Enumerating Choice Terms in Model-Based Quantifier Instantiation

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

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

Abstract. Satisfiability modulo theories (SMT) solvers are widely used for determining the satisfiability of logical formulas with respect to background theories. SMT solvers are traditionally based on first-order logic, but some also support higher-order logic. Recently, Kondylidou et al. introduced model-based quantifier instantiation with fast enumeration (MBQI-Enum), a quantifier instantiation strategy that works for both logics. A weakness of MBQI-Enum is that it does not find refutations when Hilbert choice terms are necessary. In this work, we present an extension of MBQI-Enum that enables it to reason effectively about Hilbert's choice operator. The extended strategy substantially increases the success rate of the SMT solver cvc5 on higher-order benchmarks.

1 Introduction

Satisfiability modulo theories (SMT) solvers combine a Boolean satisfiability (SAT) solver with decision procedures for interpreted theories. They work by refutation: They assume the negation of the conjecture as an axiom and try to establish the unsatisfiability (i.e., provability) of the input problem. Several SMT solvers, including Bitwuzla [22], Boolector [23], CVC4 [5], cvc5 [3], veriT [8], and Z3 [20], support quantifiers via Skolemization and instantiation.

Most SMT solvers are based on first-order logic, but a few support higher-order logic. Specifically, CVC4, its successor cvc5, and a prototype version of veriT have been extended to parse and solve higher-order problems [4]. In principle, higher-order logic is often more convenient than first-order logic for expressing problems, particularly those involving binders (e.g., λx, $\sum_i$, $\prod_i$, $\int_x$).

A crucial aspect of higher-order SMT solvers is their use of quantifier instantiation strategies. Barbosa et al. [4] partly extended the first-order E-matching strategy [19] to a higher-order setting. Recently, Kondylidou et al. [18] introduced model-based quantifier instantiation with fast enumeration (MBQI-Enum), a strategy that extends model-based quantifier instantiation (MBQI) [14] with syntax-guided synthesis (SyGuS) [24] techniques. While traditional MBQI relies only on ground terms from the MBQI model to instantiate quantified variables, MBQI-Enum broadens this approach by generating a wider range of candidate instantiations. It does so by enumerating terms guided by a SyGuS grammar, which enables it to construct more complex instantiations such as identity functions and terms involving uninterpreted symbols. MBQI-Enum was found to be the most successful strategy [18, Sect. 5] for solving higher-order problems from the TPTP [33] library.

A weakness of MBQI-Enum is that it does not attempt instantiations containing Hilbert's choice operator. A Hilbert choice expression has the form $\varepsilon x.\ \varphi$, where variable x is bound in formula φ. The entire expression denotes some value of x that satisfies φ, if such a value exists; otherwise, it denotes an arbitrary value from the type of x. Sometimes the ε operator occurs in the input problem, in which case the SMT solver must reason about it, but even if it is absent, it might be useful in an instantiation. The ε operator is reputed to be difficult to reason about because it is characterized by a higher-order axiom.

In this paper, we extend MBQI-Enum so that it instantiates quantifiers with terms that include Hilbert's choice operator. This extension requires three main modifications to MBQI-Enum. First, we augment the SyGuS grammar to include ε terms. Second, we introduce fresh Skolem symbols that represent these ε terms, since this is simpler than implementing ε terms throughout the SMT solver. Finally, we generate lemmas corresponding to instances of ε's characteristic axiom and add them to the solver.

Example 1. We compare our extended MBQI-Enum with the original. Let u be an uninterpreted sort. Let F be the input formula consisting of the injectivity axiom $\forall x, y.\ \mathsf{f}\ x = \mathsf{f}\ y \Longrightarrow x = y$ and the negated conjecture $\forall g.\ \exists z.\ g\ (\mathsf{f}\ z) \neq z$, where g has type $u \to u$ and x, y, z have type u. MBQI-Enum generates the instantiations $\lambda y.\ y$, $\lambda y.\ \mathsf{f}\ y$, $\lambda y.\ \mathsf{f}\ (\mathsf{f}\ y)$, $\ldots$ for g. As a result, the SMT solver does not terminate. By contrast, our strategy instantiates g with the term $\lambda y.\ \varepsilon x.\ y = \mathsf{f}\ x$ using the grammar. This term denotes the left inverse of f. Then the strategy introduces a fresh Skolem symbol h that represents this term. The lemma $\forall y.\ \neg\,(\exists x.\ y = \mathsf{f}\ x) \vee y = \mathsf{f}\ (\mathsf{h}\ y)$ characterizing h is added to the problem. After instantiating g with h, the solver can derive a contradiction with the injectivity axiom.

We implemented the extended MBQI-Enum strategy in cvc5. We evaluated it on higher-order TPTP benchmarks. The results show that Hilbert's choice is often useful to establish the unsatisfiability of problems. Remarkably, our extension solves a strict superset of the problems solved by the original MBQI-Enum.

We also compare our approach with other cvc5 strategies and find that it outperforms them. Finally, we compare cvc5 equipped with our strategy with the provers Satallax [10], Vampire [7], and Zipperposition [35], representing the state of the art in higher-order automated theorem proving, and find that it is highly competitive. The raw evaluation data, along with our source code and instructions for reproducing the experiments, will be made available to the artifact evaluation committee.

2 Preliminaries

Our approach builds on higher-order logic with Hilbert's choice, SMT with quantifiers, and MBQI-Enum. Below, we briefly introduce these concepts.

Higher-Order Logic with Hilbert's Choice. Monomorphic higher-order logic [1, 15], also called simple type theory [11], 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 the simply typed λ-calculus, where terms t, e are inductively defined as variables $g, x, y, z, \ldots$, symbols $\mathsf{a}, \mathsf{b}, \mathsf{f}, \mathsf{h}, \mathsf{p}, \mathsf{r} \ldots$ (possibly of function type), 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 an enclosing λ-abstraction. Terms are syntactically equal modulo α-, β-, and η-conversion, meaning, for example, that $(\lambda x.\ \mathsf{f}\ x\ x)\ \mathsf{c}$ is syntactically equal to $\mathsf{f}\ \mathsf{c}\ \mathsf{c}$. Terms of type $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow o$ are called predicates. A term of type o is called a formula.

A common extension of higher-order logic is to add *Hilbert's choice operator* ε [16, 17, 31]: If x is a variable of type τ and φ is a formula that contains x, then $\varepsilon\ (\lambda x.\ \varphi)$, abbreviated as $\varepsilon x.\ \varphi$, is a term of type τ. Intuitively, the term $\varepsilon x.\ \varphi$ denotes *some* element x of type τ for which the formula φ holds if such an element exists. If no such element exists, it denotes an arbitrary element of type τ. Unlike existential quantification, which only asserts the existence of such an element, the choice operator yields the element. Hilbert's choice operator is characterized by the axiom $(\exists x.\ \varphi(x)) \implies \varphi(\varepsilon x.\ \varphi(x))$. The operator makes it possible to denote functions that exist according to the standard semantics of higher-order logic but that otherwise cannot be expressed (cf. Example 1). Some treatments of Hilbert's choice include an axiom $\forall y, z.\ y = z \implies (\varepsilon x.\ y\ x) = (\varepsilon x.\ z\ x)$, but this is a special case of congruence, which is built into higher-order logic.

