Augmenting Model-Based Instantiation with Fast Enumeration

No Author Given

No Institute Given

Abstract. We introduce MBQI-Enum, an approach that extends model-based quantifier instantiation (MBQI) with syntax-guided synthesis (SyGuS) techniques. Our approach targets first-order theories without well-established quantifier instantiation techniques and higher-order quantifiers that can benefit from instantiations with λ-terms. By incorporating a SyGuS enumerator, it generates a broader set of candidate instantiations, including identity functions and terms containing uninterpreted symbols, thereby improving the effectiveness of MBQI.

1 Introduction

Satisfiability modulo theories (SMT) solvers combine a Boolean satisfiability (SAT) solver with decision procedures for interpreted theories. Several SMT solvers, including Bitwuzla [19], Boolector [8], cvc5 [2], veriT [7], and Z3 [18], also support quantifiers via Skolemization and instantiation. With complete instantiation strategies, SMT solvers offer a semidecision procedure for first-order logic with theories. SMT has also been partly extended to higher-order logic [4].

One successful instantiation strategy is model-based quantifier instantiation (MBQI) [11]. Briefly, it iteratively refines a candidate model constructed from the quantifier-free part of the problem. This model guides the generation of new terms for instantiating quantifiers, reducing the search space. MBQI is complete for certain fragments and tends to generate small models for satisfiable problems. But it also has some limitations: First, MBQI instantiates quantifiers only with terms that denote values in a theory. In particular, it does not consider the problem's uninterpreted symbols when creating instantiations, leading it to miss some useful instantiations. Second, for higher-order problems, MBQI cannot generate function terms that return an argument, such as the identity $\lambda x.\ x$.

Another instantiation strategy, which addresses these limitations, is syntax-guided instantiation (SyQI) [20]. It constructs a grammar for each universally quantified variable according to its type and uses enumerated terms derived from the grammar as instantiations. Its main weakness is that although it is model-based like MBQI, it uses a less scalable approach to refining models that centers around syntactic constraints.

In this paper, we propose a new strategy, MBQI-Enum, that combines the strengths of MBQI and SyQI. Specifically, our strategy augments the set of instantiations created from the MBQI model with instantiations generated by a

syntax-guided synthesis (SyGuS) grammar. It incorporates uninterpreted symbols gathered from the problem. For higher-order problems, it also considers λ-abstractions as potential instantiations. In this way, we exploit the fast model-finding capabilities of MBQI and the diversity of terms considered by SyQI.

Our work shares some similarities with Preiner et al. [23], but their approach was limited to selected first-order theories and did not handle higher-order logic. Our approach is resolutely pragmatic and does not aim at completeness.

As an example where λ-terms are needed, consider the unsatisfiable problem consisting of the axiom $\forall z.\exists x,y.\ z\ x\ y \neq x$, where z ranges over binary functions. Running cvc5 with strategies such as MBQI and higher-order E-matching [4] leads the solver to give up early. By contrast, with our strategy, cvc5 immediately finds a contradiction based on the substitution $\{z \mapsto \lambda x, y.\ x\}$. Indeed, if we instantiate z with $\lambda x, y.\ x$ in the axiom and β-reduce, we obtain $x \neq x$.

We implemented MBQI-Enum in cvc5. Our empirical evaluation finds that the strategy increases the number of solved problems for a benchmark suite consisting of various first-order SMT-LIB files by a noticeable margin. Our approach is especially useful to solve unsatisfiable problems involving theories without well-established quantifier instantiation techniques. We can also report substantial gains on higher-order TPTP benchmarks. Our source code, along with instructions for reproducing the experiments, is available as an artifact.

2 Preliminaries

Our work relies on the following pillars: higher-order logic, SMT with quantifiers, MBQI, and SyQI.

Higher-Order Logic. Monomorphic higher-order logic [1, 12], also called simple type theory [9], generalizes classical first-order logic by allowing quantification over functions. The syntax distinguishes between types and terms. Types τ are either base types κ or applications of the function arrow $\to$ to two types: $\tau_1 \to \tau_2$. The type of Booleans is denoted by o. The term language is based on Church's simply typed λ-calculus. Terms are syntactically equal modulo α-, β-, and η-conversions—for example, $(\lambda x.\ x)\ \mathsf{c}$ is syntactically equal to c. A term of type o is called a formula; a function symbol returning o is called a predicate symbol. We let $\bar{x}$ stand for $x_1, x_2, \ldots, x_n$, where $n \geq 1$.

SMT with Quantifiers. Traditionally, SMT solvers work on problems in first-order logic, but they have partly been extended to higher-order logic [4], and in this paper we consider both first- and higher-order problems. SMT solvers that support quantifiers typically do so via a combination of Skolemization and instantiation: Universal quantifiers occurring negatively can be Skolemized; universal quantifiers occurring positively are instantiated heuristically; and existental quantifiers are expressed in terms of $\forall$. To simplify the presentation, we will assume that formulas are in a normal form where possibly negated $\forall$-quantifiers all appear in a cluster at the top level—e.g., $\forall x, y.\ \neg\forall z.\ \mathsf{p}(x, y, z)$.

Algorithm 1 Main SMT loop

```
1: procedure SMT-LOOP(F)
2:     find a set of literals L where
3:         — L ⊨_p F, where L's atoms are a subset of atoms(F)
4:         — L can be partitioned into (E, Q_p, Q_n), where
5:             — E is ground and T-satisfiable
6:             — Q_p consists of universally quantified atoms
7:             — Q_n consists of negated universally quantified atoms
8:     if no such L exists then
9:         return unsat
10:    A ← INSTANTIATION-ROUND(Q_p, E) ∪ SKOLEMIZATION-ROUND(Q_n)
11:    if A = ∅ then
12:        return sat
13:    else
14:        return SMT-LOOP(F ∪ A)
15: procedure INSTANTIATION-ROUND({q_1, ..., q_n}, E)
16:    I ← ∅
17:    for each i ∈ {1, ..., n} do
18:        I ← I ∪ {q_i ⟹ q_i σ | σ ∈ INSTS(q_i, E)}
19:    return I
20: procedure INSTS(q_i, E)
21:    return a set of substitutions for q_i based on (q_i, E)
22: procedure SKOLEMIZATION-ROUND({¬q_1, ..., ¬q_n})
23:    K ← ∅
24:    for each i ∈ {1, ..., n} do
25:        K ← K ∪ {¬q_i ⟹ ¬q_i σ_{k,i}}, where σ_{k,i} maps to Skolem constants for q_i
26:    return K
```

Let T be a theory for a set of interpreted symbols, and let F be the input formula over T. The goal is to find a model of F or derive a contradiction. If F does not contain quantifiers, the quantifier-free part of the SMT solver searches for a set L of literals that propositionally satisfies F. If such an L exists, F is T-satisfiable (i.e., satisfiable with respect to T); otherwise, F is T-unsatisfiable.

