mathcal{G}^\#$ is in fact a set-of-support for $\mathcal{G}_{\mathsf{Rel}}$.

Soundness. Assume $\mathcal{G}_{\mathsf{Rel}}$ to be over a signature Σ. Let $\mathbb{M}$ be a model in the set of models $\mathcal{S}$, computed by a run of Algorithm 9 on $\mathcal{G}^\#$. According to Theorem 5.1.1, $\mathbb{M}$ corresponds to a model $\mathbb{M}^*$ of $\mathcal{G}^*$ (observe that the reduct of $\mathbb{M}$ to Σ is equivalent to the witness-reduct of $\mathbb{M}^*$).
Because $\mathbb{M}$ is homomorphically minimal, the model $\mathbb{M}^*$ is an image of the set of possible facts $\mathrm{Pos}(\mathcal{G})$ over which $\mathcal{G}^*$ is defined:

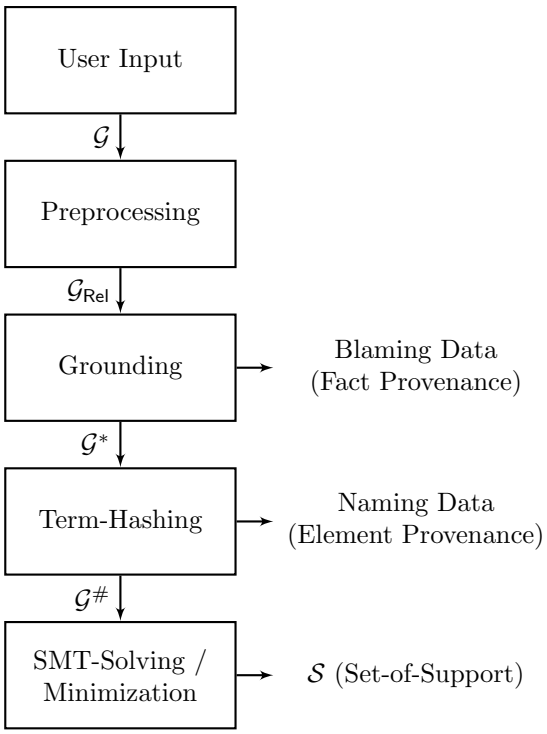

Figure 5.1: Model-construction procedure in Razor

- every element in $\mathbb{M}^*$ is denoted by a term that is mentioned in the set of possible facts.
- every fact in $\mathbb{M}^*$ is because of an axiom in $\mathcal{G}^*$; thus, it is a fact in $\text{Pos}(\mathcal{G})$.
- every equation in $\mathbb{M}^*$ is because of an axiom in $\mathcal{G}^*$; thus, it is a fact in $\text{Pos}(\mathcal{G})$.

Thus, by Theorem 5.2.2, the reduct $\mathbb{M}^-$ of $\mathbb{M}$ to Σ is a model of $\mathcal{G}_{\mathsf{Rel}}$.

Completeness. Let $\mathbb{A}$ be a model of $\mathcal{G}_{\mathsf{Rel}}$ over a signature Σ. By Theorem 4.3.2 about the Chase, a chase-model $\mathbb{C}$ (over the witnessing signature) with a witness-reduct $\mathbb{C}^-$ exists such that $\mathbb{C}^- \preccurlyeq \mathbb{A}$; and, by Theorem 5.2.3, $\mathbb{C} \models \mathcal{G}^*$. By Theorem 5.1.1, a model $\mathbb{C}^\#$ obtained by hashing the terms in $\mathbb{C}$ is a model of $\mathcal{G}^\#$ (observe that the reduct of $\mathbb{C}^\#$ to Σ is the same model as $\mathbb{C}^-$). Finally, because Algorithm 9 computes a set-of-support $\mathcal{S}$ for $\mathcal{G}^\#$ (as a consequence of Theorem 5.3.6), a model $\mathbb{M}$ exists in $\mathcal{S}$ such that $\mathbb{M} \preccurlyeq \mathbb{C}^\#$. Given the witness-reduct $\mathbb{M}^-$ of $\mathbb{M}$, one checks that $\mathbb{M}^- \preccurlyeq \mathbb{C}^-$; therefore, $\mathbb{M}^- \preccurlyeq \mathbb{A}$.

5.5 Implementation

We implemented the current version of Razor in Haskell [53], a strongly-typed and purely functional programming language. A declarative programming language, such as Haskell, is close to our mathematical description of the underlying algorithms. This enabled us to implement and evaluate a variety of algorithmic ideas, including the ones mentioned in this chapter and Chapter 8, over a rather short period of time. Haskell's strong type-system helped us achieve a better quality of code; also, it often helped us uncover algorithmic and programming flaws at compile time. Functions with no side-effects (due to purity of Haskell) helped us compose various code fragments confidently; it also helped us reproduce bugs effortlessly.

However, the lack of mutation made it impossible to efficiently implement certain algorithms, such as congruence closure (see Section 8.1.1) and incremental view maintenance. As a result of this kind of inefficiencies, we could hardly develop a clear intuition about the performance of our algorithms written in a more flexible programming language. However, we could exploit the profiling facilities of GHC [54] to identify those parts of the code that require performance improvement. Nevertheless, we found it difficult to track down the bugs in the code due to the lazy evaluation in Haskell and the lack of strong tracing facilities.

Architecture and Design. Razor's overall design consists of the following components, implemented as separate hierarchical modules in Haskell:

1. *API:* provides an API to Razor's model-finding engine. An external program, such as Razor's REPL, interacts with this module to access Razor's features.

2. *Chase:* provides an interface for an implementation of a grounding algorithm described by Section 5.2. The current implementation of this algorithm is based on a variation of the Chase from a relational algebra perspective (see Section 8.2.1).

3. *Common:* defines and implements data-structures that are used globally. In particular, this module defines the structure of models, observational facts, and provenance information.

4. *REPL:* is a command-line read-eval-print-loop that is currently the primary GUI of Razor. The REPL implements the various modes of interacting with the tool explained in Section 3.3.

5. *SAT:* defines an interface for implementations of the SMT-solving and minimization algorithms presented in Section 5.3. The current implementation uses SMTLib2 [55] as the interface to interact with Z3.

6. *Syntax:* defines the syntax and types of first-order formulas; geometric PEFs, sequents, and theories; and, TPTP formulas. This layer provides parsers for the aforementioned types as well as utility functions for syntactic manipulation including the preprocessing transformations (see Section 5.1).

7. *Tools:* implements utility structures and functions that are used freely in the implementation of the other modules.

5.5.1 Relational Grounding

The current instance of the grounding algorithm implemented into Razor utilizes relational algebra to evaluate sequents, as views, in models, as databases. This is a modification of Atlas (see Section 8.2), our earlier implementation of the Chase. The relational algebraic implementation is consistent with the general theme of our algorithms, which operate on relational theories obtained by transforming the functions in the user's input. Relationalization simplifies the computation of environments in which sequents fail in models because it does not have to deal with complex terms.

Moreover, the implementation based on relational algebra makes various efficiency techniques from the database literature available to our program. As we explain in Section 8.2.1, Razor's grounding procedure benefits from a mechanism that incrementally evaluates sequents in a growing model (or a set of facts), as it is augmented with facts. Aside from that, however, the current grounding procedure is rather elementary. Future improvements may employ techniques, such as view materialization and update detection [56], to improve the efficiency of the procedure.

A fundamentally different approach to this algorithm may utilize term-rewriting techniques that leverage sophisticated data-structures, such as directed acyclic graphs and tries, to construct the set of possible facts. We discuss a naïve implementation of the Chase based on term-rewriting in Section 8.1.

5.5.2 Interaction with the SMT-Solver

We utilize Z3 in QF_UFBV as the backend SMT-solver and use SMTLib2, a Haskell package available at [55], as the interface to interact with Z3. SMTLib2 provides an abstraction layer over the SMT-solver as a convenient monadic environment in Haskell, and it can interact with any SMT-solver that supports the SMT-LIB v.2 standard. This allows a programmer to write a Haskell program that is automatically translated to an input of the underlying SMT-solver (in our case Z3). The program then receives the solver's output as a Haskell data-structure.

Incremental SMT-Solving. The primary advantage of SMTLib2 over similar packages (*e.g.*, SMT Based Verification [57]) is that SMTLib2 supports the *incremental* mode offered by SMT-solvers such as Z3. An SMT-solver that supports the incremental model provides the user with a `(push)` command, using which the user can stack up multiple queries to the solver, and a `(pop)` command, which undoes the last "pushed" query, returning the solver to its previous state. Conceptually, an incremental SMT-solver is a stateful solver, meaning it can retrieve its previous state efficiently without recomputation. Incremental SMT-solving is crucial for the efficiency of the minimization and augmentation algorithms implemented into Razor, as explained later in this section.

Iterating over Models

Razor's SAT module delivers an abstract *iterator* for generating models. Incremental SMT-solving allows us to acquire a handle to the internal state of the solver through the iterator. The following are the main operations on the iterator:

- `satInitialize`: establishes a connection with the SMT-solver in the incremental mode.
- `satStore`: sends a ground formula to the solver, changing the internal state of the solver.
- `satSolve`: asks the solver for a model; returns the model; and, sends the chip axiom about the current model to the solver to advance the iterator.
- `satAugment`: stores a set of additional ground formulas corresponding to the augmenting facts. The augmenting facts are preceded by `(push)` for future backtracking. The augment operation results in a new iterator over the augmented models.
- `satBacktrack`: undoes an augmentation by sending a `(pop)` to the solver.
- `satClose`: closes the connection to the solver.

Incremental Minimization

Razor's minimization procedure exploits incremental SMT-solving when reducing the models returned by the SMT-solver: let $\mathcal{G}^*$ be a set of ground sequents, generated by the grounding algorithm and I be the initial SMT iterator corresponding to the state of SMT-solver after loading $\mathcal{G}^*$. In every reduction iteration, additional constraints about the negation and chip axioms of the previously generated model are sent to the solver via `satStore`.

In the absence of this feature, the minimization procedure would have to provide the solver with the entire set of constraints, starting from the first reduction iteration, for each reduction step.

5.5.3 Incremental Augmentation

Let $\mathcal{G}$ be a theory in geometric form. Let $\text{Pos}(\mathcal{G})$ and $\mathcal{G}^*$ respectively be the set of possible facts and the ground theory that are constructed for $\mathcal{G}$ by Razor's grounding algorithm. Assume that $\mathbb{M}$ is a model of $\mathcal{G}$, computed as a model of $\mathcal{G}^*$ by Razor's minimization algorithm. Given a PEF α, augmentations of $\mathbb{M}$ could be computed as a model of $\mathcal{G} \cup \{\vdash \alpha\}$ by the model-finding process illustrated by Figure 5.1. This approach, however, is inefficient because $\text{Pos}(\mathcal{G})$ and $\mathcal{G}^*$ must be recomputed. Fortunately, positivity of geometric logic enables us to reuse $\text{Pos}(\mathcal{G})$ and $\mathcal{G}^*$ when computing the augmented models.

As we explain in Section 5.5.4, the state of exploration in Razor is a triple $\langle \text{Pos}(\mathcal{G}), I, \mathcal{P} \rangle$ for a set of possible facts $\text{Pos}(\mathcal{G})$ and provenance information $\mathcal{P}$ about the facts and terms of $\text{Pos}(\mathcal{G})$. The iterator I maintains the internal state of an incremental SMT-solver, loaded with the ground instances $\mathcal{G}^*$ of $\mathcal{G}$ over $\text{Pos}(\mathcal{G})$ and additional chip axioms for generating next models of the stream. That is, the iterator I effectively points to a specific model of $\mathcal{G}$ in the current stream of models.

Razor exploits the incremental mode of the backend solver for augmentation:

1. An extension of $\mathcal{A} \equiv \mathcal{G} \cup \{\vdash \alpha\}$ is constructed.

2. Starting with $\text{Pos}(\mathcal{G})$, the grounding algorithm computes a new set of possible facts $\text{Pos}(\mathcal{A})$ (*i.e.*, $\text{Pos}(\mathcal{G}) \subseteq \text{Pos}(\mathcal{A})$).

3. The set of ground instances $\mathcal{A}^*$, generated during the computation of $\text{Pos}(\mathcal{A})$ is passed to the solver, resulting in a new iterator J.

4. Minimal models of J are generated by the minimization algorithm.

The current implementation of Razor allows for augmenting with ground atomic facts only. An augmenting fact may be defined over the elements of the current model as well as *fresh* elements introduced by the user.

5.5.4 Exploration State

Given a theory $\mathcal{G}$, the current state of exploration in Razor is kept by a triple $\langle \text{Pos}(\mathcal{G}), I, \mathcal{P} \rangle$, where $\text{Pos}(\mathcal{G})$ is a set of possible facts for $\mathcal{G}$, I is an iterator from the SAT module (see Section 5.5.2), and $\mathcal{P}$ is the provenance information about the facts and terms in $\text{Pos}(\mathcal{G})$. The history of previous augmentations is maintained by synchronizing the internal stack of a backend incremental SMT-solver and a stack $\mathcal{S}$ of possible facts and provenance information pairs:

- Every time the user asks for a next model, the iterator I in the current state is replaced by a new iterator produced by `satNext`.
- When the user augments an existing model, a new set of possible facts $\text{Pos}(\mathcal{A})$ and new provenance information $\mathcal{P}'$ about $\text{Pos}(\mathcal{A})$ is generated by the grounding procedure. Also, a call to `satAugment` stores the ground sequents generated during this run of the grounding procedure, resulting in a new iterator J (see Section 5.5.2). Consequently, the current state is set to $\langle \text{Pos}(\mathcal{A}), J, \mathcal{P}' \rangle$, and the pair of $\text{Pos}(\mathcal{G})$ and $\mathcal{P}$ is pushed into the stack $\mathcal{S}$.
- Let $\text{Pos}(\mathcal{Q})$ and $\mathcal{Q}$ be at the top of the stack $\mathcal{S}$, and J be the result of applying `satBacktrack` on the current iterator I. When the user undoes the last augmentation, the current state of exploration is set to $\langle \text{Pos}(\mathcal{Q}), J, \mathcal{Q} \rangle$; also, the pair of $\text{Pos}(\mathcal{Q})$ and $\mathcal{Q}$ is popped out of $\mathcal{S}$.

5.5.5 Bounded Search for Models

We earlier showed that theories in geometric form are as expressive as first-order theories (see Theorem 4.2.2); hence, satisfiability of theories in geometric form is undecidable. In Section 4.3.4, we presented a syntactic condition for theories in geometric form, known as weak acyclicity, under which the Chase is guaranteed to terminate. In practice, though, geometric theories are rarely weakly acyclic. That is, a workable model-finding strategy must impose a bound on the 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, however, implements a more systematic strategy for bounding the search, based on the depth of the witnessing terms that are created as the provenance of elements. Specifically, Razor implements the *partial model building* and the *controlled reuse* methods from Section 4.3.5 to bound its chase-based grounding algorithm.

