Extending SMT Solvers to Higher-Order Logic (Technical Report)

Haniel Barbosa[1], Andrew Reynolds[1], Daniel El Ouraoui[2], Cesare Tinelli[1], and Clark Barrett[3]

[1] The University of Iowa, Iowa City, USA
[2] University of Lorraine, CNRS, Inria, and LORIA, Nancy, France
[3] Stanford University, USA

Abstract. SMT solvers have throughout the years been able to cope with increasingly expressive formulas, from ground logics to full first-order logic (FOL). Nevertheless, higher-order logic (HOL) within SMT is still little explored. We propose a pragmatic extension for SMT solvers to natively support HOL reasoning without compromising performance on FOL reasoning, thus leveraging the extensive research and implementation efforts dedicated to efficient SMT solving. We show how to generalize data structures and the ground decision procedure to support partial applications and extensionality, as well as how to reconcile instantiation techniques with functional variables. We also discuss a separate approach for redesigning an SMT solver to HOL via new data structures and algorithms. We apply the pragmatic extension to the CVC4 SMT solver and redesign the veriT SMT solver. Our evaluation shows they are competitive with state-of-the-art HOL provers and often outperform the traditional encoding into FOL.

1 Introduction

Higher-order (HO) logic is a pervasive setting for reasoning about numerous real-world applications. In particular, it is widely used in proof-assistants (also known as interactive theorem provers) to provide trustworthy, formal, and machine-checkable proofs of theorems. A major challenge in these applications is to automate as much as possible the production of these formal proofs, thereby reducing the burden of proof on the users. An effective approach for stronger automation is to rely on less expressive but more automatic theorem provers to discharge some of the proof obligations. Systems such as HOLyHammer, Miz$\mathbb{AR}$, Sledgehammer, and Why3, which provide a one-click connection from proof-assistants to first-order (FO) provers, have led in recent years to considerable improvements in proof-assistant automation [17]. Such a layered approach is also used by automatic HO provers such as Leo-III [51] and Satallax [20], which regularly invoke FO provers to discharge intermediary goals depending solely on FO reasoning. However, as noted in previous work [15, 38, 57], each of these approaches has disadvantages: full encodings into FO, such as those performed by the "hammers," may lead to performance, soundness and completeness issues; while the combination of FO and HO reasoning in automatic HO provers may suffer from the HO prover itself, which is not optimized for FO proving, having to perform substantial FO reasoning in HO problems that have a large "FO component," which occurs often

in practice. We aim to overcome these shortcomings by extending Satisfiability Modulo Theories (SMT) [10] solvers, which are highly successful automatic FO provers, to natively support HOL.

The main challenges for extending SMT solvers to HOL are dealing with *partial applications* and with *functional variables*. The former mainly affects term representation and core algorithms, which assumes that all symbols are fully applied. The latter impacts quantifier instantiation techniques, which must now account for applied variables. Moreover, often HO problems can only be proven if functional variables are instantiated with synthesized λ-terms, via HO unification [27], which is undecidable.

In this paper, we present two approaches for extending SMT solvers to natively support HO reasoning (HOSMT). In the first approach, called the *pragmatic* approach (Section 3), we consider how to extend a solver with only minimal modifications to its core data structures and algorithms. This approach targets existing state-of-the-art SMT solvers with large code bases and complex data structures optimized for the FO case. In the second approach, called the *redesign* approach (Section 4), we consider how to rethink a solver's data structures and develop new algorithms. This approach may lead to better results, but seems better suited for "lightweight" solvers, i.e., less optimized solvers with smaller code bases. Moreover, the redesign approach provides more flexibility to later develop new techniques specially suited for higher-order reasoning.

A common theme of both approaches is that the instantiation algorithms are not extended with HO unification. We consider an efficient integration of these techniques to be a challenge significant enough to be explored only in a next phase of this work.

We present an extensive experimental evaluation (Section 5) of our pragmatic and redesign approaches as implemented in the state-of-the-art SMT solver CVC4 [8] and the lightweight veriT [19], respectively. Besides comparisons against state-of-the-art HO provers, we also evaluate these solvers against themselves, comparing a native HO encoding using the extensions in this paper to the base versions of the solvers with the more traditional FO encoding (not using the extensions). Our results show the extension of CVC4 complements its encoding-based counterpart and is often ahead of the state-of-the-art, as well as significant improvements for the redesigned veriT.