Quantifier Instantiation in SMT. To reason about quantifiers, SMT solvers typically rely on a combination of Skolemization and heuristic instantiation: Universal quantifiers occurring positively are instantiated, while those occurring negatively are Skolemized. Dually, existential quantifiers occurring positively are Skolemized, whereas those occurring negatively are instantiated.

Let $\mathcal{T}$ be a theory over a set of interpreted symbols, and let F be the input formula (or problem) over $\mathcal{T}$. When the main loop of the SMT solver is entered, the formula F is split into two sets whose union is equisatisfiable with F: E, a set of ground (i.e., variable-free) formulas, and Q, a set of quantified formulas from F. The quantifier-free solver, which combines a SAT solver and a number of theory solvers, enumerates potential models of $E \cup Q$. The SAT solver proposes candidate models at the propositional level—potential assignments that satisfy the formula F—while the theory solvers check these models for theory-specific conflicts. Subsequently, the instantiation module handles the quantified formulas in Q by generating instances of them. This is performed using selected instantiation strategies that compute substitutions σ for each top-level variable in every quantified formula from Q, mapping them to ground terms. For each quantified formula $\forall x.\ \psi$ and substitution σ mapping x to a ground term t, the instantiation module produces lemmas of the form $(\forall x.\ \psi) \implies \psi\{x \mapsto t\}$. These lemmas are then added to the input formula F using conjunctions.

In addition to instantiation, the SMT solver applies Skolemization to existentially quantified formulas. For each quantified formula $\exists x.\ \psi$, a fresh Skolem symbol sk is introduced and a corresponding lemma of the form $(\exists x.\ \psi) \implies \psi\{x \mapsto \mathsf{sk}\}$ is generated. These lemmas are also added to the formula F. The main SMT loop is then reentered with the new F. This iterative process continues until the SAT solver either finds a satisfying model with respect to the theory or derives a contradiction. Of course, at any point, the SMT solver may run out of time or memory.

To generate relevant ground terms for instantiating quantifiers, SMT solvers rely on instantiation strategies. For first-order logic with theories, some strategies are refutationally complete and enable SMT solvers to offer semidecision procedures: They will find a proof if one exists, although they may not terminate otherwise. Higher-order logic—which allows terms to include functions as arguments, λ-abstractions, and partial applications—remains semidecidable under Henkin semantics but is more challenging to support in practice than first-order logic. The strategies implemented in CVC4, cvc5, and veriT are pragmatic and do not seek refutational completeness [4, 18].

For higher-order logic, cvc5 initially supported only an extension of the E-matching [19] instantiation strategy. New strategies that also partly work on higher-order problems were later added to the solver. Specifically, MBQI [14] is a strategy that iteratively refines a candidate model constructed from the quantifier-free part of the formula and then attempts to refute the model by generating ground instances of the quantified formulas, using the interpretation of the quantified variables in the model. It has been used on higher-order benchmarks and has proved somewhat effective on these [18, Sect. 5]. Common techniques such as conflict-guided instantiation [30], enumerative instantiation [27], and counterexample-guided instantiation [29] can also be used on higher-order problems, but they have generally proved less effective [18, Sect. 5], which is unsurprising given that they were originally designed for first-order reasoning. SyQI [24], a syntax-guided synthesis approach, works well on some higher-order problems since it can generate terms by synthesizing candidate expressions with the help of a grammar [18, Sect. 5].

MBQI-Enum. MBQI-Enum [18] is a recent addition to cvc5 that integrates a SyGuS-based enumerator directly into MBQI. For each quantified variable in the formula, MBQI-Enum first constructs a grammar. It then iteratively enumerates candidate terms generated from the grammar and checks each resulting instance against the current model. If the instantiation refutes the model, the strategy builds a substitution by mapping the quantified variable to the current enumerated term. If the instantiation fails to refute the model, the enumerator proceeds to the next candidate term. If all candidates are exhausted without success, the strategy reverts to standard MBQI. This fallback mechanism guarantees that MBQI-Enum can, in principle, solve any problem that MBQI can solve.

A key aspect of MBQI-Enum is the construction of the SyGuS grammar that guides the enumeration. The grammar is created from a set of symbols including uninterpreted symbols extracted from the entire formula and bound variables that have not yet been instantiated. For higher-order variables, the grammar incorporates rules for generating λ-abstractions. The enumerator then systematically constructs λ-expressions by generating terms over the abstractions' argument variables.

Example 2. Let u be an uninterpreted sort. Let f be a function symbol $\mathsf{f} : u \rightarrow u$. Let F be the input formula

$$\forall y.\ \exists z.\ y\ z \neq \mathsf{f}\ z$$

where y is a variable of type $u \rightarrow u$ and z is a variable of type u. The quantified formula is split into the set $E = \emptyset$ of ground formulas and the set $Q = \{\forall y.\ \exists z.\ y\ z \neq \mathsf{f}\ z\}$ of quantified formulas. MBQI-Enum first constructs a SyGuS grammar for y to guide the enumeration process. The set of symbols is empty. The grammar consists of the rules

$$\begin{aligned} g_0 &::= \lambda x.\ g_1 \\ g_1 &::= x \mid \mathsf{f}\ g_1 \mid \mathrm{ite}(g_2, g_1, g_1) \\ g_2 &::= \mathrm{true} \mid \mathrm{false} \mid g_1 = g_1 \mid \neg\, g_2 \mid g_2 \wedge g_2 \mid g_2 \vee g_2 \end{aligned}$$

In the first iteration, MBQI-Enum enumerates terms derived from the grammar and tries them in sequence. When the enumeration reaches the term $\lambda x.\ x$, MBQI-Enum generates the substitution $\sigma = \{y \mapsto \lambda x.\ x\}$. The strategy then determines whether the instantiation of the quantified formula, after applying σ, refutes the current model. This is achieved by checking whether the negation of the formula's body, under the current substitution, is $\mathcal{T}$-satisfiable. If we instantiate y with $\lambda x.\ x$, negate, and β-reduce, we obtain $z = \mathsf{f}\ z$, which is $\mathcal{T}$-satisfiable. The substitution σ is then returned. Back in the SMT loop, the instantiation lemma $(\forall y.\ \exists z.\ y\ z \neq \mathsf{f}\ z) \implies \exists z.\ z \neq \mathsf{f}\ z$ is added to F. The new F is split into $E = \emptyset$ and $Q = \{\forall y.\ \exists z.\ y\ z \neq \mathsf{f}\ z, \exists z.\ z \neq \mathsf{f}\ z\}$. Then, the formula $\exists z.\ z \neq \mathsf{f}\ z$ is Skolemized. The quantified variable z is replaced with a fresh Skolem symbol sk_1, and the Skolemization lemma $(\exists z.\ z \neq \mathsf{f}\ z) \implies \mathsf{sk}_1 \neq \mathsf{f}\ \mathsf{sk}_1$ is added to F. As a result, the set E is updated to $\{\mathsf{sk}_1 \neq \mathsf{f}\ \mathsf{sk}_1\}$.

