EPiC Series in Computing

Volume 57, 2018, Pages 454–470

LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Towards Smarter MACE-style Model Finders

Mikoláš Janota[1,2] and Martin Suda[2]

[1] IST/INESC-ID,
University of Lisbon, Portugal
mikolas.janota@gmail.com
[2] Czech Technical University in Prague, Czech Republic,
martin.suda@cvut.cz

Abstract

Finite model finders represent a powerful tool for deciding problems with the finite model property, such as the Bernays-Schönfinkel fragment (EPR). Further, finite model finders provide useful information for counter-satisfiable conjectures. The paper investigates several novel techniques in a finite model-finder based on the translation to SAT, referred to as the MACE-style approach. The approach we propose is driven by counterexample abstraction refinement (CEGAR), which has proven to be a powerful tool in the context of quantifiers in satisfiability modulo theories (SMT) and quantified Boolean formulas (QBF).

One weakness of CEGAR-based approaches is that certain amount of luck is required in order to guess the right model, because the solver always operates on incomplete information about the formula. To tackle this issue, we propose to enhance the model finder with a machine learning algorithm to improve the likelihood that the right model is encountered. The implemented prototype based on the presented ideas shows highly promising results.

1 Introduction

Finite model finding plays an important role in a number of areas of automated reasoning. Users are often interested in models rather than in proving a theorem [19]. But even if the objective is to prove a theorem, models serve as counterexamples in the case of incorrect assumptions, which is equally important in mathematics [3] just as in other areas, such as software verification. Some high-level automated theorem proving systems use validity of formulas in many finite models as a semantic feature for *lemma selection learning* [33].

In certain fragments of first-order logic, finite model finding provides a complete decision procedure. A prominent example being the *Bernays-Schönfinkel* fragment (EPR), which itself is a generalization of other well-known fragments such as *quantified Boolean formulas (QBF)* or *Dependency QBFs (DQBF)*, cf. [21].

The state-of-the-art finite model finding enjoys a number of different paradigms. The MACE/Paradox-style translates the whole first-order logic formula into a propositional one for each considered size of the universe [6, 19]. Universal variables are eliminated by expanding

each universal quantifier into conjuncts, i.e. by exhaustive grounding. Functions are represented by the introduction of a growing number of Boolean variables corresponding to the relation defining the function.

Constraint satisfaction-based model finders, e.g. the SEM model finder [35], rely on dedicated symmetry and propagation. In the context of SMT, Reynolds et al. do not translate to SAT, but instantiate quantifiers lazily and rely on the *theory of equality with uninterpreted functions (EUF)* [27, 26]. In order to ensure that the resulting model has the desired cardinality, a model returned by EUF is shrunk by "gluing" together the calculated equivalence classes. Vakili and Day also rely on EUF but add additional constraints to achieve the right cardinality [34]. Similar constraints are used by Baumgartner et al. in function-free clause logic implemented in the Darwin system [2].

The MACE-style SAT-based approach has one clear advantage over the aforementioned ones: the cardinality of the universe does not need to be enforced by special constraints or methods. Indeed, the cardinality is implicitly captured by the encoding since SAT solvers work in a finite domain. However, the MACE-style approach blows up in space for large number of universal variables. We aim to circumvent this issue by instantiating variables *lazily* (similarly as in SMT). However, this brings about a different issue. A candidate model is always calculated from partial information and therefore the model finder needs to be lucky to hit the right one. We aim to tackle this by the introduction of *machine learning* into the model finder. Hence, the two main contributions of the paper can be summarized as follows.

1. A method for finite model finding with **CEGAR-based quantifier instantiation** anchored in **translation to SAT**.
2. An enhancement of the model finder with **machine learning** with the aim of more informed calculation of the candidate models.

The rest of the paper is organized as follows. Section 2 introduces concepts and notations used throughout the paper. Section 3 describes the model-finding algorithm. Section 5 reports on experimental results, and finally, Section 6 concludes and outlines future work.

2 Preliminaries

Standard notation and concepts from first-order logic are assumed. Throughout the paper we assume a fixed *signature* Σ consisting of a set of *function* and *predicate* symbols. Each function and predicate symbol f is assigned a unique *arity*, which is a non-negative integer. Function symbols with 0-arity are called *constants*. A function or predicate with the arity n is called n-ary.

The binary predicate symbol $\simeq$ represents equality. *Atoms*, *terms*, *literals*, and *clauses* are defined as usual. The letters $s, t, \ldots$ are used to denote terms; the letters F, α, ϕ, ψ to denote formulas; variables are denoted as $x, x_1, \ldots$ and a vector of variables as $\vec{x}$. Predicate symbols are denoted as $p, p_1, \ldots$ and function symbols are denoted as $f, f_1, \ldots$.

A formula with no quantifiers and no variables is called *ground*. A variable x in a formula $(\forall x)\phi$ is called *bound* or *universally quantified*. A variable that is not bound is called *free*. A formula is in *prenex form* if it is in the form $(Q_1 x_1 \ldots Q_n x_n)(\phi)$ where $Q_i \in \{\forall, \exists\}$ and ϕ does not contain any quantifiers. We assume that existential quantifiers are eliminated by *Skolemization*. A formula with no free variables is called *closed*.

A Σ-*structure* M consists of a non-empty *universe* U and an interpretation $\mathcal{I}$ for variables and symbols in Σ. For an n-ary function f, the interpretation of f is denoted as $\mathcal{I}(f)$ and it is

Algorithm 1: Finite model finding, general structure (possibly non-terminating)

input : First-order logic formula F
output: A model of F or $\bot$, if F proven unsatisfiable

```
1 for k ∈ 1..UniverseLimit(F) do
2 |   I ← FindModel_k(F)
3 |   if I ≠ ⊥ then
4 |   |   return I                                   // model found
5 return ⊥                                          // unsatisfiable formula
```

a total function from U^n to U. For an n-ary predicate p, the interpretation of p is denoted as $\mathcal{I}(p)$ and it is a subset of U^n. For a variable x, $\mathcal{I}(x)$ is an element of U. *Satisfiability* is defined as usual. A structure is called a *model* of a formula F if it satisfies F. A model is called *finite* if its universe is finite.

Note that since we are primarily interested in satisfiability, free variables and constants can be treated interchangeably for all practical purposes.