2 Preliminaries

Our monomorphic higher-order language $\mathscr{L}$ is defined in terms of right-associative binary *sort constructors* $\to$, $\times$ and pairwise-disjoint countably infinite sets $\mathcal{S}$, $\mathcal{X}$ and $\mathcal{F}$, of *atomic sorts*, *variables*, and *function symbols*, respectively. We use the notations $\bar{a}_n$ and $\bar{a}$ to denote the tuple $(a_1, \ldots, a_n)$ or the cross product $a_1 \times \cdots \times a_n$, depending on context, with $n \geq 0$. We extend this notation to pairwise binary operations over tuples in the natural way, e.g., for a binary operator $\bowtie$, the notation $\bar{a}_n \bowtie \bar{b}_n$ stands for $a_1 \bowtie b_1, \ldots, a_n \bowtie b_n$. A *sort* τ is either an element of $\mathcal{S}$ or a *functional sort* $\bar{\tau}_n \to \tau$ from sorts $\bar{\tau}_n = \tau_1 \times \cdots \times \tau_n$ to sort τ. The elements of $\mathcal{X}$ and $\mathcal{F}$ are annotated with sorts, so that $x : \tau$ is a variable of sort τ and $\mathsf{f} : \bar{\tau}_n \to \tau$ is an *n-ary* function symbol of sort $\bar{\tau}_n \to \tau$. We identify function symbols of sort $\bar{\tau}_0 \to \tau$ with function symbols of sort τ, which we call *constants* when τ is not a functional sort. Whenever convenient we drop the sort annotations when referring to symbols.

The set of terms is defined inductively: every variable x is a term. Given variables $\bar{x}_n : \bar{\tau}_n$ and a term $t : \tau$, then $\lambda \bar{x}_n . t$, with sort $\bar{\tau}_n \to \tau$, is a term, called a λ-*abstraction*, with $\bar{x}_n$ being its *bound* variables and t its *body*. A variable occurrence is *free* in a term if it is not bound by a λ-abstraction. A term is *ground* if it has no free variables.

Given a function symbol $\mathsf{f} : \bar{\tau}_n \to \tau$ and terms $t_1 : \tau_1, \ldots, t_n : \tau_n$, then $\mathsf{f}(\bar{t}_n) : \tau$ is a term, called an *application of* f with $\bar{t}_n$ as its *arguments* and f as its *head*. An application $\mathsf{f}(\bar{t}_m) : \tau_{m+1} \times \cdots \times \tau_n \to \tau$, for $m < n$, is a term, called a *partial application of* f. A λ-*application* is an application whose head is a λ-abstraction. The subterm relation is defined recursively: a term is a subterm of itself; if a term is an application, all subterms of its arguments are also its subterms. Note this is not the standard definition of subterms in HOL, which also includes application heads and all partial applications. The *set of all subterms in a term* t is denoted by $\mathbf{T}(t)$. We assume $\mathcal{S}$ contains a sort o, the Boolean sort, and that $\mathcal{F}$ contains Boolean constants $\top$, $\bot$, a Boolean unary function $\neg$, Boolean binary functions $\wedge$, $\vee$, and, for every sort τ, a family of equality symbols $\simeq : \tau \times \tau \to o$ and a family of symbols $\mathsf{ite} : o \times \tau \times \tau \to \tau$. These symbols are interpreted in the usual way as, respectively, logical constants, connectives, identity and *if-then-else* (ITEs). We refer to terms of sort o as *formulas* and to functions of sort $\bar{\tau} \to o$ as *predicates*. An *atom* is a total predicate application. A *literal* is an atom or its negation. We assume the language contains $\forall$ and $\exists$ as binders over formulas. We use the symbol $=$ for syntactic equality on terms. We reserve the names $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}, \mathsf{g}, \mathsf{h}, \mathsf{p}$ for function symbols; w, x, y, z for variables in general and F, G for variables of functional sort; r, s, t, u for terms; and φ, ψ for formulas. The notation $t[\bar{x}_n]$ stands for a term whose free variables are included in the tuple of distinct variables $\bar{x}_n$; $t[\bar{s}_n]$ is the term obtained from t by a simultaneous substitutions of $\bar{s}_n$ for $\bar{x}_n$.

We assume $\mathcal{F}$ contains a family $@ : (\bar{\tau}_n \to \tau) \times \tau_1 \to (\tau_2 \times \cdots \times \tau_n \to \tau)$ of *application symbols* for all $n > 1$. We use it to model (Curried) applications of terms of functional sort $\bar{\tau}_n \to \tau$. For example, given a function symbol $\mathsf{f} : \tau_1 \times \tau_2 \to \tau_3$ and application symbols $@ : (\tau_1 \times \tau_2 \to \tau_3) \times \tau_1 \to (\tau_2 \to \tau_3)$ and $@ : (\tau_2 \to \tau_3) \times \tau_2 \to \tau_3$, $@(\mathsf{f}, t_1)$ and $@(@(\mathsf{f}, t_1), t_2)$ have the same denotation, respectively, as $\lambda x_2 : \tau_2 . f(t_1, x_2)$ and $\mathsf{f}(t_1, t_2)$. To simplify the notation we may omit the $@$ symbols and write $(\ldots((\mathsf{f}, t_1), \ldots), t_n)$ to denote the application $@(\ldots(@(\mathsf{f}, t_1), \ldots), t_n)$.