By default, Razor uses the controlled reuse method, which corresponds to the *reuse* mode described in Section 3.3.1 This strategy results in (complete) models of the input theory within the given bound d, in which all unjustified equations between elements occur at depth d. The partial model building approach, known as the *pure* mode in Section 3.3.1, might not create models of the input theory. When running in the pure mode, Razor will report the existential quantifiers that failed to be instantiated (due to the bound on the search).

In some applications where real models of the theory are expected, the resulting partial models are not desirable; however, partial models are consistent with our model-exploration paradigm implemented into Razor: starting with a partial model, the user can learn about the process by which models are generated. Moreover, the user can use augmentation to develop models of her interest, possibly by equating elements of the partial model.

Chapter 6

Case-Studies and Evaluation

We present case-studies to demonstrate a user's interaction with our model-finding assistant, Razor, in the context of our framework for model-exploration. We also evaluate Razor's performance on examples from the Alloy distribution and TPTP. The current implementation of Razor is primarily a proof of concept for the framework presented in Chapter 3. At present the speed of Razor is not competitive, on large theories, with traditional model-finders with narrower goals. A long-term research question is exploring the tradeoffs between efficiency and the kind of enhanced expressivity we offer.

6.1 Case-Studies

We introduce a case-study of access control policy analysis using Razor in Section 6.1.1. This example shows how the user may utilize Razor to discover flaws in her policy, precisely by identifying rules that do not meet her expectations. In Section 6.1.2, we analyze a standard Flowlog program for Software Defined Networking using Razor; specifically, we demonstrate how the construction of minimal models and provenance information distinguishes Razor from a conventional model-finding tool.

6.1.1 Case-Study 1: Access Control

ALAS (Applied Logic and Security) and PEDS (Performance Evaluation and Distributed Systems) are research groups in the Department of Computer Science, which are located in the lab B17.

Figure 6.1 demonstrates the rules that are governing the access to B17: a member of a research group must be able to enter the labs that are assigned to the research group (rule 1). A person who has a matching key/ID card, which opens the door of a lab, can enter the lab (rules 2 and 3). If a person can enter a lab, he must have either a matching ID card or a matching key (rule 4). The central system that

```
(1)  MemberOf(p,r) & LabOf(r,l) => Enters(p,l);

(2)  HasKey(p,k) & KeyOpens(k,l) => Enters(p,l);

(3)  CardOpens(cardOf(p),l) => Enters(p,l);

(4)  Enters(p,l) => CardOpens(cardOf(p),l)
                 |  exists k. HasKey(p,k) & KeyOpens(k,l);

(5)  CardOpens(cardOf(p), l) => exists r. MemberOf(p,r) & LabOf(r,l);

(6)  HasKey(p,k) => exists e. Grants(e,p,k) & Employee(e);

(7)  Grants(e,p,k)  => HasKey(p,k)

(8)  LabOf('ALAS,'B17);

(9)  LabOf('PEDS, 'B17);

(10)  MemberOf(p,'PEDS) & HasKey(p,k) & KeyOpens(k,'B17) => Falsehood;
```

Figure 6.1: Rules governing access to B17

manages the electronic locks guarantees that a person's ID card opens the door of a lab only if he is a member of a research group assigned to that lab (rule 5). The keys to the labs, however, must be granted by an employee of the department (rule 6); once an employee grants a key to a person, the grantee is assumed to be in the possession of the key (rule 7). Finally, ALAS and PEDS are located in B17 (rules 8 and 9), but PEDS members can enter B17 only if they have matching ID cards (rule 10).

The specification in Figure 6.1 may be extended by additional sequents to ask queries about the policy.

Unauthorized Access

Consider a situation where the user is wondering if an unauthorized user, namely a "Thief", can enter B17. Our "cynical" user describes the Thief as a person who is neither a member of ALAS nor PEDS:

```
MemberOf('Thief, 'ALAS) => Falsehood;
MemberOf('Thief, 'PEDS) => Falsehood;
Enter('Thief, 'B17);
```

First Scenario—A Matching Key. The first model that Razor returns demonstrates a situation where the Thief is in the possession of a key, which can open

```
Enters   = {(p1, l1)}          'B17   = l1
Employee = {(e1)}              'PEDS  = r2
Grants   = {(e1,p1,k1)}        'ALAS  = r1
HasKey   = {(p1,k1)}           'Thief = p1
KeyOpens = {(k1,l1)}
LabOf    = {(r1,l1), (r2,l1)}
```

Figure 6.2: The Thief enters B17 with a key

```
Enters   = {(e1, l1)}          'B17   = l1
Employee = {(e1)}              'PEDS  = r2
Grants   = {(e1,e1,k1)}        'ALAS  = r1
HasKey   = {(e1,k1)}           'Thief = e1
KeyOpens = {(k1,l1)}
LabOf    = {(r1,l1), (r2,l1)}
```

Figure 6.3: An employee enters B17 with a key

B17 (Figure 6.2). Clearly, we need additional rules to restrict the people who can acquire the keys. But before fixing the policy, the user can further investigate this scenario. The user may wonder if an employee can grant a key to B17 to herself. This question may be checked by augmenting the model in Figure 6.2 with a fact that requires `p1` and `e1` to be the same person:

```
> aug p1 = e1
```

The augmentation results in a model (Figure 6.3), suggesting that the employee can indeed grant a key to herself. An easy (but not global) fix is to add a new rule that restricts the people who can receive keys to B17 to ALAS and PEDS members.

```
Grants(e, m, k) & KeyOpens(k, 'B17) => MemberOf(m, 'ALAS)
                                     |  MemberOf(m, 'PEDS);
```

Still, rule 10 from Figure 6.1 does not allow PEDS members to enter B17 using keys.

Second Scneario—A Third research group. The second model for the previous query, returned by Razor, demonstrates a situation where the Thief is a member of a third research group (other than ALAS and PEDS), which is also located in B17 (Figure 6.4). The user may ask "where did this third research group come from?", then she can look at the provenance information about `r3`:

```
> origin r3
```

```
Enter     = {(p1, l1)}          'B17   = l1
CardOpens = {(c1, l1)}          'PEDS  = r2
MemberOf  = {(e1, r3)}          'ALAS  = r1
cardOf    = {(p1, c1)}          'Thief = p1
LabOf     = {(r1, l1), (r2, l1)
            ,(r3, l1)}
```

Figure 6.4: The Thief is a member of a third research group

Razor pinpoints an instance of the causal sequence from the user's theory (rule 5 from Figure 6.1):

```
rule    :
CardOpens(cardOf(p), l) => exists r. MemberOf(p,r) & LabOf(r,l)

instance:
CardOpens(c1, l1) => MemberOf(e1, r3) & LabOf(r3, l1)
```

The research group `r3` exists because the Thief has a matching ID card. The existing policy does not restrict the research groups that may be assigned to B17. Such a restriction would force the research group `r3` to be either ALAS or PEDS. The user can test this policy fix by the following augmentation:

```
> aug r3 = r2
```

The augmentation produces no counter examples; the fix is valid.

But the user may ask "how did the Thief acquire a matching card in the first place?". The user can find an answer to this question by blaming this fact:

```
> blame CardOpens(c1, l1)
```

The resulting blaming information gives the answer:

```
rule    :
Enters(p,l) => CardOpens(cardOf(p),l)
            |  exists k. HasKey(p,k) & KeyOpens(k,l)
instance:
Enters(p1, l1) => CardOpens(c1, l1)
               |  HasKey(p1, k1) & KeyOpens(k1, l1)
```

The Thief has a card because the user's query assumed that he could enter the lab (rule 4 of Figure 6.1). He could also have a key, which is evident in the first model, discussed earlier.

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

```
TABLE learned(switchid, port, macaddr);
  ON packet_in(pkt):
    -- Learn port:mac pairs that are not already known
    INSERT (pkt.locSw, pkt.locPt, pkt.dlSrc) INTO learned WHERE
      NOT learned(pkt.locSw, pkt.locPt, pkt.dlSrc);
    -- Remove any outdated port:mac pairs
    DELETE (pkt.locSw, pt, pkt.dlSrc) FROM learned WHERE
      NOT pt = pkt.locPt;
    -- Unicast the packet when the destination is known
    DO forward(new) WHERE
      learned(pkt.locSw, new.locPt, pkt.dlDst);
    -- If the destination is not known, broadcast the packet
    DO forward(new) WHERE
      NOT learned(pkt.locSw, ANY, pkt.dlDst)
        AND NOT pkt.locPt = new.locPt;
```

Figure 6.5: The learning switch program in Flowlog

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.

The user may find it harmless to allow other research groups (such as `r3` of the previous model) to be located in B17. However, if the user finds this example contradictory to the security goals that she has in mind, she can fix the policy by adding another rule as follows:

```
LabOf(x, 'B17) => x = 'ALAS | x = 'PEDS;
```

After extending the policy with this rule and the fix from the previous scenario, Razor will not return any models for the user's query; the policy will prevent the Thief from entering B17.

6.1.2 Case-Study 2: Flowlog Program

Flowlog [58] is a language for tierless programming in Software Defined Networking (SDN) environments, specifically for programming the SDN controller. Flowlog abstracts out the complexities of the various network layers, making it possible to utilize formal methods to verify programs. In this section, we utilize Razor to reason about a standard Flowlog example that specifies a *learning switch* [1] (Figure 6.5).

[1]The Flowlog case-study presented in this section is developed by graduate student Ryan Danas.

Translation to Geometric Form

Translation of a Flowlog program to a first-order theory in geometric form requires some treatment, common to all Flowlog programs:

Timestamps. The notion of "time" is implicit in Flowlog; every event is assumed to be processed at a particular time. Our translation extends the relations that define the state of the Flowlog program with a timestamp. In principle, we could assume an infinite sequence of time, by adding an axiom that defined the successor $succ(t)$ for every time t. For the sake of efficiency, however, we define $succ(t)$ only when t corresponds to an *incoming event* E (intuitively, an incoming event is processed at the next time):

$$E(t, pkt) \vdash \exists t' \,.\, succ(t) = t'$$

State Transition. Next, we introduce a set of sequents for each relation S that maintains the *state* of the program (these are merely translations of the axioms defined in [58]): Axiom 6.1 states that a tuple in S will propagate in time unless it is deleted. Axiom 6.2 forces an inserted tuple to be available at the next time. Axiom 6.3 gives a higher priority to deletion than insertion; *i.e.*, delete overrides insert.

$$S(t, \vec{x}) \wedge succ(t) = st \quad \vdash S(st, \vec{x}) \vee S^-(t, \vec{x}) \tag{6.1}$$

$$S^+(t, \vec{x}) \wedge succ(t) = st \quad \vdash S(st, \vec{x}) \tag{6.2}$$

$$S^-(t, \vec{x}) \wedge S(t, \vec{x}) \wedge \neg S^+(t, \vec{x}) \wedge succ(t) = st \quad \vdash \neg S(st, \vec{x}) \tag{6.3}$$

Figure 6.6 is a translation of the Flowlog program in Figure 6.5. Notice that in this translation, the relation `learned_P` and `learned_M` are respectively the insert and delete tables associated to the relation *learned* (*i.e.*, $learned^+$ and $learned^-$).

Exploration by Augmentation

The augmentation feature of Razor allows the user to perform a *step-by-step* analysis of the Flowlog program. Assume that the user is interested in exploring models of the following query in the learning switch program, when a packet arrives (at some particular time):

```
exists pkt, t . packet (t, pkt);
```

Razor returns two models: a model shows a situation where the packet in question has already been learned; the other one captures a situation where the packet has not been previously learned. The user can choose either of these models and explore it forward in time. He can use augmentation to simulate the arrival of a new packet:

```
> aug packet(time1, pkt1)
```

```
packet(t, pkt) <=> learned_P(t, locSw(pkt), locPt(pkt), dlSrc(pkt))
                 |  learned(t, locSw(pkt), locPt(pkt), dlSrc(pkt));
packet(t, pkt)  & learned(t, locSw(pkt), pt, dlSrc(pkt))
                <=> learned_M(t, locSw(pkt), pt, dlSrc(pkt))
                 |  locPt(pkt)=pt;
packet(t, pkt) & learned(t, locSw(pkt), locPt(new), dlDst(pkt))
                <=> forward(t, pkt, locPt(new));
packet(t, pkt) <=> forward(t, pkt, np)
                 |  exists ANY .
                           learned( t, locSw(pkt), ANY, dlDst(pkt))
                 |  locPt(pkt) = np;
```

Figure 6.6: The learning switch program in Razor

Here `time1` is the last time in the current model (the time after arrival of the first packet) and `pkt1` is a fresh name, chosen by the user to represent a new packet. The user can explore this scenario further in time by performing other augmentations. This kind of analysis is analogous to using a debugger to trace a program step-by-step.

Models for one packet event

As mentioned, Razor returns two models for the first query of the previous section. Both models contain a single packet event `learned(time0, pkt0)`, whereby the incoming port and mac address is learned in the next time. In the first model, the packet was previously learned, whereas in the other model, the packet is going to be learned in the next time.

The user confirms that in both models, the packet is learned by the first rule. This can be done by running

```
> blame learned(time0, pkt0)
```

on the first model and

```
> blame learned_P(time0, pkt0)
```

on the second model. For both questions, Razor blames the same instance of the first sequent:

```
rule    :
packet(t, pkt) <=> learned_P(t, locSw(pkt), locPt(pkt), dlSrc(pkt))
                 |  learned(t, locSw(pkt), locPt(pkt), dlSrc(pkt));
instance:
packet(time0, pkt0) <=> learned_P(time0, sw0, pkt0, dest0)
                      |  learned(time0, sw0, pkt0, dest0)
```

Observe that neither of the (minimal) models, returned for one packet, had the other three rules fired: the deletion rule requires another packet with the same mac address to overwrite the port in the learned table. Similarly, the two forwarding rules require at least another learned packet as the destination of the packets their forward.

Provenance of Forwarding Event

Augmenting either of the initial models results in plenty of models. The combination of insertion, deletion, previously learned information, and disjunctions produce a large number of permuted states. Here, we pay attention to the models that trigger the forwarding rules; specifically, the models in which packets are forwarded to different destinations. Consider the following model:

```
packet    = {(time0, pkt0), (time1, pkt1)}
dlDst     = {(pkt0) = dest0, (pkt1) = dest1}
forward   = {(time0, pkt0, port1), (time1, pkt1, port0)}
dlSrc     = {(pkt0) = src0, (pkt1) = src1}
learned_M = {(time1, sw0, pkt1, dest0)}
locPt     = {(pkt0) = port0}, (pkt1) = port1}
learned   =
       {(time0, sw0, port0, src0) , (time0, sw0, port1, dest0),
        (time1, sw0, port0, src0}), (time1, sw0, port1, dest0),
        (time1, sw1, port0, dest1), (time1, sw1, port1, src1) ,
        (time2, sw0, port0, src0) , (time2, sw1, port0, dest1),
        (time2, sw1, port1, src1) }
locSw     = {(pkt0) = sw0, (pkt1) = sw1}
succ      = {(time0) = time1, (time1) = time2}
```

While both arriving packets have been forwarded, it is not clear *why* each was forwarded. The provenance information generated by Razor can help the user understand the reason:

```
> blame forward(time0, pkt0, port1)
```

The previous command blames the third sequent for forwarding the first packet:

```
rule    :
packet(t, pkt) & learned(t, locSw(pkt), locPt(new), dlDst(pkt))
              <=> forward(t, pkt, locPt(new));
instance:
packet(time0, pkt0) & learned(t0, sw0, port1, dest0)
              <=> forward(time0, pkt0, port1)
```

Similarly,

```
> blame forward(time1, pkt1, port0)
```

blames the fourth sequent for forwarding the second packet:

```
rule     :
packet(t, pkt) <=> forward(t, pkt, np)
                |  exists ANY .
                          learned( t, locSw(pkt), ANY, dlDst(pkt))
                |  locPt(pkt) = np;
instance :
packet(time1, pkt1)
      <=> forward(time1, pkt1, port0)
       |  exists ANY.learned(time1, locSw(pkt1), ANY, dlDst(pkt1))
       |  locPt(pkt1) = port0
```

In any Flowlog program with multiple rules firing the same outgoing events, provenance information can elucidate the rule that caused the event to happen.