In the next iteration, MBQI-Enum tries the term $\lambda x.\ \mathsf{f}\ x$ from the grammar and produces the substitution $\sigma = \{y \mapsto \lambda x.\ \mathsf{f}\ x\}$. The substitution σ is returned. The instantiation lemma $(\forall y.\ \exists z.\ y\ z \neq \mathsf{f}\ z) \implies \exists z.\ \mathsf{f}\ z \neq \mathsf{f}\ z$ is added to F. The new F is split into $E = \{\mathsf{sk}_1 \neq \mathsf{f}\ \mathsf{sk}_1\}$ and $Q = \{\forall y.\ \exists z.\ y\ z \neq \mathsf{f}\ z, \exists z.\ z \neq \mathsf{f}\ z, \exists z.\ \mathsf{f}\ z \neq \mathsf{f}\ z\}$. Then, the formula $\exists z.\ \mathsf{f}\ z \neq \mathsf{f}\ z$ is Skolemized. The quantified variable z is replaced with a fresh Skolem symbol sk_2, and the Skolemization lemma $(\exists z.\ \mathsf{f}\ z \neq \mathsf{f}\ z) \implies \mathsf{f}\ \mathsf{sk}_2 \neq \mathsf{f}\ \mathsf{sk}_2$ is added to F. As a result, the set E is updated to $\{\mathsf{sk}_1 \neq \mathsf{f}\ \mathsf{sk}_1, \mathsf{f}\ \mathsf{sk}_2 \neq \mathsf{f}\ \mathsf{sk}_2\}$. Now, the quantifier-free solver finds a contradiction.

Algorithm 3 Generalized MBQI-Enum

```
 1: function MBQI_ENUM_WITH_CHOICE($q$)
 2:     assume $q$ is $\forall y_1, \dots, y_n.\ \psi$
 3:     let $\sigma \leftarrow$ INSTS_MBQI($q$)
 4:     if $\sigma$ does not exist then
 5:         return $\emptyset$
 6:     for each $i \in \{1, \dots, n\}$ do
 7:         let $G_i \leftarrow$ MAKE_GRAMMAR($q, y_i$)
 8:         for each $j \in \{1, 2, \dots\}$ do
 9:             let $e \leftarrow$ GET_ENUM_TERM($G_i, j$)
10:             if $e$ does not exist then
11:                 break
12:             $\sigma' \leftarrow \sigma[y_i \mapsto e]$
13:             if $\neg\,\psi\sigma'$ is satisfiable then
14:                 $\sigma \leftarrow \sigma'$
15:                 break
16:     return GET_LEMMAS($\sigma$)
```

3 The Extended Strategy

Our strategy is based on MBQI-Enum, which has proved effective for higher-order reasoning. It adds three main ingredients:

- the grammar guiding the enumeration is augmented to include ε terms;
- fresh Skolem symbols are introduced to represent these terms; and
- term filtering is applied to eliminate redundant or invalid candidate terms during enumeration.

In addition to these extensions, our strategy generalizes MBQI-Enum by returning not only a substitution but also a set of lemmas. These lemmas are conjoined with the input formula F, extending it iteratively. The generalized strategy is detailed in Algorithm 3.

The strategy takes as input a quantified formula q of the form $\forall y_1, \dots, y_n.\ \psi$, where ψ does not start with a universal quantifier, and begins by computing an initial substitution σ from the model (line 3). If no such substitution exists, it returns the empty set, indicating that no instantiation could be found.

After computing σ, the strategy iterates over the quantified variables y_i in the formula q to instantiate. For each y_i, it constructs a SyGuS grammar G_i based on uninterpreted symbols appearing in the entire input formula (line 7). The grammar is further augmented to include terms of the form $\varepsilon x.\ \varphi$, where x is a variable whose type is the return type of y_i and φ is a formula over x and other free variables and symbols from the formula q. This grammar guides the enumeration of candidate terms that may be used to instantiate y_i. As a result, the SyGuS enumerator generates both ε and non-ε terms as potential substitutions for y_i.

For each candidate term e generated by the enumerator, a new substitution σ' is formed by mapping y_i to the current term e in the initial substitution σ

(line 12). The strategy then checks whether the negation of the body of the quantified formula under σ' is satisfiable in the current model (line 13). This satisfiability check serves as a semantic filter, pruning out terms that do not yield counterexamples. If the check succeeds, σ is updated and the enumeration process proceeds to the next quantified variable. Otherwise, it continues with the next candidate term. Once all quantified variables have been considered, our strategy returns both the final substitution σ and a set of lemmas, which include those introduced during the handling of ε terms and quantifier instantiations. These lemmas are subsequently added to the input formula F, extending it iteratively.

Augmented Grammar. Our strategy augments the base grammar used for term enumeration in MBQI-Enum by incorporating ε terms, allowing the generation of arbitrary formulas. Our approach is detailed in Algorithm 4.

MBQI-Enum constructs a base grammar for each universally quantified variable y_i in the input formula q using a set of symbols S from the entire input formula, such as uninterpreted symbols and bound variables (line 4). A grammar G is a triple (g_0, G, R_G), where g_0 is the initial symbol, G is a set of nonterminal symbols with $g_0 \in G$, and R_G is a set of production rules. Each rule has the form $g ::= t$, where $g \in G$ and t is a term built from symbols in the set S, nonterminals in G, free variables, and symbols from the signature of a background theory $\mathcal{T}$. The symbols in S, along with the free variables and the theory symbols from $\mathcal{T}$, act as terminal symbols in the grammar. Each grammar generates terms whose type matches the return type τ of y_i. For higher-order variables, the grammars include rules for generating λ-abstractions. In this case, the initial symbol g_0 expands to λ-terms. Whereas earlier versions of MBQI-Enum considered only terms in η-long β-normal form [18], the current cvc5 implementation allows partial application. The variables bound by λ-abstractions are terminal symbols in the grammar that generates the abstraction's body. For example, if y_i is a function variable of arity n whose arguments and result are all of the same type, the grammar includes rules such as

$$g_0 ::= \lambda x_1, \dots, x_n.\ g_1 \qquad\qquad g_1 ::= x_1 \mid \cdots \mid x_n$$

To include ε terms of the form $\varepsilon x.\ \varphi$, our strategy extends the grammar as follows. For the nonterminal g_1 in the base grammar corresponding to the type τ, a variable x of type τ is introduced (line 6) and added to the symbol set S (line 12). A new grammar is then built to generate Boolean formulas over the extended symbol set $S \cup \{x\}$ (line 7). This predicate grammar is defined as the triple (p_0, P, R_P). The symbol p_0 is the initial nonterminal, and P is a set of nonterminal symbols such that $p_0 \in P$. The set R_P contains production rules of the form $p ::= t$, where $p \in P$ and t is a term. Each term t is built from symbols in S, nonterminals in P, variables that were bound in y_i but now appear free, and symbols from the signature of the Boolean background theory (line 15). These elements act as terminal symbols in the grammar. The new grammar includes