If the formula F contains quantifiers, the quantifier-free solver cannot be directly applied. In the SMT solver's main loop, presented in Algorithm 1, the SMT solver tries to find a set L of literals whose atoms come from F and that propositionally satisfies F. Briefly, L is partitioned into a set E of ground literals and two sets of quantified literals, Q_p and Q_n. The ground literals within E must be T-satisfiable, Q_p consists of formulas of the form $\forall \bar{x}.\, \phi$, and Q_n consists of formulas of the form $\neg \forall \bar{x}.\, \phi$. If the solver is unable to find such an L, it concludes that the formula F is T-unsatisfiable.

On the other hand, if a set L of literals is found, new lemmas A are generated through the instantiation and Skolemization rounds. The instantiation round generates *instantiation lemmas*—that is, lemmas of the form $q_i \Longrightarrow q_i\sigma$—from the sets Q_p and E. For each $q_i \in Q_\mathrm{p}$, an instantiation strategy computes

substitutions σ for every top-level variable in q_i mapping them to terms. The Skolemization round generates *Skolemization lemmas* from the set Q_{n}—that is, lemmas of the form $\neg q_i \implies \neg q_i \sigma_{j,i}$, where $\sigma_{j,i}$ instantiates every top-level variable in q_i with a Skolem constant. (We abuse notation and write $(\forall \bar{x}.\ \phi)\sigma$ to mean $\phi\sigma$, syntactically conflating quantifier instantiation and substitution.) The lemmas are then added to the original formula F, and the SMT loop is called recursively. On line 12 of Algorithm 1, we assume that the instantiation strategy denoted by INSTS is *model-sound*, meaning that if INSTS returns an empty set of substitutions, it indicates that the formula F is T-satisfiable.

We now define the instantiation strategy INSTS starting with a naive approach. Subsequently, we will introduce more advanced techniques, including MBQI and SyQI, followed by our strategy, MBQI-Enum. These strategies can replace the naive approach to improve the solver's efficiency.

Definition 2. An *instantiation strategy* takes as input a set of ground terms E and a quantified formula q of the form $\forall \bar{x}.\ \phi$, and outputs a set of grounding substitutions $\{\sigma_1, \ldots, \sigma_m\}$, where the variables mapped by σ_i are exactly the variables in $\bar{x}$ for each $i \in \{1, \ldots, m\}$.

Example 3. Let $\mathsf{a} : \mathit{Int}$ and $\mathsf{p} : \mathit{Int} \to \mathit{Bool}$. Let F be the formula $(\forall y.\ \mathsf{p}\ y) \wedge \neg\, \mathsf{p}\ \mathsf{a}$, where $y : \mathit{Int}$. In the SMT loop, the sets $E = \{\neg\, \mathsf{p}\ \mathsf{a}\}$, $Q_{\mathrm{p}} = \{\forall y.\ \mathsf{p}\ y\}$, and $Q_{\mathrm{n}} = \emptyset$ are defined. The naive instantiation strategy produces substitutions mapping y to terms from E of the same type as y, as shown below:

$q \in Q_{\mathrm{p}}$	$\textsc{insts}(E, q)$
$\forall y.\ \mathsf{p}\ y$	$\{\{y \mapsto \mathsf{a}\}\}$

The instantiation lemma $(\forall y.\mathsf{p}\ y) \implies \mathsf{p}\ \mathsf{a}$ is added to F. Now, the quantifier-free solver finds a contradiction.

The most widely used strategy for quantifier instantiation is *E-matching* [17]. This is a heuristic and typically incomplete technique that choses substitutions by matching ground terms with *patterns*. Over the past decade, SMT solvers have been extended with more sophisticated approaches. *Conflict-based instantiation* [30] is another incomplete technique that attempts to find a single instantiation that would induce a ground conflict, before resorting to E-matching. *MBQI* [11] is a technique for finding instantiations that refute a candidate model, and is typically run when E-matching saturates. *Enumerative instantiation* [24] is an alternative to MBQI that focuses on finding instantiations over the current set of ground terms that are not entailed in the current context.

Quantifier instantiation strategies that target specific theories have also been proposed. *Counterexample-guided instantiation* [27] is complete for specific theories with quantifiers that admit quantifier elimination, such as linear arithmetic and bitvectors. *SyQI* [20] is a more general purpose strategy that uses syntax-guided synthesis for enumerating instantiations and is effective for theories that otherwise do not have well-established instantiation strategies.

Algorithm 4 The MBQI strategy

```
1: procedure INSTS_MBQI(q, E)
2:     assume q is ∀y_1, ..., y_n. φ
3:     let M be a model of E
4:     let M' be a model of ¬ φ^M
5:     if M' does not exist then
6:         return ∅
7:     return {{y_1 ↦ y_1^M', ..., y_n ↦ y_n^M'}}
```

MBQI. MBQI iteratively refines a candidate model M constructed from the quantifier-free part of the problem. As shown in Algorithm 4, the strategy replaces function symbols in the body ϕ of the quantified formula with their interpretation in M. If $\neg\,\phi^M$ is T-satisfiable, this means that a model M' exists, and the strategy returns the substitution $\{y_1 \mapsto y_1^{M'}, \ldots, y_n \mapsto y_n^{M'}\}$, where $y_i^{M'}$ denotes the interpretation for the variable y_i in M'.

Example 5. Let $\mathsf{a} : \mathit{Int}$ and $\mathsf{p} : \mathit{Int} \to \mathit{Bool}$. Let F be the input formula

$$(\forall y.\ \mathsf{p}\ y) \wedge 0 < \mathsf{a} < 2 \wedge \neg\,\mathsf{p}\ \mathsf{a}$$

where $y : \mathit{Int}$. In the SMT loop, our set of literals L is partitioned into $E = \{0 < \mathsf{a} < 2,\ \neg\,\mathsf{p}\ \mathsf{a}\}$, $Q_\mathrm{p} = \{\forall y.\ \mathsf{p}\ y\}$, and $Q_\mathrm{n} = \emptyset$. MBQI builds a model M from E—assume $\mathsf{a}^M = 1$ and $\mathsf{p}^M = \lambda x.\ x \neq 1$. It then considers the negation of the body of the quantified formula in Q_p under the interpretation M, which is $\neg\,(\lambda x.\ x \neq 1)\ y$, which simplifies to $y = 1$ and has a model M', where $y^{M'} = 1$. Thus, MBQI generates the substitution $\{y \mapsto 1\}$, from which the instantiation lemma $(\forall y.\ \mathsf{p}\ y) \Longrightarrow \mathsf{p}\ 1$ is constructed and added to F. Now, the quantifier-free solver finds a contradiction.