An *applicative encoding* is a well-known approach for performing HO reasoning using FO provers. This encoding converts every functional sort into an atomic sort, every n-ary symbol into a nullary symbol, and uses $@$ to encode applications. Thus, all applications, partial or not, become total, and quantification over functional variables becomes quantification over regular FO variables. We adopt Henkin's semantics [12, 31] with extensionality and choice, as is standard in automatic HO theorem proving. This logic coincides with the monomorphic fragment of higher-order TPTP, THF0 [13], modulo background theories.

2.1 SMT solvers and quantified reasoning

SMT solvers that process quantified formulas can be seen as containing three main components: a preprocessing module, a ground solver, and an instantiation module. Given an input formula φ, the preprocessing module applies various transformations (such as Skolemization and clausification) to it to obtain another, equisatisfiable, formula φ'.

The ground solver operates on the formula φ'. It abstracts all of its atoms and quantified formulas and treats them as if they were propositional variables. The solver for ground formulas provides an *assignment* $E \cup Q$, where E is a set of ground literals and Q is a set of quantified formulas appearing in φ', such that $E \cup Q$ propositionally entails φ'. The ground solver then determines the satisfiability of E according to a decision procedure for a combination of background theories, such as equality and uninterpreted functions (EUF) and linear integer arithmetic (LIA). If E is satisfiable, the instantiation module of the solver generates new ground formulas of the form $\neg(\forall \bar{x}.\ \psi) \vee \psi\sigma$ where $\forall \bar{x}.\ \psi$ is a quantified formula in Q and σ is a substitution from the variables in ψ to ground terms. We assume that all quantified formulas in Q are of the form $\forall \bar{x}.\ \varphi$ with φ quantifier-free. This can be achieved by prenex form transformation and Skolemization. These instances will be, after preprocessing, added conjunctively to the input of the ground solver, which will proceed to derive a new assignment $E' \cup Q'$, if possible. This interplay may terminate either if φ' is proven unsatisfiable or if a model is found for an assignment $E \cup Q$ that is also a model of φ'.

Extending SMT solvers to HOL can be achieved by extending these three components such thato:

1. the preprocessing module eliminates λ-expressions;
2. the ground decision procedure supports a ground extensional logic with partial applications, which we denote QF_HOSMT;
3. the instantiation module instantiates variables of functional type and takes into account partial applications and equations between functions.

We can perform each of these tasks pragmatically without heavily modifying the solver, which is useful when extending highly optimized state-of-the-art SMT solvers (Section 3). Alternatively, we can perform these extensions in a more principled way by redesigning the solver, which better suits lightweight solvers (Section 4).

3 A pragmatic extension for HOSMT

We pragmatically extend the ground SMT solver to QF_HOSMT by removing λ-expressions (Section 3.1), checking ground satisfiability (Section 3.2), and generating models (Section 3.3). Extensions to the instantiation module are discussed in Section 3.4.

3.1 Eliminating λ-expressions and partial applications of theory symbols

To ensure that the formulas that reach the core solving algorithm are λ-free, a preprocessing pass is used to eliminate λ-applications and λ-abstractions. The former are eliminated via β-reduction, with each application $(\lambda \bar{x}.\ t[\bar{x}])\ \bar{u}$ replaced by the equivalent term $t[\bar{u}]$. The substitution must rename bound variables in t to avoid capture.

Two main approaches exist for eliminating λ-abstractions: λ-lifting [33] and combinators [42]. Combinators allow λ-terms to be synthesized during solving without the need for HO unification. This translation, however, introduces a large number of quantifiers and often leads to performance loss [16, Section 6.4.2]. We instead apply λ-lifting in our pragmatic extension: λ-abstractions are replaced by a fresh function symbol, and

a quantified formula is introduced to define it in terms of the original expression. Note that this is similar to the typical approach used for eliminating ITE expressions in SMT solvers. The new function takes as arguments the variables bound by the respective λ-abstraction and the free variables occurring in its body.

Formally, λ-abstractions of the form $\lambda\bar{x}_n.\ t[\bar{x}_n, \bar{y}_m]$ of type $\bar{\tau}_n \to \tau$ with $\bar{y}_m : \bar{\upsilon}_m$ occurring in a formula φ are lifted to partial applications $@(\mathsf{f}, \bar{y}_m)$ where f is a fresh function symbol of type $\bar{\upsilon}_m \times \bar{\tau}_n \to \tau$. Moreover, the formula $\forall\bar{y}_m\bar{x}_n.\ \mathsf{f}(\bar{y}_m, \bar{x}_n) \simeq t[\bar{x}_n, \bar{y}_m]$ is added conjunctively to φ. To minimize the number of new functions and quantified formulas introduced, eliminated expressions are cached so that the same definition is reused, as is done for ITE removal. Further optimizations such as making the cache invariant to α-renaming could also be applied. We note that careful engineering is required to perform λ-lifting correctly in an SMT solver not originally designed for it. For instance, using the existing machinery for ITE removal may be insufficient, since this may not properly handle instances occurring inside binders or as the head of applications. Moreover, all the places in the SMT solver which distinguish how operations are performed depending on whether you are on the scope of a binder must be extended for the presence of λ-abstractions as well.