6.2 Performance Evaluation[2]

The current implementation of Razor is essentially a proof of concept to evaluate the usability and computability aspects of our model-exploration framework. We have not been primarily concerned with efficiency of our tool in this stage of the project: we have been content to establish that Razor is usable for many (human-crafted) theories.

We developed a set of examples, including the ones presented in Section 6.1, to demonstrate various features of our model-finding framework, implemented into Razor [3]. In addition to these examples, we translated specifications from the Alloy repository to Razor's input syntax and evaluated Razor on them (see Section 6.2.1). These experiments have given us a clear insight into Razor's usability as a tool for understanding the of models of theories.

TPTP Examples. We performed several experiments running Razor on the satisfiable TPTP problems [33]. For these experiments, Razor was bounded at depth 2 in the reuse mode. Razor's overall performance on these problems is currently not satisfactory; it frequently fails to terminate within a five-minute bound. Razor shows a relatively better performance on CAT (60%), MGT (73%), GRP (58%), PUZ (50%), and NLP (33%). Razor tends to perform better on problems that are developed by hand, have a limited number of predicates, and do not include high-arity relations.

Since the TPTP library includes solutions for its problems, used this library as a comprehensive set of test-cases for testing the correctness of Razor's implementation. This library helped us with uncovering bugs as well as discovering the sources of inefficiency in our program.

[2]The experiments and their corresponding results reported in this section are designed and computed by graduate student Ryan Danas

[3]All examples are available at `https://github.com/salmans/Razor/`

Theory	depth	# models	time
Bday(1)	unbounded	0	86 ms
Bday(2)	unbounded	1	122 ms
Bday(3)	unbounded	1	226 ms
Gene	2*	18	27.7 sec
Grade(1)	unbounded	2	681 ms
Grade(2)	unbounded	1	835 ms
Gpa	3	2	375 ms
File(1)	1	0	17.2 sec
File(2)	1*	24	2.13 sec
Java	0*	1	425 ms

Table 6.1: Razor's performance on examples from Alloy and Aluminum

6.2.1 Alloy and Aluminum Examples

We evaluated Razor on the sample specifications we had previously used to evaluate Aluminum (Section 7.4). These examples tend to have qualities that make them suitable for the kind of analysis that Razor offers: (i) they are written by users who are investigating the logical consequences of axioms that describe a system; (ii) they are human-tractable; and, (iii) they often have "surprising" models, which provoke the user's curiosity.

We manually translated these specifications from Alloy's language to Razor's input syntax in geometric form. Although the translation preserved the essence of each examples, we felt free to alter the specifications in a way that is more consistent with the inherent positivity of geometric theories. We also used Skolemization when necessary.

Figure 6.1 shows the number of models and the execution time of Razor for these examples. The first column represents the specifications birthday (Bday), genealogy (Gene), gradebook (Grade) grandpa (Gpa), filesystem (File), and java (Java). All examples except Grade are taken from the Alloy distribution. The numbers in parentheses distinguish variations of a root specification. The second column is the depth by which the search for models was bounded. The third column displays the number of models that Razor generated at the given bound. And, the last column shows the time to compute the first model—if there are any models—or to report no models.

Because we were interested in evaluating Razor's performance as a model-finder, we ran a bounded search in the reuse mode. Imposing a deeper bound (preferably unbounded) for the search is always desirable: a deeper search not only increases the chance of finding models—assuming the theory is satisfiable,— but also it results in models that are relatively closer to the homomorphically minimal ones. However, the time to compute the set of possible facts, thus, the size of the ground sequents that are passed to the SMT-solver, increases exponentially with the depth of search.

In Figure 6.1, the numbers in the second column that are labelled with * indicate the maximum depth at which Razor tractably computed models.

Reporting Unsatisfiability. Reporting unsatisfiability—at any given bound—by Razor is reassuring. File(1) tries to construct an example to show moving files in the specified filesystem is "unsafe". A conventional model-finder such as Alloy or Aluminum can guarantee that the specification has no models only up to a given bound. However, because Razor is refutationally complete, it can guarantee that moving files in the filesystem is always safe.

Computing a Set-of-Support. Razor can perform unbounded search for models of weakly acyclic theories (*e.g.*, Bday, Grade). The resulting set-of-support for a weakly acyclic theory assures the user that any other model of the theory is reducible to some model returned by Razor.

Observe that as a direct result of refutation completeness of Razor, it is not always necessary to run an unbounded search to compute a set-of-support. Specifically, if Razor returns a set of models, *without having to introduce accidental collapses* at a given depth, the resulting set of models will be a set-of-support for the input theory. In this situation, increasing the depth of search will not have any impact on Razor's output.

For instance, all three model-finders, Alloy, Aluminum and Razor, return exactly two models for the specification of the well-known "I'm my own Grandpa" example (Gpa). Since Alloy and Aluminum perform a bounded search—for models with up to 4 persons,— it is not clear if the given bound is sufficient to contain all scenarios in which a person is his own grandpa. However, because Razor generates exactly two models that are homomorphically minimal at depth 3 (this can be verified by running Razor with partial model bounding), the user can conclude that in every model of this theory, a person is his own grandpa because of only two specific combinations of relations between exactly 4 individuals.

Chapter 7

Aluminum[1]

Aluminum is a tool for systematic model exploration [32], developed as a modification of Alloy [15] and its backend model-finding engine, Kodkod [59]. Aluminum accepts inputs in Alloy's specification language and utilizes Alloy's visualizer to present models in the form of Alloy *instances*. As a preliminary implementation of our ideas for model exploration, Aluminum offers the following features:

1. *Minimal Model Construction.* Aluminum returns a set of models that are minimal under the *containment* ordering on models up to the input bounds (*i.e.*, Alloy *scopes*).

2. *Augmentation.* Aluminum allows the user to explore the consequences of *augmenting* an existing model with additional facts (*i.e.*, Alloy *tuples*).

3. *Consistent Facts.* Aluminum computes a set of facts (within the given bounds) that can be used to consistently augment a given model of the specification.

For a model returned by Aluminum, the user can be confident that every fact in the model is necessary for the model to satisfy his specification. By browsing the initial set of models, the user can quickly obtain an overall sense of the range of models engendered by the specification [32].

7.1 Minimal Model Construction

After translating the input specification to a propositional constraint (*i.e.*, a Kodkod problem) up to the given bounds, Aluminum invokes a SAT-solver (*i.e.*, SAT4J [60]) to find an initial model (*i.e.*, Kodkod instance). The initial model is then passed to the Minimize procedure, illustrated by Algorithm 10, to obtain a "minimal" model. The function Minimize repeatedly applies the Reduce procedure of Algorithm 11 until a fixed-point is reached. In every invocation of Reduce for a model $\mathbb{M}$, the

[1]The written material and the evaluation results in this chapter are primarily from [32].

SAT-solver is asked for a model that is strictly contained in $\mathbb{M}$ by adding two sets of propositional clauses to the initial propositional CNF:

- The conjunction of the negation of the propositions corresponding to the facts that are not true in $\mathbb{M}$.
- The disjunction of the negation of the propositions corresponding to the facts that are true in $\mathbb{M}$.

Intuitively, the first clause set asks the SAT-solver for a model in which every fact that is false in $\mathbb{M}$ remains false. The second clause requires a model in which at least a true fact in $\mathbb{M}$ is not true. Clearly, the resulting model (if exists) will be a submodel of $\mathbb{M}$.

Notice that because $\mathbb{M}$ may contain multiple submodels that satisfy the resulting propositional constraint, the SAT-solver chooses one of the possible submodels non-deterministically. The model returned by Minimize, a model that cannot be reduced any further, is a minimal submodel of $\mathbb{M}$.

Algorithm 10 Minimize

Require: $\mathbb{M} \models \varphi$

```
1: function MINIMIZE(φ, M)                          ▷ φ is the input formula
2:     repeat
3:         N ← M
4:         M ← REDUCE(M)
5:     until M = N                                  ▷ Cannot minimize any more
6:     return N                                     ▷ N is a minimal model for φ
```

Algorithm 11 Reduce

Require: $\mathbb{M} \models \varphi$

```
1: function REDUCE(φ, M)
2:     C ← ⋀{¬p | p is false in M}
3:     D ← ⋁{¬p | p is true in M}
4:     if there is a model N such that N ⊨ φ ∧ C ∧ D then
5:         return N
6:     else
7:         return M                                 ▷ M is minimal.
```

7.2 Consistent Facts

A *consistent* fact F for a model $\mathbb{M}$ of a theory $\mathcal{T}$ is a fact that can be added to $\mathbb{M}$ (possibly in presence of additional facts) resulting in a model $\mathbb{N}$ of $\mathcal{T}$. Aluminum

implements Algorithm 12 to compute a list all facts that are consistent with a model $\mathbb{M}$ of a formula φ.

Algorithm 12 ConsistentFacts

Require: $\mathbb{M} \models \varphi$

```
1: function ConsistentFacts(φ, M)
2:     C ← ⋀{p | p ∈ M}
3:     D ← ⋁{p | p ∉ M}
4:     R ← ∅
5:     repeat
6:         if there is a model N such that N ⊨ φ ∧ C ∧ D then
7:             F ← {p | p ∈ N and p ∉ M}
8:             R ← R ∪ F
9:             D ← D − F
10:    until no change in R return R
```

7.3 Augmentation

The minimal models that are initially returned by Aluminum are starting points for exploring the space of models. The initial models only contain facts that are *necessary* for satisfying the user's specification; yet, they do not deliver any information about *optional* facts in each model.

Augmentation allows the user to construct non-minimal models by enriching minimal models with user's optional facts. The augmentation feature of Aluminum automatically computes the set of facts that are implied by the augmenting fact in the context of the current model. If an augmentation fails with no models, the user learns about the inconsistency between the augmenting fact and the facts contained in the current model.

A key observation is that augmenting a model of a specification by a fact is merely an instance of the core problem of minimal-model generation: the result of augmentation is precisely an iterator over the minimal models of the specification given by the original specification, along with the facts of the given model plus the new augmenting facts [32]. Therefore, Algorithm 10, which is primarily used for computing minimal models, may also be used for augmentation.

7.4 Empirical Results

We compared Aluminum to Alloy numerically [32]:

Spec.	Models (Alloy)	Models (Alum.)	Cone Coverage	Min. Model Coverage	Ordinal (Alloy)	Ordinal (Alum.)
Addr	2,647	2	1	5	8	3
Bday (2)	27	1	1	3	3	1
Bday (3)	11	1	1	1	1	1
Gene	64	64	64	64	2,080	2,080
Gpa	2	2	2	2	3	3
Grade (1)	10,837	3	289	10,801	11,304	6
Grade (2)	49	3	2	12	33	6
Grade (3)	3,964	1	1	105	105	1
Hanoi (1)	1	1	1	1	1	1
Hanoi (2)	1	1	1	1	1	1
Java	1,566	3	374	1,558	4,573	6

Table 7.1: Alloy and Aluminum's coverage of minimal models and their cones

- We studied how the resulting models mathematically compare to those produced by Alloy.
- We ran experiments to evaluate how long it takes to compute minimal models using our recursive algorithm.

We conducted our experiments over the following specifications. In the tables, where a file contains more than one command, we list in parentheses the ordinal of the command used in our experiments. The following specifications are taken from the Alloy distribution: Addressbook 3a (Addr), Birthday (Bday), Filesystem (File), Genealogy (Gene), Grandpa (Gpa), Hanoi (Hanoi), Iolus (Iolus), Javatypes (Java), and Stable Mutex Ring (Mutex). In addition, we use three independent specifications: (1) Gradebook (Grade), which is defined in Figure 7.1, and enhanced with two more commands:

```
run WhoCanGradeAssignments for 3 but
1 Assignment, 1 Class, 1 Professor, 3 Student
```

and

```
run {some Class} for 3
```

(2) Continue (Cont), the specification of a conference paper manager, from our prior work. (3) The authentication protocol of Akhawe, et al.'s work [61] (Auth), a large effort that tries to faithfully model a significant portion of the Web stack.

7.4.1 Model Comparison

We considered a set of satisfiable specifications for which we could tractably enumerate *all* models. This lets us perform an exhaustive comparison of the models generated by Alloy and Aluminum. The results are shown in Figure 7.1.

```
abstract sig Subject {}
sig Student extends Subject {}
sig Professor extends Subject {}
sig Class {
      TAs: set Student,
      instructor: one Professor
}
sig Assignment {
      forClass: one Class,
      submittedBy: some Student
}
pred PolicyAllowsGrading(s: Subject,
                         a: Assignment) {
      s in a.forClass.TAs or
      s in a.forClass.instructor
}
pred WhoCanGradeAssignments() {
      some s : Subject | some a: Assignment |
         PolicyAllowsGrading[s, a]
}
run WhoCanGradeAssignments for 3
```

Figure 7.1: A simple gradebook specification

The first (data) column shows how many models Alloy generates in all. The second column shows the corresponding number of minimal models presented by Aluminum. The third column shows how many models it takes before Alloy has presented at least one model that is reachable from every minimal model from Aluminum via augmentation. Because a given model can be reachable from more than one minimal model, the number of models needed for coverage may in fact be fewer than the number of minimal models. The fourth column shifts focus to minimal models. If a user is interested in only minimal models, how many scenarios must they examine before they have encountered all the minimal ones? This column lists the earliest scenario (as an ordinal in Alloy's output, starting at 1) when Alloy achieves this. This number does not, however, give a sense of the distribution of the minimal models. Perhaps almost all are bunched near the beginning, with only one extreme outlier. To address this, we sum the ordinals of the scenarios that are minimal. That is, suppose Aluminum produces two minimal models. If the second and fifth of Alloy's models are their equivalents, then we would report a result of $2 + 5 = 7$. The fifth and sixth columns present this information for Alloy and Aluminum, respectively. The sixth column is technically redundant, because its value must necessarily be $1 + \cdots + n$ where n is the number of Aluminum models; we present it only to ease comparison. By comparing the distribution of models returned by Aluminum to those of Alloy we can see Aluminum's impact on covering the space of scenarios.

