Augmenting Model-Based Instantiation with Fast Enumeration

Lydia Kondylidou[1], Andrew Reynolds[2], and Jasmin Blanchette[1]

[1] Ludwig-Maximilians-Universität München, Munich, Germany
{l.kondylidou,jasmin.blanchette}@lmu.de
[2] The University of Iowa, Iowa City, United States
andrew.j.reynolds@gmail.com

Abstract. Satisfiability modulo theories (SMT) solvers rely on various quantifier instantiation strategies to support first- and higher-order logic. 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, our approach 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 [16], Boolector [17], cvc5 [2], veriT [7], and Z3 [15], 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].

Quantifier instantiation is a technique whereby quantified variables in a formula are instantiated with ground terms until a contradiction is found or all necessary instantiations have been generated. Consider the unsatisfiable axiom $(\forall x.\ \mathsf{p}\ x) \land \neg \mathsf{p}\ \mathsf{b}$. The SAT solver first finds a model that makes $\forall x.\ \mathsf{p}\ x$ true and $\mathsf{p}\ \mathsf{b}$ false, taken as black boxes. Then the quantifier instantiation strategy of the SMT solver might heuristically instantiate x with a, resulting in the formula $(\forall x.\ \mathsf{p}\ x) \implies \mathsf{p}\ \mathsf{a}$, which is conjoined with the axiom. Next, the SAT solver finds a new model that also makes $\mathsf{p}\ \mathsf{a}$ true. At this point, the instantiation strategy might instantiate x with b, resulting in $(\forall x.\ \mathsf{p}\ x) \implies \mathsf{p}\ \mathsf{b}$. Since the SAT solver cannot make $\mathsf{p}\ \mathsf{b}$ both true and false at the same time, the axiom conjoined with the two additional formulas is unsatisfiable at the SAT level, meaning that the original axiom is unsatisfiable in the SMT logic.

One successful instantiation strategy is model-based quantifier instantiation (MBQI) [9]. 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) [18]. This 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. [21], but their approach was limited to selected first-order theories and did not handle higher-order logic. Our approach is resolutely pragmatic; it does not aim at completeness, which can be achieved using other instantiation strategies.

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. The raw evaluation data are publicly available online.[3] Our source code, along with instructions for reproducing the experiments, is also available online.[4]

2   Preliminaries

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

---

[3] https://doi.org/10.5281/zenodo.14627782
[4] https://doi.org/10.5281/zenodo.14604430

Higher-Order Logic. Monomorphic higher-order logic [1, 10], also called simple type theory [8], 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 $\rightarrow$ to two types: $\tau_1 \rightarrow \tau_2$. The type of Booleans is denoted by o.

The term language is based on Church's simply typed λ-calculus, where terms t, e are inductively defined as variables w, x, y, z, ..., function symbols p, f, constants a, b, c, term applications $t\, t'$, and λ-abstractions $\lambda x.\, t$, where x is the bound variable and t is the body. A variable is free in a term if it is not bound by a λ-abstraction. We let $\bar{x}$ stand for $x_1, x_2, \ldots, x_n$, where $n \geq 1$. Terms are syntactically equal modulo α-, β-, and η-conversion, meaning, for example, that $(\lambda x.\, x)$ c is equal to c. Throughout this work, we assume that terms are expressed in η-long β-normal form, meaning that η-expansion and β-reduction have been applied exhaustively. The type of a term t is of the form $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau$, where τ_i are the argument types and τ is the result type. If τ is of type o, which is the distinguished Boolean type, we call t a predicate. A term of type o is called a formula.

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 existential quantifiers are expressed in terms of $\forall$ using the equivalence $(\exists x.\, \phi) \Longleftrightarrow \neg(\forall x.\, \neg\phi)$. 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., $\forall x, y.\, \neg \forall z.\, \mathsf{p}\, x\, y\, z$.