Example 6. Let $\mathsf{f} : \mathit{Int} \to \mathit{Int} \to \mathit{Bool}$. Let F be the higher-order formula

$$\forall x, y.\ y\ x \neq y\ (\mathsf{f}\ x)$$

where $x : \mathit{Int}$ and $y : \mathit{Int} \to \mathit{Int}$. The set L is partitioned into sets $E = \emptyset$, $Q_\mathrm{p} = \{\forall x, y.\ y\ x \neq y\ (\mathsf{f}\ x)\}$, and $Q_\mathrm{n} = \emptyset$. MBQI constructs a model M for E such that $\mathsf{f}^M = \lambda z.\ 0$. It then considers the negation of the body of the quantified formula in Q_p under the interpretation M. This is $\neg\, y\ x \neq y\ ((\lambda z.\ 0)\ x)$, which simplifies to $y\ x = y\ 0$ and has a model M', where $x^{M'} = 0$ and $y^{M'} = \lambda z.\ 0$. Thus, it generates the substitution $\{x \mapsto 0,\ y \mapsto \lambda z.\ 0\}$ based on the candidate model. Now, the quantifier-free solver finds a contradiction. Indeed, if we instantiate x, y with this substitution and β-reduce, we obtain $0 \neq 0$.

SyQI. SyQI uses SyGuS to choose instantiation terms. It aims to synthesize a term t for a variable x in a given formula $\forall x.\ \mathsf{p}\ x$ such that $\neg\,\mathsf{p}\ t$ holds. Each quantified variable is associated with a SyGuS grammar. The main advantage of SyQI is that, unlike MBQI, it does not require theory-specific quantifier instantiation procedures. The only parts that depend on the theory are the grammar and the T-satisfiability check for the generated instances.

3 The Method

The core idea behind our new instantiation strategy, MBQI-Enum, is to integrate a SyGuS enumerator within MBQI, thereby enabling the generation of a broader set of candidate instantiations for quantified variables.

Instantiation Strategy. Instead of restricting instantiations to ground terms derived from the current MBQI model, our strategy uses a SyGuS grammar to produce additional candidate instantiations. This grammar is not limited to ground terms with the types of the quantified variables; rather, it incorporates uninterpreted symbols gathered from the entire formula. As a result, it generates a more extensive and comprehensive language of terms.

For each quantified variable in a formula, our strategy performs iterative term enumeration. It generates candidate substitutions from the extended grammar and tests each enumerated term within the formula. For each enumerated term, the strategy tries to apply it as an instantiation for the quantified variable. For higher-order problems, it also considers λ-abstractions as candidate instantiations. If the instantiation fails, our technique continues to the next candidate until a suitable instantiation is found or all possibilities are exhausted.

When none of the candidate instantiations derived from the SyGuS enumeration prove successful, MBQI-Enum reverts to the original MBQI model-derived instantiation. This fallback mechanism ensures that our strategy can in principle solve any problem that MBQI can solve.

Our initial motivation for developing MBQI-Enum was to increase cvc5's success rate on higher-order problems. Nevertheless, incorporating uninterpreted symbols gathered from the entire formula allows our approach to extend its applicability to first-order problems using various SMT theories.

Our approach is presented in Algorithm 7. The strategy starts by invoking MBQI to generate a set of initial substitutions Σ, since the goal is to postprocess the substitutions generated by MBQI using a SyGuS enumerator. If Σ is empty, the strategy immediately returns an empty set, indicating that no valid instantiation could be found. Otherwise, it proceeds by initializing Σ to contain a single substitution σ. Next, it generates additional substitutions by extending the current one using the SyGuS enumerator.

Our strategy then iterates over the quantified variables y_i in the formula q to instantiate. For each variable, it constructs a grammar G_i used to guide the enumeration of candidate terms for substituting y_i. The enumeration starts by generating terms from G_i in a sequential manner. For each term e, the strategy creates a new substitution σ' by mapping y_i to e in the current substitution σ. It then checks whether the negation of the body ϕ under σ' is T-satisfiable. This check serves to maintain the invariant that the negation of the body of the quantified formula, after applying the current substitution, remains T-satisfiable. In other words, we want to ensure that the generated instantiation refutes the current model (cf. line 4 of Algorithm 4). If it does, σ is updated to σ', and the strategy moves on to the next variable. Once all variables have been considered, the strategy returns the refined substitution σ.

Algorithm 7 The MBQI-Enum strategy

```
1: procedure INSTS_MBQI_FAST_SYGUS(q, E)
2:     assume q is ∀y_1, ..., y_n. φ
3:     let Σ ← INSTS_MBQI(q, E)
4:     if Σ = ∅ then
5:         return ∅
6:     else
7:         let Σ ← {σ}
8:     for each i ∈ {1, ..., n} do
9:         let G_i ← CHOOSE_GRAMMAR(q, y_i)
10:        for each j ∈ {1, 2, ...} do
11:            let e ← GET_ENUM_TERM(G_i, j)
12:            if e does not exist then
13:                break
14:            σ' ← σ[y_i ↦ e]
15:            if ¬φσ' is sat then
16:                σ ← σ'
17:                break
18:    return σ
19: procedure CHOOSE_GRAMMAR(q, y_i)
20:    let F be the original input formula
21:    let S ← ∅
22:    if option_syms_global then
23:        S ← S ∪ symbols(F)
24:    if option_syms_local then
25:        S ← S ∪ symbols(q)
26:    if option_ext_vars then
27:        S ← S ∪ {y_{i+1}, ..., y_n}
28:    return grammar that generates terms of the same sort as y_i over symbols in S
```

Choice of Grammar. Term enumeration is based on a SyGuS grammar. Choosing an appropriate grammar for each quantified variable is crucial for selecting the correct instantiations. Our strategy builds a set S of symbols based on three Boolean options: *syms_global*, *ext_vars*, and *syms_local*. These options specify which symbols from the formula F will be included when constructing the SyGuS grammar.

If no options are enabled, the set S is empty. If *syms_global* is enabled, all function symbols from the entire formula F are contained in S. If *ext_vars* is enabled, S is augmented with bound variables from the formula q that are not yet instantiated. Finally, if *syms_local* is enabled, the set S also contains function symbols specifically from q. Based on these settings, our approach constructs a grammar that generates terms over the symbols in S, while ensuring that these terms are well-typed with respect to the type of y_i.

For higher-order variables, thanks to η-conversion, it is sufficient to consider only grammars that generate λ-abstractions. The variables bound by these λ-

abstractions are considered terminal symbols of the grammar that generates the abstraction's body. For example, for a function variable whose arity is n and whose arguments are the same base type as its return type, we add grammar rules of this form:

$$A ::= \lambda x_1, \ldots, x_n.\ B$$
$$B ::= x_1 \mid \cdots \mid x_n$$

Moreover, there will be additional rules for B and possibly for other nonterminal symbols of the grammar. This ensures that any λ instantiations are formed by enumerating the body B over the bound variables $x_1, \ldots, x_n$.