Even on small examples such as Grade (2), there is a noticeable benefit from Aluminum's more refined strategy. This impact grows enormously on larger models such as Java and Grade (1). We repeatedly see a pattern where Alloy gets "stuck" exploring a strict subset of cones, producing numerous models that fail to push the user to a truly distinct space of models. Even on not very large specifications (recall that Grade is presented in this paper in its entirety), the user must go over hundreds of models before Alloy presents a model from a class of models that has not shown earlier. The real danger here is that the user will have stopped exploring long before then, and will therefore fail to observe an important and potentially dangerous configuration. In contrast, Aluminum presents these at the very beginning, helping the user quickly get to the essence of the model space.

The Gene specification presents an interesting outlier. The specification is so tightly constrained that Alloy can produce *nothing but minimal models*! Indeed (and equivalently), it is impossible to augment any of these models with additional facts, as illustrated by Figure 7.4.

7.4.2 Scenario Generation

We conducted another set of experiments to evaluate Aluminum's running time [32]. All experiments were run on an OS X 10.7.4 / 2.26GHz Core 2 Duo / 4Gb RAM machine, using SAT4J version `2.3.2.v20120709`. We handled outliers using one-sided Winsorization [62] at the 90% level. The times we reported are obtained from

Spec.	Aluminum		Alloy		d
	Avg	σ	**Avg**	σ	
Bday (1)	9	8	5	3	1.57
Cont (6)	1	<1	1	<1	-0.37
Cont (8)	5	6	3	<1	5.88
File (2)	5	5	6	4	-0.28
Iolus	5,795	239	5,000	177	4.49
Mutex (3)	18,767	52	8,781	60	165.91

Table 7.2: Relative times (ms) to render an unsatisfiable result

Spec.	Aluminum			Alloy			d
	Avg	σ	N	**Avg**	σ	N	
Addr	13	12	2	8	6		0.89
Auth	800	32		110	36		19.35
Bday (2)	9	9	1	5	6		0.59
Bday (3)	8	6	1	7	6		0.17
Cont (3)	4	2	3	1	<1	9	4.86
File (1)	16	13		7	4		2.39
Gene	10	5		6	5		0.85
Gpa	9	4	2	6	5	2	0.62
Grade (1)	6	6	3	3	3		1.10
Grade (2)	3	4	3	3	4		0.01
Grade (3)	8	6	1	4	4		0.96
Java	5	4	3	2	2		1.11
Hanoi (1)	2,509	11	1	1,274	1,239	1	1.00
Hanoi (2)	14	3	1	10	2	1	2.14

Table 7.3: Relative times (ms) per scenario (minimal, in Aluminum)

Kodkod, which provides wall-clock times in *milliseconds* (ms).

All experiments were run with symmetry-breaking feature of Kodkod turned on. Numbers are presented with rounding, but statistical computations use actual data, so that values in those columns do not follow precisely from the other data shown. Every process described below was run thirty (30) times to obtain stable measurements.

To measure effect strength, we use Cohen's d [62]. Concretely, we subtract Alloy's mean from that of Aluminum, and divide by the standard deviation for Alloy. We use Alloy's in the denominator because that system is our baseline.

Because Aluminum slightly modifies Kodkod to better support symmetry-breaking, we begin by measuring the time to translate specifications into SAT problems. Across all these specifications, Aluminum's translation time falls between 81% and

113% that of Alloy, i.e., our modification has no effective impact. The absolute translation times range from 5ms (for Gradebook) to 55,945ms (for Auth). (We use commas as separators and our decimal mark is a point. Thus 55,945ms = 55.945s = almost 56 seconds.)

Though our focus is on the overhead of minimization, in the interests of thoroughness we also examine unsatisfiable queries. Figure 7.2 shows how long each tool spends in SAT-solving (ignoring translation into SAT, and then presentation) to report that there are no models. The d values show that in some cases, the time to determine unsatisfiability is much worse in Aluminum. The effect is because of the way Aluminum and Alloy handle symmetry-breaking: Aluminum splits the formula produced by Kodkod into two parts, one representing the specification and query and the other capturing symmetry-breaking, whereas Alloy keeps the formulas conjoined. The conjoined formula offers greater opportunities for optimization, which the SAT-solver exploits. Nevertheless, we note that even in some of the large effects, the *absolute* time difference is relatively small.

For satisfiable queries, we calculate the time to compute the first ten models. When the tool could not find ten models, the N column shows how many were found (and the average is computed against this N instead).

When queries are satisfiable, Aluminum's performs well compared to Alloy. First, the overall running time is small for both tools, so even large effect sizes have small wall-clock impact. Indeed, in the most extreme case, Aluminum takes only about 1.2 seconds longer, for a total time of 2.5 seconds—surely no user can read and understand a model in less time than that, so Aluminum could easily pre-compute the next model. Second, in many cases Aluminum offers many fewer models than Alloy, helping users much more quickly understand the space of models. Finally, the time taken by Kodkod to create the SAT problem can be vastly greater than that to actually solve it, which suggests that a more expensive SAT-solving step will have virtually no perceptible negative impact on the user experience.

We observe two outliers in the data. First, the time for minimization for Auth is very significant. For this specification, we found that the number of extraneous tuples eliminated during minimization is 78 on average. This shows a direct trade-off: 0.7 seconds in computing time for a possibly great impact on user comprehension. Second, the standard deviation for Alloy on Hanoi (1) looks enormous relative to the mean. This is because Kodkod is producing a second duplicate model (which in fact is discarded by the Alloy user interface) very quickly. This results in datapoints that take a long time for the first solution but close to zero for the second. We also examined the time taken by Aluminum's exploration features: how long it takes to compute the consistent facts, and how long it takes to augment a model. Since the complete space of models is enormous, we restrict attention to the first set of models produced by Aluminum.

Figure 7.4 shows the times for computing consistent facts. We computed up to ten models (five, in the case of Auth) and for each of these we determined the number of consistent facts. The first column indicates the number of consistent

Spec.	# Cons. Tupls	Cons. Tup. Time		Aug. Time
		Avg	σ	
Addr	57	45	25	4
Auth	1,335	106,456	17,268	194
Bday (2)	20	19	24	6
Bday (3)	8	9	10	2
Cont (3)	2	5	<1	3
File (1)	24	29	16	3
Gene	0	2	<1	N/A
Gpa	0	1	<1	N/A
Grade (1)	25	13	10	1
Grade (2)	6	2	1	1
Grade (3)	36	14	7	1
Java	25	17	11	2
Hanoi (1)	0	4	<1	N/A
Hanoi (2)	0	1	1	N/A

Table 7.4: Times (ms) to compute consistent tuples and to augment scenarios

facts found, averaged over the number of minimal models (*i.e.*, consistent facts per model). The zero-values are not errors: they arise because the specifications are sufficiently constrained that there is no room for augmentation beyond the initial models. The next two columns show how long it took to compute the number of consistent facts. These numbers are clearly very modest. Auth is the exception: it produces models with, on average, over 1,300 consistent tuples, more than can be explored by hand.

The last column shows how long, on average, it takes to augment a model with a consistent fact (backtracking between each augmentation). The "N/A"s correspond to specifications that have no more consistent facts, and hence cannot be augmented.

7.5 Summary

Aluminum helped us develop a practical insight into the idea of systematic model exploration. Specifically, we learned that

1. Construction of minimal models significantly reduces the number of models returned by the model-finder. This helps the user quickly get a sense of the scope of all models of the theory.

2. Aluminum's minimization algorithm based on repeated invocations of a SAT-solver efficiently computes minimal models.

3. Augmentation is an intuitive strategy for exploring the set space of all models starting from a set of minimal models.

We also realized that the feature that computes a set of consistent facts for a given model is not practically useful. The number of consistent facts dramatically increases with the size of bounds on models, which is often not human-tractable. Notice that the user could simply try augmenting the model with some fact of interest to test its consistency.

Razor vs. Aluminum

As we discuss in Chapter 5, Razor's minimization algorithm has been inspired by that of Aluminum, but the two algorithms behave differently. The domain of elements for the models that Aluminum generates is fixed by a user-specified bound, resulting in a cruder notion of containment on the models of the theory. Consequently, Aluminum does not support introducing "fresh" elements (outside of the specified domain) when augmenting a model. This considerably restricts the models that are accessible by augmentation.

Furthermore, Aluminum does not treat signatures with equality. As a result, the initial minimal models returned by Aluminum may contain "accidental" identification among elements, which cannot be justified in the user's specification. Also, the lack of treatment for equality restricts augmentation to non-equational facts.

Nevertheless, the main distinguishing feature of Razor, compared to the existing model-finding tools including Aluminum, is the construction of provenance information. As we explain in Chapter 5, Razor can compute provenance information as a direct result of a "refined" construction of the Skolem-Herbrand base of the input theory, which is computed by operational reading of sequents. In contrast, a mainstream model-finding tool computes models over a set of arbitrary elements that is fixed in advance. The lack of construction of provenance information by Aluminum was the primary motivation for the development of Razor with a completely different algorithmic approach.

Chapter 8

Preliminary Implementations

Before implementing the model-finding algorithm based on SMT-solving, presented in Chapter 5, we experimented with preliminary Chase-based model-finding algorithms. The first model-finder was a naïve implementation of the Chase of Section 8.1. This experiment helped us realize that an efficient implementation of the Chase required a sophisticated mechanism to evaluate sequents in the temporary partial models, computed during a run of the Chase. This observation inspired our next implementation, Atlas, presented in Section 8.2, which incorporated ideas from relational algebra to evaluate sequents in models.

We also realized that a naïve treatment of disjunctions by creating independent search branches was inefficient. Therefore, we developed our current model-finding algorithm that utilized SMT-solving to process disjunctions efficiently.

8.1 BranchExtend Chase

We implemented a straight-forward model-finding algorithm based on the Chase. The core idea of this implementation is to extend the input theory in geometric form, $\mathcal{G}$, with additional sequents inferred from $\mathcal{G}$ until a fixed point is reached.

This implementation maintains a branch of the Chase as a pair $(\mathcal{G}, \mathbb{M})$, consisting of a geometric theory $\mathcal{G}$ and a structure $\mathbb{M}$. The Chase consists of consecutive applications of the operations below:

Branch. Branching maps a branch of the Chase to possibly several branches, induced by disjunctions in the head of a sequent $\sigma \in \mathcal{G}$ whose body is true in $\mathbb{M}$:

$$(\mathcal{G} \cup \{\sigma\}, \mathbb{M}) \rightsquigarrow \bigcup_i \{(\mathcal{G} \cup \{\sigma\}, \mathbb{M}_i)\}$$

Each $\mathbb{M}_i$ is induced by a classical chase-step $\mathbb{M} \xrightarrow{(\sigma,\eta)} \mathbb{M}_i$ when $\mathbb{M} \not\models_\eta \sigma$.

Extend. Let $\sigma \equiv \varphi(\vec{x}) \vdash_{\vec{x}} \psi(\vec{x})$ be a sequent in $\mathcal{G}$ and $\vec{y}$ be a subset of the free variables in σ (*i.e.*, $\vec{y} \subseteq \vec{x}$). Let $\eta : \vec{y} \longrightarrow |\mathbb{M}|$ be a (partial) environment that maps the variables of $\vec{y}$ to elements of $|\mathbb{M}|$. We extend $\mathcal{G}$ by instances of σ that are induced by all possible choices of $\vec{y}$ and η:

$$(\mathcal{G} \cup \{\sigma\}, \mathbb{M}) \;\rightsquigarrow\; (\mathcal{G} \cup \{\sigma, \varphi[\eta\vec{y}] \vdash_{(\vec{x}-\vec{y})} \psi[\eta\vec{y}]\}, \mathbb{M})$$

In a run of the BranchExtend algorithm, the CPU time is primarily spent on the computation of the environment η during the Extend transformations. This is essentially an instance of the well-known *term-matching* problem, which can be done quickly [21]. However, the cost of this computation exponentially grows with the size of $\mathbb{M}$ and the number of sequents in $\mathcal{G}$, which is practically inefficient. In Section 8.2.1, we discuss an alternative solution for evaluating sequents in the current model constructed by the Chase, based on relational algebra.

Nevertheless, the primary source of inefficiency to maintain every branch of the Chase as a separate theory and model pair. This often leads to an exponential blow-up in the size of the memory consumed by the program. Furthermore, the BranchExtend is unable to share computation across the multiple branches. Consider the following example (a is a constant):

Example

$$\vdash \quad (P(a) \wedge R(a)) \vee (Q(a) \wedge R(a)) \tag{8.1}$$

$$R(x) \quad \vdash \quad S(x); \tag{8.2}$$

A run of the Chase that starts by processing the sequent 8.1 constructs two (partial) models, $\{P(\mathbf{e_1}), R(\mathbf{e_1}), a = \mathbf{e_1}\}$ and $\{Q(\mathbf{e_1}), R(\mathbf{e_1}), a = \mathbf{e_1}\}$. Any naïve implementation of the Chase, including the BranchExtend algorithm, would reprocess the sequent 8.2 in both branches. A more sophisticated algorithm, such as the current implementation of Razor (Section 5), would avoid such re-computations.

8.1.1 Models as Term-Rewrite Systems

First-order models are often implemented as a domain of elements together with a set of relational facts over closed terms over the elements. Challenges arise in presence of equality: in the presence of the equality relations, standard equality and congruence axioms must be maintained as additional invariants over models:

$$\vdash \quad x = x \tag{8.3}$$

$$x = y \quad \vdash \quad y = x \tag{8.4}$$

$$x = y \wedge y = z \quad \vdash \quad x = z \tag{8.5}$$

$$R(\ldots, x, \ldots) \wedge x = y \quad \vdash \quad R(\ldots, y, \ldots) \tag{8.6}$$

$$f(\ldots, x, \ldots) = z \wedge x = y \quad \vdash \quad f(\ldots, y, \ldots) = z \tag{8.7}$$

$$f(\vec{x}) = y \wedge f(\vec{x}) = z \quad \vdash \quad y = z \tag{8.8}$$

A straight-forward approach is to maintain equational facts as tuples of an ordinary relation in models, also to process the equality and congruence axioms as ordinary sequents. This solution, however, is inefficient in practice.

For a more efficient implementation, the BranchExtend implementation stored models as *ground term-rewrite systems* (TRS) [63]. Consequently, we implemented a *congruence closure* algorithm for efficient equational reasoning.

Congruence Closure. We adopt the definition of *congruence closure* in [47]:

Definition Fix a signature Σ and a set of constants $\mathcal{K}$.

- A *C-rule* is a rewrite rule $c \to d$, where $c, d \in \mathcal{K}$.
- A *D-rule* is a rewrite rule $f(c_1, c_2, \ldots, c_k) \to c$, where $f \in \Sigma$ is a k-ary function and $c_1, c_2, \ldots, c_k, c \in \mathcal{K}$.

