Synthesizing Axiomatizations using Logic Learning

PAUL KROGMEIER*, University of Illinois, Urbana-Champaign, USA
ZHENGYAO LIN*, University of Illinois, Urbana-Champaign, USA
ADITHYA MURALI*, University of Illinois, Urbana-Champaign, USA
P. MADHUSUDAN, University of Illinois, Urbana-Champaign, USA

Axioms and inference rules form the foundation of deductive systems and are crucial in the study of reasoning with logics over structures. Historically, axiomatizations have been discovered manually with much expertise and effort. In this paper we show the feasibility of using synthesis techniques to discover axiomatizations for different classes of structures, and in some contexts, automatically prove their completeness. For evaluation, we apply our technique to find axioms for (1) classes of frames in modal logic characterized in first-order logic and (2) the class of language models with regular operations.

CCS Concepts: • **Theory of computation** → **Logic**; *Modal and temporal logics*; *Equational logic and rewriting*; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: Learning Logics, Axiomatization, Inductive Synthesis

ACM Reference Format:
Paul Krogmeier, Zhengyao Lin, Adithya Murali, and P. Madhusudan. 2022. Synthesizing Axiomatizations using Logic Learning. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 185 (October 2022), 29 pages. https://doi.org/10.1145/3563348

1 INTRODUCTION

Several applications in programming languages, formal verification, and associated fields benefit from deductive reasoning in logics. Depending on the application domain, we need to reason with particular logics over particular classes of structures[1]. The logics are often specialized, and the classes of structures are those relevant to the problem domain, which may be known intuitively or defined precisely. Examples abound including modal logics to reason about transition systems (or Kripke structures) [Blackburn et al. 2007], temporal logics to reason about sequential or branching behavior of systems [Clarke and Emerson 1982; Pnueli 1977], logics to reason with algebraic data types in functional programming languages [Hodges 1997], logics to reason with Kleene algebras that can model packet movement in networks [Anderson et al. 2014], and separation logic to reason with pointer-based data structures in imperative programs [O'Hearn et al. 2001].

The foundations of deductive reasoning for a logic $\mathcal{L}$ over a particular class of structures C lies in the axiomatic method, which utilizes general deductive mechanisms for deriving logical

*Equal contribution
[1]Also known as *models*. We use *structure* and *model* interchangeably in this paper.

Authors' addresses: Paul Krogmeier, Department of Computer Science, University of Illinois, Urbana-Champaign, USA, paulmk2@illinois.edu; Zhengyao Lin, Department of Computer Science, University of Illinois, Urbana-Champaign, USA, zl38@illinois.edu; Adithya Murali, Department of Computer Science, University of Illinois, Urbana-Champaign, USA, adithya5@illinois.edu; P. Madhusudan, Department of Computer Science, University of Illinois, Urbana-Champaign, USA, madhu@illinois.edu.

2475-1421/2022/10-ART185
https://doi.org/10.1145/3563348

truths in $\mathcal{L}$ applied to a set of axioms A_C that captures fundamental properties of C. Whether reasoning over one particular intended structure (e.g., arithmetic or an algebraic data type) or an intended class of structures (e.g., lists or other data structures in a heap, mathematical groups, or Kleene algebras), the axiomatic method involves finding a basic set of properties that logically characterize the structures. Other properties of the structures are then *logically entailed* by the basic set of axioms, and entailment can be mechanized using a variety of reasoning methods for the logic, including proof systems and algorithmic procedures.

In special circumstances, *all* properties common to structures in C are logically entailed by a finite or recursive set of axioms, in which case the axiom system is said to be *complete*. A complete set of axioms coupled with a powerful enough deductive system then yields a complete proof system for the class of structures C.

In recent years there has been tremendous progress in two fields that are relevant to this paper—(1) automated validity checking for different logics and (2) program synthesis. Significant strides have been made in identifying (semi-)decidable logical theories and in building automatic and efficient procedures for validity[De Moura and Bjørner 2008; Kovács and Voronkov 2013]. There has also been significant progress in program and expression synthesis, where the goal is to synthesize a program or logical expression that satisfies a given set of constraints (e.g., see [Alur et al. 2013]).

Formulating axiomatizations is a difficult task typically done by humans, and most typically by researchers with considerable prowess in logic. Armed with current automated reasoning and program synthesis techniques, we ask in this paper the following (perhaps audacious) question:

Can computers help us find axiomatizations?

Intuitively, both logical reasoning and expression synthesis are useful for finding axiomatizations—axioms are logical expressions that we want to synthesize, and reasoning is needed to prove that an axiom is valid over a class of structures, as well as for other tasks, e.g., checking if an axiom is implied by another set of axioms.

In this paper, we formulate the problem of axiomatizing a class of structures in a logic. This is a *model-theoretic* formulation of axiomatization that is independent of proof systems. Given a logic $\mathcal{L}$ and a subclass C of structures within a larger class $\mathcal{S}$, an axiomatization we aim to find is a finite set A of sentences in $\mathcal{L}$ that (a) hold on all structures in C, (b) are *nontrivial* in that they do not hold over all structures in $\mathcal{S}$, and (c) are mutually independent in terms of semantic entailment with respect to $\mathcal{S}$. Such an axiomatization A is said to be *complete* if it semantically entails all sentences expressible in $\mathcal{L}$ that hold over C. That is, for any sentence $\varphi \in \mathcal{L}$ that is true for every structure in C, we have that every structure in $\mathcal{S}$ satisfying the axioms A also satisfies φ, i.e. $A \models \varphi$.

1.1 Learning-based Axiom Synthesis (LAS) Framework

The fundamental contribution of this paper is a framework and core algorithm for solving the axiom synthesis problem, consisting of logical reasoning and expression synthesis components. We propose the *Learning-based Axiom Synthesis (LAS)* framework to facilitate synthesis of sound/complete axiomatizations. For a particular logic $\mathcal{L}$ and a subclass C of a class of structures $\mathcal{S}$, this framework calls for implementing and combining the following components:

- VC: a procedure that checks the validity of formulae in $\mathcal{L}$ over the class C
- Cex: a procedure that generates a *counterexample* to rule out a formula that is *invalid* over C
- VS: a procedure that checks entailment of formulae in $\mathcal{L}$ over the class $\mathcal{S}$
- Learner: a procedure that proposes axioms in $\mathcal{L}$ using counterexamples reported by Cex

Given C and the logic $\mathcal{L}$, whose semantics over $\mathcal{S}$ is known, the LAS framework can be instantiated to axiomatize C by implementing the procedures VC, Cex, VS, and Learner. The framework specifies a core algorithm that combines these procedures to find a sound axiomatization for C.

Note that, because individual structures may have infinite domains and the class C may be infinite as well, it does not make sense to think of C as an input. Instead, the framework requires reasoning and counterexample generation procedures to be implemented for C and $\mathcal{S}$. The notion of counterexamples is rather nuanced. If structures in C are finite (even though the set C is not), we may expect Cex to directly provide structures as counterexamples for the soundness of proposed axioms. When structures are infinite, however, Cex may generate what we call *pseudo-models*, which are finite objects that rule out unsound formulae and indicate the existence of a counterexample structure (which may be infinite). The notion of pseudo-models depends on the specific setting, and we elaborate on this later.

Each procedure required by the framework serves a specific purpose for axiom synthesis. First, when Learner proposes a candidate axiom in $\mathcal{L}$, we can determine whether it is *sound*, i.e. valid over C, by using the procedure VC. Second, having synthesized a set of sound axioms A, we can check for redundancies, i.e. whether any axioms in A are logically entailed by the others, by using the procedure VS. Third, Learner is a generic expression synthesis procedure that is aware of the universal class of structures $\mathcal{S}$ and the semantics of the logic $\mathcal{L}$ over $\mathcal{S}$, but it is agnostic to the class C. To efficiently search for axioms, it hence needs counterexamples from C. Finally, each time Learner proposes a candidate that is unsound, the procedure Cex generates a pseudo-model that witnesses that the axiom is false, and Learner uses it to prune the search space for new axioms.

Since we would like to deal with expressive logics and classes of structures, we cannot avoid that the procedures may be incomplete and non-terminating in various ways. In particular, in many cases the validity procedures VC and VS will be non-terminating, though of course they must be sound: if they terminate with an answer then that answer is correct.

The LAS framework can be seen as a kind of counterexample-guided inductive synthesis (CEGIS) framework for synthesizing axiom systems [Solar Lezama 2008]. In typical synthesis problems the specification for synthesized expressions is formalized as a logical constraint. However, in axiom synthesis, axioms are a sound and independent set of statements for a class C, and it is not possible to capture this requirement as a logical constraint (e.g., as in SyGuS [Alur et al. 2013]). However, by implementing the *teacher* using the components VC, Cex, and VS, LAS facilitates a CEGIS algorithm using a learner that inductively learns expressions from counterexamples. The LAS framework and the notion of pseudo-models to learn classes with infinite models are contributions of this paper.

1.2 Complete Axiomatizations

The framework suggested above does not address the problem of finding *complete* axiomatizations. First, we need to define precisely what we mean by completeness, and there are multiple natural notions here. We could say that a set of sound axioms A is complete if the subclass of $\mathcal{S}$ that satisfies A is precisely C. This is equivalent to requiring that every structure in $\mathcal{S}$ that is not in C violates at least one axiom in A.

However, there is another natural and common notion: A is complete if every property φ (expressible in $\mathcal{L}$) that holds over C is semantically entailed by A. Note the difference: this does not demand that C is captured precisely, but rather that it is captured *up to properties expressible in* $\mathcal{L}$. In other words, if we take all structures in $\mathcal{S}$ that satisfy the axioms A, we should get the class $C' \supseteq C$ of all structures that satisfy every property that holds over C. If $C' = C$ then the above two notions coincide. (For readers familiar with the notion of *elementary class* [Hodges 1997]: this is the same notion except that instead of first-order properties it involves properties expressible in $\mathcal{L}$.)

The LAS framework involves an optional completeness checker CC. Given axioms A, the procedure CC checks whether there is a structure M that satisfies A but does not satisfy all properties common to C. Completeness checks are extremely hard to automate, in part because C' may be a complex class that is hard to understand. However, when $C' = C$, then CC need only check whether there is a structure in $\mathcal{S} \setminus C$ that satisfies the axioms, which is feasible in some cases.

1.3 Instantiations of LAS

The second contribution of this paper is an instantiation of LAS in two different settings:

- **Modal Logics** [Blackburn et al. 2006; Van Benthem 1984]. The subfield of modal logic called correspondence theory characterizes classes of Kripke structures (transition systems with propositional valuations for each state) using axioms in modal logic. We instantiate LAS in this setting to synthesize axioms for 17 different classes of structures from the literature.
- **Languages with Kleene star** [Conway 1971; Kozen 1994; Salomaa 1966]. We instantiate LAS to synthesize equational axioms for a class of structures consisting of arbitrary word languages over finite alphabets with the operations of concatenation, union, and Kleene star.

While the framework remains the same, both of the settings above have nuances. Modal logic axiomatizations for classes of Kripke structures is a natural example for our framework, especially since there are a large number of classes to study systematically. It presents unique challenges, however— axioms in modal logic have semantics that involves universal quantification over the propositional valuations, which requires us to handle *second-order* reasoning. Completeness checking is also highly nontrivial, but we are able to synthesize complete axiomatizations for all 17 classes (and with 14 of them automatically proven complete). Axiomatizing word languages with operations poses a different set of challenges— though our instantiation of LAS manages to effectively discover and reason with equational axioms, it is known that any complete finite axiomatization must go beyond equations [Redko 1964], e.g., to conditional equations, which substantially increases the complexity of reasoning procedures. In fact, the equational axioms that our tool finds are stronger than the purely equational axioms in standard axiomatizations for Kleene algebras.

Despite the nuances and complexity described above, the tool we develop for these instantiations effectively discovers sound (and in some cases complete) axiomatizations in reasonable time, and furthermore, the axioms resemble those found by human researchers, after accounting for simple, semantically-equivalent rearrangements. We believe that the relative success of our framework in two different settings shows its promise for automating axiomatizations. As logical reasoning and expression synthesis technologies improve, the framework, being parameterized over these, will also become more effective.

In summary, the contributions of this paper are: (a) a model-theoretic formulation of the axiom synthesis problem, (b) the Learning-based Axiom Synthesis framework that facilitates a CEGIS-style algorithm for automating axiom synthesis, (c) the notion of pseudo-models which can be used as counterexamples for axiom synthesis, and (d) instantiations of the LAS framework in two domains that argue its efficacy, modal logic (17 classes of structures) and equational axioms for Kleene algebras.

The paper is structured as follows. In Section 2, we show how LAS works for an illustrative example, namely, axiomatizing *equivalence relations* from structures that encode partitions. Section 3 presents our model-theoretic formulation of the axiom synthesis problem, with discussions on sound and complete axiomatizations. We present the LAS framework in Section 4 and define the components needed for reasoning, counterexample generation, and synthesis. In Sections 5 and 6, we present instantiations of the LAS framework for the settings of modal logic (correspondence theory) and languages with regular operations. The nuances in each setting, realizations of components,

implementation details, evaluation, and the axiomatizations found by our tool are reported in the corresponding sections. Section 7 presents related work and we conclude with future directions in Section 8.