Let $\mathcal{T}$ be a theory for a set of interpreted symbols, and let F be the input formula over $\mathcal{T}$. 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 $\mathcal{T}$-satisfiable set L of literals that propositionally satisfies F. If such an L exists, F is $\mathcal{T}$-satisfiable (i.e., satisfiable with respect to $\mathcal{T}$); otherwise, F is $\mathcal{T}$-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 $\mathcal{T}$-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 $\mathcal{T}$-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

---

Algorithm 1 The main SMT loop

1: **function** SMT-LOOP(F)
2:     find a set of literals L where
3:         — $L \models_{\mathrm{p}} F$, where L's atoms are a subset of atoms(F)
4:         — L can be partitioned into $(E, Q_{\mathrm{p}}, Q_{\mathrm{n}})$, where
5:             — E is ground and $\mathcal{T}$-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:     $A \leftarrow$ INSTANTIATION-ROUND(Q_{p}, E) $\cup$ SKOLEMIZATION-ROUND(Q_{n})
11:     **if** $A = \emptyset$ **then**
12:         **return** sat
13:     **else**
14:         **return** SMT-LOOP($F \cup A$)
15: **function** INSTANTIATION-ROUND($\{q_1, \ldots, q_{\mathrm{n}}\}, E$)
16:     $I \leftarrow \emptyset$
17:     **for each** $i \in \{1, \ldots, n\}$ **do**
18:         $I \leftarrow I \cup \{q_i \implies q_i\sigma \mid \sigma \in \text{INSTS}(q_i, E)\}$
19:     **return** I
20: **function** INSTS(q_i, E)
21:     **return** a set of substitutions for q_i based on (q_i, E)
22: **function** SKOLEMIZATION-ROUND($\{\neg q_1, \ldots, \neg q_{\mathrm{n}}\}$)
23:     $K \leftarrow \emptyset$
24:     **for each** $i \in \{1, \ldots, n\}$ **do**
25:         $K \leftarrow K \cup \{\neg q_i \implies \neg q_i\sigma_{k,i}\}$, where $\sigma_{k,i}$ maps to Skolem constants for q_i
26:     **return** K

---

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 $\mathcal{T}$-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 $a : Int$ and $p : Int \rightarrow Bool$. Let F be the formula $(\forall y.\, p\ y) \wedge \neg p\ a$, where $y : Int$. In the SMT loop, the sets $E = \{\neg p\ a\}$, $Q_p = \{\forall y.\, p\ y\}$, and $Q_n = \emptyset$ are defined. The naive instantiation strategy produces substitutions mapping y to terms from E of the same type as y, as shown below:

$q \in Q_p$	$\text{INSTS}(E, q)$
$\forall y.\, p\ y$	$\{\{y \mapsto a\}\}$

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

The most widely used strategy for quantifier instantiation is *E-matching* [14]. This is a heuristic and typically incomplete technique that chooses substitutions by matching ground terms with *patterns*. In Example 3, the ground term $p\ a$ matches $p\ y$ for substitution $\{y \mapsto a\}$, and hence this substitution is returned by the strategy. Over the past decade, SMT solvers have been extended with more sophisticated approaches. *Conflict-based instantiation* [26] is another incomplete technique that attempts to find a single instantiation that would induce a ground conflict, before resorting to E-matching. *MBQI* [9] is a technique for finding instantiations that refute a candidate model, and is typically run when E-matching saturates. This strategy is also able to answer "$\mathcal{T}$-satisfiable" when no such instantiation can be found. *Enumerative instantiation* [22] 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. This has similar properties to MBQI but is more tailored for unsatisfiable instances.

Quantifier instantiation strategies that target specific theories have also been proposed. *Counterexample-guided instantiation* [24] is complete for specific theories with quantifiers that admit quantifier elimination, such as linear arithmetic and bit vectors. *SyQI* [18] 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.

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 $\mathcal{T}$-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 $y_i^{M'}$ represents a term denoting the interpretation for the variable y_i in M'.