The *Bernays-Schönfinkel* fragment of first-order logic consists in the formulas $(\exists\vec{y})(\forall\vec{x})(\phi)$, where ϕ does not contain function symbols or any further quantification. The class is well-known to be decidable. In fact, its Herbrand universe is finite and it is therefore often referred to as *effectively propositional logic (EPR)*. Note that a formula F in EPR has a model if and only if it has a finite model. We remark that the complexity class of EPR is higher than SAT, in fact it is NEXP-TIME complete [18] and that it remains decidable with equality as well [21].

2.1 Machine Learning

Some basic notions from *Machine Learning* are needed. In general, by machine learning we understand the automated detection of meaningful patterns in data [29]. In particular, we are interested in the *classification* task where each vector determining values of the features is assigned a category. From a mathematical point of view, the objective is to devise a function that is given values of the features as arguments and returns a category.

Consider a set of features $\mathcal{F}_1, \ldots, \mathcal{F}_n$ with respective domains $D_1, \ldots, D_n$ and a domain of the classification category D. We define the *machine learning problem* by a *training sample*, which is a set of labeled tuples $(v_1, \ldots, v_n) \mapsto v$, where $v_i \in D_i$ are values of the individual features and v is a value from the category domain D. The output of a machine learning problem is a total function f from $D_1 \times \cdots \times D_n$ to D. The function f should be close to the labeling given by the training sample and typically is represented in some specific space. In particular, we will be using *decision trees* [23]. For further details, we refer the reader to standard literature [28, 20, 29].

3 Algorithm

The goal is to solve a closed formula F of the form $(\forall\vec{x})\phi$, where ϕ is quantifier free. Algorithm 1 outlines the general structure of the algorithm. It iterates the universe size k from 1 to a limit dependent on the given formula. Whenever a model is found for some k, the model is returned and the formula is thus shown satisfiable. If the formula F is in EPR, `UniverseLimit`(F) is

Algorithm 2: $\texttt{FindModel}_k$ with SAT-based CEGAR

input : $F = (\forall \vec{x})\phi$, $k \in \mathbb{N}^+$
output: Model $\mathcal{I}$ of $(\forall \vec{x})\phi$ of size k, if such exists, $\bot$ otherwise

```
1  α ← true
2  while true do
3      τ ← solve_k(α)                          // calculate model of α
4      if τ = ⊥ then
5          return ⊥                            // F has no model size k
6      I ← complete(τ)                         // candidate model of size k
7      μ ← solve_k(¬φ[I])                      // calculate counterexample to I
8      if μ = ⊥ then
9          return I                            // I is a model of F
10     α ← α ∧ φ[μ]                            // strengthen α
```

set as the number of constants appearing in F.[1] If F is ground, $\texttt{UniverseLimit}(F)$ is set as the number of terms in F. It is chosen as $+\infty$ in all other cases. If no model is found within the universe limit $\texttt{UniverseLimit}(F)$, the formula F is proven unsatisfiable. In the general case, however, when $\texttt{UniverseLimit}(F) = +\infty$, the algorithm may not terminate, which is inevitable due to the undecidability of the problem. Section 3.4 proposes a technique that enables stopping before reaching the limit.

One could consider a different strategy for exploring the values for k. However, if a small model exists, it is likely to be easier to calculate than a larger one. Note that in order to guarantee soundness, all the model sizes need to be considered, even in the case of EPR. For instance, for $F = (\forall x)(c_1 \simeq x \wedge c_2 \simeq x)$, we obtain the limit $\texttt{UniverseLimit}(F) = 2$ but the formula only has models of size 1.

In the MACE-like approach, as implemented in Paradox or Vampire's finite model finder, a model of a given size k is sought by translating the whole problem into a propositional representation and applying a SAT solver [19, 6, 24]. However, the translation to a propositional representation in general suffers from exponential explosion. Indeed, potentially, each universal variable gets expanded into k different constants. Hence, such translation is bound to explode for problems with large domains and large number of universal variables. Here we propose to generalize the MACE-style approach using a *lazy grounding*.

3.1 Finite Model Finding with CEGAR

The approach proposed here is anchored in *counterexample guided abstraction refinement (CEGAR)* and is summarized in Algorithm 2. For a given universe size k, Algorithm 2 runs a CEGAR loop. Throughout the course of the algorithm, the formula α represents a conjunction of some instantiations of the quantifier $(\forall \vec{x})$. The formula α is therefore always weaker than the given $F = (\forall \vec{x})\phi$. If α has no model of size k, the formula F also has no model of size k and the loop stops. If α has a model of size k, this model is used as a *candidate model* for F. Conceptually, this happens in two separate steps. Since α is ground, any model τ of α is only relevant on the ground terms that appear in α. Hence, τ needs to be *completed* into an interpretation $\mathcal{I}$ of F. In terms of the counterexample guided abstraction refinement framework, the

[1] Additional preprocessing sometimes enables reducing this number; see Section 3.3.

formula α is an *abstraction* of the original formula F and the individual instantiations are the refinement steps. Let us now discuss the details of the individual components of the algorithm.

Algorithm 2 hinges on the procedure $\texttt{solve}_k(\psi)$, which calculates a model of ψ if given a satisfiable formula; it returns $\bot$ if ψ is unsatisfiable. The procedure $\texttt{solve}_k$ is implemented by converting to SAT by standard means. The details of the implementation are discussed later on (Section 4.2).

In the following presentation we assume that the procedure $\texttt{solve}_k$ accepts the following language, capturing ground formulas of first-order logic with some additional interpreted functions (readily translatable to SAT).

$$F ::= \; p(\text{term}^*) \mid F \vee F \mid F \wedge F \mid \neg F \mid \text{term} \simeq^k \text{term} \mid \text{term} \leq^k \text{term} \tag{1}$$

$$\text{term} ::= \star_i \mid f(\text{term}^*) \mid \max^k(\text{term}, \text{term}) \mid \text{term} +^k 1 \mid \mathsf{ITE}(F, \text{term}, \text{term}) \tag{2}$$

The interpreted constant $\star_i$ for $i \in 1..k$ is interpreted as the i-th element of the universe. Hence, we assume that the universe is always of the form $\{\star_1, \ldots \star_k\}$, which also gives a natural ordering of the elements. The binary operators $\simeq^k$ and $\leq^k$ enable comparing the elements of the universe. The operations $\max^k$ and ITE (*if-then-else*) have the expected semantics. The operation $t +^k 1$ gives the element of the universe that immediately follows t if $t \neq \star_k$; it is undefined otherwise. When calling $\texttt{solve}_k$, the predicate $\simeq$, is treated as $\simeq^k$.