Algorithm 4 Augmented grammar

```
 1: function MAKE_GRAMMAR($q, y_i$)
 2:     let $F$ be the original input formula
 3:     $S \leftarrow \text{symbols}(F) \cup \text{symbols}(q) \cup \{y_{i+1}, \dots, y_n\}$
 4:     let $(g_0, G, R_G) \leftarrow$ MAKE_BASE_GRAMMAR($S, y_i$)
 5:     if choice is enabled then
 6:         let $x$ be a variable of type $\tau$
 7:         $(p_0, P, R_P) \leftarrow$ MAKE_PREDICATE_GRAMMAR($S, x, y_i$)
 8:         extend $G \leftarrow G \cup P$
 9:         extend $R_G \leftarrow \{g_0 \leftarrow \varepsilon x.\ p_0\} \cup R_G \cup R_P$
10:     return $(g_0, G, R_G)$
11: function MAKE_PREDICATE_GRAMMAR($S, x, y_i$)
12:     $S \leftarrow S \cup \{x\}$
13:     let $\bar{z} \leftarrow$ bound variables in $y_i$
14:     define $p_0$ nonterminals
15:     let $P \leftarrow \{\bar{z}\} \cup S \cup \{\text{true}, \text{false}, =, \neg, \wedge\}$
16:     define $R_P \leftarrow$ production rules
17:     return $(p_0, P, R_P)$
```

the rules

$$p_0 ::= \text{true} \mid \text{false} \mid p_1 = p_1 \mid \neg\, p_0 \mid p_0 \wedge p_0 \qquad p_1 ::= x_1 \mid \cdots \mid x_n \mid x$$

The nonterminal p_0 directly defines the body of the ε term. We define a new production rule of the form $g_1 ::= \varepsilon x.\ p_0$. The resulting grammar is constructed by extending the base grammar (g_0, G, R_G) with the predicate grammar (p_0, P, R_P) and the new production rule (lines 8–9). The final grammar is defined as $(p_0, G \cup P, \{g_1 ::= \varepsilon x.\ p_0\} \cup R_G \cup R_P)$. It can be used to enumerate both ε and non-ε terms and is thus more expressive than the original grammar. For our example, the final grammar consists of the rules

$$g_0 ::= \lambda x_1, \dots, x_n.\ g_1 \qquad g_1 ::= \varepsilon x.\ p_0 \mid x_1 \mid \cdots \mid x_n$$
$$p_0 ::= \text{true} \mid \text{false} \mid p_1 = p_1 \mid \neg\, p_0 \mid p_0 \wedge p_0 \qquad p_1 ::= x_1 \mid \cdots \mid x_n \mid x$$

Building on the base grammar with ε terms, we experimented with an additional extension. By default, the ε-augmented grammar introduces ε terms only for the nonterminal corresponding to the return type of the variable being instantiated. In contrast, this extension generalizes the construction by introducing ε terms for the nonterminals corresponding to all argument and return types of the variable. This makes the grammar more expressive, enabling the enumeration of a wider range of candidate terms, but also enlarges the search space and increases enumeration overhead. Experimental evaluation showed that a configuration of cvc5 with this additional extension improves performance on certain benchmarks but is not uniformly beneficial; the extension is therefore not included in the pseudocode or in the evaluation presented in this paper.

Algorithm 5 Lemmas for Skolem symbols

```
1: function GET_LEMMAS($\sigma$)
2:     let $L \leftarrow \emptyset$
3:     assume $\sigma = \{y_i \mapsto \lambda(x_1, \ldots, x_n).\ t\}$        ▷ $y_i$ is the variable being instantiated
4:     let $t' \leftarrow t$
5:     let $\bar{z} = (z_1 : \tau_1, \ldots, z_k : \tau_k) \subseteq \{x_1, \ldots, x_n\}$ be $\varphi$'s free variables
6:     for all subterms $e$ of $t$ of the form $\varepsilon x : \tau.\ \varphi(\bar{z}, x)$ do
7:         let $h : (\tau_1, \ldots, \tau_k) \to \tau$ be a fresh Skolem symbol
8:         let $lemma \leftarrow \forall \bar{z}.\ \neg(\exists x.\ \varphi(\bar{z}, x)) \vee \varphi(\bar{z}, \mathsf{h}\ \bar{z})$
9:         $L \cup \{lemma\}$
10:        replace $e$ in $t'$ with $\mathsf{h}\ \bar{z}$
11:    let $inst\_lemma \leftarrow (\forall y_i.\ \psi) \implies \psi\{y_i \mapsto \lambda(x_1, \ldots, x_n).\ t'\}$
12:    $L \cup \{inst\_lemma\}$
13:    return $L$
```

Skolem Symbols. The most important part of the strategy is where we take formulas containing ε terms generated by the grammar, abstract these terms into fresh Skolem symbols, and generate corresponding lemmas with triggers to guide efficient higher-order quantifier instantiation. Our approach is detailed in Algorithm 5.

Our strategy generalizes MBQI-Enum by returning a set L of lemmas in addition to the substitution. Given a substitution $\sigma = \{y_i \mapsto \lambda x_1, \ldots, x_n.\ t\}$, consider any subterm e of t of the form $\varepsilon x : \tau.\ \varphi(\bar{z}, x)$. The variables $\bar{z} = (z_1 : \tau_1, \ldots, z_k : \tau_k)$ are a subset of $\{x_1, \ldots, x_n\}$. We replace all occurrences of e with an application of a fresh Skolem symbol h to the free variables $\bar{z}$ appearing in φ (lines 6–10). The introduction of these Skolems abstracts the ε operator into symbols that can be understood by the rest of the SMT solver. Specifically, the term $\varepsilon x.\ \varphi(\bar{z}, x)$ is converted to $\mathsf{h}\ \bar{z}$, where h has type $\tau_1 \to \cdots \to \tau_k \to \tau$. Our strategy asserts the lemma $\forall \bar{z}.\ \neg(\exists x.\ \varphi(\bar{z}, x)) \vee \varphi(\bar{z}, \mathsf{h}\ \bar{z})$ (line 8), whose addition preserves satisfiability and makes the strategy more complete. This lemma guarantees that if there exists an x satisfying $\varphi(\bar{z}, x)$, then $\mathsf{h}\ \bar{z}$ satisfies φ as well.

Moreover, we can mark the Skolem application $\mathsf{h}\ \bar{z}$ as a *trigger* [12,13] in the lemma. This trigger guides E-matching, ensuring that instantiations are driven by terms that match the structure of the Skolem application, which is applied to exactly the same set of variables as in the original quantified formula.

Afterward, the instantiation lemma $(\forall y_i.\ \psi) \implies \psi\{y_i \mapsto \lambda x_1, \ldots, x_n.\ t'\}$ is generated, where t' is the term with all ε subterm occurrences replaced by $\mathsf{h}\ \bar{z}$ (line 11). Both lemmas are added to the set L. Finally, L is returned and subsequently added to the input formula F.