Another important preprocessing step makes all applications of *interpreted* symbols total: terms $\mathsf{h}(\bar{t}_m)$ where $\mathsf{h} : \bar{\tau}_n \to \tau$ is a symbol of the background theory with $n > m$ are converted to $\lambda\bar{x}_{n-m}.\ \mathsf{h}(\bar{t}_m, \bar{x}_{n-m})$, which is then λ-lifted, according to the procedure above, to an uninterpreted symbol f defined by the quantified formula $\forall\bar{y}\forall\bar{x}_{n-m}.\ \mathsf{f}(\bar{x}_{n-m}) \simeq \mathsf{h}(\bar{t}_m, \bar{x}_{n-m})$ where $\bar{y}$ collects the free variables of $\bar{t}_m$. Thus, the partial application of h is handled via the newly introduced uninterpreted function f, keeping the theory solver responsible for h agnostic to its partial application.

3.2 Extending the ground solver to QF_HOSMT

Since we operate after preprocessing in a λ-free setting in which only uninterpreted functions may occur partially applied, lifting the ground solver to QF_HOSMT amounts to extending the EUF solver to handle partial applications and extensionality.

The decision procedure for ground EUF generally adopted by SMT solvers is based on the classical congruence closure algorithm developed by Downey et al. [28] and Nelson and Oppen [39]. For better performance its implementation in most SMT solvers assumes that function symbols are fully applied. Instead of redesigning the solver to accommodate partial applications, we apply a *lazy applicative encoding*, in which only such applications are converted with the applicative encoding, whereas the traditional applicative encoding would convert every term in the formula.

Concretely, during term construction, all partial applications are converted to total applications via the binary $@$ symbols, while fully applied terms are kept in their regular representation. Determining the satisfiability of a set of EUF constraints E containing terms in both representations is done in two phases: if E is determined to be satisfiable by the regular first-order procedure, we introduce equalities between regular terms (i.e., fully applied terms without the $@$ symbol) and their applicative counterpart and recheck the satisfiability of the resulting set of constraints. However, we only introduce these equalities for regular terms which interact with partially applied ones. This interaction is characterized by function symbols appearing as members of congruence classes in the

$$\frac{t \in \mathbf{T}(E)}{t \simeq t}\ \textsc{Refl} \qquad \frac{t \simeq u}{u \simeq t}\ \textsc{Sym} \qquad \frac{s \simeq t,\ t \simeq u}{s \simeq u}\ \textsc{Trans}$$

$$\frac{\bar{t}_n \simeq \bar{u}_n \quad \mathsf{f}(\bar{t}_n),\ \mathsf{f}(\bar{u}_n) \in \mathbf{T}(E)}{\mathsf{f}(\bar{t}_n) \simeq \mathsf{f}(\bar{u}_n)}\ \textsc{Cong} \qquad \frac{t \simeq u,\ t \not\simeq u}{\bot}\ \textsc{Conflict}$$

$$\frac{\mathsf{f}(\bar{t}_n),\ \mathsf{f} \in \mathbf{T}(E)}{\mathsf{f}(\bar{t}_n) \simeq @(\ldots\,(@(\mathsf{f},\,t_1),\ldots),\,t_n)}\ \textsc{App-encode}$$

$$\frac{\mathsf{f} \not\simeq \mathsf{g} \qquad \mathsf{f},\,\mathsf{g} : \bar{\tau}_n \to \tau \qquad n > 0}{\mathsf{f}(\mathsf{sk}_1,\,\ldots,\,\mathsf{sk}_n) \not\simeq \mathsf{g}(\mathsf{sk}_1,\,\ldots,\,\mathsf{sk}_n)}\ \textsc{Extensionality}$$

where $\mathsf{sk}_1,\,\ldots,\,\mathsf{sk}_n$ are fresh symbols of respective sorts $\tau_1,\,\ldots,\,\tau_n$.

Fig. 1: Derivation rules for checking satisfiability of QF_HOSMT constraints in EUF.

E-graph, the congruence closure of E built by the EUF decision procedure. A function occurs in an equivalence class if it is an argument of an $@$ symbol or if it appears in an equality between function symbols, and thus as part of a partial application. The equalities between regular terms and their applicative encodings are kept internal to the E-graph, therefore not affecting other parts of the ground decision procedure.