Since $\texttt{solve}_k(\psi)$ is essentially just a wrapper around a SAT solver, it returns a model of ψ in the form of an assignment to the ground terms appearing in ψ. If ψ does not have a model, $\texttt{solve}_k$ returns $\bot$. The returned assignment τ assigns each term in α a value from $\{\star_1, \ldots \star_k\}$ and each atom to a value from $\{\mathit{true}, \mathit{false}\}$.

After obtaining a model τ of the abstraction α, a crucial step is to complete it into an interpretation of the original formula F. The function `complete` serves for this purpose. Since we need to be able to explicitly reason on the obtained interpretations, we assume that any model returned by `complete` is representable in the input language of $\texttt{solve}_k$. Hence, we treat an interpretation and its syntactic representation interchangeably. Conceptually, we may see an interpretation as an assignment from symbols to lambda functions in the language above. So for instance an interpretation of some binary predicate $\{(\star_0, \star_0), (\star_1, \star_1)\}$ corresponds to $\lambda xy.\, (x \simeq^k \star_0 \wedge y \simeq^k \star_0) \vee (x \simeq^k \star_1 \wedge y \simeq^k \star_1)$.

There are straightforward ways of calculating the completion of τ into $\mathcal{I}$. For an n-nary predicate $p(x_1, \ldots, x_n)$, collect into a set A_p^T all atoms with p and that are assigned to *true* by τ. Construct the following interpretation

$$\bigvee_{p(t_1,\ldots,t_n) \in A_p^T} \left(x_1 \simeq^k \tau(t_1) \wedge \cdots \wedge x_n \simeq^k \tau(t_n)\right) \tag{3}$$

Like so, all points of p that are not assigned by τ are assigned to false in $\mathcal{I}$. Alternatively, it is possible to consider to set p to false *only* in the points where τ sets it to false, i.e., assigning unassigned points to true. Functions can be completed in an analogous way by picking some default value, e.g. the value $\star_1$ (recall that the universe is always nonempty). Later on we discuss more informed ways of computing the completion (Section 3.2).

The representability of interpretations in the language of $\texttt{solve}_k$ is crucial for checking if the candidate interpretation $\mathcal{I}$ is a model of F. To model-check $\mathcal{I}$ for $(\forall \vec{x})\phi$, we look for an assignment to the variables $\vec{x}$ that would invalidate ϕ under $\mathcal{I}$. This is done by negating ϕ, replacing each occurrence of a symbol by its interpretation in $\mathcal{I}$, and calling $\texttt{solve}_k$ (line 7, Algorithm 2).

Example 1. *Consider* $F = (\forall x)(p(x) \leftrightarrow (x \simeq c) \wedge c \not\simeq d)$ *and universe size* $k = 2$*. A possible run of Algorithm 2 is as follows.*

1. *First the interpretation is chosen arbitrarily (p always true):*
   $\tau_1 = \{\}$, $\mathcal{I}_1 = \{c \mapsto \star_1, d \mapsto \star_2, p \mapsto \lambda x.\ true\}$.
2. *The element* $\star_2$ *represents a counterexample to* $\mathcal{I}_1$*:*
   $\mu_1 = \{x \mapsto \star_2\}$, $\alpha_1 = c \not\simeq d \wedge (p(\star_2) \leftrightarrow \star_2 \simeq c)$
3. *The second candidate interpretation sets p everywhere false:*
   $\tau_2 = \{p(\star_2) \mapsto false\}$, $\mathcal{I}_2 = \{c \mapsto \star_1, d \mapsto \star_2, p \mapsto \lambda x.\ false\}$
4. *Now* $\star_1$ *serves as a counterexample to* $\mathcal{I}_2$*:*
   $\mu_2 = \{x \mapsto \star_1\}$, $\alpha_2 = c \not\simeq d \wedge (p(\star_2) \leftrightarrow \star_2 \simeq c) \wedge (p(\star_1) \leftrightarrow \star_1 \simeq c)$
5. *Finally,* $\forall x$ *is fully grounded and the right interpretation is calculated:*
   $\tau_2 = \{\}$, $\mathcal{I}_2 = \{c \mapsto \star_1, d \mapsto \star_2, p \mapsto \lambda x.\, x \simeq \star_1\}$.

3.2 Learning Models

Completing a model of α into an interpretation of F is a source of nondeterminism of the CEGAR approach. In fact, the CEGAR approach seems to be going against itself in some sense. The premise is that the abstraction α is small, i.e. significantly smaller than the exponential worst-case. However, at the same time, the smaller the abstraction, the less information is obtained its models.

More specifically, for a universe of size k, a full truth table for an n-ary predicate contains k^n rows, giving thus 2^{k^n} possible interpretations. A small α will fill-in values of only few rows of this table. This effect is magnified by the existence of multiple predicates and functions in the formula. We propose to use machine learning in this context. The model of α serves as the training sample and a learning algorithm serves as the completion method.

Consider an n-ary predicate p and a model τ of α. Collect all atoms in α where p is the top-level predicate into a set A_p. Construct the following training sample:

$$\{(\tau(t_1), \ldots, \tau(t_n)) \mapsto \tau(p(t_1, \ldots, t_n)) \text{ for } p(t_1, \ldots, t_n) \in A_p\} \tag{4}$$

Recall that τ assigns to each term t_i a value from the universe $\{\star_1, \ldots, \star_k\}$ and it assigns each atom $p(t_1, \ldots, t_n)$ one of the values *true*, *false*. Hence, the training sample represents a classifier with two categories and where tuples from $\{\star_1, \ldots, \star_k\}^n$ are classified as either *true* or *false*. Since models of α must observe congruence on functions and predicates, the training sample never labels the same tuple with two different values.

In the next step, apply some machine learning algorithm on the training sample. Finally, encode the function obtained from learning into the language accepted by $\texttt{solve}_k$. The rationale for functions is analogous. To obtain a full completion of τ into an interpretation $\mathcal{I}$ of F, repeat this process for every predicate and function symbol of F separately.

3.2.1 Learning with Decision Trees

In the current implementation learning of models is realized as follows. For an n-ary predicate $p(\text{arg}_1, \ldots, \text{arg}_n)$ construct a *decision tree* [23] and consider the set of branches that lead to *true* and define the interpretation of the predicate as the disjunction over these branches.

For the concrete form of the decision tree, we consider two alternatives. In the first alternative, each node of the decision tree is labeled by the argument arg_i on which to decide and then the edges from this node to its children are labeled with all the possible values of arg_i. Hence, each node in the tree has exactly k children.