2 EXAMPLE: AXIOMATIZING EQUIVALENCE RELATIONS

We begin by illustrating LAS in a very simple setting. We want to synthesize the axioms of the relations that characterize equi-membership in partitions of sets (which we familiarly call equivalence relations, with axioms for reflexivity, symmetry, and transitivity). This example covers many aspects of our framework (except completeness). It illustrates how the axiomatization problem is defined, how to build the components for reasoning, and also the axiomatization produced by our tool.

The *equi-membership* relation for a partition of a set is a binary relation that relates two elements precisely when they occupy the same partition cell. Let us model a partition P of a set X by a function $f : X \to X$, where for each element $c \in f(X) \subseteq X$ we have one cell $P_c := \{x \in X : f(x) = c\}$. In other words, two elements belong to the same set in a partition if and only if f maps them to the same elements. Intuitively, our goal is to find a (preferably small) set of axioms expressed as first-order logic sentences that captures the theory of relations R defined using such partitions. More precisely, we want an axiomatization that uses sentences referring only to R (and not to "f" or " = ") which are true in *every* structure that satisfies:

$$\psi := \forall x.\forall y.\, (R(x, y) \leftrightarrow f(x) = f(y)). \tag{1}$$

The target class C hence consists of structures that satisfy (1), each being a set of elements together with interpretations for f, =, and R, with = possessing the usual meaning. Each structure in C therefore interprets R as the equi-membership relation defined by the partition that puts two elements x and y in the same cell if $f(x) = f(y)$.

Suppose we have an axiom synthesizer that proposes candidate axioms. Leaving aside how to obtain the synthesizer, we must at least be able to determine whether a candidate axiom is truly an axiom, i.e., whether it is true in each structure in C. In this example, the target class is characterized in first-order logic by (1). Checking that a candidate formula φ is in fact a true axiom requires checking that every structure $M \in C$ makes φ true, which corresponds to the validity of

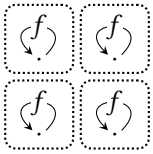
$$\psi \to \varphi. \tag{2}$$

Validity in the class C, which is equivalent to the first-order validity of (2) for this example, corresponds to what we call *soundness* and is checked in our framework by a component we call VC. We refer to formulae that are true for all structures in C as *sound axioms*. As an example, suppose the synthesizer proposes the candidate

$$\varphi := \forall x.\forall y.\, R(x, y).$$

This is not a sound axiom, because any partition with at least two cells satisfies (1) but not φ, and therefore does not satisfy (2). In this setting, we can implement VC using any semi-decision procedure for first-order logic validity. Note, however, that we cannot hope to mechanically prove that a candidate axiom is *not* sound. We must be content with admitting axioms that we can prove sound and discarding those that we cannot prove.

Can we do better than merely filtering candidate axioms for soundness? A key idea in synthesis is the use of counterexamples to guide search. If we can find *counterexamples that witness unsoundness*, then we can use them to rule out many other unsound candidates. For any unsound φ like the one above, there must be a structure $M \in C$ such that $M \not\models \varphi$ (though it may be difficult or impossible to automatically find one, and it may not be finitely representable). If we can indeed effectively find

$M :=$ 

Rules out: $\forall xy.\, R(x,y)$,
$\forall xyz.\, (R(x,y) \vee R(y,z) \vee R(x,z))$,
$\exists x.\forall y.\, R(x,y)$, ...

Fig. 1. A counterexample structure M that encodes a partition with 4 singleton cells. It witnesses unsoundness of the candidate axiom $\forall x.\forall y.\, R(x,y)$ and many others.

counterexamples, then we can maintain a set of counterexample structures in a counterexample-guided synthesis loop. In each iteration, we can query the synthesizer for candidate axioms that are true in all counterexample structures found up to that point. Candidate axioms are then subjected to a soundness check, and if proven, added to a growing set of axioms. Continuing with our example above, suppose we begin with an empty set of counterexample structures and the first candidate axiom is $\varphi := \forall x.\forall y.\, R(x,y)$. After failing to prove it sound, we may generate a counterexample structure M with, say, four solitary elements in their own partition cells, as shown in Figure 1. In the following iterations, all candidate axioms proposed by the synthesizer are required to be true in M. Notice that this is a favorable counterexample because it rules out several unsound candidates beyond φ. For instance, the unsound axiom

$$\forall x.\forall y.\forall z.\, (R(x,y) \vee R(y,z) \vee R(x,z))$$

is eliminated by M, but would not have been eliminated by a partition consisting of two elements in their own cells.

Building a counterexample synthesizer, which we call Cex, is a hard problem. In general, there is no procedure that delivers a counterexample even when one exists. However, since counterexamples guide search but are not crucial for progress, we resort to heuristics to find them. For example, we can query a constraint solver to find structures of small, bounded size (or finitary witnesses for infinite ones, which we call *pseudo-models*) that witness the unsoundness of candidate axioms.

Beyond soundness and efficiency, we also want axioms to be *independent* of each other. In other words, no single axiom should be logically implied by the other axioms. For example, suppose we have discovered the axiom for transitivity,

$$\forall x.\forall y.\forall z.(R(x,y) \wedge R(y,z) \rightarrow R(x,z)),$$

and the synthesizer proposes a similar axiom

$$\forall w.\forall x.\forall y.\forall z.(R(w,x) \wedge R(x,y) \wedge R(y,z) \rightarrow R(w,z)).$$

Both axioms are sound, but we would like to discard the latter because it is logically implied by the former. The problem of checking independence, i.e., whether a set of axioms $\{\psi_1, \ldots, \psi_n\}$ implies a candidate axiom φ, can be solved in this setting by checking the validity of

$$\bigwedge_i \psi_i \rightarrow \varphi. \tag{3}$$

We can build an independence checker using VS, which is a validity procedure for arbitrary structures and signatures, and in this case, it can be realized using a semi-decision procedure for first-order validity. If we are able to prove (3), then we discard φ. Otherwise, we add it to the growing set of axioms.

Despite our (unavoidable) reliance on semi-decision procedures and heuristics, our tool is able to find axiomatizations effectively. Given the description of the target class (1), our tool finds axioms for reflexivity, symmetry, and transitivity (with slightly different forms than usual):

$\forall x.\, R(x,x)$
$\forall x.\forall y.\, (R(x,y) \leftrightarrow R(y,x))$
$\forall x.\forall y.\forall z.\, (R(x,y) \rightarrow (R(x,z) \rightarrow R(y,z)))$.

The reader may be wondering: at what point does the synthesis loop stop? When is an axiomatization *complete*, or good enough? What does *good enough* mean? These are interesting and tricky questions (as discussed in Section 1), and answers depend heavily on the specific problem, which makes automation very hard.

In the case of relations defined by a partition, proving completeness of a set of axioms A turns out to be subtle. Intuitively, we want to know whether for any structure that satisfies A there is a function f such that Equation (1) holds (a second-order quantification over functions). The difficulty with proving this, intuitively, is that one needs to select a representative from each equivalence class of R. For example, if $R(a,b)$ holds, then it should be the case that $f(a) = f(b)$, but there are many choices for this element. Proving the existence of a suitable function f is difficult to automate (in general it seems to require the axiom of choice), and we did not implement a completeness checker for this example.

3 THE AXIOM SYNTHESIS PROBLEM

In this section, we define the axiomatization problem. We introduce some preliminaries in Section 3.1 and describe the problem in Section 3.2.

3.1 Preliminaries

In our formulation, the problem of axiom synthesis is parameterized by an *abstract logic* $\mathcal{L}$, defined as follows:

Definition 3.1 (Abstract Logic)**.** An *abstract logic* $\mathcal{L}$ is a tuple $(\mathcal{F}, \mathcal{S}, \models)$ where

- $\mathcal{F}$ is a set of *formulae*.
- $\mathcal{S}$ is a class of *models*.
- $\models \subseteq \mathcal{S} \times \mathcal{F}$ is the binary *satisfaction relation*. □

Satisfaction and Entailment. We say M *satisfies* φ if $M \models \varphi$ holds. Equivalently, we say φ *holds in* M or M *is a model of* φ or φ *is true in* M. Let $M \in \mathcal{S}$ be a model, $C \subseteq \mathcal{S}$ be a subclass of models, $T \subseteq \mathcal{F}$ be a subset of formulae, and $\varphi \in \mathcal{F}$ be a formula. We say M satisfies T, written $M \models T$, to mean $M \models \varphi$ for every $\varphi \in T$. We lift this to a class of models C and write $C \models \varphi$ (or $C \models T$) if every model $M \in C$ satisfies φ (resp. satisfies T). We also use $\models$ to denote logical entailment. We say φ is entailed by T, written $T \models \varphi$, to mean that every model of T satisfies φ.

Theories and Models. A *theory* is a set of formulae $T \subseteq \mathcal{F}$ that is entailment closed, i.e., for every $\varphi \in \mathcal{F}$, if $T \models \varphi$, then $\varphi \in T$. The theory of a class of models C, denoted by $\mathrm{Th}(C)$, is the set $\{\varphi \in \mathcal{F} \mid C \models \varphi\}$ of formulae that hold in all models in C. The dual of this view is the class $\{M \in \mathcal{S} \mid M \models T\}$ consisting of all models that satisfy a set of formulae T, which we denote by $\mathrm{Mod}(T)$. Observe that the larger the theory, the smaller its class of models, i.e., if $T \subseteq T'$ then $\mathrm{Mod}(T') \subseteq \mathrm{Mod}(T)$. In fact, taking the two partially-ordered sets $(2^{\mathcal{F}}, \subseteq)$ and $(2^{\mathcal{S}}, \supseteq)$, the monotonic functions $\mathrm{Mod} : 2^{\mathcal{F}} \rightarrow 2^{\mathcal{S}}$ and $\mathrm{Th} : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{F}}$ form a monotone Galois connection [Lawvere 1969; Smith 2010], since for any set of formulae T and class C we have $T \subseteq \mathrm{Th}(C) \Leftrightarrow \mathrm{Mod}(T) \supseteq C$.

$\mathcal{L}$-Elementary Class. One consequence of the above Galois connection is that for any $C \subseteq \mathcal{S}$ we have $\mathrm{Mod}(\mathrm{Th}(C)) \supseteq C$. We distinguish a special case, namely, when $\mathrm{Mod}(\mathrm{Th}(C)) = C$. We refer to such a class C as $\mathcal{L}$-elementary. One can see from the above definitions that a class is $\mathcal{L}$-elementary

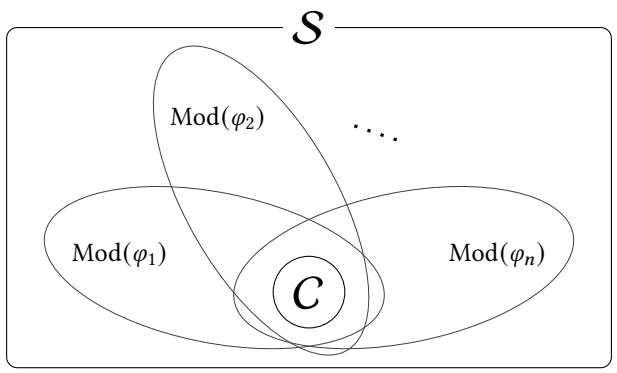


Fig. 2. Model-theoretic axiomatization.

if and only if it can be defined as $\text{Mod}(T)$ for some theory T. This is inspired by the concept of an elementary class from model theory [Hodges 1993], and it informs our definition of completeness in Section 3.2.

3.2 A Model-Theoretic Formulation of Axiom Synthesis

The model-theoretic axiomatization problem is parameterized by an abstract logic $\mathcal{L} = (\mathcal{F}, \mathcal{S}, \models)$. The objective is to axiomatize a *target class* of models $C \subseteq \mathcal{S}$ using formulae in $\mathcal{F}$. We use the word *axiom* for any formula in $\mathcal{F}$ that is true in all models in C, equivalently, $\varphi \in \text{Th}(C)$[2].

We typically want a mutually-independent set of axioms, i.e., one where no single axiom follows from the others. A set of formulae $T \subseteq \mathcal{F}$ is *mutually independent* if $T \setminus \{\varphi\} \not\models \varphi$ for every $\varphi \in T$.

Definition 3.2 (Finite Axiomatization)**.** Given an abstract logic $\mathcal{L} = (\mathcal{F}, \mathcal{S}, \models)$ and target class $C \subseteq \mathcal{S}$, a finite set $A \subseteq \mathcal{F}$ is said to be an axiomatization of C if $C \models A$. Additionally, an axiomatization is said to be *non-vacuous* if A contains no tautologies over $\mathcal{S}$, and it is *mutually independent* if A is also mutually independent.

We develop algorithms for synthesizing sound, non-vacuous, and mutually-independent finite axiomatizations in this work[3].

In the space of models, axiomatization can be viewed as overapproximating C as a subclass of $\mathcal{S}$ using independent axioms A. Consider Figure 2. We see that an axiom φ corresponds to a class of models $\text{Mod}(\varphi) \supseteq C$ such that φ is satisfied everywhere in that class. As the set A grows, the space of models shrinks to

$$\text{Mod}(A) = \bigcap_{\varphi \in A} \text{Mod}(\varphi),$$

as fewer and fewer models satisfy every axiom. The dual perspective gives the *theory of the axioms* $\text{Th}(A) = \{\varphi \in \mathcal{F} \mid A \models \varphi\}$, which grows with A. For any axiomatization A, the theory of A is contained in the theory of C, i.e., $\text{Th}(A) \subseteq \text{Th}(C)$.

Axiomatizations are equipped with a natural partial order on their theories.