Example 1. Given $\mathsf{f} : \tau \times \tau \to \tau$, $\mathsf{g}, \mathsf{h} : \tau \to \tau$ and $\mathsf{a} : \tau$, consider the set of constraints $E = \{@(\mathsf{f}, \mathsf{a}) \simeq \mathsf{g},\ \mathsf{f}(\mathsf{a}, \mathsf{a}) \not\simeq \mathsf{g}(\mathsf{a}),\ \mathsf{g}(\mathsf{a}) \simeq \mathsf{h}(\mathsf{a})\}$. We have that E is initially found to be satisfiable. However, since f and g occur partially applied, we augment the set of constraints with a correspondence between the HO and FO applications of f, g:

$$E' = E \cup \{@(@(\mathsf{f}, \mathsf{a}), \mathsf{a}) \simeq \mathsf{f}(\mathsf{a}, \mathsf{a}),\ @(\mathsf{g}, \mathsf{a}) \simeq \mathsf{g}(\mathsf{a})\}$$

When determining the satisfiability of E' the equality $@(@(\mathsf{f}, \mathsf{a}), \mathsf{a}) \simeq @(\mathsf{g}, \mathsf{a})$ will be derived by congruence and hence $\mathsf{f}(\mathsf{a}, \mathsf{a}) \simeq \mathsf{g}(\mathsf{a})$ will be derived by transitivity, leading to a conflict. Notice that we do not require equalities between fully applied terms whose functions *do not* appear in the E-graph and their equivalent in the applicative encoding. In particular, the equality $\mathsf{h}(\mathsf{a}) \simeq @(\mathsf{h}, \mathsf{a})$ is not introduced in this example. •

We formalize the above procedure via the calculus in Figure 1. The derivation rules operate on a current set E of constraints. A derivation rule can be applied to E if its premises are met. A rule's conclusion either adds an equality literal to E or replaces it by $\bot$ to indicate unsatisfiability. A rule application is *redundant* if its conclusion leaves E unchanged. A constraint set is *saturated* if it admits only redundant rule applications. The rules are applied to increase an initial set of constraints E_0 until $\bot$ is derived or until saturation.

Rules REFL, SYM, TRANS, CONG and CONFLICT are standard for EUF decision procedures based on congruence closure, i.e., the smallest superset of a set of equations that is closed under entailment in the theory of equality. The rule APP-ENCODE equates a full application to its applicative encoding equivalent, and it is applied only to applications of functions which occur as subterms in E. As mentioned above, this can only

be the case if the function itself appear as an argument of an application, which happens when it is partially applied (as argument of @ or $\simeq$).

Rule EXTENSIONALITY is similar to how extensionality is handled in decision procedures for extensional arrays [24, 52]. If two non-nullary functions are disequal in E, then a witness of their disequality is introduced. The extensionality property is characterized by the axiom $\forall \bar{x}_n.\ \mathsf{f}(\bar{x}_n) \simeq \mathsf{g}(\bar{x}_n) \Leftrightarrow \mathsf{f} \simeq \mathsf{g}$, for all functions f and g of same type. The rule ensures the left-to-right direction of the axiom (the opposite one is ensured by APP-ENCODE together with the congruence closure rules). To simplify the presentation we assume that, for every term $@(\dots\ (@(\mathsf{f},\ t_1), \dots),\ t_m) : \bar{\tau}_n \to \tau \in \mathbf{T}(E)$, there is a fresh symbol $\mathsf{f}' : \bar{\tau}_n \to \tau$ such that $@(\dots\ (@(\mathsf{f},\ t_1), \dots),\ t_m) \simeq \mathsf{f}' \in E$. Thus, we do need to define another EXTENSIONALITY rule for terms such as $@(\dots\ (@(\mathsf{f},\ t_1), \dots),\ t_{n_1}) \not\simeq @(\dots\ (@(\mathsf{g},\ u_1), \dots),\ u_{n_2})$.

Example 2. Consider the function symbols $\mathsf{f}, \mathsf{g} : \tau \to \tau$, $\mathsf{a} : \tau$, and the set of constraints $E = \{\mathsf{f} \simeq \mathsf{g},\ \mathsf{f}(\mathsf{a}) \not\simeq \mathsf{g}(\mathsf{a})\}$. The constraints are initially satisfiable with respect to the congruence closure rules, however, since $\mathsf{f}, \mathsf{g} \in \mathbf{T}(E)$, the rule APP-ENCODE will be applied twice to derive $\mathsf{f}(\mathsf{a}) \simeq @(\mathsf{f},\ \mathsf{a})$ and $\mathsf{g}(\mathsf{a}) \simeq @(\mathsf{g},\ \mathsf{a})$. Then via CONG, from $\mathsf{f} \simeq \mathsf{g}$ we infer $@(\mathsf{f},\ \mathsf{a}) \simeq @(\mathsf{g},\ \mathsf{a})$, which leads to a conflict via transitivity. •

Decision procedure Any derivation strategy for the calculus that does not stop until it saturates or generates $\bot$ yields a decision procedure for the satisfiability of QF_HOSMT constraints in the EUF theory, according to the following results for the calculus.