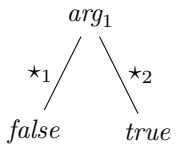
In the second alternative for a decision tree, each node is labeled by a predicate $(\text{arg}_i \leq \star_j)$ and has two children: one for inputs where $\text{arg}_i \leq \star_j$ and one for inputs where $\text{arg}_i \not\leq \star_j$. Note that arg_i may appear multiple times on a single branch in this alternative, whereas this is not the case in the first alternative. In both alternatives, the decision tree is learned by the standard algorithm *ID3* [23]. The following example demonstrates the process.

Example 2. *Let* $F = (\forall x_1, \ldots, x_n)(p(x_1, \ldots, x_n) \leftrightarrow (x_1 \simeq c))$ *and* $k = 2$. *Instantiate by* $(\star_1, \star_1, \ldots, \star_1)$ *and* $(\star_2, \star_1, \ldots, \star_1)$ *yielding the following abstraction* α

$$(p(\star_1, \star_1, \ldots, \star_1) \leftrightarrow \star_1 \simeq^k c) \wedge (p(\star_2, \star_1, \ldots, \star_1) \leftrightarrow \star_2 \simeq^k c).$$

Consider a model of α, $\tau = \{c \mapsto \star_2, p(\star_1, \star_1, \ldots, \star_1) \mapsto \mathit{false}, p(\star_2, \star_1, \ldots, \star_1) \mapsto \mathit{true}\}$. *Completing* p *by the straightforward disjunction operation (see (3)) only gives* p *as* $\{(\star_2, \star_1, \ldots, \star_1)\}$.

To use machine learning, we consider the features $arg_1, \ldots, arg_n$ *corresponding to the arguments of the predicate* p. *Interpreting the assignment* τ *as a training sample yields the following:* $\{(\star_1, \star_1, \ldots, \star_1) \mapsto \mathit{false}, (\star_2, \star_1, \ldots, \star_1) \mapsto \mathit{true}\}$. *Applying a learning algorithm on this training sample use the following simple decision tree.*



The decision tree identifies that the values of the predicate can be classified by looking only at the first argument. Semantically, the tree corresponds to the branches $a_1 \simeq^k \star_1$ *leading false and* $a_1 \simeq^k \star_2$ *leading true. Finally, we get the interpretation* $p = \lambda a_1, \ldots, a_n.(a_1 \simeq^k \star_2)$. *Alternatively, one can use the negative branches:* $p = \lambda a_1, \ldots, a_n.(a_1 \not\simeq^k \star_1)$.

3.3 Symmetry Breaking

For symmetry breaking we follow the approach proposed by Claessen and Sörensson (Paradox) [6, Sec. 6]. However, the realization of the symmetries needs to be adapted to our setting.

Consider all the constants of F in some arbitrary order $c_1, \ldots, c_n$. We wish to express that the constants are allocated values consecutively from $\star_1$, i.e., $c_1 = \star_1$ and $c_i = \star_j$ only if there exists $i' < i$ s.t. $c_{i'} = \star_{j-1}$. This is ensured by adding two types of constraints. First we express that c_i is assigned an element in the range $\star_1, \ldots, \star_{\min(i,k)}$ by adding the constraint $c_i \leq^k \star_{\min(i,k)}$. In particular, we get $c_1 \leq^k \star_1$ or equivalently $c_1 \simeq^k \star_1$ for $i = 1$. Second, we ensure that there are no "holes" in the assignments by adding the constraint $c_{i+1} \leq \max^k(c_1, .., c_i) +^k 1$.

In the case of an EPR formula, we need only at most as many elements of the universe as there are constants, i.e. n. Further, since the universe sizes are visited exhaustively (in an increasing order), it is sound and complete to restrict $\texttt{solve}_k$ to look for solutions where the constants use up all the elements of the universe. This is guaranteed by the additional constraint $\max^k(c_1, .., c_n) \simeq^k k$. A special case is when $n = k$, where we simply add the equalities $c_i \simeq^k \star_i$.

However, this constraint can be only applied in the case of EPR as in general first-order logic, further elements of the universe may be needed as values for non-constant terms, e.g. $(\forall x)(f(x) \not\simeq x)$ requires universe size 2.

Just as in Paradox, before applying the symmetries, we partition all the constants into equivalence classes so that constants from different classes "do not interact" [6, Sec. 7]. These equivalence classes can be seen as inferred sorts. The symmetries above are applied to these classes individually. The exception is the last symmetry, where we need to express that the universe is used up by all the constants across all the classes. Taking into account these equivalence classes also enables lowering the upper bound on the universe size that needs to be explored in order to guarantee completeness. For EPR the upper bound $\texttt{UniverseLimit}(F)$ is taken as the maximum of the sizes of these classes.

3.4 Stopping Early

The top-level loop in Algorithm 1 explores the sizes of the universe from 1 to the limit $\texttt{UniverseLimit}(F)$. In some cases, however, one may deduce that F is unsatisfiable earlier. We propose a simple stopping criterion. Once α is shown unsatisfiable for some k, count the number of terms occurring in it (these terms are guaranteed to be ground). If this number is less or equal to k, the top-level for-loop can stop, i.e. it is not necessary to explore larger universe sizes.

Example 3. *Let $F = f(c) \simeq c \wedge (\forall x)(f(x) \not\simeq x)$. Instantiate x with $x \mapsto \star_1$ and also consider the symmetry breaker $c \simeq \star_1$ yielding the following ground problem*

$$c \simeq \star_1 \wedge f(c) \simeq c \wedge f(\star_1) \not\simeq \star_1$$

The ground formula has the ground terms $\{c, \star_1, f(c), f(\star_1)\}$. Consequently, it is sufficient to stop for the universe size $k = 4$.

It is easy to observe that this stopping criterion is sound i.e., the criterion is applicable only if F is unsatisfiable. Indeed, since α is ground and it contains at most $m \leq k$ terms, it must be the case that if it has a model, it also has a model of size at most m. This means that if α does not have a model of size at most k, it is unsatisfiable. Since α is weaker than F, it must mean that F is also unsatisfiable.