Example 8. Let $\mathsf{a} : \mathit{Int}$ and $\mathsf{p} : \mathit{Int} \to \mathit{Bool}$. Let F be the input formula

$$(\forall y.\ \neg\, \mathsf{p}\ (y\ \mathsf{a})) \wedge \mathsf{p}\ \mathsf{a}$$

where $y : \mathit{Int} \to \mathit{Int}$. The set L is partitioned into $E = \{\mathsf{p}\ \mathsf{a}\}$, $Q_\mathrm{p} = \{\forall y.\ \neg\, \mathsf{p}\ (y\ \mathsf{a})\}$, and $Q_\mathrm{n} = \emptyset$. MBQI generates substitutions such as $\{y \mapsto \lambda x.\ 0\}$, $\{y \mapsto \lambda x.\ 1\}$, ... As a result, the solver does not terminate. In contrast, MBQI-Enum augments these instantiations based on enumeration. On the first iteration of the SMT loop, MBQI-Enum considers the set $\Sigma = \{\{y \mapsto \lambda x.\ 0\}\}$ consisting of the first substitution generated by MBQI. Our strategy first constructs a grammar for y. The set of symbols S is empty. The grammar is

$$\mathcal{A} ::= \lambda x.\ \mathcal{B}$$
$$\mathcal{B} ::= x \mid 0 \mid 1 \mid \mathcal{B} + \mathcal{B} \mid \mathcal{B} - \mathcal{B} \mid \mathrm{ite}(\mathcal{B}, \mathcal{B}, \mathcal{B})$$
$$\mathcal{C} ::= \mathsf{true} \mid \mathsf{false} \mid \mathcal{B} = \mathcal{B} \mid \mathcal{B} \leq \mathcal{B} \mid \neg \mathcal{C} \mid \mathcal{C} \wedge \mathcal{C} \mid \mathcal{C} \vee \mathcal{C}$$

Next, MBQI-Enum enumerates terms derived from the grammar and creates the substitution $\sigma' = \{y \mapsto \lambda x.\ x\}$ by updating σ with the enumerated term $\lambda x.\ x$ from the grammar. Our strategy then checks whether the negation of the body of the quantified formula after applying σ' is T-satisfiable. Indeed, if we instantiate y with $\lambda x.\ x$ and β-reduce, we obtain $\mathsf{p}\ \mathsf{a}$, which is T-satisfiable. The substitution σ is then updated to $\{y \mapsto \lambda x.\ x\}$ and returned. Back in the SMT loop, the instantiation lemma $(\forall y.\ \neg\, \mathsf{p}\ (y\ \mathsf{a})) \Longrightarrow \neg\, \mathsf{p}\ \mathsf{a}$ is added to F. Now, the quantifier-free solver finds a contradiction.

The candidate substitutions generated by MBQI-Enum (M) are listed below:

Iter.	$q \in Q_\mathrm{p}$	E	M(q, E)	New E
1	$\forall y.\ \neg\, \mathsf{p}\ (y\ \mathsf{a})$	$\{\mathsf{p}\ \mathsf{a}\}$	$\{\{y \mapsto \lambda x.\ x\}\}$	$\{\mathsf{p}\ \mathsf{a}, \neg\, \mathsf{p}\ \mathsf{a}\}$

The first column shows the number of the SMT loop iteration. The second column shows the quantified formula q and the third column shows the set E of ground literals before every iteration. The fourth column shows the possible selection of substitutions of y that are considered with MBQI-Enum and the fifth column shows the set E after every iteration.

In this example, MBQI-Enum was able to terminate in the first iteration, since it found a substitution for y that immediately leads to a refutation, whereas MBQI considers a repeating pattern of instantiations that leads to a timeout.

Another useful candidate substitution would have been $\{y \mapsto \lambda x.\ \mathsf{a}\}$. MBQI-Enum would have found this substitution as well if it had not terminated after finding $\{y \mapsto \lambda x.\ x\}$.

In an informal, preliminary evaluation on higher-order TPTP benchmarks, we determined that the best-performing configuration enables *ext_vars* and *syms_local* and leaves *syms_global* disabled. The following examples are based on this setup.

Example 9. Let $\mathsf{p} : \mathit{Int} \to \mathit{Bool}$ and $\mathsf{f} : \mathit{Int} \to \mathit{Int}$. Let F be the input formula

$$\forall y.\ \neg\, \forall z.\ \neg\, \mathsf{p}\ (y\ z) \vee \mathsf{p}\ (\mathsf{f}\ z)$$

where $y : \mathit{Int} \to \mathit{Int}$ and $z : \mathit{Int}$. The set L is partitioned into sets $E = \emptyset$, $Q_\mathrm{p} = \{\forall y.\ \neg\, \forall z.\ \neg\, \mathsf{p}\ (y\ z) \vee \mathsf{p}\ (\mathsf{f}\ z)\}$, and $Q_\mathrm{n} = \emptyset$. MBQI generates substitutions such as $\{y \mapsto \lambda x.\ 0\}, \{y \mapsto \lambda x.\ 1\}, \ldots$. As a result, the solver does not terminate. In contrast, MBQI-Enum adds the first substitution generated by MBQI, $\{y \mapsto \lambda x.\ 0\}$, to the set Σ and proceeds to postprocess it. Our strategy first constructs a grammar for y using the function symbols from q. The set of symbols used is $S = \{\mathsf{f}, \mathsf{p}\}$. The grammar is

$$\begin{aligned}
&\mathcal{A} ::= \lambda x.\ \mathcal{B}\\
&\mathcal{B} ::= x \mid \mathsf{f}\ \mathcal{B} \mid 0 \mid 1 \mid \mathcal{B} + \mathcal{B} \mid \mathcal{B} - \mathcal{B} \mid \mathrm{ite}(\mathcal{C}, \mathcal{B}, \mathcal{B})\\
&\mathcal{C} ::= \mathsf{true} \mid \mathsf{false} \mid \mathcal{B} = \mathcal{B} \mid \mathcal{B} \leq \mathcal{B} \mid \mathsf{p}\ \mathcal{B} \mid \neg\, \mathcal{C} \mid \mathcal{C} \wedge \mathcal{C} \mid \mathcal{C} \vee \mathcal{C}
\end{aligned}$$

In the first iteration, MBQI-Enum generates the substitution $\sigma = \{y \mapsto \lambda x.\ x\}$. The instantiation lemma $(\forall y.\ \neg\, \forall z.\ \neg\, \mathsf{p}\ (y\ z) \vee \mathsf{p}\ (\mathsf{f}\ z)) \Longrightarrow \neg\, \forall z.\ \neg\, \mathsf{p}\ z \vee \mathsf{p}\ (\mathsf{f}\ z)$ is added to F. The set L is now partitioned into $E = \emptyset$, $Q_\mathrm{p} = \{\forall y.\ \neg\, \forall z.\ \neg\, \mathsf{p}\ (y\ z) \vee \mathsf{p}\ (\mathsf{f}\ z)\}$, and $Q_\mathrm{n} = \{\neg\, \forall z.\ \neg\, \mathsf{p}\ z \vee \mathsf{p}\ (\mathsf{f}\ z)\}$. Next, the quantifier in Q_n is Skolemized. The Skolemization lemma $(\neg\, \forall z.\ \neg\, \mathsf{p}\ z \vee \mathsf{p}\ (\mathsf{f}\ z)) \Longrightarrow \mathsf{p}\ \mathsf{sk}_1 \wedge \neg\, \mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_1)$ is added to F. As a result, the set E is updated to $\{\mathsf{p}\ \mathsf{sk}_1, \neg\, \mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_1)\}$.