Definition 3.3 (Precision Order on Axiomatizations)**.** Let A and A' be axiomatizations of C according to Definition 3.2. We say A is *more precise* than A' if $\text{Th}(A) \supset \text{Th}(A')$.

[2]In practice, we focus on discovering *simple* (e.g., short) axioms.

[3]Note that mutual independence entails non-vacuity since tautologies are entailed by the empty set of axioms. However, we state non-vacuity explicitly as it is a basic property of axiomatizations.

We focus on finding axiomatizations that are as precise as possible, with the ideal scenario being a *complete axiomatization*.

As discussed in Section 1, although it is tempting to think of a complete axiomatization as an axiomatization A such that $\mathrm{Mod}(A) = C$, it may happen that $C \subset C^* := \mathrm{Mod}(\mathrm{Th}(C))$, i.e., C is not $\mathcal{L}$-elementary, and hence the models of any axiomatization of C will always be a strict superset of C. We hence use the following notion of completeness:

Definition 3.4 (Complete Finite Axiomatization)**.** A finite axiomatization A is said to be a complete axiomatization for C if $\mathrm{Th}(A) = \mathrm{Th}(C)$.

This definition allows for completeness with respect to the expressive power of $\mathcal{L}$. Since $\mathrm{Th}(A) \subseteq \mathrm{Th}(C)$ for any axiomatization A, a complete axiomatization is the most precise, i.e., has the largest possible theory among all axiomatizations of C.

Discussion on axioms for proof systems. While our formulation is model-theoretic, it is of course useful in the context of proof systems as well. A proof system is a set of facts along with rules that allow one to infer judgments of the form $\Gamma \vdash \varphi$, which mean that φ is derivable from Γ using the rules. Let us assume a proof system that is sound and *strongly complete* over all models $\mathcal{S}$, or, formally, $\Gamma \vdash \varphi \Leftrightarrow \Gamma \models \varphi$. An axiomatization is then sound for the proof system as well: if A is an axiomatization of C per Definition 3.2 and $A \vdash \varphi$, then $\varphi \in \mathrm{Th}(C)$. Observe that the completeness criterion in this case also reduces to our definition using the completeness of the proof system. Let A be an axiomatization that is complete for C per Definition 3.4, i.e., $A \models \varphi \Leftrightarrow \varphi \in \mathrm{Th}(C)$. Using the soundness and strong completeness of the proof system, we get that $A \vdash \varphi \Leftrightarrow \varphi \in \mathrm{Th}(C)$. Therefore A is also complete for the theory of C with respect to the proof system. In the rest of the paper, we do not explicitly consider axioms for proof systems; we concentrate only on the model-theoretic formulation.

4 LEARNING-BASED AXIOM SYNTHESIS FRAMEWORK

In this section, we describe the Learning-based Axiom Synthesis (LAS) framework to synthesize precise or complete axiomatizations. We argue that effective axiom synthesis algorithms can be built by implementing the LAS framework for domains of interest. We describe the atomic components of the framework in Section 4.1, the high-level algorithm which uses the components in Section 4.2, and in Section 4.3 we formulate a constraint-based synthesizer used for our instantiations of LAS (Sections 5 and 6).

4.1 Components of the LAS Framework

We now develop the set of independent functional components that underlie LAS. Instantiating LAS for a given domain requires the implementation of these components for that domain. The components are parameterized by an abstract logic $\mathcal{L} = (\mathcal{F}, \mathcal{S}, \models)$ and a target class $C \subseteq \mathcal{S}$, which we fix throughout this section. We describe the components below.

- **Soundness checker**: A procedure dubbed VC (*validity over* C), which takes as input a formula $\varphi \in \mathcal{F}$ and determines whether φ is an axiom for C, i.e. whether $C \models \varphi$

Although soundness can be determined using an axiomatization in $\mathcal{L}$, note that having an axiomatization in $\mathcal{L}$ is certainly not a requirement for building VC. We explore two approaches in this paper. For the domain of modal logic, we build VC using an axiomatization in first-order logic, the axioms of which do not indicate the modal axioms in any reasonable sense. For the domain of languages with Kleene star, we build VC using a reference implementation of operations on regular languages.

- **Independence checker**: A procedure dubbed VS (*validity over* $\mathcal{S}$), which takes a set of axioms $A \subseteq \mathcal{F}$ and an axiom $\varphi \in \mathcal{F}$ as inputs and determines whether φ is independent of A, i.e. whether $A \not\models \varphi$. Note that the entailment is over all models in $\mathcal{S}$.
- **Counterexample generator**: A procedure dubbed Cex, which takes as input a formula $\varphi \in \mathcal{F}$ that is not an axiom of C and produces a *pseudo-model* pm as a counterexample to φ.

A natural notion of a counterexample for φ is a model $M \in C$ such that $M \not\models \varphi$. In many domains, however, it may be the case that most (or even all) models in C are infinite (see Section 6). Therefore, we require for such domains the definition of finitely-representable objects called *pseudo-models*, along with a *witness map* $\mathcal{W}$ that associates to every pseudo-model a set of formulae that it rules out. A pseudo-model pm is said to be a *counterexample to* φ if and only if $\varphi \in \mathcal{W}(pm)$. Pseudo-models *witness the existence of models* in C that rule out unsound candidate axioms, and often we can precisely describe these models with a mapping $\mathcal{M}$ that maps a pseudo-model pm to the set $\mathcal{M}(pm) \subseteq C$ of models indicated by pm. When we have the mapping $\mathcal{M}$, one natural definition for $\mathcal{W}(pm)$ is the one that rules out any candidate that is false in all models from $\mathcal{M}(pm)$, i.e. $\mathcal{W}(pm) = \{\varphi \in \mathcal{F} \mid \forall M \in \mathcal{M}(pm),\ M \not\models \varphi\}$. These notions are reminiscent of those from abstract formulations of learning which associate with any sample a *set* of concepts that the sample is consistent with (e.g. [Löding et al. 2016]).

As an example, suppose we are working with a class of models that have as their domain the set of integers. Consider an unsound candidate axiom $\varphi := \forall x.\, f(x) = x$, where f is uninterpreted. One possible pseudo-model pm might specify that $f(1) = 2$ while leaving f unspecified everywhere else. We can take $\mathcal{M}(pm)$ to be the set of all models that have $f(1) = 2$, with $\mathcal{W}$ stipulating that $\varphi \in \mathcal{W}(pm)$ because φ is already false with the partially defined f (and extending f cannot help). Thus pm is a finitely-presented counterexample to φ, but it is not a model, since it does not specify the value of f on all integers. The notion of pseudo-models will vary by domain, and we present the specifics where relevant.

- **Completeness checker**: A procedure dubbed CC, which takes as input a set of axioms A and determines whether A is a complete axiomatization in the sense of Definition 3.4.

As discussed earlier, building CC is difficult, especially when we have $C \subset C^* := \mathrm{Mod}(\mathrm{Th}(C))$. Therefore, we view the completeness checker as an optional component in the LAS framework, and typically we would only consider implementing it when $C^* = C$, where we can make use of knowledge about the target class C. In general, we aim to find axiomatizations that are as precise as possible.

- **Formula Synthesizer**: A procedure dubbed $\mathsf{Learner}_{\mathcal{W}}$, which takes as input a set PM of pseudo-models and a set *Avoid* of formulae and synthesizes a formula $\varphi \in \mathcal{F}$ such that $\varphi \notin \mathcal{W}(pm)$ for every $pm \in PM$ and $\varphi \notin Avoid$. Note that the procedure is parameterized by the domain-specific map $\mathcal{W}$. We describe a formulation for $\mathsf{Learner}_{\mathcal{W}}$ using SMT solvers in Section 4.3.

We demonstrate in the following sections that, by building the various components described above, it is possible to realize effective axiom synthesis for different domains using a single framework. Building effective components is crucial to the success of axiom synthesis in LAS; we make contributions in this respect by implementing variants of the above components for two domains with complex requirements.

4.2 The Core LAS Algorithm

Here we describe the core algorithm for the LAS framework, which utilizes the components described in Section 4.1. We make two simplifications for presentation: (1) we assume all components implement decision procedures and (2) we exclude completeness checking, given that it is optional

parameters: Logic $\mathcal{L} = (\mathcal{S}, \mathcal{F}, \models)$, target class $C \subseteq \mathcal{S}$, and *timeout*
imports: VC, VS, Cex, Learner$_{\mathcal{W}}$ over $\mathcal{L}, C$
output: Axioms $A \subseteq \mathcal{F}$ for C

```
1:  procedure LAS:
2:      A, PM, Avoid ← ∅
3:      repeat
4:          φ ← Learner_W(PM, Avoid)    // Get a proposal not ruled out by counterexamples
5:          sound ← VC(φ)
6:          if sound then
7:              independent ← VS(A, φ)
8:              if independent then
9:                  A ← A ∪ {φ}
10:             else                    // Rule out φ and continue
11:                 Avoid ← Avoid ∪ {φ}
12:                 continue
13:         else                        // Get counterexample and continue
14:             pm ← Cex(φ)
15:             PM ← PM ∪ {pm}
16:     until timeout
17:     return A
```

Algorithm 1. Core LAS Algorithm.

and there are many possible variants for incorporating it within the algorithm. After describing the core algorithm we discuss how it works in general without the simplifications.

The core LAS algorithm is presented in Algorithm 1. It is parameterized by a logic $\mathcal{L}$ and target class C, for which we must implement the soundness checker VC, independence checker VS, counterexample generator Cex, and the formula synthesizer Learner$_{\mathcal{W}}$.

The algorithm maintains a set A of discovered axioms, a set PM of pseudo-models, and a set *Avoid* of formulae, all initially empty. It synthesizes axioms in a loop until a timeout is reached (lines 3-16), at which point it returns A. In a given iteration of the loop, with discovered axioms $A = \{\psi_1, \psi_2, \ldots, \psi_n\}$, it ensures ψ_i is independent of $\{\psi_j \mid 1 \leq j < i\}$, assuming the ψ_i are enumerated in order of discovery. Additionally, for every $\psi \in A$, the algorithm ensures that $\psi \notin \mathcal{W}(pm)$ for every $pm \in PM$.

At the head of the loop, the algorithm queries Learner for a new candidate φ which is neither ruled out by the current counterexample pseudo-models nor a member of the set *Avoid* of formulae (line 4). The algorithm checks whether φ is sound using VC (line 5). If sound, it checks for independence from A using VS (line 7). If φ is independent, the algorithm adds it to A and continues to the next iteration of the loop to find more axioms (line 9). If not independent, i.e., φ is entailed by A, the algorithm discards φ (adding it to the *Avoid* set to prevent it from being proposed in the future) and goes back to the head of the loop to get another candidate (lines 11-12). If the soundness check fails, the algorithm queries Cex for a pseudo-model pm such that $\varphi \in \mathcal{W}(pm)$ and adds it to PM before returning to the head of the loop (lines 14-15).

We now discuss some technical details about how the algorithm works in the general case without the simplifications.

Nonterminating Procedures. In general, the components described in Section 4.1 may only be realizable as nonterminating procedures. For example, in Section 5 we instantiate the framework

for modal logics, where VS is implemented with a procedure for first-order logic validity, which can only be a *semi*-decision procedure that halts and returns true on valid formulae but may not terminate on invalid formulae. Therefore, we modify each component to take an additional input $fuel \in \mathbb{N}$ that models a resource bound and ensures termination. For example, we modify VC in this way so that for every $fuel \in \mathbb{N}$ and $\varphi \in \mathcal{F}$ we have that $\mathsf{VC}(\varphi, fuel)$ always terminates, saying either that φ is an axiom or is not an axiom, or else *Unknown*. This allows us to stage the various procedures as terminating sub-procedures which we can call iteratively with increasing *fuel*.

With this modification, we must also extend the algorithm to make decisions when a component returns *Unknown*. If Cex returns *Unknown* (line 14), we can discard φ after adding it to the *Avoid* set, which rules it out from future proposals, and continue searching for more axioms without adding a counterexample. If Learner returns *Unknown*, we can increase its *fuel* or exit and return A. If VC returns *Unknown* on a given axiom φ, we could add φ to the list of axioms and emit a warning that φ may not be sound. We could also discard φ (risking that we do not find a useful, sound axiom) or increase *fuel* and rerun VC on φ. Similarly, if we cannot prove independence from A for a given sound axiom using VS, we must choose between running again with more *fuel*, keeping the potentially redundant axiom, or discarding it. Which choices are effective will vary by setting, and we leave them up to the implementation.

Mutual Independence. Observe that the axioms A returned by the algorithm are not necessarily mutually independent. The algorithm only ensures that each axiom is independent from those discovered before it. This is a practical choice, since finding a minimal mutually-independent subset of A that covers A could require an exponential number of calls to VS. In many cases it may be preferable to have a larger set of simple axioms rather than a smaller set of complex axioms. However, if we assume that the Learner component outputs candidates in order of increasing complexity, it may happen that a simple axiom discovered early may be entailed by a combination of more complex axioms discovered later. One way to address this is to run the algorithm to obtain a set of axioms $A = \{\psi_1, \psi_2, \ldots, \psi_n\}$, and then use VS to remove ψ_1 from A if it is entailed by $A \setminus \{\psi_1\}$. We can repeat this process with ψ_2 and $A \setminus \{\psi_1\}$, and continue until each remaining axiom is not entailed by the others, using only a linear number of calls to VS.