The abstract congruence closure is a term-rewrite system (TRS) that captures the equality relation over a set of constants by a set of C-rules and D-rules:

Definition Fix a signature Σ and a set of constants $\mathcal{K}$ disjoint from Σ. Let $\mathcal{E}$ be a set of ground equations over Terms$(\Sigma \cup \mathcal{K})$. A ground TRS $\mathcal{R}$, containing only C-rules and D-rules over Σ and $\mathcal{K}$, is a *congruence closure* for $\mathcal{E}$ if and only if:

(i) every constant $c \in \mathcal{K}$ represents some $t \in$ Terms(Σ) in $\mathcal{R}$.

(ii) $\mathcal{R}$ is ground convergent.

(iii) for all terms t and u in Terms(Σ), $t \leftrightarrow_E^* u$ if and only if $t \downarrow = u \downarrow$ in $\mathcal{R}$.

The problem of constructing a congruence closure TRS for a set of equations E is a well-studied problem with several efficient algorithms [47, 63–66]. An efficient implementation of any of the existing algorithms requires sharing and mutation, which are not available to pure functional programming languages such as Haskell. For an experiment, we implemented a naïve algorithm based on the transformations suggested in [47].

Data-Structure for Models. Fix a geometric theory $\mathcal{G}$ over signature Σ, and let E be a set of equations generated by $\mathcal{G}$ in a run of the Chase. A TRS model $\mathbb{M} \equiv (\mathcal{K}, \mathcal{R})$ consists of a set of constants, $\mathcal{K}$, and a congruence closure TRS, $\mathcal{R}$, where:

- $\mathcal{K} = |\mathbb{M}| \cup \{\mathsf{True}\}$, where True is a special constant.
- For any $t, t' \in$ Terms$(\Sigma \cup \mathcal{K})$, $s \leftrightarrow_E^* t$ if and only if $s \downarrow = t \downarrow$ in $\mathcal{R}$.
- Let $R(t_1, t_2, \ldots, t_k)$ be a an atomic formula where $R \in \Sigma$, and $t_1, t_2, \ldots, t_k \in$ Terms(Σ). $R(t_1, t_2, \ldots, t_k)$ is true in $\mathbb{M}$ if and only if $R(t_1, t_2, \ldots, t_k)$ rewrites to True in $\mathcal{R}$; *i.e.*, $(R(t_1, t_2, \ldots, t_k)) \downarrow_{\mathcal{R}} = \mathsf{True}$.

8.2 Atlas

As mentioned earlier, the main cost of applying a chase-step on a sequent $\sigma = \varphi \vdash_{\vec{x}} \psi$ and a model $\mathbb{M}$ is due to computation of an environment η in which $\mathbb{M}$ does not satisfy σ. Because it requires testing σ in $\mathbb{M}$ for every map from $\vec{x}$ to $|\mathbb{M}|$, the cost of computation grows exponentially with the size of $|\mathbb{M}|$. To address this problem, we implemented Atlas, a variation of the Chase from a relational algebraic perspective, whereby sequents in an input theory were evaluated, as queries, in first-order models, as databases. This idea significantly improved the efficiency of constructing the environment η during a chase-step.

Due to inefficient treatment of disjunctions, we did not stay with Atlas as our final implementation; however, we founded the grounding algorithm (Section 5.2) of Razor based on an implementation of the Chase from a relational algebra perspective.

8.2.1 Relational Algebra for the Chase

As a result of Codd's theorem [67], a PEF φ can be regarded as a database *view* with a relational expression V_φ. Consequently, a geometric sequent $\varphi \vdash_{\vec{x}} \psi$ is a pair of *union-compatible* views, V_φ and V_ψ, with $\vec{x}$ as attributes.

Let $V_\varphi(\mathbb{M})$ denote the set of tuples returned by evaluating V_φ in a model $\mathbb{M}$ as database. Then, $\mathbb{M}$ satisfies the sequent $\varphi \vdash_{\vec{x}} \psi$ if and only if $V_\varphi(\mathbb{M}) \subseteq V_\psi(\mathbb{M})$. That is, a chase-step on the sequent $\varphi \vdash_{\vec{x}} \psi$ in model $\mathbb{M}$ is as a procedure that inserts the tuples of $V_\varphi(\mathbb{M}) - V_\psi(\mathbb{M})$ into $V_\psi(\mathbb{M})$, an instance of a *view update problem*. By adopting the relational algebra perspective, an environment η is implicitly constructed as a mapping from $\vec{x}$ to the tuples of $V_\varphi(\mathbb{M}) - V_\psi(\mathbb{M})$.

Conversion to Relational Algebra

After relationalization, the heads and bodies of the resulting sequents are converted to pairs of relational expressions by the standard transformations of first-order logic to relational algebra [67]. A relational expression V_φ corresponding to a geometric formula φ is inductively defined by the following:

$$V_\varphi = \begin{cases} R & \varphi = R(\vec{x}) \\ V_\alpha \bowtie V_\beta & \varphi = \alpha \wedge \beta \\ V_\alpha + V_\beta & \varphi = \alpha \vee \beta \\ \Pi_x(V_\alpha) & \varphi = \exists x.\alpha \end{cases}$$

A geometric sequent $\varphi \vdash \psi$ maps to a pair of relational expressions, V_φ and V_ψ. For a standard geometric sequent, $\varphi \vdash \bigvee_i \psi_i$, we found it more convenient to keep every disjunct ψ_i by a separate relational expression; thus, we maintain a standard sequent by a single relational expression V_φ in its body and a set of relational expressions $\bigcup_i \{V_{\psi_i}\}$—which in theory can be infinite due to infinitary disjunctions— in its head.

Notation. We overload the connective $\vdash$ to denote a geometric sequent $V_\varphi \vdash V_\psi$ as a pair of relational expressions. We also write $V_\varphi \vdash \bigcup_i \{\psi_i\}$ for a standard sequent.

Models as Databases. After relationalizing the input theory $\mathcal{G}$ (by the transformations in Section 5.1.1), we can store a model $\mathbb{M}$ of $\mathcal{G}$ as a database, consisting of a set of relational tables. Every row $\langle \mathbf{e_1}, \ldots, \mathbf{e_k} \rangle$ of a table T_R assigned to a k-ary relation R represents a relational fact $R_k(\mathbf{e_1}, \ldots, \mathbf{e_k})$ in $\mathbb{M}$.

Consider a relational expression V_φ for a geometric formula φ. Let $V_\varphi(\mathbb{M})$ be the result of evaluating V_φ (as query) in a model $\mathbb{M}$ (as database). Observe that for $\vec{v} \equiv \langle v_1, \ldots, v_k \rangle = \mathrm{FV}(\varphi)$ and every k-tuple $\vec{\mathbf{e}} \equiv \langle \mathbf{e_1}, \ldots, \mathbf{e_k} \rangle \in V_\varphi(\mathbb{M})$, an environment $\eta : \vec{v} \longrightarrow \vec{\mathbf{e}}$ makes φ true in $\mathbb{M}$; *i.e.*, $\mathbb{M} \models_\eta \varphi$. It thus follows that a geometric sequent $\sigma \equiv V_\varphi \vdash V_\psi$ is true in a database model $\mathbb{M}$ if and only if $V_\varphi(\mathbb{M}) \subseteq V_\psi(\mathbb{M})$. This leads us to a variation of the Chase based on relational algebra.

Database Chase

Algorithm 13 is a variant of the classical Chase from a relational algebra perspective. Similar to the standard Chase illustrated by Section 2, a non-deterministic run of Algorithm 13 constructs a model as a relational database.

Algorithm 13 Database Chase

```
1:  function DBChase(𝒢)
2:      𝕄 ← ∅                                          ▷ start with an empty model
3:      while 𝕄 ⊭ 𝒢 do
4:          choose (V_φ ⊢_x⃗ ⋃_{i=1}^{n} {V_ψi}) ∈ 𝒢 such that V_φ(𝕄) ⊈ V_ψi(𝕄) (for all i)
5:          if n = 0 then
6:              fail                                   ▷ The sequent has ⊥ in head
7:          choose a view V_ψj (1 ≤ j ≤ n)
8:          for each t ∈ (V_φ(𝕄) − V_ψj(𝕄)) do
9:              𝕄 ← Update(V_ψj, t, 𝕄)                 ▷ update V_ψj with tuple t in 𝕄
10:     return 𝕄
```

Algorithm 13 reduces to the two following well-studied problems in the database literature:

1. *View Update*: the update to the view V_{ψ_j} on line 9 is a straigh-forward view update problem, limited to inserting tuples.

2. *View Maintenance*: as a result of inserting t into V_{ψ_j} in line 9, every view in the heads and bodies of all sequent in $\mathcal{G}$ must be updated according to this last update to $\mathbb{M}$. This is an instance of a view maintenance problem.

Incremental View Maintenance

Implementing the Chase based on relational algebra makes a range of techniques, developed in the database community, available to improve the efficiency of the Chase. Inspired by the idea of *incremental view maintenance*, our relational implementation of the Chase incrementally evaluates the body of sequents in models, based on algebraic differencing of relational expressions. For every sequent with a relational expression V_φ in its body, we compute a differential expression ΔV_φ by the device described below. The set ΔV_φ can then be used to compute the changes in the body of the sequent after each update to the model. Consequently, Algorithm 13 can efficiently compute the set of tuples that must be inserted into the head of the sequent by a chase-step.

Differential Expression. Assume that $V \equiv V^{exp}(T_1, \ldots, T_n)$ is a relational expression over tables $T_1, \ldots, T_n$. Also, let $\Delta T_1, \ldots, \Delta T_n$ denote *positive* changes, induced by inserting tuples, to $T_1, \ldots, T_n$. A relational expression ΔV, defined by the following induction, captures the changes to V:

- $\Delta(\sigma_{C(Y)}(T)) = \sigma_{C(Y)}(\Delta T)$
- $\Delta(\Pi_X(T)) = \Pi_X(\Delta T)$
- $\Delta(S \bowtie_{C(Y)} T) = (\Delta S \bowtie_{C(Y)} T) \cup (\Delta S \bowtie_{C(Y)} T) \cup (\Delta S \bowtie_{C(Y)} \Delta T)$

Here, $\sigma_{C(Y)}(T)$ selects those tuples in T for which $C(Y)$ holds; $\Pi_X(T)$ projects T onto a set of attributes X; and, $S \bowtie_{C(Y)} T$ is equivalent to $\sigma_{C(Y)}(S \times T)$. A procedural approach to the previous computation is discussed in [56].

8.2.2 Scheduling

Atlas maintains the state of every run of the Chase on an input (relational) theory $\mathcal{G}$ by a *problem*, a structure consisting of a model $\mathbb{M}$ and a queue $\mathcal{Q}$ of sequents. The model $\mathbb{M}$ is the database that has been constructed during the current run of the Chase, according to Algorithm 13. The queue $\mathcal{Q}$ is used to schedule the sequents of $\mathcal{G}$ in a fair manner.

In every execution of Atlas, the global state of computation is a *pool* of problems, initialized with a single problem with an empty model. In every cycle, the program chooses a problem $(\mathcal{Q}, \mathbb{M})$ from the pool; selects a sequent σ from the head of $\mathcal{Q}$; computes a model $\mathbb{M}'$ by a chase-step $\mathbb{M} \xrightarrow{(\sigma,\eta)} \mathbb{M}'$ (for some environment η such that $\mathbb{M} \not\models_\eta \sigma$); reschedules σ at the end of the queue yielding $\mathcal{Q}'$; and, restores $(\mathcal{Q}', \mathbb{M}')$ into the pool. Once a model $\mathbb{M}$ of $\mathcal{G}$ is obtained, a problem $(\mathcal{Q}, \mathbb{M})$ is removed from the pool and $\mathbb{M}$ is presented to the user. Also, the problem is removed from the pool if the Chase fails on $\mathbb{M}$.

Performing a chase-step for a problem $(\mathcal{Q}, \mathbb{M})$ with $\sigma \equiv \varphi \vdash \bigvee_{i=1}^{n} \psi_i$ at the head of $\mathcal{Q}$ yields a set of problems $P_i = (\mathcal{Q}', \mathbb{M}_i)$ for $1 \leq i \leq n$. Each model $\mathbb{M}_i$ is the

result of making a disjunct ψ_i true in $\mathbb{M}$, enabling us to follow the consequences of every choice of disjunct as a separate problem in the pool. This simple idea helps the performance of the program drastically.

In practice, we use a priority queue to schedule the sequents of $\mathcal{G}$ in a problem: this allows the scheduler to prioritize sequents with $\bot$ in the head over the the other sequents that may extend the current model with new facts and elements. This helps Atlas terminate a failing branch before wasting unnecessary chase-steps on the other sequents of the theory.

Strategies for Scheduling the Problems. The performance of Atlas is greatly influenced by the scheduling strategy that selects a problem from the pool. We implemented three scheduling strategies as described below:

1. *Depth First Search (DFS):* the pool of problems is a stack; a problem is restored at the top of the stack once processed. In practice, DFS shows a better average performance to return the first model. However, it may suffer non-termination in a given Chase branch.

2. *Breadth First Search (BFS):* the pool of problems is a queue; a problem is restored at the end of the queue once processed. As predicted, BFS needs a long time to return the first model. However, it guarantees to return a finite model of the theory if it exists.

3. *Round Robin (RR):* a problem is processed for k times, then it is scheduled at the end of the pool. In theory, RR is expected to combine the quickness of DFS and the stability of BFS; however, finding the right k to adjust RR is not a trivial problem. A decent strategy is to grow k exponentially with the number of problems in the pool.

Chapter 9

Related Work

The development of algorithms for the generation of finite models is an active area of research. Two well-known methods are "MACE-style" tools, which reduce the problem to be solved into propositional logic and employ a SAT-solver, and "SEM-style"tools, which work directly in first-order logic (see Section 9.1). A common application is the computation of counterexamples to conjectures that arise in modeling or software verification. Particularly, model-finding supports a form of "lightweight" analysis of software whereby a user understands the implications of a software artifact by studying examples of its execution (see Section 9.2). Our framework for model-exploration has been inspired by this line of research.

The theoretical foundation of Razor, our model-finding assistant for exploring models, is geometric logic. Geometric logic was previously applied in a number of theorem-proving and formal verification applications (see Section 9.3). The crucial difference with the current work is of course the fact that we focus on model-finding and exploration.

9.1 Finite Model-Finding

It is well-known that model-finding for first-order logic is undecidable. Finite models of a first-order theory, however, can be found by exhaustive search. Given a first-order theory $\mathcal{T}$ and a natural number b as a *bound* on the size of models, finite model-finding is conventionally the problem of finding a model $\mathbb{M}$ with a domain of size less than or equal to b.

The two prominent approaches to finite model-finding are MACE-style [2] and SEM-style [20] algorithms. MACE-style model-finders are characterized by the use of propositional SAT-solving, whereas the SEM-style ones, which search for models directly at the first-order level.