Proposition 1 (Termination). *Every sequence of non-redundant rule applications is finite.*

Proof. The congruence closure rules do not introduce new terms, therefore their non-redundant applications are bounded by the number of terms in E. Applications of APP-ENCODE are bounded by the number of function symbols in $\mathbf{T}(E)$. These are bounded by the number of partially applied functions initially in E and, ultimately, by the number of total applications in $\mathbf{T}(E)$, which is also finite. Applications of EXTENSIONALITY are bounded by the number of disequalities between function symbols in E. Applications of EXTENSIONALITY do not introduce such disequalities, as the introduced disequalities are in terms of total applications. Since only a finite number of function symbols may be added to E by APP-ENCODE, there is a finite number of applications of EXTENSIONALITY to be made for any E. Thus, as the last two rules are the only ones that introduce new terms and both have a bounded number of applications, the calculus is terminating.

Proposition 2 (Refutation Soundness). *A constraint set is unsatisfiable if $\bot$ is derivable from it.*

Proposition 3 (Solution Soundness). *Every saturated constraint set is satisfiable.*

3.3 Model generation for ground formulas

When our decision procedure for QF_HOSMT saturates, it can produce a first-order model M as a witness for the satisfiability of its input. Typically, the models generated

by SMT solvers for theories in first-order logic map uninterpreted functions $\mathsf{f} : \bar{\tau}_n \to \tau$ to functions, denoted $M(\mathsf{f})$, of the form

$$\lambda \bar{x}_n.\ \mathsf{ite}(x_1 \simeq t_1^1 \wedge \ldots \wedge x_n \simeq t_n^1, s_1, \\ \ldots, \mathsf{ite}(x_1 \simeq t_1^{m-1} \wedge \ldots \wedge x_n \simeq t_n^{m-1}, s_{m-1},\ s_m) \ldots)$$

abusing notation to refer to the interpretation of $M(t_j^i)$ as t_j^i, in which every entry but the last corresponds to an application $\mathsf{f}(t_1^i,\ \ldots,\ t_n^i)$, modulo congruence, occurring in the problem. In other words, functions are interpreted in models M as almost constant functions.

In the presence of partial applications, this scheme can sometimes lead to functions with exponentially many entries. For example, consider the satisfiable formula

$$\begin{aligned} &\mathsf{f}_1(\mathsf{a}) \simeq \mathsf{f}_1(\mathsf{b}) \wedge \mathsf{f}_1(\mathsf{b}) \simeq \mathsf{f}_2 \\ \wedge\ &\mathsf{f}_2(\mathsf{a}) \simeq \mathsf{f}_2(\mathsf{b}) \wedge \mathsf{f}_2(\mathsf{b}) \simeq \mathsf{f}_3 \\ \wedge\ &\mathsf{f}_3(\mathsf{a}) \simeq \mathsf{f}_3(\mathsf{b}) \wedge \mathsf{f}_3(\mathsf{b}) \simeq \mathsf{c} \end{aligned}$$

in which $\mathsf{f}_1 : \tau \times \tau \times \tau \to \tau$, $\mathsf{f}_2 : \tau \times \tau \to \tau$, $\mathsf{f}_3 : \tau \to \tau$, and $\mathsf{a}, \mathsf{b}, \mathsf{c} : \tau$. To produce the model values of f_1 as a list of total applications with three arguments into an element of the interpretation of τ, we would need to account for 8 cases. In other words, we require 8 ite cases to indicate $\mathsf{f}_1(x, y, z) \simeq \mathsf{c}$ for all inputs where $x, y, z \in \{\mathsf{a}, \mathsf{b}\}$. The number of entries in the model is exponential on the "depth" of the chain of functions that each partial application is equal to, which can make model building unfeasible if just a few functions are chained as in the above example. This is a pattern we have seen in practice in problems originating from interactive theorem provers.

To avoid such an exponential behavior, model building assigns values for functions in terms of the other functions that their partial applications are equated to. In the above example f_1 would have only two model values, depending on its application's first argument being a or b, by using the model values of f_2 applied on its two other arguments. In other words, we construct $M(\mathsf{f}_1)$ as the term:

$$\lambda xyz.\ \mathsf{ite}(x \simeq \mathsf{a},\ M(\mathsf{f}_2)(y,\ z), \mathsf{ite}(x \simeq \mathsf{b},\ M(\mathsf{f}_2)(y,\ z), \_))$$

where $M(\mathsf{f}_2)$ is the model for f_2 and _ is an arbitrary value. The model value of f_2 would be analogously built in terms of the model value of f_3. This guarantees a polynomial construction for models in terms of the number of constraints in the problem in the presence of partial applications.