Completeness Check. If it is possible to check completeness, then the algorithm can use completeness as a termination condition. However, it may not make practical sense to check completeness each time a new axiom is added to A as it could be expensive. Other choices include checking completeness at the end of synthesis or checking at regular intervals. Again, these are choices that depend on the domain and are left up to the implementation.

4.3 Realizing the Learner using an SMT Solver

We implement a generic learner parameterized by a logic (syntax and semantics), which synthesizes expressions that are not ruled out by any of a given set of pseudo-models. We use established constraint solving techniques to synthesize formulae of bounded height (we assume that the logic has a bounded number of constants). Note that the constraints encoding whether a formula is ruled out by a pseudo-model are determined by the domain-specific map $\mathcal{W}$. We thus require that, for any pseudo-model pm, the constraints encoding $\varphi \notin \mathcal{W}(pm)$ are expressible as a quantifier-free first-order formula[4]. Consequently, the synthesis of bounded-depth logical expressions reduces to quantifier-free satisfiability and can be accomplished (often) with an SMT solver. More precisely, given a bound b on the depth of expressions, we can use a set of Boolean variables V_b to encode

[4]Evaluating operators like universal quantification on a pseudo-model with a finite domain can be expressed as a quantifier-free formula using a conjunction over the elements of the domain of the pseudo-model.

choices for how the nodes in the parse tree of the expression are to be filled. For any fixed pseudo-model pm, we then write a formula $Allowed_{pm}(V_b)$ that constrains the expression (represented by an assignment to V_b) so that it is *not* ruled out by pm. We then check the satisfiability of the formula:

$$\bigwedge_{pm \in PM} Allowed_{pm}(V_b)$$

conjuncted with a constraint that rules out all the formulae in the set *Avoid*. If satisfiable, we can extract the expression from the assignment to V_b in the satisfying model. We implement this learner for the required logic in each setting.

5 AXIOMATIZING CLASSES OF FRAMES IN MODAL LOGIC

In this section, we instantiate the LAS framework for the setting of modal logic and the problem of axiomatizing first-order definable classes of frames, which are first-order structures over a single binary "accessibility" relation R. Section 5.1 provides background, Section 5.2 explains the components of the framework and discusses nuances for this particular setting, and Section 5.3 describes details of the implementation and the axiomatizations we find.

5.1 Modal Logic and Correspondence Theory

We now briefly review the syntax and semantics of propositional modal logic over Kripke structures. We are interested in axiomatizing properties of the accessibility relation for Kripke structures using modal logic formulae. In particular, we aim to find axiomatizations of first-order definable properties that are classically studied in correspondence theory. We review some background from correspondence theory following the syntax and semantics of propositional modal logic.

5.1.1 Syntax and Semantics of Modal Logic. We consider propositional modal logic over Kripke structures (henceforth called models) of the form $M = (W, R, V)$, consisting of a set of *worlds* W an *accessibility relation* $R \subseteq W \times W$, and a *valuation* $V : W \to \mathcal{P}(Prop)$ that maps worlds to a subset of propositions from a finite set *Prop*. We refer to a pair $F = (W, R)$ as a *frame*, and in the context of a specific model $M = (W, R, V)$ we refer to F as the frame of M. Frames are the basic object we aim to axiomatize.

Formulae in the logic are given by the following grammar:

$$\varphi \ ::= \ \top \mid p \in Prop \mid \neg\varphi \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid \Box\varphi \mid \Diamond\varphi$$

The formulae $\Diamond\varphi$ and $\Box\varphi$ can be read in the usual way as "it is possible that φ" and "it is necessary that φ", respectively. The semantics of modal logic defines when a formula is true at a given world $w \in W$ in a model $M = (W, R, V)$:

$$\begin{aligned}
&M, w \models \top \text{ always}\\
&M, w \models p \text{ iff } p \in V(w)\\
&M, w \models \neg\varphi \text{ iff } M, w \not\models \varphi\\
&M, w \models \varphi \vee \varphi' \text{ iff } M, w \models \varphi \text{ or } M, w \models \varphi'\\
&M, w \models \varphi \wedge \varphi' \text{ iff } M, w \models \varphi \text{ and } M, w \models \varphi'\\
&M, w \models \Box\varphi \text{ iff } M, w' \models \varphi \text{ for every } (w, w') \in R\\
&M, w \models \Diamond\varphi \text{ iff } M, w' \models \varphi \text{ for some } (w, w') \in R
\end{aligned}$$

These are standard syntax and semantics of propositional modal logic [Blackburn et al. 2006].

5.1.2 Correspondence Theory in Modal Logic. Modal Correspondence Theory [Van Benthem 1984] studies connections between modal logic formulae and classical first-order properties of frames. As an example, the modal formula $\Box p \rightarrow p$ corresponds to the *reflexive frames*, i.e., those for which $\forall x.R(x,x)$ holds when we treat a frame $F = (W, R)$ as a first-order structure. The precise correspondence is the following:

Fact 5.1. For any frame $F = (W, R)$, the accessibility relation R is reflexive if and only if the modal formula $\Box p \rightarrow p$ is true in $M = (W, R, V)$ for all valuations V and worlds w.

Many correspondences of this kind are known to exist between modal formulae and the accessibility relation, and they rely on the notion of *frame validity*, defined as follows:

Definition 5.1 (Frame Validity)**.** A modal formula is valid in a frame $F = (W, R)$, written $F \models_f \varphi$, if $(W, R, V), w \models \varphi$ for every world $w \in W$ and every valuation $V : W \rightarrow \mathcal{P}(Prop)$.

For intuition, let us work through the proof of Fact 5.1.

Proof of Fact 5.1. *Soundness ($\Rightarrow$).* Suppose $F = (W, R)$ is a reflexive frame, i.e., $\forall x.R(x,x)$ is true, and let V, w be an arbitrary valuation and world, respectively. Suppose $(W, R, V), w \models \Box p$, i.e., every world accessible from w has p true under V (if not, the implication is already true). We have that $(W, R, V), w \models p$ holds because $R(w, w)$ holds. Note that this direction of the proof corresponds to what VC checks in our framework.

Completeness ($\Leftarrow$). We prove the contrapositive. Note: the intuition behind this direction is that we pick a w for which $R(w, w)$ is false, and from the form of the axiom we pick a valuation V that makes the axiom false at w. (We explain how we automate some of this intuition in Section 5.2.) Proof: suppose F is not reflexive. That means there is some $w \in W$ where $R(w, w)$ does not hold. Let V be a valuation that makes p false at w and true at all w' for which $R(w, w')$ holds. Then $(W, R, V), w \models \Box p$, but $(W, R, V), w \not\models p$. □

Our goal now is to instantiate the LAS framework to find modal logic formulae that characterize various classes of frames, as described above. Note that the axioms we aim to find in this setting can be interpreted as axiom schemas. For instance, the formula $\Box p \rightarrow p$ can soundly be interpreted as a schema $\Box\alpha \rightarrow \alpha$, where α is a placeholder for a modal formula, i.e., all instances of this schema are valid in the class of reflexive frames, and the same can be said of the other classes we axiomatize. We refer the reader to [Van Benthem 1984] for more about correspondence theory.

5.2 Instantiating the Framework for Modal Logic

We now instantiate the framework from Section 4 for the problem of synthesizing modal logic axioms that characterize a target class of frames. We aim to axiomatize classes of frames that are definable in first-order logic, i.e., classes defined by a first-order logic sentence ψ over the relation symbols $R, =$, e.g., reflexive frames defined by $\psi := \forall x.R(x,x)$. Because modal logic formulae can be translated to first-order logic, the components make use of existing validity procedures for first-order logic. We discuss this translation next.

Definition 5.2 (Translation of Modal Logic to First-Order Logic)**.** Given a modal logic formula φ, we can translate φ into a first-order logic formula $\psi(x)$ such that $(W, R, V), w \models \varphi$ if and only if $M' \models \psi(w)$, where M' extends the frame (W, R) (as a first-order structure) with interpretations for unary predicates P, one for each proposition $p \in Prop$. Each predicate P holds for the worlds w for which $p \in V(w)$. The translated formula $\psi(x) := \mathsf{ml2fo}_x(\varphi)$ is defined inductively in the structure of φ as shown below, with x, y, etc. drawn from an infinite supply of fresh variables:

$$\mathsf{ml2fo}_x(\top) = \top \qquad \mathsf{ml2fo}_x(\varphi \wedge \varphi') = \mathsf{ml2fo}_x(\varphi) \wedge \mathsf{ml2fo}_x(\varphi')$$
$$\mathsf{ml2fo}_x(p) = P(x) \qquad \mathsf{ml2fo}_x(\Box\varphi) = \forall y.\, \big(R(x,y) \rightarrow \mathsf{ml2fo}_y(\varphi)\big)$$
$$\mathsf{ml2fo}_x(\neg\varphi) = \neg\mathsf{ml2fo}_x(\varphi) \qquad \mathsf{ml2fo}_x(\Diamond\varphi) = \exists y.\, \big(R(x,y) \wedge \mathsf{ml2fo}_y(\varphi)\big)$$
$$\mathsf{ml2fo}_x(\varphi \vee \varphi') = \mathsf{ml2fo}_x(\varphi) \vee \mathsf{ml2fo}_x(\varphi')$$

Recall that the LAS framework involves the procedures VC, VS, and Cex, whose respective purposes are to check soundness, check independence, and generate counterexamples for candidate axioms. In this setting, we aim to find axioms for a subclass of frames, and thus $\mathcal{S}$ is the class of all frames and $\mathcal{C}$ is a subclass of $\mathcal{S}$, e.g., reflexive frames. The logic $\mathcal{L}$ is modal logic with *frame validity* for the entailment relation (Definition 5.1). In this setting, we also implement a completeness checker CC. We explain each component next and discuss some details for the Learner in Section 5.3.

Instantiating VC. We reduce the soundness condition for modal logic axioms over a class of first-order definable frames $\mathcal{C}$ to the validity problem in first-order logic (more precisely, the relational theory of equality and uninterpreted relations) by using the first-order characterization for $\mathcal{C}$. Recall the proof of soundness for Fact 5.1. Suppose we are axiomatizing the class of reflexive frames, which are characterized by the first-order sentence $\forall x.R(x,x)$. To check the proposed axiom $\Box p \rightarrow p$ is sound (true in all frames from $\mathcal{C}$), we can check the validity of the first-order sentence $\forall x.R(x,x) \rightarrow (\forall x.\mathsf{ml2fo}_x(\Box p \rightarrow p))$. More generally, for a class of frames defined by a first-order sentence ψ, we check the soundness of a candidate modal axiom φ by checking the validity of

$$\mathsf{sound}(\psi, \varphi) \coloneqq \psi \rightarrow \forall x.\mathsf{ml2fo}_x(\varphi).$$

Observe that the ml2fo translation turns propositions into uninterpreted unary relations, and thus first-order validity of the translated formula requires the formula to be true for all interpretations of the unary relations. This corresponds to quantifying over all valuations. We can thus implement VC using any semi-decision procedure for validity in first-order logic (see Section 5.3 for details).

Instantiating Cex. As discussed above, we reduce soundness to the relational theory of equality and uninterpreted relations, which is recursively enumerable but undecidable. Since satisfiability is not recursively enumerable, it is straightforward to show there can be no complete procedure to find counterexample models for an unsound candidate φ, i.e., a model of $\neg\mathsf{sound}(\psi, \varphi)$. However, our intuition is that finite counterexample models are enough to efficiently synthesize many modal axiomatizations. Given a candidate φ that is not sound, the component Cex produces a frame $F = (W, R) \in \mathcal{C}$ of a small, bounded size such that $F \not\models_f \varphi$. Note that in this setting we can take pseudo-models to simply be finite frames F, with $\mathcal{M}(F) = \{F\}$ and $\varphi \in \mathcal{W}(F) \Leftrightarrow F \not\models_f \varphi$.

Instantiating VS. Given a set of modal axioms $A = \{\varphi_1, \ldots, \varphi_n\}$ and a candidate axiom φ, we want to check that φ is independent of A. That is, we want to check that there is a frame $F = (W, R)$ such that $F \models_f \varphi_i$ for each i but $F \not\models_f \varphi$. Checking the existence of such a frame (i.e., independence) does not reduce to first-order validity, but is captured by the second-order formula

$$\exists(W,R).\, \bigwedge_i \forall P.\forall x.\mathsf{ml2fo}_x(\varphi_i) \;\wedge\; (\exists P'.\exists x.\neg\mathsf{ml2fo}_x(\varphi)),$$

where P, P' are sequences of second-order variables corresponding to the unary predicates produced by the ml2fo translations.

We approximate independence by checking whether there is such a frame $F = (W, R)$ of some small, bounded size. As for finding counterexamples to soundness, our intuition is that small witnesses will exist to show the independence of modal axioms. We note that this heuristic could cause independent axioms to be discarded if we insist on finding independence proofs and there are only large frames witnessing independence. In principle, we can mitigate that risk by increasing the size bound, but this proved unnecessary in experiments.

Instantiating CC. We now describe a heuristic procedure for completeness. Let us assume a first-order characterization ψ for a class of frames and a set of sound modal logic axioms A. Let us also assume, without loss of generality, that the propositions in each axiom are disjoint, and let φ be the conjunction of the modal axioms in A.

The formula φ is complete for the class of frames described by ψ if for all frames F, whenever $F \models_f \varphi$ (frame validity Definition 5.1) then $F \models \psi$ (first-order validity). In other words, we need to check the validity (over all frames) of the second-order sentence

$$(\forall P.\forall w.\mathsf{ml2fo}_w(\varphi)) \rightarrow \psi. \tag{4}$$

where once again P is a sequence of second-order variables introduced by $\mathsf{ml2fo}_w(\varphi)$.