In the next iteration, the substitution σ is modified to $\{y \mapsto \lambda x.\ \mathsf{f}\ x\}$, incorporating the enumerated term $\lambda x.\ \mathsf{f}\ x$ from the grammar. The instantiation lemma $(\forall y.\ \neg\, \forall z.\ \neg\, \mathsf{p}\ (y\ z) \vee \mathsf{p}\ (\mathsf{f}\ z)) \Longrightarrow \neg\, \forall z.\ \neg\, \mathsf{p}\ (\mathsf{f}\ z) \vee \mathsf{p}\ (\mathsf{f}\ z)$ is then added to F. After Skolemization, the set E is augmented with $\{\mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_2), \neg\, \mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_2)\}$. The quantifier-free solver finds a contradiction. Indeed, if we instantiate y with $\lambda x.\ \mathsf{f}\ x$ in F, β-reduce, and Skolemize z, we obtain $\mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_2) \wedge \neg\, \mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_2)$.

The candidate substitutions generated by MBQI-Enum (M) are listed below:

Iter.	$q \in Q_\mathrm{p}$	E	$\mathrm{M}(q, E)$	New E
1	$\forall y.\ \neg\, \forall z.\ \neg\, \mathsf{p}\ (y\ z) \vee \mathsf{p}\ (\mathsf{f}\ z)$	$\emptyset$	$\{\{y \mapsto \lambda x.\ x\}\}$	$\{\mathsf{p}\ \mathsf{sk}_1, \neg\, \mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_1)\}$
2	$\forall y.\ \neg\, \forall z.\ \neg\, \mathsf{p}\ (y\ z) \vee \mathsf{p}\ (\mathsf{f}\ z)$	$\{\mathsf{p}\ \mathsf{sk}_1, \neg\, \mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_1)\}$	$\{\{y \mapsto \lambda x.\ \mathsf{f}\ x\}\}$	$\{\mathsf{p}\ \mathsf{sk}_1, \neg\, \mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_1)$ $\mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_2), \neg\, \mathsf{p}\ (\mathsf{f}\ \mathsf{sk}_2)\}$

In this example, our strategy terminated in the second iteration, whereas MBQI would lead the solver to time out.

Example 10. In this example, our strategy is run with and without the *syms_global* option enabled. Let u be an uninterpreted sort, and let $\mathsf{a} : u$ and $\mathsf{b} : u$. Let F be the input formula

$$(\forall x, y, z.\ x\ y = x\ z) \wedge \mathsf{a} \neq \mathsf{b}$$

where $x : (u \to u) \to u$, $y : u \to u$, and $z : u \to u$. The set L is partitioned into $E = \{\mathsf{a} \neq \mathsf{b}\}$, $Q_\mathrm{p} = \{\forall x, y, z.\ x\ y = x\ z\}$, and $Q_\mathrm{n} = \emptyset$. MBQI-Enum fails to construct a grammar for a variable x, y, or z that has any terms of the same type as the variable. As a result, it cannot generate any substitutions, and the solver gives up early. In contrast, when the *syms_global* option is enabled in MBQI-Enum, function symbols from the entire formula F are used to construct a grammar for each variable x, y, and z. For these variables, the set of symbols is $\{\mathsf{a}, \mathsf{b}\}$. The grammar for x follows:

$$\begin{aligned}
\mathcal{A} &::= \lambda w.\ \mathcal{B} \\
\mathcal{B} &::= w\ \mathcal{B} \mid \mathsf{a} \mid \mathsf{b} \mid \mathrm{ite}(\mathcal{C}, \mathcal{B}, \mathcal{B}) \\
\mathcal{C} &::= \mathsf{true} \mid \mathsf{false} \mid \mathcal{B} = \mathcal{B} \mid \neg\mathcal{C} \mid \mathcal{C} \wedge \mathcal{C} \mid \mathcal{C} \vee \mathcal{C}
\end{aligned}$$

(Since w is of unary function type, we pass an argument corresponding to the nonterminal $\mathcal{B}$ in the second grammar rule.) The grammar for y and z follows:

$$\begin{aligned}
\mathcal{A} &::= \lambda w.\ \mathcal{B} \\
\mathcal{B} &::= w \mid \mathsf{a} \mid \mathsf{b} \mid \mathrm{ite}(\mathcal{C}, \mathcal{B}, \mathcal{B}) \\
\mathcal{C} &::= \mathsf{true} \mid \mathsf{false} \mid \mathcal{B} = \mathcal{B} \mid \neg\mathcal{C} \mid \mathcal{C} \wedge \mathcal{C} \mid \mathcal{C} \vee \mathcal{C}
\end{aligned}$$

Our strategy then enumerates terms derived from the grammar and builds the substitutions $\{x \mapsto \lambda w.\ w\ \mathsf{b}\}, \{y \mapsto \lambda w.\ w\}$, and $\{z \mapsto \lambda w.\ \mathsf{a}\}$. The instantiation lemma $(\forall x, y, z.\ x\ y = x\ z) \Longrightarrow \mathsf{b} = \mathsf{a}$ is added to F. Now, the quantifier-free solver finds a contradiction. Indeed, if we instantiate x with $\lambda w.\ w\ \mathsf{b}$, y with $\lambda w.\ w$, and z with $\lambda w.\ \mathsf{a}$ in F and β-reduce, we obtain $\mathsf{a} = \mathsf{b} \wedge \mathsf{a} \neq \mathsf{b}$.

The candidate substitutions generated by MBQI-Enum (M) are listed below:

Iter.	$q \in Q_\mathrm{p}$	E	$\mathrm{M}(q, E)$	New E
1	$\forall x, y, z.\ x\ y = x\ z$	$\{\mathsf{a} \neq \mathsf{b}\}$	$\emptyset$	$\{\mathsf{a} \neq \mathsf{b}\}$

The same information is provided for MBQI-Enum with the option *syms_global* enabled (M+g) below:

Iter.	$q \in Q_\mathrm{p}$	E	$(\mathrm{M}+\mathrm{g})(q, E)$	New E
1	$\forall x, y, z.\ x\ y = x\ z$	$\{\mathsf{a} \neq \mathsf{b}\}$	$\{\{x \mapsto \lambda w.\ w\ \mathsf{b}\}, \{y \mapsto \lambda w.\ w\}, \{z \mapsto \lambda w.\ \mathsf{a}\}\}$	$\{\mathsf{a} \neq \mathsf{b}, \mathsf{a} = \mathsf{b}\}$

In this example, MBQI-Enum with *syms_global* was able to terminate in the first iteration, since it found a substitution for the quantified variables that leads to a refutation. By contrast, MBQI-Enum does not terminate since it cannot build any substitutions.

4 Implementation and Heuristics

We implemented MBQI-Enum as an extension of cvc5's implementation of MBQI. Our strategy is invoked after the current MBQI strategy returns a candidate instantiation (line 3 of Algorithm 7).