3.5 Instantiation with Original Constants

Algorithm 2 always instantiates the vector of variables $\vec{x}$ with elements of the universe $\star_i$. Since we are concerned with finite models, this is clearly both sound and complete. However, it is not necessarily advantageous. For instance, in $\neg p(c) \wedge (\forall x)p(x)$ instantiating x with c rather than $\star_1$ gives a more appropriate strengthening. To achieve such instantiations we proceed as follows. For a considered candidate model $\mathcal{I}$, and a counterexample μ such that $\mu(x) = \star_i$ and $\mathcal{I}(c_j) = \star_i$ for some x, c_j, and i, we change μ to $\mu' = \mu[x \mapsto c_j]$. The operation is sound as these instantiations are in fact instantiations with members of the Herbrand universe.

Using original constants for instantiations combined with the early-stop criterion (Section 3.4) may lead to lowering the necessary universe size.

Example 4. *Instantiating $F = f(c) \simeq c \wedge (\forall x)(f(f(x)) \not\simeq x)$ with $x \mapsto c$ gives the ground problem $f(c) \simeq c \wedge f(f(c)) \not\simeq c$ with the ground terms $\{c, f(f(c)), f(c)\}$. This makes the universe size $k = 3$ sufficient to stop.*

We remark that a similar technique was used by Ge and de Moura in SMT to obtain complete instantiation for certain types of theories [12].

4 Implementation

4.1 Non-prenex Input

Algorithm 2 is easily adapted to operate on non-prenex input, similarly as in [14]. This is practically useful in clausal inputs where the input is of the form $(\forall \vec{x}_1)(C_1) \wedge \cdots \wedge (\forall \vec{x}_n)(C_n)$. In such case, the implementation calculates a candidate model $\mathcal{I}$ as before and tests it against all the individual conjuncts. Instantiation is then carried out based on the counterexamples that were obtained from those conjuncts that are not satisfied by $\mathcal{I}$. Hence, the model-checking routine, can be seen as a function returning a set of counterexamples rather than a single one. If this set is empty, it means that the candidate interpretation is indeed a model. This operation is characterized by the following equations.

$$\texttt{check}(\psi \wedge \phi) := \texttt{check}(\phi) \cup \texttt{check}(\psi)$$
$$\texttt{check}(\psi \vee \phi) := \textbf{let } S_1 = \texttt{check}(\psi),\ S_2 = \texttt{check}(\phi) \textbf{ in } \emptyset \textbf{ if } (S_1 = \emptyset \textbf{ or } S_2 = \emptyset) \textbf{ else } (S_1 \cup S_2)$$
$$\texttt{check}((\forall \vec{x})\psi) := \textbf{let } \tau = \texttt{solve}_k(\neg\psi[\mathcal{I}]) \textbf{ in } \{(\tau, \psi)\} \textbf{ if } (\tau \neq \bot) \textbf{ else } \emptyset$$

In the case of conjunctions it would be possible to return only the failing conjunct, but that appears to be overly conservative for typical FOL benchmarks.

4.2 Deciding Ground Formulas ($\texttt{solve}_k$)

To implement a solver for ground formulas we use one of the standard translations to SAT. Given a fixed size of the universe k, each ground term t is represented using the unary encoding by k fresh Boolean variables. The unary representation enables straightforward encoding of the interpreted predicates and functions $\star_i$, $\simeq^k$, $\leq^k$, $\max^k$, $+^k 1$.

Congruence on terms predicates is guaranteed by *Ackermann reduction* (also known as *Ackermannization*) [7, 1]. For each pair of terms s and t that appear in the formula and have the same top-level function symbol f of arity n with the corresponding arguments $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ gives the constraint:

$$\left(\bigwedge_{i \in 1..n} a_i \simeq^k b_i \right) \rightarrow (s \simeq^k t) \tag{5}$$

Congruence for predicates is ensured analogously by generating constraints for pairs of atoms with the same top-level predicate. Even though the Ackermann reduction is polynomial, it can produce large number of additional constraints. Hence, we add the Ackermann constraints (5) in a *lazy fashion*. Initially, the SAT solver is called on the formula with no Ackermann constraints generated. If the SAT solver returns a congruent satisfying assignment, it is returned. If it is not congruent, Ackermann constraints are added for those terms and atoms that break congruence and the process is repeated (c.f. [9, 7, 4]).

5 Experimental Evaluation

The evaluated prototype is implemented in *C++* and minisat 2.2 [10] is used as the backend solver. All experiments were run on the StarExec compute cluster [30] with the time limit 600 s. We refer to our prototype as QFM.

Solver	#Solved	#Solved SAT	#Solved UNSAT
iProver	1265	403	862
Z3	1139	338	801
vampire-fm	1072	363	709
QFM+CEGAR+learn	1025	340	685
QFM+CEGAR	1019	334	685
cvc-fm	1014	328	686
QFM+expand	933	340	593
cvc-epr	879	185	694

Table 1: Summary for EPR instances.

For the evaluation we have selected two sets of benchmarks both from the TPTP library [31]. The first group are the EPR problems (marked with the `*EPR` tags). The second group consists in all *satisfiable* and *counter-satisfiable* problems that are not contained in the first group.

The following state-of-the-art tools were included into the comparison. The automated FOL prover *iProver* [17]; the SMT solver *Z3* [8]; Vampire's finite model finder [24], implemented according to *Paradox* [6]; and the SMT solver *CVC4* [27, 26]. CVC4 was tested in a finite-model finding mode (switch `--finite-model-find`) [27, 26] and also on the EPR instances with EPR quantifier instantiation (switch `--quant-epr`).

We have experimented with a number of configurations of QFM, not all are reported on in detail. Symmetry breaking (Section 3.3) was helpful overall and therefore is always present. The technique of refining by original constants (Section 3.5) was helpful in the EPR instances, but harmful on the non-EPR instances. We report on the results of QFM in 3 main configurations:

- QFM+CEGAR, which corresponds to Algorithm 2 with straightforward model completion (see equation(3)),
- QFM+CEGAR+learning, which corresponds to Algorithm 2 where model completion is calculated by machine learning (Section 2.1),
- QFM+expand, which corresponds to simply expanding all universal variables, i.e., to calling $\texttt{solve}_k$ just once after exhaustive grounding.

5.1 EPR instances

Table 1 summarizes the results for all the solvers across the considered EPR instances. The solver iProver dominates this benchmarks set in all categories, i.e. in total numbers, satisfiable, and unsatisfiable instances solved. Interestingly, while Z3 places 2$^{\text{nd}}$ in terms of total number of instances solved, it solves less satisfiable instances than both Vampire-FM and QFM+CEGAR+learn, i.e. Z3's strength lies in the unsatisfiable EPR problems. CVC4-EPR and QFM+expand perform poorly in this benchmarks set.