Following a common pattern for completeness proofs in correspondence theory, we can try to prove the contrapositive of Equation (4): assume the first-order characterization ψ does not hold on some frame and then try to find a specific valuation and world that violate φ.

Using this intuition from manual proofs, we reduce the problem to a stronger version where we try to find a *finite* set of worlds $\{w_1, \ldots, w_n\}$ and a valuation on them that violates φ. For worlds outside $\{w_1, \ldots, w_n\}$, we simply assume that the valuation uniformly assigns the same default value v_{def}. For instance, suppose we only have one atomic proposition p. Then, instead of the second-order sentence (4), we can check the validity of the following first-order sentence:

$$(\forall w_1, \ldots, w_n.\forall v_1, \ldots, v_n, v_{\text{def}}.\forall w.\mathsf{ml2fo}'_w(\varphi)) \rightarrow \psi \tag{5}$$

where $v_1, \ldots, v_n, v_{\text{def}}$ range over Booleans (modeling the valuation of P on $w_1, \ldots, w_n$ and the default value), and where $\mathsf{ml2fo}'_w(\varphi)$ is $\mathsf{ml2fo}_w(\varphi)$ where each predicate occurrence $P(w)$ is defined as:

$$P(w) := ite(w = w_1, v_1, ite(w = w_2, v_2, \ldots ite(w = w_n, v_n, v_{\text{def}}) \ldots)).$$

Notice that the validity of Formula 5 implies the validity of Formula 4. Furthermore, Formula 5 is in first-order logic (as valuations have been replaced by a finite set of Boolean variables), and we can use automatic procedures for first-order logic to solve validity. This approximate checking for completeness works well in practice (in fact, it works for 14/17 of the modal axiomatizations we explore in Section 5.3).

Table 1. First-order descriptions used in Table 2.

FO Description	**Definition**
Reflexive	$\forall x.R(x,x)$
Transitive	$\forall x,y,z.(R(x,y) \wedge R(y,z)) \rightarrow R(x,z)$
Symmetric	$\forall x,y.R(x,y) \rightarrow R(y,x)$
Euclidean	$\forall x,y,z.(R(x,y) \wedge R(x,z)) \rightarrow (R(y,z) \wedge R(z,y))$
Functional	$\forall x,y,z.(R(x,y) \wedge R(x,z)) \rightarrow y = z$
Shift Reflexive	$\forall x,y.R(x,y) \rightarrow R(y,y)$
Dense	$\forall x,y.R(x,y) \rightarrow \exists z.R(x,z) \wedge R(z,y)$
Serial	$\forall x.\exists y.R(x,y)$
Convergent	$\forall x,y,z.(R(x,y) \wedge R(x,z)) \rightarrow \exists w.R(y,w) \wedge R(z,w)$

5.3 Implementation and Evaluation

We implemented the procedures VC, VS, Cex, and CC as described in previous sections, reducing the problems to SMT queries in Z3 [De Moura and Bjørner 2008]. We built an axiom synthesizer (Learner) for modal logic and combined the components according to the general algorithm in Algorithm 1. The implementation can be found in our GitHub repository [Krogmeier et al. 2022a].

5.3.1 Implementation Details. Let ψ be a first-order description for a class of frames for which we want to synthesize modal axioms, let φ be a candidate modal axiom, and let A be a set of sound modal axioms that have already been synthesized.

For soundness, VC generates an SMT query checking whether $\psi \wedge \neg\forall P.\forall w.\mathsf{ml2fo}_w(\varphi)$ is satisfiable, where P stands for a sequence of unary relation symbols corresponding to propositions in φ. This is equivalent to $\psi \wedge \exists P.\exists w.\neg\mathsf{ml2fo}_w(\varphi)$. As discussed in Section 5.2, the translated formula $\mathsf{ml2fo}_w(\varphi)$ replaces propositions with uninterpreted unary relations, and thus the quantification of P can be removed, giving the formula $\psi \wedge \exists w.\neg\mathsf{ml2fo}_w(\varphi)$ in the first-order theory of equality and uninterpreted relations. If this formula is not satisfiable, then φ is sound; otherwise, we proceed to generate a counterexample as in Algorithm 1.

Table 2. Synthesis results for modal logics. For each logic, we synthesized modal axioms from the first-order description of the logic (see Table 1 for the formulae used). The Reference Axioms column shows canonical axioms studied in the literature; the LN column shows the time taken by the learner Learner$_{\mathcal{W}}$; the CX column shows the time taken by the counterexample generator Cex; the SN column shows the time taken by the soundness checker VC; the CM column shows the time taken by the completeness check. Times are given in seconds.

Logic	**FO Description** (See Table 1)	**Synthesized Axioms**	**Reference Axioms**	**LN**	**CX**	**SN**	**CM**
(M)	Reflexive	$\neg\alpha \vee \Diamond\alpha$	$\Box\alpha \rightarrow \alpha$	9.9	0.00	0.06	0.03
(4)	Transitive	$\Box\alpha \rightarrow \Box\Box\alpha$	$\Box\alpha \rightarrow \Box\Box\alpha$	22.7	0.01	0.03	1.01
(B)	Symmetric	$\Diamond\Box\alpha \rightarrow (\Diamond\alpha \rightarrow \alpha)$	$\alpha \rightarrow \Box\Diamond\alpha$	25.0	0.01	0.09	0.05
(5)	Euclidean	$\Diamond\Box\alpha \rightarrow \Box\alpha$	$\Diamond\alpha \rightarrow \Box\Diamond\alpha$	29.7	0.01	0.13	timeout
(□M)	Shift Reflexive	$\Box(\alpha \rightarrow \Diamond\alpha)$	$\Box(\Box\alpha \rightarrow \alpha)$	27.7	0.01	0.14	0.04
(D)	Serial	$\Diamond\top$	$\Box\alpha \rightarrow \Diamond\alpha$	26.9	0.00	0.03	0.04
(C)	Convergent	$\Diamond\Box\alpha \rightarrow \Box\Diamond\alpha$	$\Diamond\Box\alpha \rightarrow \Box\Diamond\alpha$	23.4	0.05	0.07	timeout
(CD)	Functional	$\Diamond\alpha \rightarrow \Box\alpha$	$\Diamond\alpha \rightarrow \Box\alpha$	46.8	0.01	0.04	0.06
(C4)	Dense	$\Diamond\alpha \rightarrow \Diamond\Diamond\alpha$	$\Box\Box\alpha \rightarrow \Box\alpha$	24.2	0.01	0.13	0.05
(K45)	(4) + (5)	$\Diamond\Diamond\alpha \rightarrow \Diamond\alpha$ $\neg\Diamond\alpha \vee \Box\Diamond\alpha$	$\Box\alpha \rightarrow \Box\Box\alpha$ $\Diamond\alpha \rightarrow \Box\Diamond\alpha$	27.0	0.01	0.15	0.82
(KB5)	(B) + (5)	$\neg\alpha \vee \Box\Diamond\alpha$ $\Box\Diamond\alpha \vee \Box\neg\alpha$	$\alpha \rightarrow \Box\Diamond\alpha$ $\Diamond\alpha \rightarrow \Box\Diamond\alpha$	17.7	0.00	0.14	43.06
(D4)	(D) + (4)	$\Box\Diamond\alpha \rightarrow \Diamond\alpha$ $\Diamond\Diamond\alpha \rightarrow \Diamond\alpha$	$\Box\alpha \rightarrow \Diamond\alpha$ $\Box\alpha \rightarrow \Box\Box\alpha$	55.2	0.01	0.33	0.17
(D5)	(D) + (5)	$\Diamond\top$ $\Box\neg\alpha \vee \Box\Diamond\alpha$	$\Box\alpha \rightarrow \Diamond\alpha$ $\Diamond\alpha \rightarrow \Box\Diamond\alpha$	30.8	0.01	0.19	11.66
(D45)	(D) + (4) + (5)	$\Diamond\top$ $\Box\Diamond\alpha \vee \Box\neg\alpha$ $\Diamond\Diamond\alpha \rightarrow (\Box\alpha \vee \Diamond\alpha)$	$\Box\alpha \rightarrow \Diamond\alpha$ $\Box\alpha \rightarrow \Box\Box\alpha$ $\Diamond\alpha \rightarrow \Box\Diamond\alpha$	43.5	0.01	0.15	6.41
(DB)	(D) + (B)	$\alpha \rightarrow \Diamond\Diamond\alpha$ $\neg\alpha \vee \Box\Diamond\alpha$	$\Box\alpha \rightarrow \Diamond\alpha$ $\alpha \rightarrow \Box\Diamond\alpha$	36.2	0.01	0.18	0.05
(M4)	(M) + (4)	$\Diamond\alpha \vee \neg\alpha$ $\Diamond\Diamond\alpha \rightarrow \Diamond\alpha$	$\Box\alpha \rightarrow \alpha$ $\Box\alpha \rightarrow \Box\Box\alpha$	19.3	0.00	0.10	0.74
(M5)	(M) + (5)	$\Diamond\alpha \vee \neg\alpha$ $\Box\Box\alpha \vee \Diamond\neg\alpha$ $\Diamond\Box\alpha \rightarrow \alpha$	$\Box\alpha \rightarrow \alpha$ $\Diamond\alpha \rightarrow \Box\Diamond\alpha$	13.1	0.00	0.14	timeout

For independence, VS queries the SMT solver to find a bounded frame witnessing $A \not\models \varphi$. If such a bounded frame exists, then the axiom is *independent*, and we discard the proposal otherwise.

We optimize the implementation for this domain by merging the SMT query for checking independence and the SMT query for synthesis (i.e. Learner). This is sound since we proceed with a candidate axiom only if both queries are satisfiable. We found that not combining these queries resulted in the synthesis of a large number of candidates that were immediately ruled out by the independence checker. This essentially makes the algorithm enumerate all semantically equivalent variants of a formula and it resulted in performance similar to naive enumeration (which we show is less efficient in Section 5.3.3). The use of a single query also results in only a small number of non-independent axioms being proposed, as we show in our evaluation (see Figure 3).

For counterexamples, Cex also queries the SMT solver to find a bounded frame satisfying ψ but not φ. In our evaluation, we use a size bound of 4. Note that since the counterexample frames are bounded, VS and Cex are guaranteed to terminate. However, VC tries to decide a first-order sentence using a semi-decision procedure so it may not terminate. The *fuel* parameter determines the timeout (Section 4.2) in this case.

5.3.2 Evaluation Results. We attempted 17 historically-studied classes of modal logic frames, and we were able to synthesize complete axiomatizations for all of them. Our completeness procedure in Section 5.2 verifies automatically that the axiomatization is complete with respect to the FO description for 14 classes. We manually verified the completeness for the other 3 classes.

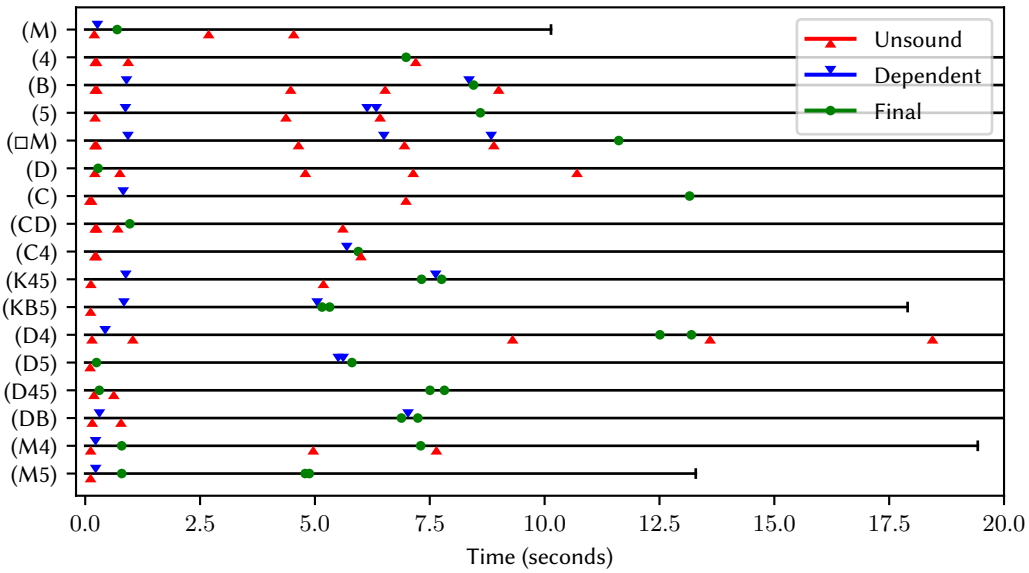


Fig. 3. Distribution of candidate axioms proposed for each modal logic. The x-axis shows the time at which each axiom was proposed, and each horizontal line shows the duration of synthesis (not including the completeness check) cut off at 20 seconds (see Table 2 for the full duration). "Unsound" refers to axioms that failed the soundness checker VC; "Dependent" refers to axioms that were proven sound but were entailed by later proposals and pruned in a post-processing pass; "Final" indicates the axioms that our tool outputs given in Table 2.

We used a laptop with a 4-core (8-thread) Intel CPU i7-8550U and 16 GB of memory. Table 2 shows the results of our evaluation with the synthesized axioms, reference axioms studied in the literature, and the breakdown of time spent in each component. Figure 3 shows the distribution of candidate axioms proposed for each modal axiomatization, including unsound proposals.