Extensionality and finite sorts Model construction assigns different values to terms not asserted equal. Therefore, if non-nullary functions $\mathsf{f}, \mathsf{g} : \bar{\tau}_n \to \tau$ occur as terms in different congruence classes but are not asserted disequal, we ensure they are assigned different model values by introducing disequalities of the form $\mathsf{f}(\bar{\mathsf{sk}}_n) \not\simeq \mathsf{g}(\bar{\mathsf{sk}}_n)$ for fresh $\bar{\mathsf{sk}}_n$. This is necessary because model values for functions are built based on their applications occurring in the constraint set. However, such disequalities are only always guaranteed to be satisfied if $\bar{\tau}_n, \tau$ are infinite sorts.

Example 3. Let E be a saturated set of constraints s.t. $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3 : \tau \to o \in \mathbf{T}(E)$ and $E \not\models \mathsf{p}_1 \simeq \mathsf{p}_2 \vee \mathsf{p}_1 \simeq \mathsf{p}_3 \vee \mathsf{p}_2 \simeq \mathsf{p}_3 \vee \mathsf{p}_1 \not\simeq \mathsf{p}_2 \vee \mathsf{p}_1 \not\simeq \mathsf{p}_3 \vee \mathsf{p}_2 \not\simeq \mathsf{p}_3$. In the congruence

closure of E the functions $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3$ each occur in a different congruence class but are not asserted disequal, so model construction would, in order to build their model values, introduce disequalities $\mathsf{p}_1(\mathsf{sk}_1) \not\simeq \mathsf{p}_2(\mathsf{sk}_1)$, $\mathsf{p}_1(\mathsf{sk}_2) \not\simeq \mathsf{p}_3(\mathsf{sk}_2)$, and $\mathsf{p}_2(\mathsf{sk}_3) \not\simeq \mathsf{p}_3(\mathsf{sk}_3)$, for fresh $\mathsf{sk}_1, \mathsf{sk}_2, \mathsf{sk}_3 : \tau$. However, if τ has cardinality one these disequalities make E unsatisfiable, since $\mathsf{sk}_1, \mathsf{sk}_2, \mathsf{sk}_3$ must be equal and o has cardinality 2. •

To prevent this issue, whenever the set of constraints E is saturated, we introduce, for every pair of functions $\mathsf{f}, \mathsf{g} : \bar{\tau}_n \to \tau \in \mathbf{T}(E)$ s.t. $n > 0$ and $E \not\models \mathsf{f} \simeq \mathsf{g} \vee \mathsf{f} \not\simeq \mathsf{g}$, the splitting lemma $\mathsf{f} \simeq \mathsf{g} \vee \mathsf{f} \not\simeq \mathsf{g}$. In the above example this would amount to add the lemmas $\mathsf{p}_1 \simeq \mathsf{p}_2 \vee \mathsf{p}_1 \not\simeq \mathsf{p}_2$, $\mathsf{p}_1 \simeq \mathsf{p}_3 \vee \mathsf{p}_1 \not\simeq \mathsf{p}_3$, and $\mathsf{p}_2 \simeq \mathsf{p}_3 \vee \mathsf{p}_2 \not\simeq \mathsf{p}_3$, thus ensuring that the decision procedure detects the inconsistency before saturation.

3.4 Extending the quantifier instantiation module to HOMST

The main quantifier instantiation techniques in SMT solving are trigger-based [25], conflict-based [7, 45], model-based [30, 47], and enumerative [44]. Lifting any of them to HOSMT presents its own challenges, often related to performing HO unification. We focus here on extending the E-matching [23] algorithm, the keystone of trigger-based instantiation, the most commonly used technique in modern SMT solvers. In this technique, instantiations are chosen for quantified formulas φ based on *triggers*. A trigger is a term (or set of terms) containing the free variables occurring in φ. Matching a trigger term against ground terms in the current set of assertions E results in a substitution that is used to instantiate φ.

The presence of higher-order constraints poses several challenges for E-matching. First, notice that the $@$ symbol is an overloaded operator. Applications of this symbol can be selected as terms that appear in triggers. Special care must be taken so that applications of $@$ are not matched with ground applications of $@$ whose arguments have different types. Second, functions can be equated in higher-order logic. As a consequence, a match may involve a trigger term and a ground term with different head symbols. Third, since we use a lazy applicative encoding, our ground set of terms may contain a mixture of partially and fully applied function applications. Thus, our indexing techniques must be robust to handle combinations of the two. The following example demonstrates the last two challenges.