For each variable, our algorithm chooses a grammar (line 9) and initializes a term enumeration data structure. Since the choice of the grammar is fixed over the course of solving, the grammar is constructed only once. Our implementation uses the utility for fast SyGuS enumeration described in Reynolds et al. [26] as a black box. Since the grammar for each variable is fixed, we can cache the enumeration and invoke this utility only on line 11 of Algorithm 7, when j is larger than the number of terms we have generated on a previous run, where we notice that a term that was skipped in a previous call to this method may be incorporated into instantiations on later calls.

On line 15 of Algorithm 7, we use cvc5's ability to call a copy of itself as a subsolver. As an optimization, this satisfiability check can be avoided if the query to check simplifies to "true" or "false."

5 Evaluation

We extensively evaluated our cvc5 implementation of MBQI-Enum both on higher- and on first-order benchmarks.

Setup. As base configuration, we used our best-performing setup: MBQI-Enum with the options *ext_vars* and *syms_local* enabled by default. We denote this configuration by cvc5[M].

We first compare the performance of the base configuration against traditional instantiation techniques: cvc5[e], which uses enumerative instantiation [25]; cvc5[s], which uses SyQI [20]; cvc5[c], which uses counterexample-guided instantiation [28]; and cvc5[m], which uses MBQI [11].

Additionally, for higher-order problems, we also include a comparison with the state-of-the-art provers Vampire [16] and Zipperposition [10]. For Vampire, we used its portfolio mode, while Zipperposition was tested in its so-called "best" mode, since it does not include a portfolio mode.

Next, we compare the base configuration on first-order benchmarks to two state-of-the-art SMT solvers: Z3 [18], the only SMT solver aside from cvc5 that supports all the logics handled by our implementation, and Bitwuzla [19], which supports only logics without arithmetic.

Finally, we compare the performance of all four MBQI-Enum configurations on both higher-order and first-order problems. In this evaluation, we toggled one option at a time: cvc5[M−x] denotes MBQI-Enum with *ext_vars* disabled, cvc5[M−L] denotes MBQI-Enum with *syms_local* disabled, and cvc5[M+g] denotes MBQI-Enum with *syms_global* enabled.

We performed all experiments on a system with a 40x Intel Xeon Silver 4114 processor at 2.20 GHz and 192 GB of RAM using Debian Bookworm as the operating system. We used a time limit of 60 seconds for each benchmark.

Table 1. MBQI-Enum vs. other strategies, Vampire, and Zipperposition on TPTP TH0 benchmarks

	Vampire	Zipperposition	cvc5[e]	cvc5[s]	cvc5[m]	cvc5[M]
Satisfiable	6	0	72	78	121	**129**
Unsatisfiable	**1757**	1499	1643	1304	1637	1670
Total	1763	1499	1715	1382	1758	**1799**
Unknown	0	0	350	38	127	59
Timeouts	999	1263	697	1342	877	904

Higher-Order Problems. The higher-order part of the experiments was carried out on monomorphic higher-order problems (TH0) from version 9.0.0 of the TPTP library [31]. The benchmark set consists of 2762 problems. From the 3962 TH0 problems, we excluded 1200 benchmarks that one or more systems could not parse (e.g., because they use arithmetic).

The results are summarized in Table 1. In this and the following tables, bold indicates the best-performing system. Notably, our approach achieves the highest total count of solved benchmarks, surpassing the nearest competitor by 36 solved problems. Overall, it outperforms all other cvc5 strategies as well as Zipperposition in higher-order logic and solves 87 fewer unsatisfiable problems than Vampire. Our strategy's advantage over Zipperposition likely stems from using Zipperposition's "best" mode for the evaluation, since it does not include a portfolio mode. Remarkably, our strategy managed to solve 129 satisfiable problems, whereas Vampire solved only 6, and Zipperposition none.

The performance of our approach shows a significant improvement over traditional instantiation techniques, particularly compared with MBQI, which was previously the best-performing cvc5 configuration for higher-order problems. Specifically, our strategy successfully solved an additional 41 problems without any losses. When compared to SyQI, our strategy solved 417 more benchmarks while also incurring no losses.

Table 2 shows the evaluation of the different configurations of MBQI-Enum on higher-order problems. We see that all three options are beneficial, but for *syms_global* the difference is only two problems. (In our preliminary evaluation,

Table 2. MBQI-Enum configurations on TPTP TH0 benchmarks

	cvc5[M]	cvc5[M−x]	cvc5[M−l]	cvc5[M+g]
Satisfiable	**129**	**129**	122	**129**
Unsatisfiable	1670	1665	1655	**1672**
Total	1799	1794	1777	**1801**
Unknown	59	65	88	56
Timeouts	904	903	897	905

we had found *syms_global* to be slightly harmful, which is why we disabled it by default.)

First-Order Problems. The experiments on first-order problems were conducted on the SMT-LIB benchmarks [5], as of April 2024, focusing on logics that include quantifiers. We specifically considered logics involving theories such as floating-point arithmetic, linear and nonlinear arithmetic, and bit vectors. Overall, we include the logics BV (bit vectors), FP (floating-point arithmetic), LIA (linear integer arithmetic), LRA (linear real arithmetic), NIA (non-linear integer arithmetic), NRA (non-linear real arithmetic), and their combinations: BVFP, BVFPLRA, and FPLRA. We also incorporated the ABV (arrays and bit vectors) and UFBV (uninterpreted functions with bit vectors). In total, our benchmark set consists of 21 605 problems.

Table 3. MBQI-Enum vs. other techniques, Bitwuzla, and Z3 on SMT-LIB benchmarks

Library	Bitwuzla		Z3		cvc5[e]		cvc5[s]		cvc5[c]		cvc5[m]		cvc5[M]	
	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
ABV	378	47	386	103	21	**926**	617	234	17	90	**790**	135	729	157
ABVFP	24	0	29	2	10	3	15	0	13	0	**31**	0	**31**	0
BV	**638**	5060	547	4993	202	4837	316	4919	439	**5195**	610	4834	593	5076
BVFP	**179**	**12**	163	5	26	**12**	102	4	106	0	167	4	169	7
FP	129	2172	16	1789	111	1588	**149**	2034	113	2003	115	2053	116	**2222**
UFBV	25	**120**	**41**	103	8	103	9	108	8	52	23	83	21	105
Subtotal	1373	7411	1182	6995	378	7469	1208	7299	696	7340	**1736**	7109	1659	**7567**
SAT+UNSAT	8784		8177		7847		8507		8036		8845		**9226**	
ABVFPLRA			**55**	**3**	14	2	18	1	18	2	41	2	42	2
BVFPLRA			203	16	80	**25**	120	24	111	24	220	24	**226**	**25**
FPLRA			24	0	20	0	24	0	22	0	**37**	0	36	0
LIA			140	230	12	170	**150**	236	**150**	**266**	**150**	167	149	239
LRA			**756**	**1360**	468	1117	477	1131	591	1302	545	1123	557	1157
NIA			65	**144**	16	43	49	45	64	**144**	67	47	**80**	61
NRA			**3**	**3806**	1	3802	1	3783	**3**	3801	**3**	3712	**3**	3802
Total			2428	12554	989	12628	2047	12519	1655	**12879**	**2799**	12184	2752	12853
SAT+UNSAT			14982		13617		14566		14534		14983		**15605**	

The results, summarized in Table 3, show that our approach performs remarkably well against all other cvc5 configurations, as well as against Bitwuzla and Z3, across all SMT logics. Notably, it achieves the highest total count of benchmarks solved, surpassing the nearest competitor by 623 solved problems. Our strategy solves the most problems in ABVFP and BVFPLRA and achieves the highest number of unsatisfiable benchmarks solved in FP and the highest number of satisfiable benchmarks solved in NIA.