Further insights can be gained from the cactus plot Figure 1. Recall that a cactus plot contains a point (x, t) if for the given solver and CPU time t there exists x instances that the solver solves within that time limit. For the sake of readability, in all the cactus plots we exclude instances that were solved by all the considered solvers within the time limit of 60 seconds. From the cactus plot we can see that QFM+CEGAR+learn starts to give longer times earlier than most of the solvers, but eventually it is able to overtake CVC4's finite model

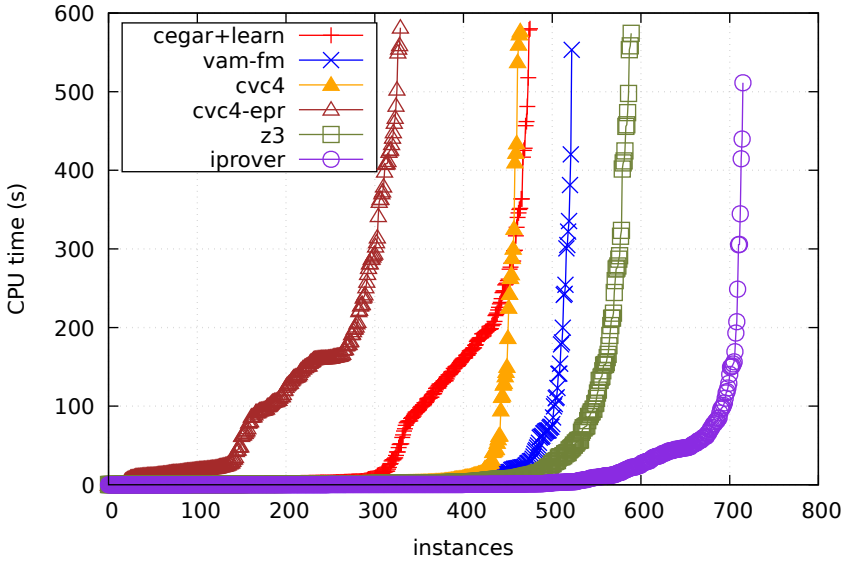


Figure 1: All solvers on EPR problems, with 739 not-trivial instances, with 549 trivial instances.

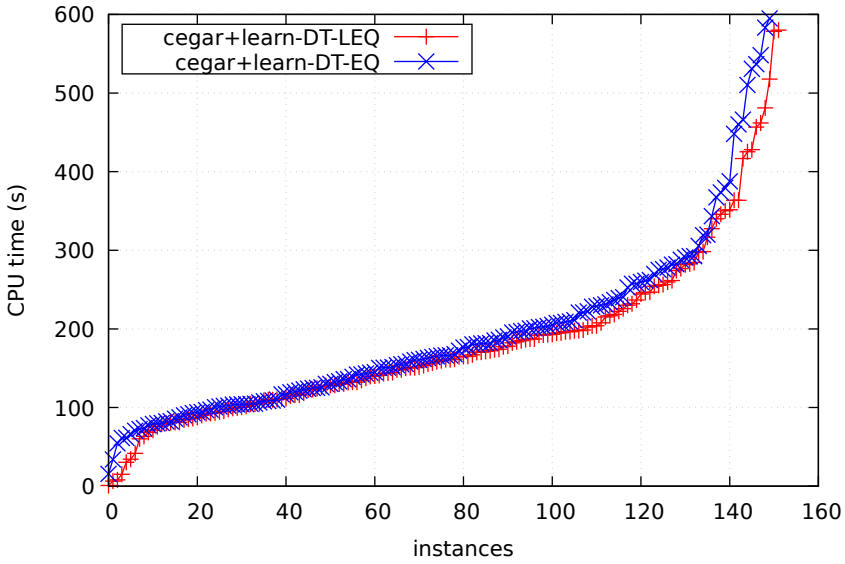


Figure 2: Comparing two alternatives of decision trees on EPR with 185 non-trivial and 873 trivial instances

finder. The slower performance of QFM could be attributed to the fact that it is a much less mature tool compared to the other ones.

Cactus plots Figure 2 and Figure 3 compare the two alternatives of decisions trees (see Section 3.2.1). Here the alternative relying on the splitting on the $\leq$ predicate is clearly favourable. While the difference is relatively small, it is remarkable that already a small improvement in

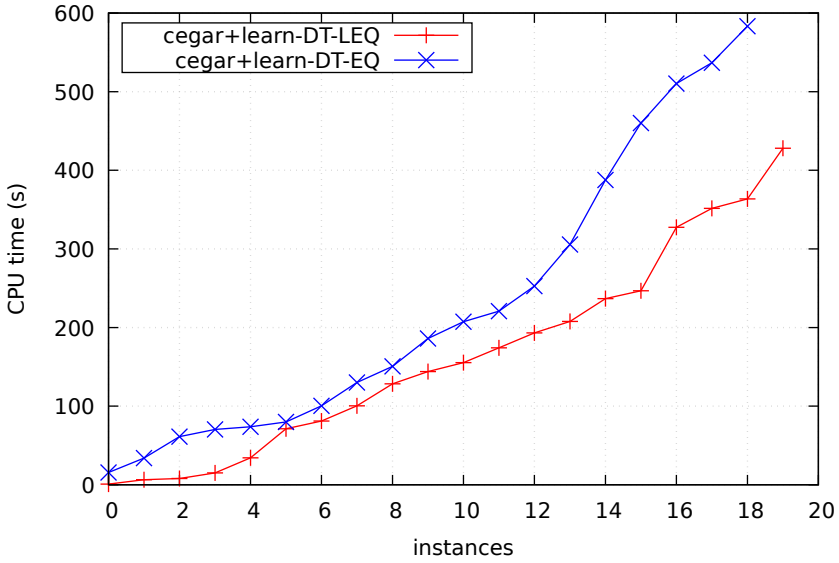


Figure 3: Comparing two alternatives of decision trees on EPR satisfiable with 33 non-trivial and 320 trivial instances