In our evaluation, we considered grammars that allowed all possible modal logic formulae with one proposition, and the height of formulae was increased incrementally until 3. Counterexample frames were bound to at most 4 worlds.

In all of the cases, the synthesizer effectively produces the complete axioms in less than a minute. We believe that the main reason for this effectiveness is that we have an ideal type of counterexample in modal logic: extremely small frames are already able to characterize a class with a first-order description.

The axiomatizations synthesized by the tool, modulo small syntactic equivalences, correspond closely to the reference axiomatizations we see in the literature (see Table 2). For the class of serial frames, the tool found the axiom $\Diamond\top$ while the reference axiom is $\Box\alpha \rightarrow \Diamond\alpha$. We verified that the tool's simpler axiom is indeed sound and complete. There may be aesthetic reasons why humans avoid axioms with constants such as $\top$, and there are several other frames where our tool generates axioms with constants.

For classes that are characterized by conjunctions of multiple first-order properties, note that our tool does not know this fact, and often synthesizes a different set of axioms than the union of the axioms for each property, such as (M5).

The completeness procedure failed in some cases. Recall that for completeness, we need to check the validity of the second-order sentence from Equation (4), which we handled using an incomplete reduction to Equation (5). But this reduction is not enough for some of the cases we evaluated. For example, to prove completeness for the convergence axiom $\varphi = \Diamond\Box\alpha \rightarrow \Box\Diamond\alpha$ (corresponding to the first-order description $\psi = (R(x,y) \wedge R(x,z)) \rightarrow \exists w.R(y,w) \wedge R(z,w)$), one has to show that for any non-convergent frame (satisfying $\neg\psi$), we can find a valuation V such that $\neg\varphi$ holds for some world w. To do this, we have to first find a witness to $\neg\psi$, i.e., a world w and two worlds v_1 and v_2 with $R(w,v_1)$ and $R(w,v_2)$, such that v_1 and v_2 have disjoint successors in R. Then we can pick an atomic proposition α, and assign α to all successors of v_1 (which may be infinite), and $\neg\alpha$ to all successors of v_2. Then φ is not true at w because $\Diamond\Box\alpha$ is true, but $\Box\Diamond\alpha$ is false. This proof requires us to find a valuation that may vary on an infinite set of worlds (all the successors of v_1 and v_2) as opposed to the finite-varying valuation in the reduction to Equation (5).

5.3.3 Comparison with Brute-force Enumeration. We evaluated a synthesis procedure that uses brute-force enumeration without counterexamples for synthesis rather than constraint solving with counterexamples. In terms of our core algorithm (Algorithm 1), this corresponds to implementing VC (needed for soundness) and VS (needed for independence checking), but skipping Cex and having the Learner component simply enumerate formulae.

In this evaluation, Learner enumerates all modal logic formulae up to height 3 (which is the maximum height used in our previous evaluation in Section 5.3.2). For each enumerated candidate, we perform the soundness check using VC and the independence check using VS as before.

We ran the enumerative procedure for all 17 modal axiomatizations (the enumeration is naive and does not rule out any kind of symmetries), on a server with 32 Intel Xeon 8124M CPUs and 72 GB of memory. We used a timeout of 3 hours for each modal axiomatization setting.

In all 17 settings, brute force enumeration was unable to exhaustively search the space for axiomatizations, while our technique was able to exhaustively search the space within a minute. However, one can argue that exhausting the search space is unnecessary if the synthesizer can produce a provably complete axiomatization earlier. This would require running the completeness checker

each time an axiom is added. Indeed, enumeration does produce provably complete axiomatizations for 13 cases early on, and the time to reach such a complete axiomatization was less than a minute in 5 cases and about 8-11 minutes in the other 8 cases. Note that this does not account for the additional time spent checking completeness with each discovered axiom. However, for the 4 remaining settings where completeness cannot be checked automatically, an enumerative tool that implements completeness checks with each discovered axiom would have to run until the timeout of 3 hours. More importantly, the enumerative tool's final axiomatizations (terminated at 3 hours) were not complete for two settings ((C) and (□M)), whereas our tool found complete axiomatizations for all settings within a minute.

This evaluation suggests that although the synthesized axioms in each setting are short, brute force enumeration is not promising and is unlikely to scale to more complex settings, while using counterexamples to guide search with constraint-solving is significantly faster. Moreover, early termination is not possible when completeness checks fail, and exhaustive enumeration takes prohibitively long in these cases.

6 AXIOMATIZING LANGUAGES WITH KLEENE STAR

In this section, we instantiate the LAS framework to find axioms for *languages with Kleene star*. Each model in this case consists of a set of languages over a finite alphabet that is closed under the usual operations from formal language theory. Section 6.1 reviews some background from language theory, Section 6.2 discusses nuances for this setting, and Section 6.3 describes our implementation and results.

6.1 Language Models

Languages of finite words and the operations of concatenation, union, and Kleene closure are fundamental concepts in computer science. Much work went into the discovery of axioms for reasoning with these concepts (e.g. [Conway 1971; Kozen 1994; Salomaa 1966]), and we are interested in discovering such axioms *de novo*. In particular, we want to axiomatize a class of algebraic structures over the signature $\tau = (\cdot, +, *, 1, 0)$, which we refer to as the class of *language models*. The symbols $\cdot$ and $+$ are binary function symbols corresponding to concatenation and union, $*$ is a unary function symbol corresponding to Kleene closure, and 1 and 0 are constants corresponding to the singleton language containing the empty word and the empty language, respectively. The domain of a language model over an alphabet Σ consists of languages of words over Σ, i.e., $D \subseteq \mathcal{P}(\Sigma^*)$. We review the standard language theory interpretations for τ-symbols below.

Let x, y be languages over Σ. The operation $+ : D \times D \to D$ is interpreted as union:

$$x + y \coloneqq x \cup y$$

The operation $\cdot : D \times D \to D$ is interpreted as the concatenation of languages:

$$xy = x \cdot y \coloneqq \{w_1 w_2 \mid w_1 \in x, w_2 \in y\}$$

where $w_1 w_2$ denotes the concatenation of words defined in the usual way. The n-fold concatenation of a language x with itself is given by:

$$x^0 \coloneqq \{\epsilon\} \qquad x^{n+1} \coloneqq x^n x,$$

where ϵ is the empty word. The constants 0 and 1 denote the empty language $\emptyset$ and the singleton language $\{\epsilon\}$, respectively. Finally, the operation $* : D \to D$ forms the closure of a language under concatenation with itself:

$$x^* = *(x) \coloneqq \bigcup_{i \in \mathbb{N}} x^i$$

For a fixed alphabet Σ, we refer to these models as Σ-*language models*. We write t_i^M for the language denoted by t in a model M under variable assignment i. For example, if M contains the languages a^*, $\{\epsilon\}$, and $\emptyset$, then for an assignment i with $i(x) = a^*$ and $i(y) = \{\epsilon\}$ we have $(xy)_i^M = a^*$.

Note that language models need not consist of regular languages, and many equations hold in some models and not others. For instance, we may have a model consisting of the context-free language $L = \{a^n b^n \ : \ n \in \mathbb{N}\}$, as well as the languages formed by closing the domain under concatenation, union, and Kleene closure (and also adding the languages for 0 and 1). In this model, it happens that the equation $xy = yx$ is true. But of course, this is not true for all language models: for example, we can take x to be the language $\{a\}$ and y to be the language $\{b\}$ in any model that contains those languages.

6.2 Instantiating the Framework for Language Models

We now describe how the main components of the framework are instantiated to find equational first-order logic axioms for language models. Following the discussion above, we can identify the class $\mathcal{S}$ to be all models over the signature $\tau = (\cdot, +, *, 1, 0)$, with C consisting of all language models over finite alphabets, as defined in Section 6.1. Note that to build the components for this setting we make use of only basic knowledge about the target class, which we discuss next.

It so happens that the equational theory (the set of all true universally-quantified equations) of the class of Σ-language models coincides with a distinguished language model called Reg_Σ, whose domain consists of all regular languages over Σ. The problem of axiomatizing the equational theory of Reg_Σ has a long history. It was first posed by Kleene [Kleene 1956], with several contributions toward axiomatization (e.g., [Conway 1971; Redko 1964; Salomaa 1966]), and culminated in Kozen's finite axiomatization consisting of conditional and unconditional equations that were proven complete for the equational theory [Kozen 1994]. Though we do not aim *a priori* to rediscover precisely that axiomatization, it informs our choice to focus on finding *equational* axioms. Thus the set of formulae $\mathcal{F}$ consists of universally-quantified equations in first-order logic over the signature τ. We invite the reader in the remainder of this section to start fresh and naively explore what is necessary to find axioms in this setting, and in particular, how to build VC and Cex.

Instantiating VC. Our goal is to build a procedure that checks whether a candidate equational axiom is true in the class C of language models. Suppose in particular that we want to prove that a (universally-quantified) equation in n variables is true for all language models (over arbitrary finite alphabets Σ). Assume we have a formula $\forall x.t(x) = t'(x)$, for some τ-terms t, t' and a sequence of variables x, and we want to prove that for any language model M, we have $t = t'$ for any assignment of variables to languages from the domain of M. Observe first that it is *necessary* that $t = t'$ holds when each variable x_i is assigned the singleton language $\{x_i\}$, where we treat each variable as a distinct alphabet symbol. If not, then the equation does not hold in any language model containing these n singleton languages $\{x_i\}$. Call this singleton assignment s.

It turns out this condition is also *sufficient* for an equation to be true in all language models (observed by Gischer [Gischer 1985], see also a detailed proof in [Hopcroft et al. 2006]). The argument runs as follows. Suppose for contradiction that the equation is true under the singleton assignment s in some suitable model N, but the languages denoted by t and t' are different for another assignment i in a (possibly different) model M. Then, without loss of generality, we can assume there is a word $w \in t_i^M$ and $w \notin t_i'^M$. Since the equation holds under s in N, the terms t, t' must generate the same words over x when treated as regular expressions over the alphabet x. It follows by a straightforward induction on t that membership of a word in the language t_i^M is

witnessed by such a word over x (say w is witnessed by w_x). That is, the language t_i^M has the form:

$$\bigcup_{w_x \in t_s^N} i(w_x).$$

Membership of the word w in t_i^M is witnessed by $w \in i(w_x)$ for some word $w_x \in t_s^N$, but this is a contradiction because $w_x \in t_s'^N$ and thus $w \in i(w_x) \subseteq t_i'^M$. Thus, for equational axioms, the VC component can simply check that t and t' are equivalent as regular expressions over x.

The preceding observation reduces the soundness of equational axioms to the equivalence of regular expressions. We note that Kozen's complete axiomatization [Kozen 1994] involves two conditional equations. Building VC for conditional equations is more difficult, as it would likely require proofs by induction. We leave such automation to future work, but note that it does not fall outside the scope of the framework.

Instantiating Cex. It follows from above that the counterexample generator for false equations over language models can always provide as a counterexample a finite prefix of a canonical language model. For example, for the false equation $xy = x$, any language model over $\Sigma = \{a, b\}$ that has the languages $\{a\}$, $\{b\}$, and $\{ab\}$ witnesses that the equation is false. Of course, such a model must also contain the languages $\{aa\} = \{a\} \cdot \{a\}$, $\{bb\} = \{b\} \cdot \{b\}$, and many others (it must be closed under the operations). Intuitively, the counterexample models, though infinite, can be witnessed finitely. As discussed in Section 4, we formalize this with the concept of *pseudo-models*, and in this context *language pseudo-models*.

Definition 6.1 (Language pseudo-model)**.** A *language pseudo-model* over an alphabet Σ is a model over $\tau = (\cdot, +, *, 1, 0)$. Its domain D is finite and consists of Σ-languages, and each operation is a partial function on the domain. For every $x \in D^i$, each operation f of arity i is either undefined or else $f(x)$ is the language given by the standard interpretation from language theory.

The counterexample generator Cex produces language pseudo-models as counterexamples, and the Learner can propose any equation that is not false in the pseudo-models it has seen so far, with satisfaction defined as follows.

Definition 6.2 (Satisfaction in a language pseudo-model)**.** An equation $t = t'$ is *false* in a language pseudo-model M just when t_i^M and $t_i'^M$ are both defined and $t_i^M \neq t_i'^M$ for some variable assignment i. Otherwise, $t = t'$ is *true* in M (or *satisfied* by M), written as $M \models_p t = t'$.

For every language pseudo-model M and equation φ, we have:

$$\mathcal{M}(M) = \mathit{extensions}(M) \qquad \text{and} \qquad \varphi \in \mathcal{W}(M) \Leftrightarrow M \not\models_p \varphi \Leftrightarrow \forall M' \in \mathcal{M}(M),\ M' \not\models \varphi,$$

where $\mathit{extensions}(M)$ denotes all language models that contain the domain of M and that agree with the operations of M wherever they are defined. As an example, if the Learner proposes the false equation $xy = x$, then Cex may produce a pseudo-model of size 3 with domain

$$D = \{\{a\}, \{b\}, \{ab\}\}$$

and an interpretation of concatenation such that $\{a\} \cdot \{b\} = \{ab\}$ and all other operations on all other elements are undefined. Such a pseudo-model is enough to show the equation $xy = x$ is false using an assignment that maps x to $\{a\}$ and y to $\{b\}$. Note that this pseudo-model does not rule out, for example, the false equation $xx = x$, because xx is undefined for every x.