Filtering. To avoid redundant instantiations, term filtering is applied during enumeration. Enumerated terms already present in the set of candidate terms are discarded, ensuring that terms equivalent under rewriting are considered only once. For terms of the form $\varepsilon x.\ \varphi$, the bound variable x must occur in the body φ.

Terms where x does not appear are excluded, preventing the introduction of trivial instantiations. Moreover, since cvc5 already performs extensive theory-specific filtering by default, no additional filtering is required in our setting.

Example 6. We return to Example 1 and present it in more detail. Let u be an uninterpreted sort. Let F be the input formula

$$(\forall x, y.\ \mathsf{f}\ x = \mathsf{f}\ y \Longrightarrow x = y) \Longrightarrow (\exists g.\ \forall z.\ g\ (\mathsf{f}\ z) = z)$$

where g has type $u \to u$ and x, y, z have type u. The formula states that if f is injective, then it has a left inverse. To prove F by contradiction, we negate it and obtain $(\forall x, y.\ \mathsf{f}\ x = \mathsf{f}\ y \Longrightarrow x = y) \wedge (\forall g.\ \exists z.\ g\ (\mathsf{f}\ z) \neq z)$. The quantified formula is split into the set $E = \emptyset$ of ground formulas and the set $Q = \{\forall x, y.\ \mathsf{f}\ x = \mathsf{f}\ y \Longrightarrow x = y,\ \forall g.\ \exists z.\ g\ (\mathsf{f}\ z) \neq z\}$ of quantified formulas.

MBQI-Enum generates a grammar for g using the set of symbols $S = \{\mathsf{f}\}$. The grammar consists of the rules

$$g_0 ::= \lambda y.\ g_1 \qquad\qquad g_1 ::= y \mid \mathsf{f}\ g_1$$

Based on this grammar, MBQI-Enum generates the substitutions $\{g \mapsto \lambda y.\ y\}$, $\{g \mapsto \lambda y.\ \mathsf{f}\ y\}$, $\{g \mapsto \lambda y.\ \mathsf{f}\ (\mathsf{f}\ y)\}$, ... for g. Since this enumeration does not capture inverse functions, the SMT solver does not terminate.

Our strategy addresses this by constructing a richer grammar. We introduce a variable x of type u and extend the symbol set to $S \cup \{x\}$. We define the following grammar:

$$p_0 ::= \text{true} \mid \text{false} \mid p_1 = p_1 \mid \neg\, p_0 \mid p_0 \wedge p_0 \qquad\qquad p_1 ::= y \mid x \mid \mathsf{f}\ p_1$$

We augment the MBQI-Enum grammar by adding a rule to produce $\varepsilon x.\ p_0$. The result consists of the rules for p_0 and p_1 above and the following rules:

$$g_0 ::= \lambda y.\ g_1 \qquad\qquad g_1 ::= \varepsilon x.\ p_0 \mid y \mid \mathsf{f}\ g_1$$

Using this grammar, our strategy generates the term $\lambda y.\ \varepsilon x.\ y = \mathsf{f}\ x$, which denotes a left inverse of f, leading to the substitution $\{g \mapsto \lambda y.\ \varepsilon x.\ y = \mathsf{f}\ x\}$.

Corresponding to this term, we introduce a fresh Skolem symbol $\mathsf{h} : u \to u$. Our strategy asserts the following lemma:

$$\forall y.\ \neg\, (\exists x.\ y = \mathsf{f}\ x) \vee y = \mathsf{f}\ (\mathsf{h}\ y) \qquad (\ell_1)$$

This lemma states that for any variable y, if there exists an x such that $y = \mathsf{f}\ x$ (i.e., y is in the image of f), then applying f to $\mathsf{h}\ y$ must yield y. The original enumerated term $\lambda y.\ \varepsilon x.\ y = \mathsf{f}\ x$ is thus rewritten to the term $\lambda y.\ \mathsf{h}\ y$. The substitution $\sigma = \{g \mapsto \lambda y.\ \mathsf{h}\ y\}$ is considered. Next, the instantiation lemma

$$(\forall g.\ \exists z.\ g\ (\mathsf{f}\ z) \neq z) \Longrightarrow (\exists z.\ \mathsf{h}\ (\mathsf{f}\ z) \neq z) \qquad (\ell_2)$$

is generated by applying the substitution σ followed by β-reduction. The set of lemmas $L = \{(\ell_1), (\ell_2)\}$ is constructed. Each lemma in L is then added to the input formula F. After, Skolemization introduces a fresh symbol sk corresponding to the existentially quantified variable z. The lemma

$$(\exists z.\ \mathsf{h}\ (\mathsf{f}\ z) \neq z) \Longrightarrow \mathsf{h}\ (\mathsf{f}\ \mathsf{sk}) \neq \mathsf{sk} \qquad (\ell_3)$$

is added to F. The new F is split into $E = \{\mathsf{h}\ (\mathsf{f}\ \mathsf{sk}) \neq \mathsf{sk}\}$ and $Q = \{\forall x{,}y.\ \mathsf{f}\ x = \mathsf{f}\ y \Longrightarrow x = y,\ \forall g.\ \exists z.\ g\ (\mathsf{f}\ z) \neq z,\ \forall y.\ \neg\,(\exists x.\ y = \mathsf{f}\ x) \lor y = \mathsf{f}\ (\mathsf{h}\ y),\ \exists z.\ \mathsf{h}\ (\mathsf{f}\ z) \neq z\}$.

The strategy proceeds to instantiate the quantified formulas in Q. Quantified variables whose type is not a function type are instantiated using terms from the set E. First, the strategy instantiates x and y in the injectivity axiom $\forall x{,}y.\ \mathsf{f}\ x = \mathsf{f}\ y \Longrightarrow x = y$ with sk and $\mathsf{h}\ (\mathsf{f}\ \mathsf{sk})$, respectively. Then it instantiates x and y in the lemma $\forall y.\ \neg\,(\exists x.\ y = \mathsf{f}\ x) \lor y = \mathsf{f}\ (\mathsf{h}\ y)$ with sk and $\mathsf{f}\ \mathsf{sk}$, respectively. These instantiations arise from E-matching, which matches each lemma's pattern against the terms already present in the equivalence classes maintained by the SMT solver. The instantiation lemmas

$$(\forall y.\ \neg\,(\exists x.\ y = \mathsf{f}\ x) \lor y = \mathsf{f}\ (\mathsf{h}\ y)) \Longrightarrow \mathsf{f}\ \mathsf{sk} \neq \mathsf{f}\ \mathsf{sk} \lor \mathsf{f}\ \mathsf{sk} = \mathsf{f}\ (\mathsf{h}\ (\mathsf{f}\ \mathsf{sk})) \qquad (\ell_4)$$
$$(\forall x, y.\ \mathsf{f}\ x = \mathsf{f}\ y \Longrightarrow x = y) \Longrightarrow \mathsf{f}\ \mathsf{sk} \neq \mathsf{f}\ (\mathsf{h}\ (\mathsf{f}\ \mathsf{sk})) \lor \mathsf{sk} = \mathsf{h}\ (\mathsf{f}\ \mathsf{sk}) \qquad (\ell_5)$$

are added to F. The set E is updated to $\{\mathsf{h}\ (\mathsf{f}\ \mathsf{sk}) \neq \mathsf{sk},\ \mathsf{f}\ \mathsf{sk} = \mathsf{f}\ (\mathsf{h}\ (\mathsf{f}\ \mathsf{sk})),\ \mathsf{f}\ \mathsf{sk} \neq \mathsf{f}\ (\mathsf{h}\ (\mathsf{f}\ \mathsf{sk})) \lor \mathsf{sk} = \mathsf{h}\ (\mathsf{f}\ \mathsf{sk})\}$. Now, the quantifier-free solver finds a contradiction.