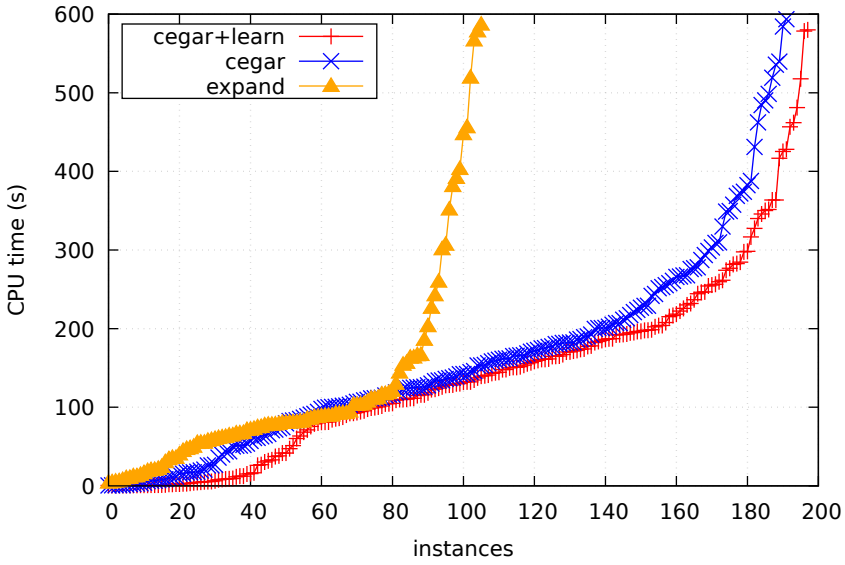


Figure 4: QFM on EPR problems, 231 non-trivial solved instances, with 827 trivial

the learning algorithm gives an improvement in the model finder.

Finally, the cactus plots Figure 4 and Figure 5 compare our prototype to itself. Overall, expanding fully universally quantified variables from the get-go gives the worst performance and learning is helpful. Interestingly, there is a big difference between satisfiable and unsatisfiable instances. CEGAR outperforms full expansion by a factor of two on all the instances combined,

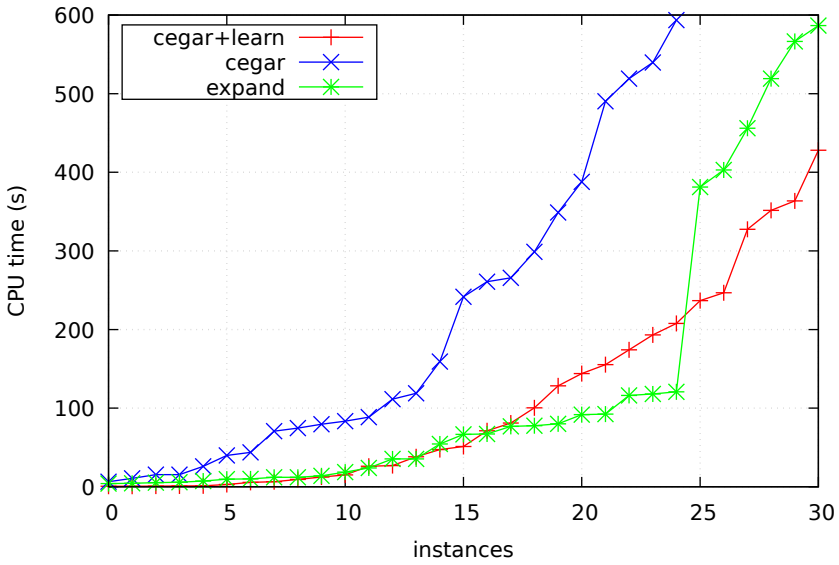


Figure 5: Configurations of QFM on EPR satisfiable instances, 44 non-trivial solved instances, with 309 trivial instances

Solver	**#Solved**
vampire-fm	1282
QFM+CEGAR	1133
iProver	1093
cvc-fm	1023
QFM+expand	985
QFM+CEGAR+learning	816
Z3	632

Table 2: Summary for non-EPR instances.

but if only satisfiable instances are taken into account, only CEGAR+learning outperforms the expansion approach.

5.2 Non-EPR instances

The overall results are summarized in Table 2 and in the cactus plot Figure 6. Interestingly, the results are quite different from the EPR ones. Here Vampire's finite model finder has the best performance while Z3 performs poorly on these instances. QFM+CEGAR clearly outperforms QFM with expansion. In fact, QFM+CEGAR places 2nd overall. However, learning for QFM on these instances is harmful. It is harmful to the extent that it performs more poorly than the full expansion approach. This suggests that the learning procedure we are using (Section 2.1) works well in predicates, but does not when it comes to learning functions with larger ranges.

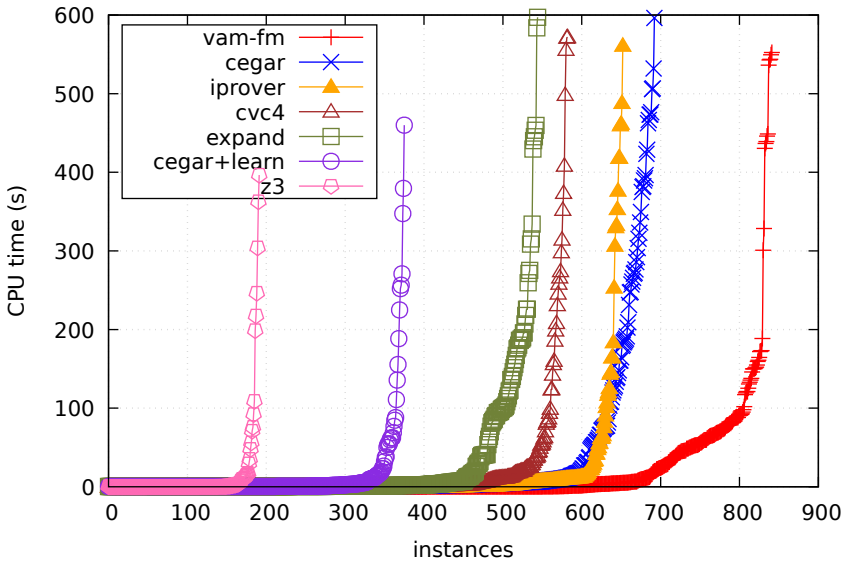


Figure 6: All solvers on non-EPR instances, with 880 non-trivial and 440 trivial instances.

6 Summary and Future Work

In this paper we present an approach for finite model finding that is anchored in a translation to SAT, as in MACE and Paradox, but instantiates quantifiers lazily by relying on counterexample guided abstraction refinement (CEGAR). Counterexample-driven instantiation has been used in a number of contexts in recent years, mainly in SMT [12, 25, 11, 22], but also in QBF [15, 13, 32]. In the context of certain theories, it has been shown that the procedure can be refutationally complete [12]. Here we postulate that CEGAR-based (finite) model finders suffer from a significant weakness in satisfiable problems. Since the abstraction of the formula contains only a small portion of the information about the formula, the algorithm requires a certain amount of luck to guess the right model. Purely statistically speaking, it is nearly impossible.