Example 5. Let $a : Int$ and $p : Int \rightarrow Bool$. Let F be the input formula

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

where $y : Int$. In the SMT loop, our set of literals L is partitioned into $E = \{0 < a < 2, \neg p\ a\}$, $Q_p = \{\forall y.\, p\ y\}$, and $Q_n = \emptyset$. MBQI builds a model M from E—assume $a^M = 1$ and $p^M = \lambda x.\, x \neq 1$. (We abuse notation by denoting values using λ-terms.) It then considers the negation of the body of the quantified

---
Algorithm 4 The MBQI strategy

1: **function** INSTS_MBQI(q, E)
2:     **assume** q is $\forall y_1, \ldots, y_n.\ \phi$
3:     **let** M be a model of E
4:     **if** $\neg \phi^M$ is unsatisfiable **then**
5:         **return** $\emptyset$
6:     **let** M' be a model of $\neg \phi^M$
7:     **return** $\{\{y_1 \mapsto y_1^{M'}, \ldots, y_n \mapsto y_n^{M'}\}\}$
---

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 $(\forall y.\mathsf{p}\ y) \implies \mathsf{p}\ 1$ is constructed and added to F. Now, the quantifier-free solver finds a contradiction.

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

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

where $x : Int$ and $y : Int \to Int$. 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 and 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 $\mathcal{T}$-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 strategy produces a useless instance according to the current model, it 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 extends our approach's 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 $\mathcal{T}$-satisfiable. This check serves to maintain the invariant that the negation of the body of the quantified formula, after applying the current substitution, remains $\mathcal{T}$-satisfiable. In other words, we want to ensure that the generated instantiation refutes the current model (cf. line 6 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.

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

---

Algorithm 7 The MBQI-Enum strategy

1: **function** INSTS_MBQI_FAST_SYGUS(q, E)
2:      **assume** q is $\forall y_1, \ldots, y_n. \phi$
3:      **let** $\Sigma \leftarrow$ INSTS_MBQI(q, E)
4:      **if** $\Sigma = \emptyset$ **then**
5:          **return** $\emptyset$
6:      **else**
7:          **let** $\Sigma \leftarrow \{\sigma\}$
8:      **for each** $i \in \{1, \ldots, n\}$ **do**
9:          **let** $G_i \leftarrow$ CHOOSE_GRAMMAR(q, y_i)
10:          **for each** $j \in \{1, 2, \ldots\}$ **do**
11:              **let** $e \leftarrow$ GET_ENUM_TERM(G_i, j)
12:              **if** e does not exist **then**
13:                  **break**
14:              $\sigma' \leftarrow \sigma[y_i \mapsto e]$
15:              **if** $\neg \phi \sigma'$ is sat **then**
16:                  $\sigma \leftarrow \sigma'$
17:                  **break**
18:      **return** σ
19: **function** CHOOSE_GRAMMAR(q, y_i)
20:      **let** F be the original input formula
21:      **let** $S \leftarrow \emptyset$
22:      **if** option_syms_global **then**
23:          $S \leftarrow S \cup \text{symbols}(F)$
24:      **if** option_syms_local **then**
25:          $S \leftarrow S \cup \text{symbols}(q)$
26:      **if** option_ext_vars **then**
27:          $S \leftarrow S \cup \{y_{i+1}, \ldots, y_n\}$
28:      **return** grammar that generates terms of the same sort as y_i over symbols in S

---

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, since we use η-long β-normal form, 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 of 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 some of the grammar's other nonterminal symbols. 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 $\mathsf{a} : Int$ and $\mathsf{p} : Int \rightarrow Bool$. Let F be the input formula

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

where $y : Int \rightarrow Int$. 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 $\{y \mapsto \lambda x. \, 0\}$, $\{y \mapsto \lambda x. \, 1\}$, $\ldots$. 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. MBQI-Enum first constructs a grammar for y. The set of symbols S is empty. The grammar is

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

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. The strategy then checks whether the negation of the body of the quantified formula after applying σ' is $\mathcal{T}$-satisfiable. Indeed, if we instantiate y with $\lambda x. \, x$ and β-reduce, we obtain $\mathsf{p} \, \mathsf{a}$, which is $\mathcal{T}$-satisfiable. The substitution σ is then updated to $\{y \mapsto \lambda x. \, x\}$ and returned. Back in the SMT loop, the instantiation lemma $(\forall y. \neg \mathsf{p} \, (y \, \mathsf{a})) \implies \neg \mathsf{p} \, \mathsf{a}$ 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	$\mathrm{M}(q, E)$	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 most successful configuration enables *ext_vars* and

syms_local and leaves *syms_global* disabled. The following examples are based on this configuration.