For these SMT theories, our strategy is a clear improvement over previous instantiation strategies. When compared to MBQI, it successfully solved an additional 704 problems while incurring a loss of 82 problems across all logics. The raw evaluation data also reveals a notable reduction in timeouts, decreasing from 3035 to 2436.

Our strategy also substantially outperforms enumerative instantiation and SyQI across most benchmark categories. Both of these share an enumerative nature. The former relies on evolving ground terms within the current context, while the latter employs a fixed grammar derived from the initial set of terms. Overall, enumerative instantiation and SyQI perform significantly better than MBQI in unsatisfiable benchmarks (+444), but they underperform in satisfiable benchmarks (−752). This highlights the need for a hybrid approach that combines model-based and enumerative techniques. Our strategy incorporates the enumerative aspects of SyQI while enhancing the model-based features of MBQI to generate instantiations. This is likely why it outperforms all the mentioned configurations. Our strategy also matches or outperforms counterexample-guided instantiation in most logics. However, in logics such as LRA, counterexample-guided instantiation is expected to perform better due to its specialized handling of such theories.

Compared with Bitwuzla and Z3, our approach performs very well across various logics, often closely matching or even surpassing both competitors. Overall, our strategy solved 442 more benchmarks than Bitwuzla and 623 more than Z3 in total. When compared to Bitwuzla, our strategy leads in ABV, ABVFP, and in the number of unsatisfiable benchmarks solved in FP, though it lags slightly behind in BVFP, UFBV, and in the number of satisfiable benchmarks solved in BV. This may be because MBQI-Enum is primarily designed for deriving contradictions. As for Z3, our strategy outperforms it across most logics, except for ABVFPLRA and LRA, where Z3's better performance is likely attributable to its well-established instantiation strategies for real arithmetic.

Finally, the evaluation of the different configurations of MBQI-Enum in first-order SMT-LIB benchmarks across different theories is shown in Table 4. We observe that most configurations perform similarly; however, the configuration of MBQI-Enum without the *syms_local* option enabled shows significantly poorer performance.

In summary, our approach was highly effective in first-order SMT-LIB benchmarks, achieving the highest number of benchmarks solved. With further refinements tailored to specific logics, we suspect that its performance could be improved further.

6 Related Work

Mainstream approaches for quantifier instantiation in SMT are typically centered around E-matching [17]. Conflict-based instantiation [3,13,30] can improve the solver's ability to answer "T-unsatisfiable" by prioritizing instantiations that induce quantifier-free conflicts. As a whole, these techniques are generally incomplete and do not target specific background theories. For satisfiable instances, Ge and de Moura [11] introduced MBQI, which is complete for certain fragments. Finite model finding [29] is a variant of this technique that targets quantified formulas whose domains are small and finite. Approaches for quantified formulas

Table 4. MBQI-Enum configurations in SMT-LIB benchmarks

Library	cvc5[M]		cvc5[M-x]		cvc5[M-l]		cvc5[M+g]	
	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
ABV	729	157	729	157	647	135	**737**	**159**
ABVFPLRA	**42**	2	**42**	2	41	2	38	2
ABVFP	31	0	31	0	31	0	**32**	0
BV	593	5076	594	5075	**616**	4842	600	**5086**
BVFP	169	**7**	169	**7**	167	4	**176**	2
BVFPLRA	226	**25**	226	**25**	220	24	**231**	**25**
FP	**116**	2222	**116**	2222	115	2089	**116**	**2223**
FPLRA	36	0	36	0	**37**	0	36	0
LIA	149	**239**	149	**239**	**150**	167	149	**239**
LRA	**557**	**1157**	**557**	**1157**	546	1129	**557**	**1157**
NIA	**80**	**61**	**80**	**61**	68	47	78	56
NRA	3	**3802**	3	**3802**	3	3712	3	**3802**
UFBV	21	**105**	21	**105**	**24**	88	17	99
Total	2752	**12 853**	2753	12 852	2665	12 239	**2770**	12 850

in higher-order logic are discussed by Barbosa et al. [4], but, in contrast to this work, they are based on (higher-order) E-matching.

Other approaches for higher-order logic, notably in Vampire [16] and Zipperposition [10], rely on superposition. Vampire has been initially extended to handle higher-order reasoning using applicative first-order logic with combinators. Since this proved insufficient for problems requiring complex unifiers, its superposition calculus was later enhanced with native λ-abstractions and a depth-bounded version of higher-order unification [6]. As for Zipperposition, it also uses a superposition calculus that directly supports higher-order terms. It tackles the challenge of higher-order unification by using techniques such as pattern unification and heuristics to manage undecidability issues.

Certain background theories admit quantifier elimination, which can be handled using domain-specific instantiation strategies. Specifically, efficient and complete instantiation procedures have been developed for quantified linear arithmetic [27] and quantified bitvectors [21]. These techniques require specific knowledge of the background theory.

Other recent works on quantifier instantiation have pursued enumeration as a pragmatic means for discovering useful instantiations. Reynolds et al. [24] introduced enumerative instantiation as an alternative to MBQI, which primarily focused on first-order logic in the empty theory. This technique has been further studied in more recent works, where more advanced selection strategies are used for instantiations, including those based on machine learning [14, 15, 22].

The closest related works to ours are counterexample-guided model synthesis [23] and SyQI [20], which both focus on enumerative approaches for finding useful instantiations in rich background logics. The former work was implemented in the Boolector [8] solver; it was limited to selected first-order theories

and did not handle higher-order logic. The latter work can potentially be used for any theory but does not leverage MBQI for guiding the instantiation procedure. Our evaluation shows that our MBQI-Enum strategy generally outperforms SyQI overall.

7 Conclusion

We presented a new strategy, MBQI-Enum, for instantiating quantifiers in SMT solvers. It extends MBQI with the SyGuS enumerator, thereby augmenting the number of instantiations considered at every iteration. The main strength of our strategy is that it combines the fast model-finding capabilities of MBQI and the diversity of terms considered by SyQI. We implemented the strategy in cvc5 and found that it helps solve many first- and higher-order problems from SMT-LIB and TPTP for which cvc5 previously either timed out or gave up early.