Example 7. Let u_1, u_2 be uninterpreted sorts. Let F be the input formula

$$(\forall x.\ \exists y.\ \mathsf{r}\ x\ y) \Longrightarrow (\exists g.\ \forall x.\ \mathsf{r}\ x\ (g\ x))$$

where g has type $u_1 \to u_2$, x has type u_1, and y has type u_2. The formula states that for every total relation r on $u_1 \times u_2$, there exists a corresponding function from u_1 to u_2. To prove F by contradiction, we negate it and obtain $(\forall x.\ \exists y.\ \mathsf{r}\ x\ y) \land (\forall g.\ \exists x.\ \neg\,\mathsf{r}\ x\ (g\ x))$. The quantified formula is split into the set $E = \emptyset$ of ground formulas and the set $Q = \{\forall x.\ \exists y.\ \mathsf{r}\ x\ y,\ \forall g.\ \exists x.\ \neg\,\mathsf{r}\ x\ (g\ x)\}$ of quantified formulas.

MBQI-Enum generates a grammar for g using the set of symbols $S = \{\mathsf{r}\}$. The grammar consists of the rules

$$g_0 ::= \lambda y.\ g_1 \qquad\qquad g_1 ::= t$$

Based on this grammar, MBQI-Enum generates the substitution $\{g \mapsto \lambda y.\ t\}$, where t is a ground term of type u_2. As a result, the SMT solver terminates with an unknown status.

In contrast, with our extension, the solver behaves as follows. Let x be a variable of type u_2. We extend the symbol set to $S \cup \{x\}$. Consider the following rules:

$$\begin{aligned} p_0 &::= \text{true} \mid \text{false} \mid p_1 = p_1 \mid p_2 = p_2 \mid \mathsf{r}\ p_1\ p_2 \mid \neg\,p_2 \mid p_2 \land p_2 \\ p_1 &::= y \\ p_2 &::= x \end{aligned}$$

We augment the MBQI-Enum grammar by adding a rule to produce $\varepsilon x.\ p_0$. The resulting grammar consists of the rules for p_0, p_1, and p_2 above together with

$$g_0 ::= \lambda y.\ g_1 \qquad\qquad g_1 ::= \varepsilon x.\ p_0 \mid t$$

Using this grammar, our strategy generates the term $\lambda y.\ \varepsilon x.\ \mathsf{r}\ y\ x$, leading to the substitution $\{g \mapsto \lambda y.\ \varepsilon x.\ \mathsf{r}\ y\ x\}$.

Corresponding to this term, we introduce a fresh Skolem symbol $\mathsf{h} : u_1 \to u_2$. Our strategy asserts the following lemma:

$$\forall y.\ \neg\,(\exists x.\ \mathsf{r}\ y\ x) \lor \mathsf{r}\ y\ (\mathsf{h}\ y) \tag{ℓ_1}$$

The original enumerated term $\lambda y.\ \varepsilon x.\ \mathsf{r}\ y\ x$ is thus rewritten to the term $\lambda y.\ \mathsf{h}\ y$. The substitution $\sigma = \{y \mapsto \lambda y.\ \mathsf{h}\ y\}$ is considered. Next, the instantiation lemma

$$(\forall g.\ \exists x.\ \neg\,\mathsf{r}\ x\ (g\ x)) \Longrightarrow (\exists x.\ \neg\,\mathsf{r}\ x\ (\mathsf{h}\ x)) \tag{ℓ_2}$$

is generated by applying the substitution σ followed by β-reduction. The set of lemmas $L = \{(\ell_1), (\ell_2)\}$ is constructed. Each lemma in L is then added to the input formula F. After, Skolemization introduces a fresh symbol sk corresponding to the existentially quantified variable x. The lemma

$$(\exists x.\ \neg\,\mathsf{r}\ x\ (\mathsf{h}\ x)) \Longrightarrow \neg\,\mathsf{r}\ \mathsf{sk}\ (\mathsf{h}\ \mathsf{sk}) \tag{ℓ_3}$$

is added to F. The new F is split into $E = \{\neg\,\mathsf{r}\ \mathsf{sk}\ (\mathsf{h}\ \mathsf{sk})\}$ and $Q = \{\forall x.\ \exists y.\ \mathsf{r}\ x\ y,\ \forall g.\ \exists x.\ \neg\,\mathsf{r}\ x\ (g\ x),\ \forall y.\ \neg\,(\exists x.\ \mathsf{r}\ y\ x) \lor \mathsf{r}\ y\ (\mathsf{h}\ y),\ \exists x.\ \neg\,\mathsf{r}\ x\ (\mathsf{h}\ x)\}$.

The strategy proceeds to instantiate the quantifiers in Q. Quantified variables of nonfunction type are instantiated using terms from the set E. First, the strategy instantiates x and y in the lemma $\forall y.\ \neg\,(\exists x.\ \mathsf{r}\ y\ x) \lor \mathsf{r}\ y\ (\mathsf{h}\ y)$ with $\mathsf{h}\ \mathsf{sk}$ and sk, respectively. Then it instantiates x and y in the axiom $\forall x.\ \exists y.\ \mathsf{r}\ x\ y$ with sk, and $\mathsf{h}\ \mathsf{sk}$, respectively. These instantiations arise from E-matching, which matches each lemma's pattern against the terms already present in the equivalence classes maintained by the solver. The instantiation lemmas

$$(\forall y.\ \neg\,(\exists x.\ \mathsf{r}\ y\ x) \lor \mathsf{r}\ y\ (\mathsf{h}\ y)) \Longrightarrow \neg\,\mathsf{r}\ \mathsf{sk}\ (\mathsf{h}\ \mathsf{sk}) \lor \mathsf{r}\ \mathsf{sk}\ (\mathsf{h}\ \mathsf{sk}) \tag{ℓ_4}$$

$$(\forall x.\ \exists y.\ \mathsf{r}\ x\ y) \Longrightarrow \mathsf{r}\ \mathsf{sk}\ (\mathsf{h}\ \mathsf{sk}) \tag{ℓ_5}$$

are added to F. The set E is updated to $\{\neg\,\mathsf{r}\ \mathsf{sk}\ (\mathsf{h}\ \mathsf{sk}),\ \mathsf{r}\ \mathsf{sk}\ (\mathsf{h}\ \mathsf{sk})\}$. Now, the quantifier-free solver finds a contradiction.