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

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

where $y : Int \to Int$ and $z : Int$. 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 $\{y \mapsto \lambda x. \, 0\}, \{y \mapsto \lambda x. \, 1\}, \dots$, and the solver goes on forever. 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

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

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 $\{a, b\}$. The grammar for x follows:

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

(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:

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

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

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

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

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

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

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, default 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 by Reynolds et al. [23] 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 on both higher- and first-order benchmarks.

Setup. As the base configuration, we use the setup that we found to be the most successful in a preliminary evaluation: 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 [22]; cvc5[s], which uses SyQI [18]; cvc5[c], which uses counterexample-guided instantiation [24]; and cvc5[m], which uses MBQI [9].

Additionally, for higher-order problems, we also include a comparison with the state-of-the-art provers Vampire [6] and Zipperposition [28]. For Vampire, we used its portfolio mode, while Zipperposition was run in its so-called "best" mode, since it does not include a portfolio.

Next, we compare the base configuration on first-order benchmarks with three state-of-the-art SMT solvers: Z3 [15], the only SMT solver besides cvc5 that supports all the logics handled by our implementation; Bitwuzla [16], which supports only logics without arithmetic; and Boolector [17], which implements the most closely related approach to ours, counterexample-guided model synthesis [21], but focuses only on the theory of bit vectors.

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 40-core Intel Xeon Silver 4114 processor at 2.20 GHz and with 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 and provers 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

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

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 [27]. 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 most successful 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 instead of a portfolio. Remarkably, our strategy manages to solve 129 satisfiable problems, whereas Vampire solves only 6, and Zipperposition none.

Although our approach is based on both MBQI and SyQI, it is considerably stronger than either of those techniques used individually. Specifically, compared with MBQI, which was until now the most successful strategy in cvc5, MBQI-Enum solves an additional 41 problems without any losses. When compared with SyQI, our strategy solves 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, 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] from April 2024, focusing on logics that support quantifiers. We included logics involving theories such as floating-point arithmetic, linear and nonlinear arithmetic, and bit vectors. Overall, we consider the logics BV (bit vectors), FP (floating-point arithmetic), LIA (linear integer arithmetic), LRA (linear real arithmetic), NIA (nonlinear integer arithmetic), NRA (nonlinear real arithmetic), and their combinations: BVFP, BVFPLRA, and FPLRA. We also incorporate 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 and solvers on SMT-LIB benchmarks

Library	Boolector		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	SAT	UNSAT
BV	585	4980	**642**	5066	547	5001	202	4836	313	4911	437	**5190**	611	4835	594	5076
SAT+UNSAT	5565		**5708**		5548		5038		5224		5627		5446		5670	
ABV			378	47	387	103	21	**930**	618	237	17	90	**790**	135	731	157
ABVFP			24	0	29	2	10	**3**	15	0	13	0	**31**	0	**31**	0
ABVFPLRA			32	1	**55**	**3**	14	2	18	1	18	2	41	2	42	2
BVFP			**179**	**12**	164	5	26	**12**	102	3	106	0	167	4	169	7
BVFPLRA			**233**	**25**	205	18	80	**25**	119	24	111	24	220	24	226	**25**
FP			130	2176	17	1802	113	1591	99	2040	116	2015	115	2059	116	**2223**
FPLRA			**37**	0	23	0	20	0	24	0	22	0	**37**	0	36	0
UFBV			26	**120**	**43**	103	8	103	9	108	8	52	23	84	21	105
Subtotal			1681	7447	1470	7037	494	7502	1317	7324	848	7373	**2035**	7143	1966	**7595**
SAT+UNSAT			9128		8507		7996		8641		8221		9178		**9561**	
LIA					140	230	12	170	**150**	236	**150**	**266**	**150**	167	149	239
LRA					**760**	**1361**	468	1117	478	1130	593	1303	545	1123	557	1161
NIA					65	**144**	16	43	49	45	64	**144**	67	47	**80**	61
NRA					**3**	**3806**	1	3802	1	3785	**3**	3801	**3**	3712	**3**	3802
Total					2438	12 578	991	12 634	1995	12 520	1658	**12 887**	**2800**	12 192	2755	12 858
SAT+UNSAT					15 016		13 625		14 515		14 545		14 992		**15 613**	