9.1.1 MACE-Style Model-Finding

Given an input first-order theory $\mathcal{T}$ over a signature Σ and a bound b on the size of models, a MACE-style model-finder translates $\mathcal{T}$ to a set of propositional clauses equisatisfiable to $\mathcal{T}$ up to b. The bound b corresponds to a domain of elements $\mathcal{D}$ of size b. The propositional variables in the propositional clause set represent values for the entries of operation tables for the predicates and functions of Σ. Solutions of the propositional clauses, as a satisfiability problem, correspond to values of functions and predicates in a first-order model of $\mathcal{T}$.

The Algorithm. MACE-style model-finding proceeds by converting the input first-order theory $\mathcal{T}$ to a CNF, possibly in presence of Skolemization. The next step eliminates the function symbols in $\mathcal{T}$ by *flattening* and *relationalization*, using transformations similar to the ones described in Section 5.1.1. The resulting relational theory $\mathcal{T}_{\mathsf{Rel}}$ is equisatisfiable $\mathcal{T}$ [48, 68, 69]. Next, $\mathcal{T}_{\mathsf{Rel}}$ (in CNF) is translated to an equisatisfiable "propositional" clause set over the input domain of elements $\mathcal{D}$, provided as the bound on the search. The resulting propositional problem is then fed to a SAT-solver.

Saturation. *Saturation* is the process that generates all posssible instances of the relational first-order CNF, $\mathcal{T}_{\mathsf{Rel}}$, over the input domain $\mathcal{D}$ [5, 48, 69]: for every clause $C(\vec{x})$ and environment $\eta : \vec{x} \to \mathcal{D}$, a propositional clause $C(\eta\vec{x})$ is generated. The resulting propositional CNF, will then be added to a set of *integrity* clauses to form the input to the SAT-solver.

Integrity Clause Generation. The saturated propositional CNF is equisatisfiable to the input first-order formula only in presence of additional clauses that enforce *integrity* constraints for the functions of the input signature Σ, over the given finite domain $\mathcal{D}$. The integrity constraints fall into the two following categories: [69]:

1. *Unity Clauses:* for every function f of artiy k in Σ, and any pair of elements $\mathbf{c}, \mathbf{d} \in \mathcal{D}$ ($\mathbf{c} \neq \mathbf{d}$), a set of clauses in the form of $f(\vec{\mathbf{e}}) \neq \mathbf{c} \vee f(\vec{\mathbf{e}}) \neq \mathbf{d}$ is generated, where $\vec{\mathbf{e}}$ is a k-tuple over $\mathcal{D}$.

   The set of unity clauses ensure that for every argument vector over the domain of elements, every function f gets at most one value.

2. *Totality Clauses:* for every function f of arity k in Σ, a set of clauses in the form of $f(\vec{\mathbf{e}}) = \mathbf{d}$ are generated, where $\mathbf{d} \in \mathcal{D}$ and $\vec{\mathbf{e}}$ is a k-tuple over $\mathcal{D}$.

   The totality clauses, interpret f as a total function.

Paradox. Paradox is a quintessential MACE-style model-finding tool [69]. Paradox is an influential model-finder that introduced new ideas to the MACE-style technology, including: (i) a reduction technique, known as *term definitions*, (ii) a method for incremental extension of the search bound, known as *incremental model-finding*, (iii) *static isomorphism elimination*, and (iv) *sort inference* in the context of model-finding.

Kodkod. Kodkod accepts a multi-sorted first-order formula, possibly with transitive closure, and a set of upper and lower bounds on the input sorts as input, and it constructs models for the input formula between the given bounds [68]. The primary features of Kodkod include (i) support for partial models, (ii) a mechanism for detecting and eliminating isomorphic models (known as, symmetry breaking), and (iii) a sophisticated sharing detection algorithm for efficient propositionalization.

Kodkod is a popular model-finding engine, designed to serve as a component that can be easily incorporated into other tools. A variety of model-finding analyzers including Alloy [15, 70], Margrave [13, 71], and Nitpick [11, 72] use Kodkod for their backend solver.

9.1.2 SEM-Style Model-Finding

Unlike the MACE-style algorithm, the SEM-style approach finds models of the input theory $\mathcal{T}$, over a signature Σ, directly at the first-order level (without propositionalization). SEM-style model-finders implement a *backtracking* search algorithm to build interpretation tables, for the functions and relations of Σ, which correspond to finite models of $\mathcal{T}$ up to a given bound b. Every cell of such operation tables stores a value for a ground term or a ground atomic formula [20, 48].

The Algorithm. Given a first-order formula $\mathcal{T}$ (often in CNF) over a signature Σ and a finite domain $\mathcal{D}$, a SEM-style search with backtracking interprets the functions and the predicates of Σ in models of $\mathcal{T}$. Every cell of an interpretation table for a k-ary relation R in Σ holds a Boolean value for $R(\vec{\mathbf{e}})$, where $\vec{\mathbf{e}}$ is a k-tuple over $\mathcal{D}$. Every cell of a table that interprets a k-ary function f in Σ holds a value over $\mathcal{D}$, which represents the value of $f(\vec{\mathbf{e}})$, for a k-tuple $\vec{\mathbf{e}}$ in f.

Value Assignment. The state of computation is maintained as a tuple: $S \equiv (\mathcal{A}, \mathcal{B}, \mathcal{C})$: $\mathcal{A}$ is the set of cells to which values have been assigned, $\mathcal{B}$ is the set of unassigned cells, and $\mathcal{C}$ is a set of constraints over the values of different cells. Initially, all cells are in $\mathcal{B}$, $\mathcal{A}$ is empty, and $\mathcal{C}$ is the input theory $\mathcal{T}$. In this setting, model-finding is the process of moving cells from $\mathcal{B}$ to $\mathcal{A}$, using a recursive depth-first search algorithm, by assigning values to the cells in $\mathcal{B}$ in a way that is consistent with $\mathcal{C}$. After every value assignment, $\mathcal{C}$ is extended with an additional constraint that records the value assignments. The constraints in $\mathcal{C}$, therefore, force further restrictions on values that the remaining unassigned cells can receive.

The search procedure succeeds if all cells can successfully be moved from $\mathcal{B}$ to $\mathcal{A}$. In the case of a contradiction within a branch, the search backtracks and chooses different value-assignments. The search for models (up to the given bound) fails if successful value-assignments for all cells cannot be found [20].

Constraint Propagation. A naïve SEM-style search in the space of all possible interpretations is computationally expensive. The cost of such a naïve search grows exponentially with the number of elements in the input domain $\mathcal{D}$. In order to compensate for the theoretical inefficiency, SEM-style model-finders are often equipped with powerful *constraint propagation* engines, comparable to those developed by propositional SAT-solvers. The constraint propagation engines make use of the logical consequences of value-assignments to diminish the search space for future assignments. This is done by either forcing or restricting the values that can be assigned to the unassigned cells [20, 48].

Common heuristic algorithms that have been utilized by SEM-style model-finders include the *first-fail* principle, *forward checking*, and *isomorphism elimination* [20].

9.1.3 Other Methods

A somewhat different approach to model-finding arises out of "instance based" methods for proof search, which can be adapted to compute finite models [49, 73]. The core idea of this approach is to reduce the model-finding problem to a first-order clause set, and to employ theorem-proving techniques to decide whether a model exists. For efficient equational reasoning, instance based solvers may utilize SMT-solvers [74].

The techniques based on bottom-up model generation [51] read implications as rules that are repeatedly applied to construct models for the input theory. Similar to our algorithm for Razor, bottom-up model generation is not parameterized by the domain size of models, thus, it is refutationally complete. This approach also constructs "small" models, and it makes minimal identification among the elements of models.

Baumgartner and Suchanek [45] present a set of transformations to convert a first-order theory to a *disjunctive logic program*, which can be fed to a bottom-up model-finder. The transformations treat existential quantifiers by standard Skolemization as the basic option. The "recycling option" avoids creation of fresh Skolem terms if an existing Skolem term satisfies the existentailly quantified formula. The "loop check" option tries to reuse existing Skolem terms when possible. These strategies are comparable to the ones presented in Section 4.3.5, used to bound the search by Razor. The authors also provide a set of transformations to efficiently eliminate equality when the underlying model-finder does not offer equational reasoning.

9.2 Lightweight Model-Finding

Inspired by the automated analysis provided by model-checkers [75–80], model-finding has been applied as a "lightweight formal method" [12], where it helps users understand formal specifications. In this setting, models of a theory are *examples* that illustrate the behavior of a software or a hardware specification, written in a declarative formal language. We refer to the approach whereby a user develops her intuition about her specification in interaction with a model-finder as *lightweight model-finding*.

Lightweight model-finders such as Alloy [15, 70] and Margrave [13, 15, 81] are closer in spirit to Razor in the sense that they facilitate an environment in which the user can verify the design of a system through examples of the system's execution. However, the existing work in the lightweight model-finding community is not primarily driven by the idea of *exploring all models* of the input theory.

Alloy. Alloy [15, 70] is a general-purpose specification language and an analyzing tool that has been applied to a broad range of problems, including: design and analysis of software systems [82–89], automated software testing [90–92], business process modelling [93–95], network configuration [96–98], security analysis [99–101], and other applications [102–106]. Alloy allows its users to describe structures in a specification language based on first-order relational algebra and explore examples of the specified structures in a graphical visualizer [70]. Alloy utilizes Kodkod [59], an efficient first-order relational model-finder, to generate examples for the input specification.

The user's interaction with the Alloy analyzer comprises three major activities [107]:

- *Simulation.* the analyzer simulates a specified system by constructing finite models of its specification.
- *Checking.* the analyzer checks if the specified system satisfies a given property by constructing counterexamples that refute the property.
- *Debugging.* the analyzer helps the user to debug an over-constrained specification by highlighting contradictory constraints in the specification [108].

Margrave. Margrave [13, 71, 81] is a tool for analyzing access-control policies. Given a set of policy rules and a query about the given policy, Margrave computes a set of "scenarios" that witness the queried behavior [13]. Margrave allows the user to learn about the consequences of configuration edits, overlaps, and conflicts. Margrave also helps the user identify rules that must be "blamed" for a particular behavior of the policy. The user can utilize the aforementioned features of Margrave to verify the compliance of a given policy with initial security goals. Similar to Alloy, Margrave too relies on Kodkod for model-finding.

CPSA. The Cryptographic Protocol Shapes Analyzer (CPSA) [14, 109, 110] is a tool for analyzing cryptographic protocols. Given some initial behavior corresponding to various parties that are involved in a session of a cryptographic protocol, CPSA enumerates all possible executions, namely *shapes*, of the protocol that are compatible with the given behavior. By investigating all possible shapes, generated by CPSA, the user can discover complete runs of the protocol that correspond to anomalies or attacks.

A distinguishing feature of CPSA, among the mainstream model-finders, is that it generates minimal models, *i.e.*, minimal shapes. However, its application and underlying algorithms are different from those of conventional model-finders.

9.2.1 Lightweight Model-Finding Research

While the primary focus of the automated theorem-proving is to improve the performance of model-finders researchers in the Alloy community have been building a body of research to improve the "quality" of models that are presented to the user. Although development of new ideas for visualizing and presenting the models to the user may be included in this line of research [111–113], we are primarily interested in the underlying model-finding technology that facilitates construction of more effective models for lightweight model-finding.

Automatic Bound Computation

Due to undecidability of first-order logic, model-finders, such as Alloy, hold the user accountable for making a trade-off between the efficiency and completeness of the model-finding procedure. The user is responsible to provide precise bounds that are loose enough to contain *interesting* models and tight enough to ensure the efficiency of the search. Finding the sweet-spot between completeness and efficiency is a domain-specific matter, demanding an experienced user. Furthermore, achieving this balance varies from one problem to another even within the same domain.

An ongoing line of research in the Alloy community focuses on automatically computing lower-bounds and upper-bounds on the sizes of models when possible. While attempts have been made to leverage automatic bound computation to improve the efficiency of the model construction [114–116], the primary goal of this branch of research is to mitigate the burden of bound computation for the user. Momtahan [117] computes an upper-bound on the size of a single sort, as a function of bounds specified by the user on the other sorts, for a restricted fragment of Alloy's core language. Abadi *et al.* [118] identify a decidable fragment of sorted logic whereby the theories written in this fragment are guaranteed to have a finite Herbrand universe. Finally, Nelson *et al.* [119] establish a syntactic condition that generalizes Bernays-Schnöfinkel-Ramsey fragment of first-order logic to ensure the Finite Model Property in a many-sorted framework. The authors provide a linear-time algorithm for deciding the given condition and a polynomial-time algorithm

for computing the bound sizes. They implement their algorithms into Margrave.

Razor, on the other hand, computes the domain of elements by enumerating the terms in the witnessing signature of the input theory. This strategy does not require the user to fix a domain of elements in advanced. Notice that although we need to bound the search for termination, by providing a limit on the depth of the witness terms, our algorithm is not inherently bounded.

As another consequence of inductive enumeration of witnessing terms, Razor is refutationally complete. Conventional model-finders do not distinguish inconsistency in the user's theory form the absence of models up to the given bound. Specifically, it is always probable to see models of the theory by increasing the bound size. In contrast, if Razor terminates with no models (regardless of the depth of search), the user can be confident that his theory is inconsistent and has no models at all.

Debugging Specifications

Another challenging problem of lightweight model-finding is to provide debugging facilities by which a user can correct his specification once he encounters models that do not demonstrate a desired behavior. *Core extraction* and its variations [120–122] are the central facility for debugging Alloy specifications [108]. When a given specification is unsatisfiable, Alloy utilizes the "unsatisfiable cores", provided by the underlying SAT-solver, to identify contradictory constraints. This feature helps the user detect accidental overconstraints and correct the specification accordingly.

In an attempt comparable to the construction of unsatisfiable cores, Wittocx *et al.* [123] construct debugging traces for unsatisfiable theories in situations when *model expansion* is the underlying technique for constructing models. The authors argue that their approach results in more tractable proofs for unsatisfiability when the sizes of the unsatisfiable cores are relatively large.

Finally, Bendersky *et al.* present a debugging visualizer, inspired by programming language debuggers, which generates debugging traces for DynAlloy [106], an extension of Alloy with procedural actions [124]. These traces are constructed based on DynAlloy's atomic actions, which are defined in terms of preconditions and postconditions using standard Alloy *predicates*.

Razor presents provenance information, which can trace any piece of information in a model to a line of the user's specification. The user may effectively use this mechanism to detect bugs in his specification that are responsible for unexpected pieces of information in a given model. An extension of this idea is to construct provenance information for unsatisfiable theories, which is a subject of future work.

Partial Models

Inspired by the work in the model-checking community [125, 126], researchers are investigating the use of *partial* models in lightweight model-finding when part of a solution is already known [127, 128]. These partial models capture known in-

formation about solutions to the user's problem, which are to be completed by the model-finding tool. Partial models are proven to help the efficiency of model generation [68]. Partial models are also useful to serve as regression test-cases for incremental development of specifications [129]. In the context of target-oriented model-finding, a partial model is extended to support the specification of a target solution when model-finding is employed to construct solutions that are as close as possible to the target [130].

Kodkod, Alloy's model-finding backend, is able to exploit the information in initial partial models to efficiently construct full models [68]. Montaghami and Rayside [129, 131] made the Kodkod partial models explicitly available to Alloy users.