Instantiating VS. Unlike for modal logic, validity in the class $\mathcal{S}$ can be stated directly in first-order logic, and thus VS is instantiated as a semi-decision procedure for first-order validity. Given axioms $\varphi_1, \ldots, \varphi_n$, checking that a candidate axiom φ is independent from the axioms φ_i amounts to checking that $\psi \coloneqq \bigwedge_i \varphi_i \to \varphi$ is not valid, or equivalently, that $\neg\psi$ is satisfiable. Since satisfiability is hard to tackle directly, we instead choose to take *failure to prove dependence* as a proxy for

independence. We use VS to attempt a proof of ψ. If the proof fails we add the axiom φ to the growing set of axioms, and otherwise we discard it.

Completeness. It is known that the equational theory of language models has no complete *finite* axiomatization in terms of only equations [Redko 1964]. And as mentioned, Kozen's complete axiomatization [Kozen 1994] involves two conditional equations. The proof of completeness relies on the uniqueness of minimal deterministic finite automata for regular languages and involves algebraically encoding the determinization and minimization constructions for such automata. Automating the discovery of such a proof is very difficult and beyond the scope of this paper.

6.3 Implementation and Evaluation

We implemented this instantiation of the framework following the general algorithm in Algorithm 1. Using this algorithm, we obtain a synthesizer for sound equations over language models with the operations of concatenation, union, and Kleene star. The implementation can be found in our GitHub repository [Krogmeier et al. 2022a].

6.3.1 Implementation Details. The implementations of VC, VS, and Cex are based on the SMT solver Z3 [De Moura and Bjørner 2008]. We discuss their implementation details in this section.

Suppose we have already synthesized a set A of (universally-quantified) equations, and suppose that $t = t'$ is a candidate equation generated by Learner. VC employs the decision procedure described in Section 6.2 to check the validity of $t = t'$ by checking the equivalence of two regular expressions representing t and t'. We check the equivalence of regular expressions by encoding an SMT query over the theory of strings and using Z3. If $t = t'$ is not valid, Cex generates a large enough pseudo-model as described in Definition 6.1. We pre-compute a finite portion of the canonical model of regular languages with Kleene star corresponding to small regular expressions. When an axiom cannot be proven valid we look up this pseudo-model for an instantiation that witnesses the non-validity of the candidate.

For VS, we use a procedure called natural proofs [Löding et al. 2018] to check the entailment $A \models t = t'$. Natural proofs are semi-decision procedures for checking the validity of first-order formulae using systematic quantifier instantiation. VS instantiates A with ground terms up to a certain height and generates an SMT query whose unsatisfiability would imply $A \models t = t'$. This query is a quantifier-free formula over the theory of uninterpreted functions, and hence the SMT solver is guaranteed to terminate. If the query is satisfiable, it may still be the case that $A \models t = t'$, but we treat it as if $t = t'$ is independent from A to avoid missing axioms.

We optimized our algorithm for this domain in our implementation by merging the SMT query by VS and the SMT query by Learner (line 4 of Algorithm 1). This optimized version is equivalent to the original algorithm since the core algorithm only proceeds with a candidate equation if both queries are satisfiable.

6.3.2 Evaluation Results. Recall that our signature is $\tau = (\cdot, +, *, 1, 0)$. We ran three passes of the algorithm to synthesize equations with increasingly larger term grammars:

(1) τ-terms of height 1 with 2 free variables,
(2) τ-terms of height 2 with 2 free variables, and
(3) τ'-terms of height 2 with 3 free variables, where τ' is τ without the constants 0 and 1.

The set of axioms found in each pass is first pruned for redundant axioms that may be entailed by others in the set. Recall that our independence check using VS only ensures that axioms that are proposed later are not entailed by those that were proposed earlier; the converse may not be true. This additional pruning is done using the first-order theorem prover Vampire [Kovács and Voronkov 2013]. We treat the symbols in the signature as uninterpreted functions and ask whether

Table 3. Synthesis statistics for language models.

	# of Axioms		Time (seconds)			
Pass	**New**	**Pruned**	**Synthesis**	**Pruning**	Cex	**Total**
1	12	3	0.6	0.6	0.4	1.6
2	25	14	136.4	88.4	227.4	452.2
3	12	17	1821.5	70.0	760.2	2651.7

any axioms in the set are entailed by the others. The final set of axioms after pruning in each pass is used as an initial set of axioms in subsequent passes. Note that in this evaluation we run the core algorithm (Algorithm 1) multiple times, i.e., once for each pass.

Table 3 presents some statistics about our evaluation, including the number of axioms synthesized in each pass, the total time taken in each pass, and a breakdown of the time spent per component. The evaluation was performed on a machine with a 4-core (8-thread) Intel CPU i7-8550U and 16 GB of memory. Our tool synthesizes the following axioms (post pruning after all passes):

(1) $0 = 0b$
(2) $0 = b0$
(3) $0^* = 1$
(4) $0^* + b^* = (1 + b)^*$
(5) $00 + (a + b) = (b + a) + a$
(6) $(b^*)^* = b^*(b + 1)$
(7) $b^* + (1 + b) = b^*$
(8) $(a + a)a^* = a^*a$
(9) $(1a)(1 + b) = ab + a$
(10) $a + a = a$
(11) $(b + b)^* = b^*b^*$
(12) $b+(c+a) = (c+a)+(a+b)$
(13) $(ac)b = a(cb)$
(14) $(a + a)(b + c) = ac + ab$
(15) $cb + ab = (c + a)b$

The axioms for this class of models are expected to be similar to those of Kleene algebras from the literature, and we compared them to Kozen's axioms [Kozen 1994] (see also [Conway 1971]). The axioms synthesized by our tool are quite different from these reference axioms. As noted in Section 6.2, however, Kozen used *two* kinds of axioms: equations and conditional equations (axioms formulated as inequalities can be reformulated as equations). Handling soundness for conditional equations is more complex, and we did not tackle it in this work.

Kozen's axioms are *complete* for the equational theory of regular languages under concatenation, union, and Kleene star (all valid equations are semantically entailed by Kozen's axioms). Since our axioms are valid on the same class, Kozen's axioms imply our axioms by completeness.

On the other direction, it turns out that if we consider only the unconditional equational axioms in [Kozen 1994, Section 2, Axioms (3) - (15)], then our axioms are stronger. That is, our axioms imply all of the unconditional equational axioms in [Kozen 1994], but the converse is not true. We used Vampire [Kovács and Voronkov 2013] to automatically verify that our axioms (3), (4), (6), (8), and (11) are not implied by the unconditional equational axioms in [Kozen 1994], and Vampire was able to produce finite counterexample models. This result shows that our technique has the potential to discover new and useful axioms. The equational axioms discovered in our work may already have applications; there are several rewrite engines and solvers that use equational axioms to reason with regular expressions where this expanded set of axioms could be useful.

6.3.3 Comparison with Brute-force Enumeration. Similar to the experiments for modal logics, we tried to use a brute-force enumerative Learner, instead of a constraint-solving-based one that learns from counterexamples. The enumerative version took about 25 hours to exhaust the axiom search space, while our tool took only 50 minutes. The enumeration scanned through ~10 million equations with height-2 terms and 2 free variables (corresponding to pass 2), and ~1.3 million equations with height-2 terms and 3 free variables (corresponding to pass 3). Note that there are more equations in pass 2 than in pass 3 because pass 3 does not have constant terms 0 and 1 in the

signature. Note also that since no finite set of equational axioms can be complete, early termination on completeness is impossible in this setting. We again conclude that enumerative approaches are unlikely to scale for axiom synthesis.

7 RELATED WORK

The axiomatization of logics and classes of structures has a rich history in the mathematical literature. But the term *axiomatization* is used to describe many different kinds of problems. To our knowledge, ours is the first work to study automated axiomatization of classes of structures.

One class of problems rooted in the work of Leśniewski, Tarski, and Łukasiewicz [Łukasiewicz and Tarski 1930; Rezus 2020; Tarski 1938] is to find *simple* axioms for algebraic structures (e.g., groups) for which axiomatizations are already known. The objective is to find axiomatizations that are shorter (as short as a single axiom) or that use a different set of operators (e.g., a division operator for groups). Work by William McCune and contemporaries [Kunen 1992; McCune and Padmanabhan 1996; McCune et al. 2005, 2003; McCune and Sands 1996; McCune et al. 2002; Mccune 1993; Neumann 1981; Padmanabhan 1969] studies using computers to find simple axiomatizations for various algebraic structures.

The problem of automated theory discovery or theory exploration has also been studied [Buchberger et al. 2006; Drâmnesc et al. 2015; Drămnesc and Jebelean 2012; Johansson 2017; Johansson et al. 2014; Mccasland et al. 2017; Singher and Itzhaky 2021; Valbuena and Johansson 2015]. In theory exploration, one has a set of axioms A that defines a theory of interest, and the goal is to find formulae φ that belong to the theory of A, i.e., $A \models \varphi$, with a preference for finding interesting or complex theorems, e.g., discovering Sylow theorems given group axioms. The model-theoretic axiomatization problem can be thought of as a dual problem to theory exploration since it requires finding axioms given the theory, i.e., to find A, such that $A \models \varphi$ for every φ in the theory of some class of structures C.

Synthesizing logical formulae from example structures is a fundamental problem that has seen recent theoretical and practical progress. First-order logic formulae (with quantifiers) have been used to express invariants and prove correctness for complex distributed protocols [Hance et al. 2021; Koenig et al. 2020], and synthesis algorithms have been proposed based on either constraint solving or tree automata emptiness procedures [Koenig et al. 2020; Krogmeier and Madhusudan 2022]. The Learner component of the LAS framework would benefit from improvements to algorithms for learning logical formulae.

Also related is the field of program synthesis, especially the Programming By Example (PBE) paradigm [Gulwani 2011; Polozov and Gulwani 2015]. Many different techniques [Reynolds et al. 2019; Solar Lezama 2008] proposed in the program synthesis literature have proven useful for other kinds of synthesis problems. Common formats and frameworks for synthesis, like SyGuS [Alur et al. 2013; Reynolds et al. 2019], and more recently the SemGuS framework [D'Antoni et al. 2021; Kim et al. 2021], have spurred advances in algorithms for synthesizing expressions over logical specifications, which LAS can benefit from.

SMT solvers [Barrett et al. 2011; De Moura and Bjørner 2008] provide automated reasoning for many first-order logics and we use them in our implementation. Improvements in SMT solving can improve our existing implementation and also inform the design of components in the LAS framework for new domains. Using first-order theorem provers [Kovács and Voronkov 2013] in implementing LAS is also an interesting direction to be explored.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we develop (1) a model-theoretic formulation of axiom synthesis, (2) the LAS framework for solving the axiom synthesis problem using computational reasoning, counterexample

generation, and synthesis components, and (3) an instantiation of the framework and implementation showing its effectiveness for discovering axioms in the realms of modal logic and the class of language models. We believe our work opens a new research direction for computers to generate axiomatizations that were hitherto found by humans. Our framework is, of course, constrained by the current state-of-the-art in automated reasoning and expression synthesis. However, it will grow more effective as techniques for these areas evolve and improve in the future.

We believe the LAS framework has many interesting applications. A key future direction is to explore applications of axiom synthesis that serve a purpose beyond the current state-of-the-art. One possibility would be to explore axiom synthesis in a domain that has not been axiomatized by humans so far. The second possibility would be to use LAS to alleviate the burden of finding axioms in situations where the problem is tedious and perhaps not interesting to a human. For example, the semantics of CPU instruction sets can be seen as a logic with intricate semantics that can change frequently (see [Heule et al. 2016] for work on synthesizing semantics of such instruction sets). We can find equational axioms to express one set of instructions in terms of another, which can help in finding constant-time implementations that avoid timing attacks (e.g. [Dinesh et al. 2022]). As another example, we can learn axioms that aid in downstream verification tasks, e.g., one of our anonymous reviewers pointed out recent work on proving the equivalence of database programs that involved a manual axiomatization of the theory of relational algebra with updates [Wang et al. 2018]. In this domain, one can imagine using a reference implementation as the soundness checker VC and using a weaker logic such as the theory of uninterpreted functions to build VS. However, counterexample generation and, more importantly, the right grammar for synthesizing axioms is unclear. Yet another application is to learn axioms to aid in metamorphic testing, where program modules are tested against sequences or compositions of function calls (e.g. $f(x, y) = f(y, x)$ may be known to be true, even though the functional specification of f is hard to write). The axioms in this case (e.g. equational specifications) can be used to test other implementations (the work in [Smith et al. 2017] explores similar ideas). In these examples, the class C consists of a single structure corresponding to a reference implementation for the operations, and VC can find proofs of soundness against the reference implementation (or simply resort to testing).

These examples would use LAS as a means to an end, e.g. synthesis, verification, or testing. Another class of applications would use LAS in settings where axiomatization is an end goal in itself. For instance, we could try to discover properties or laws about a complex system by manipulating it and observing how it behaves, similar to work on the Bacon system that re-discovered simple laws of physics from observed data [Langley 1981], or more recent work on predicting the motion of objects in complex artificial environments [Wu and Tegmark 2019]. In such settings, the target class can be thought of as a single structure. But since the system may not be fully known or analyzable we must resort to testing rather than proofs of soundness. Applying the technique of this paper to discover interpretable axioms in such domains would also be interesting.