The results, summarized in Table 3, show that our approach performs remarkably well against all other cvc5 configurations, as well as against Boolector, Bitwuzla, and Z3, across all SMT logics. Notably, it achieves the highest total count of benchmarks solved, surpassing the nearest competitor by 597 solved problems. Our strategy solves the most satisfiable problems in ABVFP and NIA and achieves the highest number of unsatisfiable benchmarks solved in FP.

For the evaluated SMT theories, our strategy is a clear improvement over previous instantiation strategies. When compared with MBQI, it successfully solves an additional 701 problems while incurring a loss of 79 problems across all logics. The raw evaluation data also reveals a notable reduction in timeouts, decreasing from 3491 to 2343.

Our strategy also substantially outperforms enumerative instantiation and SyQI across most benchmark categories. Both of these strategies 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 clearly better than

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	731	157	731	157	647	135	**737**	**159**
ABVFP	31	0	31	0	31	0	**32**	0
ABVFPLRA	**42**	2	**42**	2	41	2	38	2
BV	594	5076	594	5075	**616**	4843	600	**5087**
BVFP	169	**7**	169	**7**	167	4	**176**	2
BVFPLRA	226	**25**	226	**25**	220	24	**231**	**25**
FP	**116**	2223	**116**	2223	115	2092	**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	89	17	99
Total	2755	**12 858**	2755	12 857	2665	12 244	**2771**	12 854

MBQI on unsatisfiable benchmarks (+442), but they underperform on satisfiable benchmarks (−805). 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 Boolector, our strategy outperforms it on both satisfiable and unsatisfiable benchmarks, solving 105 more benchmarks overall. Compared with Bitwuzla and Z3, our approach performs very well across various logics, often closely matching or even surpassing both competitors. Overall, our strategy solves 433 more benchmarks than Bitwuzla and 597 more than Z3 in total. Bitwuzla is generally stronger for satisfiable problems, which is not surprising because MBQI-Enum is primarily designed for deriving contradictions. Z3's higher success rate for real arithmetic is likely attributable to its well-established instantiation strategies for these theories.

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

In summary, our approach is highly effective on first-order SMT-LIB benchmarks, solving the highest number of benchmarks. With 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 [14]. Conflict-based instantiation [3,11,26] can improve the solver's ability to answer "$\mathcal{T}$-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 [9] introduced MBQI, which is complete for certain fragments. Finite model finding [25] is a variant of this technique that targets quantified formulas whose domains are small and finite. Approaches for quantified formulas 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 [6] and Zipperposition [28], 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 [24] and quantified bit vectors [19]. 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. [22] 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 [12,13,20].

The closest related works to ours are counterexample-guided model synthesis [21] and SyQI [18], which both focus on enumerative approaches for finding useful instantiations in rich background logics. The former was implemented in the Boolector [17] 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. MBQI generates very specific instances; by resorting to a grammar, the terms in our instantiations are more abstract and therefore tend to lead to more useful instances. 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, vol. 27. 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: Fishman, D., Rosu, G. (eds.) TACAS 2022. LNCS, vol. 13243, pp. 415–442 (2022)
3. Barbosa, H., Fontaine, P., Reynolds, A.: Congruence closure with free variables. In: Legay, A., Margaria, T. (eds.) TACAS 2017, Part II. LNCS, vol. 10206, pp. 214–230. Springer (2017)
4. Barbosa, H., Reynolds, A., Ouraoui, D.E., Tinelli, C., Barrett, C.: Extending SMT solvers to higher-order logic. In: Fontaine, P. (ed.) CADE 2019. LNCS, vol. 11716, pp. 35–54. Springer (2019)
5. Barrett, C., Fontaine, P., Tinelli, C.: The satisfiability modulo theories library (SMT-LIB) (2016)
6. Bhayat, A., Suda, M.: A higher-order Vampire (short paper). In: Benzmüller, C., Heule, M.J.H., Schmidt, R.A. (eds.) IJCAR 2024. LNCS, vol. 14739, pp. 75–85. Springer (2024)
7. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-solver. In: Schmidt, R.A. (ed.) CADE 2009. LNCS, vol. 5663, pp. 151–156. Springer (2009)

8. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic **5**(2), 56–68 (1940)

9. Ge, Y., de Moura, L.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 306–320. Springer (2009)

10. Gordon, M.J.C.: Introduction to the HOL system. In: Archer, M., Joyce, J.J., Levitt, K.N., Windley, P.J. (eds.) TPHOL 1991. pp. 2–3. IEEE (1991)

11. 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.) VMCAI 2021. LNCS, vol. 12597, pp. 534–555. Springer (2021)

12. Jakubův, J., Janota, M., Piotrowski, B., Piepenbrock, J., Reynolds, A.: Selecting quantifiers for instantiation in SMT. In: Graham-Lengrand, S., Preiner, M. (eds.) SMT 2023. CEUR Workshop Proceedings, vol. 3429, pp. 71–77. CEUR-WS.org (2023)

13. Janota, M., Barbosa, H., Fontaine, P., Reynolds, A.: Fair and adventurous enumeration of quantifier instantiations. In: FMCAD 2021. pp. 256–260. IEEE (2021)

14. de Moura, L., Bjørner, N.: Efficient E-matching for SMT solvers. In: Pfenning, F. (ed.) CADE 2007. LNCS, vol. 4603, pp. 183–198. Springer (2007)

15. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)

16. Niemetz, A., Preiner, M.: Bitwuzla. In: Enea, C., Lal, A. (eds.) CAV 2023. LNCS, vol. 13965, pp. 3–17. Springer (2023)

17. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0. Journal on Satisfiability, Boolean Modeling and Computation **9**, 53–58 (2014)

18. Niemetz, A., Preiner, M., Reynolds, A., Barrett, C., Tinelli, C.: Syntax-guided quantifier instantiation. In: Groote, J.F., Larsen, K.G. (eds.) TACAS 2021. LNCS, vol. 12652, pp. 145–163. Springer (2021)

19. Niemetz, A., Preiner, M., Reynolds, A., Barrett, C.W., Tinelli, C.: Solving quantified bit-vectors using invertibility conditions. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018, Part II. LNCS, vol. 10982, pp. 236–255. Springer (2018)

20. Piepenbrock, J., Janota, M., Urban, J., Jakubův, J.: First experiments with neural cvc5. In: Bjørner, N.S., Heule, M., Voronkov, A. (eds.) LPAR 2024. EPiC Series in Computing, vol. 100, pp. 264–277. EasyChair (2024)

21. Preiner, M., Niemetz, A., Biere, A.: Counterexample-guided model synthesis. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 264–280. Springer (2017)

22. Reynolds, A., Barbosa, H., Fontaine, P.: Revisiting enumerative instantiation. In: Beyer, D. (ed.) TACAS 2018. LNCS, vol. 10806, pp. 112–131. Springer (2018)

23. 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.) CAV 2019, Part II. LNCS, vol. 11562, pp. 74–83. Springer (2019)

24. 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.) CAV 2015. LNCS, vol. 9207, pp. 198–216. Springer (2015)

25. Reynolds, A., Tinelli, C., Goel, A., Krstić, S.: Finite model finding in SMT. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 640–655. Springer (2013)

26. Reynolds, A., Tinelli, C., de Moura, L.: Finding conflicting instances of quantified formulas in SMT. In: FMCAD 2014. pp. 195–202. IEEE (2014)

27. Sutcliffe, G.: The TPTP problem library and associated infrastructure—from CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning **59**(4), 483–502 (2017)

28. Vukmirović, P., Bentkamp, A., Blanchette, J., Cruanes, S., Nummelin, V., Tour-ret, S.: Making higher-order superposition work. Journal of Automated Reasoning **66**(4), 541–564 (2022)