4 Implementation

We developed our strategy as an extension of cvc5's implementation of MBQI-Enum. Our extension refines the selection of grammars for term enumeration, introduces a mechanism for managing choice terms generated by these grammars, and incorporates a filtering step to discard redundant or unsuitable terms.

For each quantified variable, MBQI-Enum constructs a SyGuS grammar that specifies the space of candidate instantiations (Algorithm 4, line 4). Our implementation augments these grammars with terminal rules for generating choice terms. Specifically, for each nonterminal type that qualifies, we add a corresponding expression of the form $\varepsilon x.\ \varphi$, which selects a value satisfying a predicate over that type (lines 5–9). This allows the enumerator to synthesize Skolem-style instantiations directly from the grammar.

We use the existing implementation of the fast SyGuS-based enumerator for term generation. This enumerator considers only one term from each equivalence class up to rewriting [28]. During the enumeration process, ε terms are constructed within cvc5's internal abstract syntax tree representation, thereby exploiting cvc5's built-in rewriter. For example, $\varepsilon x.\ x = t$, where x does not occur free in t, is rewritten to t.

Beyond these rewrite-based simplifications, our implementation applies additional heuristics for filtering ε terms. In particular, it explicitly discards any term of the form $\varepsilon x.\ \varphi$ where x does not occur free in φ. These heuristics are implemented via a callback from the SyGuS enumerator.

The enumerator constructs ε terms that are immediately purified by our implementation. Recall that each ε term is replaced with a fresh Skolem symbol, and an accompanying lemma is generated to relate the Skolem to the original ε term (Algorithm 5, lines 6–10). For example, given a term $\varepsilon x.\ \varphi$, the procedure introduces a fresh Skolem h and asserts that either $\varphi(\mathsf{h})$ holds or the quantified condition that motivated the ε term is already satisfied. An important heuristic is that our approach designates the Skolem application itself as a trigger for the generated quantified formula, thereby ensuring that instantiations during solving are guided by the intended terms.

During instantiation, terms containing ε terms are first normalized through the purification and filtering procedures described above before being considered as candidate substitutions. The generalized strategy, shown in Algorithm 3, iteratively builds substitutions by enumerating terms from the augmented grammars (lines 6–12), tests them in the current model (line 13), and commits suitable assignments (line 14). Notably, unlike the standard MBQI-Enum procedure, our strategy returns not only substitutions but also auxiliary lemmas (line 16).

5 Evaluation

We extensively evaluated our cvc5 implementation of the extended MBQI-Enum on higher-order benchmarks.

Since the publication of Kondylidou et al. [18], MBQI-Enum has been developed further independently of our work on choice. In our evaluation, by "original MBQI-Enum" we mean the most recent version of the strategy, excluding our modifications related to ε. This strategy includes the following enhancements:

- MBQI-Enum now supports partial applications, allowing the strategy to consider candidate terms that partially apply functions during enumeration.

- Timeouts have been added to subsolver invocations used for checking candidate instantiations, preventing individual candidates from stalling the overall SMT loop. In addition, there is an option that completely prohibits nested MBQI calls, which avoids potential nontermination caused by nested subsolver invocations.
- MBQI-Enum now incorporates a guess-and-test loop with a progress-aware instantiation cache: When a proposed instantiation fails to refute the current model, it is cached to avoid rechecking the same substitution later, thereby ensuring that the strategy always makes progress.
- Skolemization is now performed eagerly for negatively occurring universally quantified variables that appear earlier in the formula than positively occurring ones.
- MBQI-Enum now ensures that once all but the last variable of a quantified formula have been assigned consistent candidates, the remaining variable is always instantiated, guaranteeing that the process produces complete instantiations.

Setup. We denote our configuration of MBQI-Enum extended with choice by cvc5[C]. We compare our strategy against several established quantifier instantiation strategies in cvc5: cvc5[s], which uses SyQI [24]; cvc5[m], which implements MBQI [14]; cvc5[M], which implements the original MBQI-Enum [18]; cvc5[hoelim], which eagerly rewrites higher-order constraints into first-order form and applies full quantifier saturation; and cvc5[foinst], which relies on exhaustive first-order-style instantiation and axiomatic handling of higher-order applications.

We also include a comparison with the state-of-the-art provers Satallax [10], Vampire [7], and Zipperposition [35]. Satallax was run with its default settings. Vampire was run in portfolio mode, as its higher-order reasoning configuration automatically enables this mode and disabling it makes the prover much less competitive. For Zipperposition, we enabled several options that were extensively evaluated by Bozec and Blanchette [9] and shown to achieve the best overall performance on higher-order benchmarks.

All experiments were conducted on a machine with a 40-core Intel Xeon Silver 4114 CPU running at 2.20 GHz, equipped with 192 GB of RAM and running Debian 12 (Bookworm). Each benchmark was executed with a timeout of 60 seconds.

Higher-Order Problems. The experiments were carried out on monomorphic higher-order problems (TH0) from version 9.1.0 of the TPTP library [33]. The benchmark set consists of 3757 problems. From the 3962 TH0 problems, we excluded 205 benchmarks that one or more systems could not parse (e.g., because of the use of interpreted arithmetic).

To support reasoning with choice, we implemented external parser support for the TPTP operators for ε and the definite description operator ι. Although the TPTP library contains very few problems with choice constructs, they might arise in users' problems in practice.

Table 1. Extended MBQI-Enum vs. other provers on TPTP TH0 benchmarks

	Satallax	Vampire	Zipperposition	cvc5[C]
Satisfiable	196	14	0	**204**
Unsatisfiable	2162	**2303**	2083	2178
Total	2358	2317	2083	**2382**
Unknown	15	16	0	154
Timeouts	1384	1424	1674	1221

Table 2. Extended MBQI-Enum vs. other cvc5 strategies on TPTP TH0 benchmarks

	cvc5[hoelim]	cvc5[foinst]	cvc5[s]	cvc5[m]	cvc5[M]	cvc5[C]
Satisfiable	14	37	88	188	204	**204**
Unsatisfiable	2082	2126	1657	2074	2155	**2178**
Total	2096	2163	1745	2262	2359	**2382**
Unknown	0	125	45	289	158	154
Timeouts	1661	1469	1967	1206	1240	1221

The results are shown in Table 1 and Table 2, where bold indicates the most successful system. Notably, our approach achieves the highest total count of solved benchmarks, surpassing the nearest competitor by 24 solved problems and the baseline MBQI-Enum by 23.

Table 1 compares our strategy (cvc5[C]) with external state-of-the-art provers. Here, our approach achieves the highest total number of solved benchmarks, with 2382 problems, surpassing Satallax, Vampire, and Zipperposition. It outperforms the nearest competitor by 24 benchmarks. In particular, our strategy solves 204 satisfiable problems, while Satallax solves 196, Vampire solves 14, and Zipperposition solves none. For unsatisfiable problems, Vampire performs the best, solving 2303 benchmarks, whereas our strategy solves 2178, Satallax 2162, and Zipperposition 2083. Overall, our strategy demonstrates strong performance across both satisfiable and unsatisfiable problems compared with the state-of-the-art provers.