In principle, an inductive model-finding algorithm based on the (classical) Chase naturally computes partial models of the theory during its execution. The incomplete models that are computed during a run of the Chase can be sent to (complete) models of the theory by homomorphism (Lemma 4.3.1). In fact, the augmentation feature of Razor exploits this facility: the model being augmented maybe seen as a partial model with regard to the resulting model.

Having said that, exploiting the property mentioned above is more complicated in the two-phase algorithm implemented into Razor, where the Chase-like computation is done before the models are computed by the SMT-solver.

Model Selection

Most *interesting* first-order theories have many models; in fact, the majority of first-order logic specifications have an infinite number of models. Even when models are constrained to be finite —for example, by imposing a size-bound— effective model-finders should identify the best models to present to the user, determine the order by which the models are presented to the user, and facilitate a framework for the user to explore the space of all models.

Alloy attempts to exclude isomorphic models [68] on the grounds that isomorphic models do not provide additional information. Regardless of isomorphism elimination (*i.e.*, symmetry breaking), Alloy allows the underlying SAT-solver to dictate a random order of model presentation. The random nature of the presentation allows users to navigate among models with no semantics associated with the order of navigation. A trend in Alloy's random model presentation is to present smaller models first. As Jackson [70, page 7] explains:

> The tool's selection of instances is arbitrary, and depending on the preferences you've set, may even change from run to run. In practice, though, the first instance generated does tend to be a small one. This is useful, because the small instances are often pathological, and thus more likely to expose subtle problems.

Given the benefits associated with presenting the smaller models first, it is natural to explore the possibility of forcing a systematic order of presentation.

Aluminum [32] is our first attempt to explore the models of a theory with the intention of forcing a systematic order on models. Aluminum is a variation of Alloy and its model-finding engine, Kodkod, which utilizes the underlying SAT-solver to recursively reduce an initial model returned by the SAT-solver to a "minimal" model. Aluminum considers a model to be minimal if it is minimal with respect to the containment relation on models. Consequently, Aluminum allows the user to "augment" the minimal models with additional facts in order to "explore" the space of all models of the theory.

In contrast to the containment relation on models used by Aluminum, Razor utilizes the ordering relation determined by homomorphism [132]. Like Aluminum, Razor relies on augmentation for exploring the space of all models. In target-oriented model-finding [130], partial models are extended to contain information about target models to be approximated by the model-finder. In the context of scenario exploration target-oriented model-finding can be employed to investigate the consequences of modifications to a specification.

9.3 Geometric Logic

Geometric logic as a logic of observable properties was developed by Abramsky [37] and has been explored as a notion of specification by several authors [34, 133]. Geometric logic for theorem-proving was introduced in [134]. In [73] deNivelle generalized the setting of [134] to incorporate equality, and introduced a technique to augment the underlying theory as the system "learns" lemmas.

The Chase. The Chase is a well-known algorithm in the database community [16–19]. In the context of a *data-exchange* problem, where *tuple-generating* and *equality-generating* dependencies are essentially restricted forms of geometric sequents, the Chase has been applied to extend an instance over a given source schema to an instance over a target schema [17, 43].

The earlier versions of the Chase worked with disjunction-free dependencies, yielding exactly one (universal) model by a successful run. Deutch and his colleagues developed a version of the Chase that allowed disjunctions [42], resulting in a set-of-support for the input theory by a successful run. However, challenges arise in managing the complexity that arises due to disjunctions and equations. The algorithm implemented into Razor has been primarily aimed to address these challenges.

Theorem Proving. Fisher and Bezem [135] develop *Geolog*, a language for theorem-proving in geometric logic. A Geolog program consists of a set of axioms, goal conditions, and failure conditions, expressed as geometric sequents. The authors also define an abstract *Skolem* machine that utilizes a Chase-like algorithm to decide whether any of the goal or failure conditions follow from the given set of axioms.

Bezem and Coquand [134] implement a proof theory in *coherent* logic —a fragment of geometric logic that prohibits infinitary disjunctions— using Prolog [136]. The suggested proof theory exploits the observational properties of coherent logic for two primary advantages:

- Positivity of coherent logic makes it possible to constructively build a proof for an axiom, whereas resolution-based theorem-proving, which transforms the axiom into equisatisfiable ones.
- Theorem-proving in coherent logic is efficient because one never needs to introduce extra witnesses for an existential quantifier, if a witness already exists.

Formal Validation and Verification. The observational properties of geometric logic suggests geometric logic as a viable specification language [34]. In fact, logical axioms in a variety of applications including policy analysis [137], protocol analysis [44, 133, 138], and description logic [45] tend to be in geometric form.

Geometric logic has been utilized to study the connections between the properties of components and the properties of systems that are made of those components [139, 140]. The observational properties of geometric logic play a central role in preserving properties as they propagate to the system level.

Chapter 10

Conclusion and Future Work

This thesis presents a framework for exploring finite models of first-order theories. The framework suggests a novel approach for systematic exploration of the space of models as well as construction of provenance information for individual models. We suggest navigating between models along the paths induced by the homomorphisms on models. The homomorphism preorder on models measures their information content; that is, a model that is minimal under this preorder contains a minimum amount of information that is essential for satisfying the theory. Such minimal models are desirable as they do not contain distracting information. Moreover, every piece of information in a homomorphically minimal model can be justified by the axioms in the user's theory.

We implemented our framework in Razor, a model-finding assistant that provides the user with an interactive REPL for exploring the models of a theory $\mathcal{G}$ in geometric form. Razor initially returns a set-of-support for $\mathcal{G}$, a stream of all (finite) minimal models in the homomorphism ordering. Starting with the initial minimal models, the user can explore the space of all (finite) models of $\mathcal{G}$ via augmentation. Razor's model-finding algorithm is inspired by the Chase, an algorithm developed in the database community, which computes a set-of-support for theories in geometric form. Razor utilizes a variation of the Chase to compute a set of possible facts for $\mathcal{G}$, which can be thought of a refined version of the Herbrand base for $\mathcal{G}$. Consequently, Razor utilizes an SMT-solver to construct models of $\mathcal{G}$ that are homomorphically minimal. Razor employs a minimization algorithm, inspired by that of Aluminum [32], to reduce an arbitrary model of $\mathcal{G}$ to a homomorphically minimal one.

Unlike a conventional model-finder, Razor computes provenance information for the elements and facts of the models it returns. The provenance computation is done by constructing terms that witness the elements of models in an extended witnessing language. Razor can also compute provenance information for the elements and the facts of augmented models with regards to the user's original theory and the (optional) augmenting facts. Our model-finding algorithm is refutationally complete, thus, is not inherently bounded. However, Razor uses a subtle notion of

bound on the depth of witnessing terms to make the search tractable.

We exercised Razor on sample specifications of access control policies, Flowlog programs, and software design artifacts. Developing a comprehensive set of case-studies and sample specifications to exercise Razor in other application areas is an ongoing research.

Performance Enhancement. An obvious area of future work is to improve Razor's performance on large theories. Razor's performance is currently not competitive with the state of the art model-finders on large theories from TPTP [141] and the Alloy distribution [15]. Our main focus, however, even in the long term, is on theories developed by hand, which are human readable, have a limited number of predicates, and rarely include high arity relations. A natural source of such theories is the Alloy repository. A long-term research question is exploring the tradeoffs between efficiency and the kind of enhanced expressivity we offer.

The current algorithm based on SMT-solving has significantly improved the efficiency of the tool, compared to our preliminary implementations of the Chase. Still, major engineering improvements in the following areas will remain as a future work:

1. Exploiting the work in the SMT community, specifically the support for universal quantifiers [142].

2. Utilizing an in-memory database (*e.g.*, Berkeley DB [143]) to efficiently evaluate sequents in the set of possible facts, during a run of the grounding procedure. The use of a traditional database system makes a wide range of techniques from the database literature available to us. Specifically, materializing the views induced by the formulas in the head of sequents, leveraging efficient view update algorithms, and utilizing the existing techniques for identifying relevant updates [56] will improve the performance of Razor's grounding algorithm.

3. Leveraging the techniques developed in the model-finding community—*e.g.*, the ones mentioned in [69, 144]—to reduce the size of the ground instances that are passed to the SMT-solver.

Support for Other Languages. An ongoing project focuses on the support for specifications written in other languages, which are not necessarily in geometric form. This includes development of parsers to allow input in native formats such as Description Logic, firewall specifications, XACML, languages for software-defined networking, and cryptographic protocols. As discussed earlier, converting free-form first-order theories to geometric form is a straight-forward process in presence of Skolemization. However, it is often preferable to avoid Skolemization in order to maintain the language of the user's specification. Nonetheless, the primary challenge of translating specifications from other languages to Razor's core syntax will be the

presentation of provenance information, computed in the extended signature with Skolem functions.

Feature Enhancement. The current interface to interact with Razor is rather primitive. Future work includes the development of a more sophisticated GUI for presenting the space of all models as well as individual models, with regards to the goals of our exploration framework. Another extension is to enhance Razor's augmentation feature. The current implementation of Razor only allows for augmenting models with atomic facts, but in theory, augmentation can work with arbitrary positive-existential formulas.

Bibliography

[1] S. Feferman, J. W. Dawson, Jr., S. C. Kleene, G. H. Moore, R. M. Solovay, and J. van Heijenoort, Eds., *Kurt Godel: Collected Works. Vol. 1: Publications 1929-1936*. Oxford University Press, Inc., 1986.

[2] W. Mccune, "A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems," Argonne National Laboratory, Tech. Rep., 1994.

[3] L. Kovács and A. Voronkov, "First-Order Theorem Proving and Vampire," in *International Conference on Computer Aided Verification*, 2013.

[4] P. Baumgartner, A. Fuchs, and C. Tinelli, "Darwin: A Theorem Prover for the Model Evolution Calculus," in *IJCAR Workshop on Empirically Successful First Order Reasoning*, 2004.

[5] T. Tammet, "Gandalf," *Journal of Automated Reasoning*, 1997.

[6] D. W. Loveland, "Mechanical Theorem-Proving by Model Elimination," *Journal of the ACM*, 1968.

[7] B. Selman, H. J. Levesque, and D. G. Mitchell, "A New Method for Solving Hard Satisfiability Problems," in *National Conference on Artificial Intelligence*, 1992.

[8] J. D. Phillips and D. Stanovský, "Automated Theorem Proving in Quasigroup and Loop Theory," *AI Communications*, 2010.

[9] E. Zawadzki, G. J. Gordon, and A. Platzer, "An Instantiation-Based Theorem Prover for First-Order Programming," in *AI Statistics*, 2011.

[10] H. Chu and D. A. Plaisted, "Model Finding In Semantically Guided Instance-Based Theorem Proving," *Fundamenta Informaticae*, 1994.

[11] J. C. Blanchette, "Nitpick: A Counterexample Generator for Isabelle/HOL Based on the Relational Model Finder Kodkod," in *Logic for Programming, Artificial Intelligence, and Reasoning*, 2010, pp. 20–25.

[12] D. Jackson, "Lightweight Formal Methods," in *International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity*, 2001.

[13] T. Nelson, C. Barratt, D. J.Dougherty, K. Fisler, and S. Krishnamurthi, "The Margrave Tool for Firewall Analysis," in *USENIX Large Installation System Administration Conference*, 2010.

[14] S. F. Doghmi, J. D. Guttman, and F. J. Thayer, "Searching for Shapes in Cryptographic Protocols," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2007.

[15] "Alloy Analyzer." [Online]. Available: http://alloy.mit.edu/alloy/

[16] D. Maier, A. O. Mendelzon, and Y. Sagiv, "Testing Implications of Data Dependencies," *ACM Transactions on Database Systems*, 1979.

[17] C. Beeri and M. Y. Vardi, "A Proof Procedure for Data Dependencies," *Journal of the ACM*, 1984.

[18] A. Deutsch and V. Tannen, "XML Queries and Constraints, Containment and Reformulation," *ACM Symposium on Theory Computer Science*, 2005.

[19] S. Abiteboul, R. Hull, and V. Vianu, Eds., *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[20] J. Zhang and H. Zhang, "SEM: a system for enumerating models," in *International Joint Conference on Artificial Intelligence*, 1995.

[21] J. Harrison, *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[22] L. de Moura, B. Dutertre, and N. Shankar, "A Tutorial on Satisfiability Modulo Theories," in *International Conference on Computer Aided Verification*, 2007.

[23] L. de Moura and N. Bjørner, "Satisfiability Modulo Theories: An Appetizer," in *Formal Methods: Foundations and Applications, Brazilian Symposium on Formal Methods*, 2009.

[24] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability: Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009.

[25] C. Barrett, A. Stump, and C. Tinelli, "The SMT-LIB Standard: Version 2.0," Department of Computer Science, The University of Iowa, Tech. Rep., 2010, available at `www.SMT-LIB.org`.

[26] "SMT-LIB." [Online]. Available: http://smt-lib.org

[27] "Boolector." [Online]. Available: http://fmv.jku.at/boolector/

[28] "UCLID." [Online]. Available: http://uclid.eecs.berkeley.edu

[29] "Yices." [Online]. Available: http://yices.csl.sri.com

[30] "Z3." [Online]. Available: http://z3.codeplex.com

[31] "Razor." [Online]. Available: http://salmans.github.io/Razor/

[32] T. Nelson, S. Saghafi, D. J. Dougherty, K. Fisler, and S. Krishnamurthi, "Aluminum: Principled Scenario Exploration Through Minimality," in *International Conference on Software Engineering*, 2013.

[33] G. Sutcliffe, "The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0," *Journal of Automated Reasoning*, 2009.

[34] S. Vickers, "Geometric Logic as a Specification Language," in *Theory and Formal Methods*, 1994.

[35] ——, "Geometric Theories and Databases," in *Applications of Categories in Computer Science*, 1992.

[36] ——, "Geometric Logic in Computer Science," in *Theory and Formal Methods*, 1993.

[37] S. Abramsky, "Domain Theory in Logical Form," *Annals of Pure and Applied Logic*, 1991.

[38] B. Rossman, "Existential Positive Types and Preservation under Homomorphisms," in *IEEE Logic in Computer Science*, 2005.

[39] R. Goldblatt, *Topoi: the Categorical Analysis of Logic*. North-Holland, 1984.

[40] S. Vickers, "Topical Categories of Domains," *Mathematical Structures in Computer Science*, 1999.

[41] M. Makkai and G. Reyes, *First-Order Categorical Logic: Model-Theoretical Methods in the Theory of Topoi and Related Categories*. Springer-Verlag, 1977.

[42] A. Deutsch, A. Nash, and J. B. Remmel, "The Chase Revisited," in *Symposium on Principles of Database System*, 2008.

[43] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa, "Data Exchange: Semantics and Query Answering," *ACM Symposium on Theory Computer Science*, 2005.

[44] D. J. Dougherty and J. D. Guttman, "Decidability for Lightweight Diffie-Hellman Protocols," in *IEEE Computer Security Foundations Symposium*, 2014.