DATA AVAILABILITY STATEMENT

We have prepared a publicly available Docker image [Krogmeier et al. 2022b] for reproducing our evaluations in Sections 5.3 and 6.3.

ACKNOWLEDGMENTS

The last author dedicates this paper as a Festschrift article in honor of the academic career of Prof. R. Ramanujam (“Jam”) at the Institute of Mathematical Sciences, Chennai, India, who was a mentor and also introduced him to correspondence theory for modal logics. This work is supported in part by a research grant from Amazon, a Discovery Partners Institute (DPI) science team seed grant, a Google Faculty Fellowship, and NSF Grant No. 1801369.

REFERENCES

Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2013. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design*. IEEE, Portland, OR, USA, 1–8. https://doi.org/10.1109/FMCAD.2013.6679385

Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. 2014. NetKAT: Semantic Foundations for Networks. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '14)*. Association for Computing Machinery, New York, NY, USA, 113–126. https://doi.org/10.1145/2535838.2535862

Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification* (Snowbird, UT) *(CAV'11)*. Springer-Verlag, Berlin, Heidelberg, 171–177. https://doi.org/10.1007/978-3-642-22110-1_14

Patrick Blackburn, Johan F. A. K. van Benthem, and Frank Wolter. 2006. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., USA.

Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter (Eds.). 2007. *Handbook of Modal Logic*. Studies in logic and practical reasoning, Vol. 3. Elsevier, Amsterdam, Netherlands. https://doi.org/10.1016/S1570-2464(07)80002-4

Bruno Buchberger, Adrian Crăciun, Tudor Jebelean, Laura Kovács, Temur Kutsia, Koji Nakagawa, Florina Piroi, Nikolaj Popov, Judit Robu, Markus Rosenkranz, and Wolfgang Windsteiger. 2006. Theorema: Towards computer-aided mathematical theory exploration. *Journal of Applied Logic* 4, 4 (2006), 470–504. https://doi.org/10.1016/j.jal.2005.10.006

Edmund M. Clarke and E. Allen Emerson. 1982. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs*, Dexter Kozen (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 52–71.

John H. Conway. 1971. *Regular algebra and finite machines*. Chapman and Hall, London, UK.

Loris D'Antoni, Qinheping Hu, Jinwoo Kim, and Thomas Reps. 2021. Programmable Program Synthesis. In *Computer Aided Verification*, Alexandra Silva and K. Rustan M. Leino (Eds.). Springer International Publishing, Cham, 84–109. https://doi.org/10.1007/978-3-030-81685-8_4

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08)*, Vol. 4963. Springer, Budapest, Hungary, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

Sushant Dinesh, Grant Garrett-Grossman, and Christopher W. Fletcher. 2022. SynthCT: Towards Portable Constant-Time Code. In *29th Annual Network and Distributed System Security Symposium (NDSS '22)*. The Internet Society, San Diego, CA, USA, 1–18. https://doi.org/10.14722/ndss.2022.24215

Isabela Drâmnesc, Tudor Jebelean, and Sorin Stratulat. 2015. Theory exploration of binary trees. In *2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY)*. IEEE, Subotica, Serbia, 139–144. https://doi.org/10.1109/SISY.2015.7325367

Isabela Drămnesc and Tudor Jebelean. 2012. Theory Exploration in Theorema: Case Study on Lists. In *2012 7th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE, Timisoara, Romania, 421–426. https://doi.org/10.1109/SACI.2012.6250041

Jay Loren Gischer. 1985. *Partial Orders and the Axiomatic Theory of Shuffle (Pomsets)*. Ph. D. Dissertation. Stanford University, Stanford, CA, USA. AAI8506191.

Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-Output Examples. *SIGPLAN Not.* 46, 1 (jan 2011), 317–330. https://doi.org/10.1145/1925844.1926423

Travis Hance, Marijn Heule, Ruben Martins, and Bryan Parno. 2021. Finding Invariants of Distributed Systems: It's a Small (Enough) World After All. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*. USENIX Association, Boston, MA, USA, 115–131. https://www.usenix.org/conference/nsdi21/presentation/hance

Stefan Heule, Eric Schkufza, Rahul Sharma, and Alex Aiken. 2016. Stratified Synthesis: Automatically Learning the X86-64 Instruction Set. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Santa Barbara, CA, USA) *(PLDI '16)*. Association for Computing Machinery, New York, NY, USA, 237–250. https://doi.org/10.1145/2908080.2908121

Wilfrid Hodges. 1993. *Model Theory*. Cambridge University Press, Cambridge, UK. https://doi.org/10.1017/CBO9780511551574

Wilfrid Hodges. 1997. *A Shorter Model Theory*. Cambridge University Press, USA.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA.

Moa Johansson. 2017. Automated Theory Exploration for Interactive Theorem Proving:. In *Interactive Theorem Proving*, Mauricio Ayala-Rincón and César A. Muñoz (Eds.). Springer International Publishing, Cham, 1–11.

Moa Johansson, Dan Rosén, Nicholas Smallbone, and Koen Claessen. 2014. Hipster: Integrating Theory Exploration in a Proof Assistant. In *Intelligent Computer Mathematics*, Stephen M. Watt, James H. Davenport, Alan P. Sexton, Petr Sojka, and Josef Urban (Eds.). Springer International Publishing, Cham, 108–122. https://doi.org/10.1007/978-3-319-08434-3_9

Jinwoo Kim, Qinheping Hu, Loris D'Antoni, and Thomas Reps. 2021. Semantics-Guided Synthesis. *Proc. ACM Program. Lang.* 5, POPL, Article 30 (jan 2021), 32 pages. https://doi.org/10.1145/3434311

S. C. Kleene. 1956. *Representation of Events in Nerve Nets and Finite Automata.* Princeton University Press, Princeton, NJ, USA, 3–42.

Jason R. Koenig, Oded Padon, Neil Immerman, and Alex Aiken. 2020. First-Order Quantified Separators. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI '20)*. Association for Computing Machinery, New York, NY, USA, 703–717. https://doi.org/10.1145/3385412.3386018

Laura Kovács and Andrei Voronkov. 2013. First-Order Theorem Proving and Vampire. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–35.

D. Kozen. 1994. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Information and Computation* 110, 2 (1994), 366–390. https://doi.org/10.1006/inco.1994.1037

Paul Krogmeier, Zhengyao Lin, Adithya Murali, and P. Madhusudan. 2022a. LAS Framework Implementation. https://github.com/zhengyao-lin/fol-synthesis/tree/oopsla22.

Paul Krogmeier, Zhengyao Lin, Adithya Murali, and P. Madhusudan. 2022b. Synthesizing Axiomatizations using Logic Learning. https://doi.org/10.5281/zenodo.7072506

Paul Krogmeier and P. Madhusudan. 2022. Learning Formulas in Finite Variable Logics. *Proc. ACM Program. Lang.* 6, POPL, Article 10 (jan 2022), 28 pages. https://doi.org/10.1145/3498671

Kenneth Kunen. 1992. Single axioms for groups. *Journal of Automated Reasoning* 9, 3 (01 Dec 1992), 291–308. https://doi.org/10.1007/BF00245293

Pat Langley. 1981. Data-driven discovery of physical laws. *Cognitive Science* 5, 1 (1981), 31–54. https://doi.org/10.1016/S0364-0213(81)80025-0

F. William Lawvere. 1969. Adjointness in Foundations. *Dialectica* 23, 3/4 (1969), 281–296. http://www.jstor.org/stable/42969800

Christof Löding, P. Madhusudan, and Daniel Neider. 2016. Abstract Learning Frameworks for Synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems*, Marsha Chechik and Jean-François Raskin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 167–185. https://doi.org/10.1007/978-3-662-49674-9_10

Christof Löding, P. Madhusudan, and Lucas Peña. 2018. Foundations for natural proofs and quantifier instantiation. *PACMPL* 2, POPL (2018), 10:1–10:30. https://doi.org/10.1145/3158098

J. Łukasiewicz and A. Tarski. 1930. *Untersuchungen über den Aussagenkalkül.* Vol. 23. Comptes Rendus des Séances de la Sociélé des Scierices et des Lettres des Varsovie Classe III, Warsaw, Poland. 30–50 pages.

R. L. Mccasland, A. Bundy, and P. F. Smith. 2017. MATHsAiD: Automated Mathematical Theory Exploration. *Applied Intelligence* 47, 3 (oct 2017), 585–606. https://doi.org/10.1007/s10489-017-0954-8

W. McCune and R. Padmanabhan. 1996. Single identities for lattice theory and for weakly associative lattices. *Algebra Universalis* 36 (12 1996), 436–449. https://doi.org/10.1007/BF01233914

W. McCune, R. Padmanabhan, M. A. Rose, and R. Veroff. 2005. Automated discovery of single axioms for ortholattices. *algebra universalis* 52, 4 (01 Feb 2005), 541–549. https://doi.org/10.1007/s00012-004-1902-0

William McCune, R. Padmanabhan, and Robert Veroff. 2003. Yet another single law for lattices. *algebra universalis* 50, 2 (01 Dec 2003), 165–169. https://doi.org/10.1007/s00012-003-1832-2

W. McCune and A. D. Sands. 1996. Computer and Human Reasoning: Single Implicative Axioms for Groups and for Abelian Groups. *The American Mathematical Monthly* 103, 10 (1996), 888–892. http://www.jstor.org/stable/2974613

William McCune, Robert Veroff, Branden Fitelson, Kenneth Harris, Andrew Feist, and Larry Wos. 2002. Short Single Axioms for Boolean Algebra. *Journal of Automated Reasoning* 29, 1 (01 Mar 2002), 1–16. https://doi.org/10.1023/A:1020542009983

William W. Mccune. 1993. Single axioms for groups and Abelian groups with various operations. *Journal of Automated Reasoning* 10 (1993), 1–13. https://doi.org/10.1007/BF00881862

B.H. Neumann. 1981. Another single law for groups. *Bulletin of the Australian Mathematical Society* 23, 1 (1981), 81–102. https://doi.org/10.1017/S0004972700006912

Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. 2001. Local Reasoning about Programs That Alter Data Structures. In *Proceedings of the 15th International Workshop on Computer Science Logic (CSL '01)*. Springer-Verlag, Berlin, Heidelberg, 1–19. https://dl.acm.org/doi/10.5555/647851.737404

R Padmanabhan. 1969. On Single Equational-Axiom Systems for Abelian Groups. *Journal of the Australian Mathematical Society* 9, 1-2 (1969), 143–152. https://doi.org/10.1017/S144678870000570X

Amir Pnueli. 1977. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS '77)*. IEEE Computer Society, USA, 46–57. https://doi.org/10.1109/SFCS.1977.32

Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: A Framework for Inductive Program Synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Pittsburgh, PA, USA) *(OOPSLA 2015)*. Association for Computing Machinery, New York, NY, USA, 107–126. https://doi.org/10.1145/2814270.2814310

V.N. Redko. 1964. On defining relations for the algebra of regular events. *Ukrainian Mathematical Journal* 16, 2 (1964), 120–126. In Russian.

Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Cesare Tinelli, and Clark Barrett. 2019. CVC4SY: Smart and Fast Term Enumeration for Syntax-Guided Synthesis. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV '19) (Lecture Notes in Computer Science, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 74–83. https://doi.org/10.1007/978-3-030-25543-5_5

Adrian Rezus. 2020. *Witness Theory: Notes on λ-calculus and Logic.* College Publications, London, UK.

Arto Salomaa. 1966. Two Complete Axiom Systems for the Algebra of Regular Events. *J. ACM* 13, 1 (1966), 158–169. https://doi.org/10.1145/321312.321326

Eytan Singher and Shachar Itzhaky. 2021. Theory Exploration Powered by Deductive Synthesis. In *Computer Aided Verification*, Alexandra Silva and K. Rustan M. Leino (Eds.). Springer International Publishing, Cham, 125–148. https://link.springer.com/chapter/10.1007/978-3-030-81688-9_6

Calvin Smith, Gabriel Ferns, and Aws Albarghouthi. 2017. Discovering Relational Specifications. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 616–626. https://doi.org/10.1145/3106237.3106279

Peter Smith. 2010. The Galois Connection between Syntax and Semantics. https://www.logicmatters.net/resources/pdfs/Galois.pdf

Armando Solar Lezama. 2008. *Program Synthesis By Sketching.* Ph. D. Dissertation. EECS Department, University of California, Berkeley. https://dl.acm.org/doi/10.5555/1714168

Alfred Tarski. 1938. Ein Beitrag zur Axiomatik der Abelschen Gruppen. *Fundamenta Mathematicae* 30, 11 (1938), 253–256.

Irene Lobo Valbuena and Moa Johansson. 2015. Conditional Lemma Discovery and Recursion Induction in Hipster. *Electronic Communications of the EASST* 72 (2015), 1–15. https://doi.org/10.14279/tuj.eceasst.72.1009

Johan Van Benthem. 1984. *Correspondence Theory.* Springer Netherlands, Dordrecht, 167–247. https://doi.org/10.1007/978-94-009-6259-0_4

Yuepeng Wang, Isil Dillig, Shuvendu K. Lahiri, and William R. Cook. 2018. Verifying Equivalence of Database-Driven Applications. *Proc. ACM Program. Lang.* 2, POPL, Article 56 (dec 2018), 29 pages. https://doi.org/10.1145/3158144

Tailin Wu and Max Tegmark. 2019. Toward an artificial intelligence physicist for unsupervised learning. *Physical review. E* 100 3-1 (2019), 033311.