Several aspects of our approach present opportunities for future work. First, we could improve performance by enhancing the quantifier-free solver to better integrate with our instantiation approach. Moreover, although our instantiation technique is designed to be generic, we could tailor it to individual SMT logics. Finally, we could develop more sophisticated instantiation strategies for higher-order logic. By designing methods that can more intelligently navigate the space of enumerated terms, we should be able to improve the solver's ability to handle complex higher-order problems.

References

1. Andrews, B.: An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof. Springer, 2nd edn. (2002)
2. Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength smt solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 415–442. Springer International Publishing, Cham (2022)
3. Barbosa, H., Fontaine, P., Reynolds, A.: Congruence closure with free variables. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10206, pp. 214–230 (2017). https://doi.org/10.1007/978-3-662-54580-5_13, https://doi.org/10.1007/978-3-662-54580-5_13
4. Barbosa, H., Reynolds, A., El Ouraoui, D., Tinelli, C., Barrett, C.: Extending smt solvers to higher-order logic. In: Fontaine, P. (ed.) Automated Deduction – CADE 27. pp. 35–54. Springer International Publishing, Cham (2019)
5. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)

6. Bhayat, A., Suda, M.: A higher-order vampire (short paper). In: Benzmüller, C., Heule, M.J., Schmidt, R.A. (eds.) Automated Reasoning. pp. 75–85. Springer Nature Switzerland, Cham (2024)
7. Bouton, T., Caminha B. de Oliveira, D., Déharbe, D., Fontaine, P.: verit: An open, trustable and efficient smt-solver. In: Schmidt, R.A. (ed.) Automated Deduction – CADE-22. pp. 151–156. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
8. Brummayer, R., Biere, A.: Boolector: An efficient smt solver for bit-vectors and arrays. In: Kowalewski, S., Philippou, A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 174–177. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
9. Church, A.: A formulation of the simple theory of types. The Journal of Symbolic Logic **5**(2), 56–68 (1940), https://www.jstor.org/stable/2266170
10. Cruanes, S., contributors: Zipperposition: a clausal superposition theorem prover. https://github.com/sneeuwballen/zipperposition (2024), gitHub repository
11. Ge, Y., de Moura, L.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification. pp. 306–320. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
12. Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem-Proving Environment for Higher-Order Logic. Cambridge University Press (1993)
13. Hoenicke, J., Schindler, T.: Incremental search for conflict and unit instances of quantified formulas with e-matching. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12597, pp. 534–555. Springer (2021). https://doi.org/10.1007/978-3-030-67067-2_24, https://doi.org/10.1007/978-3-030-67067-2_24
14. Jakubuv, J., Janota, M., Piotrowski, B., Piepenbrock, J., Reynolds, A.: Selecting quantifiers for instantiation in SMT. In: Graham-Lengrand, S., Preiner, M. (eds.) Proceedings of the 21st International Workshop on Satisfiability Modulo Theories (SMT 2023) co-located with the 29th International Conference on Automated Deduction (CADE 2023), Rome, Italy, July, 5-6, 2023. CEUR Workshop Proceedings, vol. 3429, pp. 71–77. CEUR-WS.org (2023), https://ceur-ws.org/Vol-3429/short10.pdf
15. Janota, M., Barbosa, H., Fontaine, P., Reynolds, A.: Fair and adventurous enumeration of quantifier instantiations. In: Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, October 19-22, 2021. pp. 256–260. IEEE (2021). https://doi.org/10.34727/2021/ISBN.978-3-85448-046-4_35, https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_35
16. Kovács, L., Voronkov, A.: The vampire prover. In: International Conference on Computer Aided Verification. pp. 292–298. Springer (2013)
17. de Moura, L., Bjørner, N.: Efficient e-matching for smt solvers. In: Pfenning, F. (ed.) Automated Deduction – CADE-21. pp. 183–198. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
18. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
19. Niemetz, A., Preiner, M.: Bitwuzla. In: Enea, C., Lal, A. (eds.) Computer Aided Verification. pp. 3–17. Springer Nature Switzerland, Cham (2023)
20. Niemetz, A., Preiner, M., Reynolds, A., Barrett, C., Tinelli, C.: Syntax-guided quantifier instantiation. In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms

for the Construction and Analysis of Systems. pp. 145–163. Springer International Publishing, Cham (2021)
21. Niemetz, A., Preiner, M., Reynolds, A., Barrett, C.W., Tinelli, C.: Solving quantified bit-vectors using invertibility conditions. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10982, pp. 236–255. Springer (2018). https://doi.org/10.1007/978-3-319-96142-2_16, https://doi.org/10.1007/978-3-319-96142-2_16
22. Piepenbrock, J., Janota, M., Urban, J., Jakubuv, J.: First experiments with neural cvc5. In: Bjørner, N.S., Heule, M., Voronkov, A. (eds.) LPAR 2024: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning, Port Louis, Mauritius, May 26-31, 2024. EPiC Series in Computing, vol. 100, pp. 264–277. EasyChair (2024). https://doi.org/10.29007/H5DR, https://doi.org/10.29007/h5dr
23. Preiner, M., Niemetz, A., Biere, A.: Counterexample-guided model synthesis. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10205, pp. 264–280 (2017). https://doi.org/10.1007/978-3-662-54577-5_15, https://doi.org/10.1007/978-3-662-54577-5_15
24. Reynolds, A., Barbosa, H., Fontaine, P.: Revisiting enumerative instantiation. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 112–131. Springer International Publishing, Cham (2018)
25. Reynolds, A., Barbosa, H., Fontaine, P.: Revisiting enumerative instantiation. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 112–131. Springer International Publishing, Cham (2018)
26. Reynolds, A., Barbosa, H., Nötzli, A., Barrett, C.W., Tinelli, C.: cvc4sy: Smart and fast term enumeration for syntax-guided synthesis. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11562, pp. 74–83. Springer (2019). https://doi.org/10.1007/978-3-030-25543-5_5, https://doi.org/10.1007/978-3-030-25543-5_5
27. Reynolds, A., Deters, M., Kuncak, V., Tinelli, C., Barrett, C.: Counterexample-guided quantifier instantiation for synthesis in smt. In: Kroening, D., Păsăreanu, C.S. (eds.) Computer Aided Verification. pp. 198–216. Springer International Publishing, Cham (2015)
28. Reynolds, A., Deters, M., Kuncak, V., Tinelli, C., Barrett, C.: Counterexample-guided quantifier instantiation for synthesis in smt. In: Kroening, D., Păsăreanu, C.S. (eds.) Computer Aided Verification. pp. 198–216. Springer International Publishing, Cham (2015)
29. Reynolds, A., Tinelli, C., Goel, A., Krstić, S.: Finite model finding in smt. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 640–655. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
30. Reynolds, A., Tinelli, C., de Moura, L.: Finding conflicting instances of quantified formulas in smt. In: 2014 Formal Methods in Computer-Aided Design (FMCAD). pp. 195–202 (2014). https://doi.org/10.1109/FMCAD.2014.6987613
31. Sutcliffe, G.: The tptp problem library and associated infrastructure. Journal of Automated Reasoning **43**(4), 337–362 (2009)