[45] P. Baumgartner and F. M. Suchanek, "Automated Reasoning Support for First-Order Ontologies," *Principles and Practice of Semantic Web Reasoning*, 2006.

[46] K. Claessen and J. Hughes, "QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs," in *International Conference on Functional Programming*, 2000.

[47] L. Bachmair, A. Tiwari, and L. Vigneron, "Abstract Congruence Closure," *Journal of Automated Reasoning*, 2003.

[48] T. Tammet, "Finite Model Building: Improvements and Comparisons," in *International Conference on Automated Deduction*, 2003.

[49] P. Baumgartner, A. Fuchs, H. De Nivelle, and C. Tinelli, "Computing Finite Models by Reduction to Function-Free Clause Logic," *Journal of Applied Logic*, 2009.

[50] R. Manthey and F. Bry, "SATCHMO: A Theorem Prover Implemented in Prolog," in *International Conference on Automated Deduction*, 1988.

[51] P. Baumgartner and R. A. Schmidt, "Blocking and Other Enhancements for Bottom-Up Model Generation Methods," in *International Joint Conference on Automated Reasoning*, 2006.

[52] F. Bry and A. Yahya, "Minimal Model Generation with Positive Unit Hyper-Resolution Tableaux," in *Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, 1996.

[53] P. Hudak, J. Hughes, S. Peyton Jones, and P. Wadler, "A history of Haskell: Being Lazy with Class," in *Conference on History of Programming Languages*, 2007.

[54] "The Glasgow Haskell Compiler." [Online]. Available: https://www.haskell.org/ghc/

[55] "SMTLib2." [Online]. Available: https://github.com/hguenther/smtlib2

[56] J. A. Blakeley, P.-A. Larson, and F. W. Tompa, "Efficiently Updating Materialized Views," in *International Conference on Management of Data*, 1986.

[57] "The SBV Package." [Online]. Available: https://hackage.haskell.org/package/sbv

[58] T. Nelson, A. D. Ferguson, M. Scheer, and S. Krishnamurthi, "Tierless Programming and Reasoning for Software-Defined Networks," *USENIX Symposium on Networked Systems Design and Implementation*, 2014.

[59] "Kodkod." [Online]. Available: http://alloy.mit.edu/kodkod/

[60] D. L. Berre and A. Parrain, "The SAT4J library, release 2.2," *Journal on Satisfiability, Boolean Modeling and Computation*, 2010.

[61] D. Akhawe, A. Barth, P. Lam, J. Mitchell, and D. Song, "Towards a Formal Foundation of Web Security," in *IEEE Computer Security Foundations Symposium*, 2010.

[62] R. Wilcox, *Introduction to Robust Estimation and Hypothesis Testing.* Academic Press, 2012.

[63] F. Baader and T. Nipkow, *Term Rewriting and All That.* Cambridge University Press, 1998.

[64] R. E. Shostak, "Deciding Combinations of Theories," in *International Conference on Automated Deduction*, 1982.

[65] J. H. Gallier, P. Narendran, D. A. Plaisted, S. Raatz, and W. Snyder, "An Algorithm for Finding Canonical Sets of Ground Rewrite Rules in Polynomial Time," *Journal of the ACM*, 1993.

[66] W. Snyder, "A Fast Algorithm for Generating Reduced Ground Rewriting Systems from a Set of Ground Equations," *Journal of Symbolic Computation*, 1993.

[67] E. F. Codd, "Relational Completeness of Data Base Sublanguages," *Database Systems*, 1972.

[68] E. Torlak and D. Jackson, "Kodkod: A Relational Model Finder," *Tools and Algorithms for the Construction and Analysis of Systems*, 2007.

[69] K. Claessen and N. Sörensson, "New Techniques that improve MACE-Style Finite Model Finding," in *CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications*, 2003.

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

[71] "Margrave." [Online]. Available: http://www.margrave-tool.org

[72] "Nitpick." [Online]. Available: http://www4.in.tum.de/~blanchet/nitpick.html

[73] H. de Nivelle and J. Meng, "Geometric Resolution: A Proof Procedure Based on Finite Model Search," in *International Joint Conference on Automated Reasoning*, 2006.

[74] K. Korovin and C. Sticksel, "iProver-Eq: An Instantiation-Based Theorem Prover with Equality," in *International Joint Conference on Automated Reasoning*, 2010.

[75] E. M. Clarke and E. A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic," in *Logic of Programs Workshop*, 1982.

[76] J. Queille and J. Sifakis, "Specification and Verification of Concurrent Systems in CESAR," in *International Symposium on Programming*, 1982.

[77] M. Y. Vardi and P. Wolper, "Reasoning About Infinite Computations," *Information and Computation*, 1994.

[78] "BLAST." [Online]. Available: http://forge.ispras.ru/projects/blast/

[79] "NuSMV." [Online]. Available: http://nusmv.fbk.eu

[80] "Spin." [Online]. Available: http://spinroot.com/spin/whatispin.html

[81] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and Change-Impact Analysis of Access-Control Policies," in *International Conference on Software Engineering*, 2005.

[82] G. Soares, M. Mongiovi, and R. Gheyi, "Identifying Overly Strong Conditions in Refactoring Implementations," in *International Conference on Software Maintenance*, 2011.

[83] S. M. A. Shah, K. Anastasakis, and B. Bordbar, "From UML to Alloy and Back Again," in *International Workshop on Model-Driven Engineering, Verification and Validation*, 2009.

[84] P. N. Devyanin, A. V. Khoroshilov, V. V. Kuliamin, A. K. Petrenko, and I. V. Shchepetkov, "Formal Verification of OS Security Model with Alloy and Event-B," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2014.

[85] L. K. Dillon, R. E. K. Stirewalt, B. Sarna-starosta, and S. D. Fleming, "Developing an Alloy framework akin to OO frameworks," in *Alloy Workshop*, 2006.

[86] E. Zulkoski, C. Kleynhans, M. Yee, D. Rayside, and K. Czarnecki, "Optimizing Alloy for Multi-objective Software Product Line Configuration," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2014.

[87] A. Shaffer, M. Auguston, C. Irvine, and T. Levin, "A Security Domain Model for Implementing Trusted Subject Behaviors," 2008.

[88] K. Anastasakis, B. Bordbar, and J. M. Kster, "Analysis of Model Transformations via Alloy," 2007.

[89] F. Mostefaoui and J. Vachon, "Design-Level Detection of Interactions in Aspect- UML Models Using Alloy," *Journal of Object Technology*, 2007.

[90] S. Khurshid and D. Marinov, "TestEra: Specification-based Testing of Java Programs Using SAT," *Automated Software Engineering*, 2004.

[91] E. Uzuncaova, S. Khurshid, and D. Batory, "Incremental Test Generation for Software Product Lines," *IEEE Transactions on Software Engineering*, 2010.

[92] F. Rebello de Andrade, J. Faria, A. Lopes, and A. Paiva, "Specification-Driven Unit Test Generation for Java Generic Classes," in *Integrated Formal Methods*, 2012.

[93] I. Rychkova, G. Regev, and A. Wegmann, "Using Declarative Specifications in Business Process Design," *International Journal on Computational Science*, 2008.

[94] A. Wegmann, L. son Lê, and L. Hussami, "A Tool for Verified Design using Alloy for Specification and CrocoPat for Verification," 2006.

[95] C. Wallace, "Using Alloy in Process Modeling," 2003.

[96] F. A. Maldonado-Lopez, J. Chavarriaga, and Y. Donoso, "Detecting Network Policy Conflicts Using Alloy," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2014.

[97] S. Mirzaei, S. Bahargam, R. Skowyra, A. Kfoury, and A. Bestavros, "Using Alloy to Formally Model and Reason About an OpenFlow Network Switch," 2013.

[98] S. Narain, A. Poylisher, and R. Talapade, "Network Single Point of Failure Analysis via Model Finding," in *Alloy Workshop*, 2006.

[99] S. Pai, Y. Sharma, S. Kumar, R. M. Pai, and S. Singh, "Formal Verification of OAuth 2.0 Using Alloy Framework," in *International Conference on Communication Systems and Network Technologies*, 2011.

[100] T. Wang and D. Ji, "Active Attacking Multicast Key Management Protocol Using Alloy," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2012.

[101] C. L. Chen, P. S. Grisham, S. Khurshid, and D. E. Perry, "Design and Validation of a General Security Model with the Alloy Analyzer," in *Alloy Workshop*, 2006.

[102] B. Fraikin, M. Frappier, and R. St-Denis, "Modeling the Supervisory Control Theory with Alloy," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2012.

[103] A. Vakili and N. Day, "Temporal Logic Model Checking in Alloy," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2012.

[104] T. Giannakopoulos, D. J. Dougherty, K. Fisler, and S. Krishnamurthi, "Towards an Operational Semantics for Alloy," in *World Congress on Formal Methods*, 2009.

[105] R. Gheyi, T. Massoni, and P. Borba, "A Theory for Feature Models in Alloy," in *Alloy Workshop*, 2006.

[106] M. Frias, J. Galeotti, C. Pombo, and N. Aguirre, "DynAlloy: Upgrading Alloy with Actions," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2005.

[107] E. Torlak, M. Taghdiri, G. Dennis, and J. P. Near, "Applications and Extensions of Alloy: Past, Present and Future," *Mathematical Structures in Computer Science*, 2013.

[108] I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri, "Debugging Overconstrained Declarative Models Using Unsatisfiable Cores," in *International Conference on Automated Software Engineering*, 2003.

[109] J. D. Ramsdell and J. D. Guttman, "CPSA: A Cryptographic Protocol Shapes Analyzer," MITRE Corporation, 2009. [Online]. Available: http://hackage.haskell.org/package/cpsa

[110] "CPSA." [Online]. Available: https://hackage.haskell.org/package/cpsa

[111] D. Rayside, F. S. Chang, G. Dennis, R. Seater, and D. Jackson, "Automatic Visualization of Relational Logic Models," *Electronic Communications of the EASST*, 2007.

[112] A. Zaman, I. Kazerani, M. Patki, B. Guntoori, and D. Rayside, "Improved Visualization of Relational Logic Models," University of Waterloo, Tech. Rep., 2013.

[113] L. Gammaitoni and P. Kelsen, "Domain-Specific Visualization of Alloy Instances," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2014.

[114] P. Ponzio, N. Rosner, N. Aguirre, and M. F. Frias, "Efficient Tight Field Bounds Computation Based on Shape Predicates," in *International Symposium on Formal Methods*, 2014.

[115] J. P. Galeotti, N. Rosner, C. G. L. Pombo, and M. F. Frias, "Distributed SAT-Based Computation of Relational Tight Bounds," 2010.

[116] Y. Feng, "Disjunction of Regular Timing Diagrams," Master's thesis, Worcester Polytechnic Institute, 2010.

[117] L. Momtahan, "Towards a Small Model Theorem for Data Independent Systems in Alloy," *Electric Notes on Theory of Computer Science.*, 2005.

[118] A. Abadi, A. Rabinovich, and M. Sagiv, "Decidable Fragments of Many-sorted Logic," in *Logic for Programming, Artificial Intelligence, and Reasoning*, 2007.

[119] T. Nelson, D. Dougherty, K. Fisler, and S. Krishnamurthi, "Toward a More Complete Alloy," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2012.

[120] E. Torlak, F. S.-H. Chang, and D. Jackson, "Finding Minimal Unsatisfiable Cores of Declarative Specifications," in *International Symposium on Formal Methods*, 2008.

[121] N. D'Ippolito, M. F. Frias, J. P. Galeotti, E. Lanzarotti, and S. Mera, "Alloy+HotCore: A Fast Approximation to Unsat Core," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2010.

[122] V. Schuppan, "Towards a Notion of Unsatisfiable Cores for LTL," in *Fundamentals of Software Engineering*, 2010.

[123] J. Wittocx, H. Vlaeminck, and M. Denecker, "Debugging for Model Expansion," in *International Conference on Logic Programming*, 2009.

[124] P. Bendersky, J. P. Galeotti, and D. Garbervetsky, "The DynAlloy Visualizer," in *Latin American Workshop on Formal Methods*, 2014, pp. 59–64.

[125] H. R. Andersen, "Partial Model Checking (Extended Abstract)," in *IEEE Logic in Computer Science*, 1995.

[126] M. Huth and S. Pradhan, "Consistent Partial Model Checking," *Electronic Notes on Theoretical Computer Science*, 2004.

[127] D. Saccà and C. Zaniolo, "Partial Models and Three-Valued Models in Logic Programs with Negation," in *International Conference on Logic Programming and Nonmonotonic Reasoning*, 1991.

[128] M. Famelis, S. Ben-David, M. Chechik, and R. Salay, "Partial Models: A Position Paper," in *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation*, 2011.

[129] V. Montaghami and D. Rayside, "Extending Alloy with Partial Instances," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2012.

[130] A. Cunha, N. Macedo, and T. Guimarães, "Target Oriented Relational Model Finding," in *Fundamental Approaches to Software Engineering*, 2014.

[131] V. Montaghami and D. Rayside, "Staged Evaluation of Partial Instances in a Relational Model Finder," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2014.

[132] S. Saghafi and D. J. Dougherty, "Razor: Provenance and Exploration in Model-Finding," in *Workshop on Practical Aspects of Automated Reasoning*, 2014.

[133] J. D. Guttman, "Security Theorems via Model Theory," *EXPRESS: Expressiveness in Concurrency*, 2009.

[134] M. Bezem and T. Coquand, "Automating Coherent Logic," in *Logic for Programming, Artificial Intelligence, and Reasoning*, 2005, pp. 246–260.

[135] J. Fisher and M. Bezem, "Skolem Machines," *Fundamenta Informaticae*, 2009.

[136] J. Wielemaker, T. Scherijvers, M. Triska, and T. Lager, "SWI-Prolog," *Theory and Practice of Logic Programming*, 2012.

[137] S. Saghafi, T. Nelson, and D. J. Dougherty, "Geometric Logic for Policy Analysis," in *Workshop on Automated Reasoning in Security and Software Verification*, 2013.

[138] J. D. Guttman, "Establishing and Preserving Protocol Security Goals," *Journal of Computer Security*, 2014.

[139] V. Sofronie-Stokkermans and K. Stokkermans, "Modeling Interactions by Sheaves and Geometric Logic," in *International Symposium on Fundamentals of Computation Theory*, 1999.

[140] V. Sofronie-Stokkermans, "Sheaves and Geometric Logic and Applications to Modular Verification of Complex Systems," *Electronic Notes on Theoretical Computer Science*, 2009.

[141] "TPTP." [Online]. Available: http://www.cs.miami.edu/~tptp

[142] A. Reynolds, C. Tinelli, A. Goel, and S. Krstic, "Finite Model Finding in SMT," in *International Conference on Computer Aided Verification*, 2013.

[143] M. A. Olson, K. Bostic, and M. Seltzer, "Berkeley DB," in *USENIX Annual Technical Conference*, 1999.

[144] S. Schulz, "A Comparison of Different Techniques for Grounding Near-Propositional CNF Formulae," in *The Florida Artificial Intelligence Society Conference*, 2002.