Table 2 compares cvc5[C] with other cvc5 quantifier instantiation strategies. Our configuration solves 23 unsatisfiable benchmarks that the baseline MBQI-Enum cannot, with no benchmarks lost, while solving the same number of satisfiable problems (204). This shows that extending MBQI-Enum with choice consistently improves solver performance. Overall, when compared against all other available quantifier instantiation strategies, our strategy achieves the strongest performance, demonstrating that it is considerably more effective on higher-order benchmarks than any other individual technique in cvc5.

In addition to its overall performance, our technique also solves several problems not solved by its competitors: 143 problems not solved by Vampire, 296 not solved by Zipperposition, 235 not solved by Satallax, and 21 not solved by any other cvc5 strategy. When considering all systems, cvc5[C] solves exactly one benchmark that no other system can solve.

6 Related Work

Besides MBQI-Enum, other quantifier instantiation strategies work on higher-order problems, notably E-matching, MBQI, and SyQI. In higher-order E-matching, quantified variables are instantiated by matching their patterns against ground terms available in the current context—that is, the set of ground terms introduced during proof search. This technique has been extended to support higher-order features, such as functional arguments [3]. Higher-order MBQI constructs candidate models that provide interpretations for functions. This enables the solver to generate ground instantiations of quantified formulas in an attempt to refute the current model [14]. Higher-order SyQI further improves the instantiation process by selecting candidate terms using syntactic templates and heuristics derived from the problem's structure, guided by a fixed grammar [24]. None of these approaches provides special support for Hilbert's choice.

Other higher-order provers support Hilbert's choice in various ways. Satallax implements specialized tableau inference rules that directly reason about ε terms in a sound and refutationally complete way [2]. Leo-III integrates ε into its extensional higher-order paramodulation calculus and supports Henkin semantics with choice [32]. Zipperposition includes a dedicated choice axiom within its higher-order superposition calculus, allowing it to reason about ε terms as needed [6]. Similarly, Vampire supports Hilbert's choice either through a Leo-III-style inference or by introducing the choice axiom [7].

Hilbert's choice is also present in proof assistants such as Isabelle/HOL [25], Lean [21], and Rocq [34] (formerly known as Coq). In Isabelle/HOL, the choice operator is introduced as an axiom as part of the system's core libraries. In systems based on dependent type theory, such as Lean and Rocq, choice is introduced via classical axioms, which should be avoided in constructive definitions and proofs. Even in Isabelle/HOL, some users, such as Paulson [26], prefer to avoid choice:

> Pragmatists may argue that in verification nobody cares whether choice is used or not. However, pragmatists should be concerned that reasoning about ε-terms is tricky.

Reasoning about choice is indeed tricky, but cvc5 can do it now.

7 Conclusion

We presented an SMT quantifier instantiation strategy that reasons natively about Hilbert's choice operator. The strategy extends MBQI-Enum. It can be used to solve problems that contain Hilbert's choice but also problems that do not (cf. Example 1). We implemented the approach in cvc5. The empirical results on higher-order TPTP benchmarks show that our strategy is clearly superior to the original MBQI-Enum.

We believe that there is potential for further improving the strategy. We observed that the ε terms we need, often occur too late in the term enumeration (which is by increasing term size), since they tend to be complex. Better term ordering heuristics and term filtering could help the SMT solver enumerate the desired terms earlier—before the timeout.

References

1. Andrews, B.: An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof, APLS, vol. 27. Springer (2022)
2. Backes, J., Brown, C.E.: Analytic Tableaux for Higher-Order Logic with Choice **47**, 451–479 (2011)
3. 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. Springer (2022)
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.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer (2011)
6. Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P.: Superposition for Higher-Order Logic. In: Blanchette, J. (ed.) Journal of Automated Reasoning. vol. 67. Springer (2023)
7. 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)
8. 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)
9. Bozec, T., Blanchette, J.: Iterative Monomorphisation **15979**, 269–286 (2025)
10. Brown, C.E.: Satallax: An Automatic Higher-Order Prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning. LNCS, vol. 7364, pp. 111–117. Springer (2012)
11. Church, A.: A formulation of the Simple Theory of Types **5**, 56–68 (1940)
12. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking **52**(3), 365–473 (2005)
13. Dross, C., Conchon, S., Paskevic, A.: Reasoning with Triggers **20**, 22–31 (2012)
14. 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)
15. Gordon, M.J.C.: Introduction to the HOL System. In: Archer, M., Joyce, J.J., Levitt, K.N., Windley, P.J. (eds.) HOL 1991. pp. 2–3. IEEE (1991)
16. Hilbert, D., Ackermann, W.: Grundzüge der theoretischen Logik, GL, vol. 27. Springer (1928)

17. Hilbert, D., Bernays, P.: Grundlagen der Mathematik I, GL, vol. 40. Springer (1934)
18. Kondylidou, L., Reynolds, A., Blanchette, J.: Augmenting Model-Based Instantiation with Fast Enumeration. In: Gurfinkel, A., Heule, M. (eds.) TACAS 2025. LNCS, vol. 15696, pp. 85–103. Springer (2025)
19. 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)
20. 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)
21. de Moura, L., Ullrich, S.: The Lean 4 Theorem Prover and Programming Language. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS, vol. 12699, pp. 625–635. Springer (2021)
22. Niemetz, A., Preiner, M.: Bitwuzla. In: Enea, C., Lal, A. (eds.) CAV 2023. LNCS, vol. 13965, pp. 3–17. Springer (2023)
23. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0 **9**, 53–58 (2014)
24. 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)
25. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
26. Paulson, L.C.: Defining Functions on Equivalence Classes **7**, 658–675 (2019)
27. Reynolds, A., Barbosa, H., Fontaine, P.: Revisiting Enumerative Instantiation. In: Beyer, D. (ed.) TACAS 2018. LNCS, vol. 10806, pp. 112–131. Springer (2018)
28. 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)
29. 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)
30. Reynolds, A., Tinelli, C., de Moura, L.: Finding conflicting instances of quantified formulas in SMT. In: FMCAD 2014. pp. 195–202. IEEE (2014)
31. Simpson, S.G.: The Epsilon Calculus and Conservative Extensions **25**(3), 193–202 (1984)
32. Steen, A.: Extensional Paramodulation for Higher-Order Logic and its Effective Implementation Leo-III. Ph.D. thesis (2020)
33. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure—From CNF to TH0, TPTP v6.4.0 **59**, 483–502 (2017)
34. The Rocq Development Team: The Rocq Prover (2025)
35. Vukmirović, P., Bentkamp, A., Blanchette, J., Cruanes, S., Nummelin, V., Tourret, S.: Making Higher-Order Superposition Work **66**, 541–564 (2022)