To tackle this issue, we propose to enhance the model finder with a machine learning algorithm whose purpose is to identify patterns that might emerge in the partial solution and that will eventually bring the solver closer to the desired model.

The initial experiments on our prototype indicate that the model finder benefits both from CEGAR and machine learning. However, the experiments also show several drawbacks in the current prototype.

While in many cases our prototype already outperforms state-of-the-art CEGAR-based SMT solvers (CVC4 and Z3), its overall performance is poorer than the original full-grounding Paradox-style approach, as implemented in Vampire. It is not obvious why that is the case as CEGAR outperforms our own implementation of exhaustive grounding. It is the subject of future work to investigate more carefully where the difference is coming from—it might simply be a more careful handling of the SAT solver or similar implementation-related issues.

In terms of the learning approach, the experiments indicate that learning is helpful for predicates, but harmful for functions. One explanation for this could be that the chosen form of the hypothesis space is inadequate for learning functions. A solution here might be to simply

apply learning only on predicates, but some other techniques can be envisioned. For instance, rather than learning a function by a single decision tree, use multiple ones. Or, use a different structure than a decision tree. In general, the learned functions could be built from a richer language than now. This opens a number of avenues for future work.

Last but not least, we plan to extend the current work for multi-sorted logic [5, 24] and incorporate more advanced preprocessing techniques, such as blocked-clause elimination [16].

Acknowledgments.

This work was supported by Portuguese national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013. The second author was supported by the ERC Consolidator grant AI4REASON 649043, ERC Starting grant SYMCAR 639270, and the Austrian research project FWF S11409-N23. The work was supported by the European Regional Development Fund under the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466).

References

[1] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[2] Peter Baumgartner, Alexander Fuchs, Hans de Nivelle, and Cesare Tinelli. Computing finite models by reduction to function-free clause logic. *Journal of Applied Logic*, 7(1):58 – 74, 2009. Special Issue: Empirically Successful Computerized Reasoning.

[3] Jasmin Christian Blanchette. Nitpick: A counterexample generator for Isabelle/HOL based on the relational model finder Kodkod. In Andrei Voronkov, Geoff Sutcliffe, Matthias Baaz, and Christian G. Fermüller, editors, *Short papers for 17th International Conference on Logic for Programming, Artificial intelligence, and Reasoning*, volume 13 of *EPiC Series in Computing*, pages 20–25. EasyChair, 2010.

[4] Robert Brummayer and Armin Biere. Lemmas on demand for the extensional theory of arrays. *JSAT*, 6(1-3):165–201, 2009.

[5] Koen Claessen, Ann Lillieström, and Nicholas Smallbone. Sort it out with monotonicity - translating between many-sorted and unsorted first-order logic. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction*, volume 6803 of *Lecture Notes in Computer Science*, pages 207–221. Springer, 2011.

[6] Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

[7] Leonardo Mendonça de Moura and Nikolaj Bjørner. Model-based theory combination. *Electr. Notes Theor. Comput. Sci.*, 198(2):37–49, 2008.

[8] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[9] Bruno Dutertre and Leonardo Mendonça de Moura. The Yices SMT Solver, 2006.

[10] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing (SAT)*, 2003.

[11] Azadeh Farzan and Zachary Kincaid. Strategy synthesis for linear arithmetic games. *Proc. ACM Program. Lang.*, 2(POPL):61:1–61:30, December 2017.

[12] Yeting Ge and Leonardo Mendonça de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2009.

[13] Mikoláš Janota, William Klieber, João Marques-Silva, and Edmund Clarke. Solving QBF with counterexample guided refinement. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 114–128, 2012.

[14] Mikoláš Janota, William Klieber, Joao Marques-Silva, and Edmund Clarke. Solving QBF with counterexample guided refinement. *Artificial Intelligence*, 234:1–25, 2016.

[15] Mikoláš Janota and Joao Marques-Silva. Abstraction-based algorithm for 2QBF. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 230–244, 2011.

[16] Benjamin Kiesl, Martin Suda, Martina Seidl, Hans Tompits, and Armin Biere. Blocked clauses in first-order logic. In Thomas Eiter and David Sands, editors, *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46 of *EPiC Series in Computing*, pages 31–48. EasyChair, 2017.

[17] Konstantin Korovin. iProver - an instantiation-based theorem prover for first-order logic (system description). In *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, pages 292–298, 2008.

[18] Harry R. Lewis. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21(3):317 – 353, 1980.

[19] William McCune. A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical report, Argonne National Laboratory, 1994.

[20] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[21] Ruzica Piskac, Leonardo de Moura, and Nikolaj Bjorner. Deciding effectively propositional logic with equality. Technical report, Microsoft Research, December 2008.

[22] Mathias Preiner, Aina Niemetz, and Armin Biere. Counterexample-guided model synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 264–280, 2017.

[23] John Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[24] Giles Reger, Martin Suda, and Andrei Voronkov. Finding finite models in multi-sorted first-order logic. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2016.

[25] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett. Counterexample-guided quantifier instantiation for synthesis in SMT. In *Computer Aided Verification (CAV)*, pages 198–216, 2015.

[26] Andrew Reynolds, Cesare Tinelli, Amit Goel, and Sava Krstić. Finite model finding in SMT. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 640–655. Springer, 2013.

[27] Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstić, Morgan Deters, and Clark Barrett. Quantifier instantiation techniques for finite model finding in SMT. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2013.

[28] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2010.

[29] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

[30] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec: A cross-community infrastructure for logic solving. In *Proc. of the 7th Int. Joint Conference on Automated Reasoning IJCAR*, volume 8562 of *LNCS*, pages 367–373. Springer, 2014.

[31] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[32] Kuan-Hua Tu, Tzu-Chien Hsu, and Jie-Hong R. Jiang. Qell: Qbf reasoning with extended clause learning and levelized sat solving. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 343–359, Cham, 2015. Springer International Publishing.

[33] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. MaLARea SG1 – machine learner for automated reasoning with semantic guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2008.

[34] Amirhossein Vakili and Nancy A. Day. Finite model finding using the logic of equality with uninterpreted functions. In *Formal Methods (FM)*, pages 677–693, 2016.

[35] Jian Zhang and Hantao Zhang. SEM: a system for enumerating models. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI*, pages 298–303. Morgan Kaufmann, 1995.