Example 4. Consider E containing the equality $@(\mathsf{f}, \mathsf{a}) \simeq \mathsf{g}$ and the term $\mathsf{f}(\mathsf{a}, \mathsf{b})$ where $\mathsf{f} : \tau \times \tau \to \tau$ and $g : \tau \to \tau$. Notice that $\mathsf{g}(x)$ is equivalent modulo E to the term $\mathsf{f}(\mathsf{a}, \mathsf{b})$ under the substitution $x \mapsto \mathsf{b}$. Such a match is found by indexing all terms that are applications of *either* $@(\mathsf{f}, \mathsf{a})$ or g in a common term index. This ensures that when we find matches for $\mathsf{g}(x)$, the application $\mathsf{f}(\mathsf{a}, \mathsf{b})$, whose applicative counterpart is $@(@(\mathsf{f}, \mathsf{a}), \mathsf{b})$, is considered.

We extended the regular first-order E-matching algorithm of CVC4 with the extensions mentioned in this section. Extensions to the other instantiation techniques of CVC4, such as model-based quantifier instantiation, are left as future work.

Extending expressivity via axioms Even though not synthesizing λ-expressions prevents us from fully lifting the above instantiation techniques to HOL, we remark that, as we see in Section 5, this pragmatic extension very often can prove HO theorems, many times even at higher rates than full-fledged HO provers. Nevertheless, success rates can be improved by using well-chosen axioms to prove problems that otherwise cannot be proved without synthesizing λ-expressions.

Example 5. Consider the ground formula $\varphi = \mathsf{a} : \tau \not\simeq \mathsf{b} : \tau$ and the quantified formula $\psi = \forall F, G : \tau \to \tau.\ F \simeq G$. Intuitively ψ establishes that all functions of sort $\tau \to \tau$ are equal. However, this is inconsistent with φ, which forces τ to contain at least two elements and therefore $\tau \to \tau$ to contain at least four functions. For a prover to detect this inconsistency it must apply the instantiation $\{F \mapsto (\lambda w.\ \mathsf{a}),\ G \mapsto (\lambda w.\ \mathsf{b})\}$ to ψ, which requires performing HO unification. However, adding the axiom

$$\forall F : \tau \to \tau, x, y : \tau.\ \exists G : \tau \to \tau.\ \forall z : \tau.\ G(z) \simeq \mathsf{ite}(z \simeq x,\ y,\ F(z)) \qquad \text{(SAX)}$$

makes the problem provable without the need to synthesize λ-expressions. •

We denote the above axiom as the *store axiom* (SAX) because it simulates how arrays are updated via the store operation. As we note in Section 5, introducing this axiom for all functional sorts occurring in the problem often allows our pragmatically extended solver to prove a problem that it would not otherwise been able to.

Extensionality and quantifiers We note that even though we handle extensionality in a complete way in the extended ground solver the same guarantees cannot be made when quantifiers are present, even without functional quantification. The following example shows that not only HO unification is necessary to address incompleteness of SMT solvers in extensional HOL.

Example 6. The constraint set $\{\mathsf{h}(\mathsf{f}) \simeq \mathsf{b},\ \mathsf{h}(\mathsf{g}) \not\simeq \mathsf{b},\ \forall x.\ \mathsf{f}(x) \simeq \mathsf{a},\ \forall x.\ \mathsf{g}(x) \simeq \mathsf{a}\}$, with $\mathsf{h} : \tau \to \tau \to \tau$, $\mathsf{f}, \mathsf{g} : \tau \to \tau$, $\mathsf{a}, \mathsf{b} : \tau$, is unsatisfiable since it establishes, via the quantified formulas, that $\mathsf{f} \simeq \mathsf{g}$, but also, via the ground constraints, that $\mathsf{f} \not\simeq \mathsf{g}$. However only a ground decision procedure closed under entailment w.r.t. disequalities, which SMT solvers are well known not to implement, would derive the disequality $\mathsf{f} \not\simeq \mathsf{g}$, which, via an application of the extensionality rule, would lead to the derivation of $\mathsf{f}(\mathsf{sk}) \not\simeq \mathsf{g}(\mathsf{sk})$ and the subsequent instantiation of the quantified formulas that would lead to a conflict.

Towards higher-order E-matching. Let $\mathsf{p}, \mathsf{q} : \tau \to o$ and $\mathsf{f} : \tau \times \tau \to \tau$ and consider the constraints

$$E = \{\mathsf{q}(\mathsf{f}(\mathsf{a}, \mathsf{b})),\ \neg\mathsf{p}(k(\mathsf{a}, \mathsf{a}))\}$$
$$Q = \{\forall (F : \tau \times \tau \to \tau)\,(y, z : \tau).\ \mathsf{p}(F(y, z)) \vee \neg\mathsf{q}(f(\mathsf{b}, y))\}$$

The above problem can be found unsatisfiable with e.g. the instantiation $\{F \mapsto \lambda w_1 w_2.\mathsf{f}(\mathsf{a}, w_1),\ y \mapsto \mathsf{a},\ z \mapsto \mathsf{a}\}$. First-order E-matching is not capable of finding instantiations as the one above, since it does not derive new lambda expressions. To address this issue we have

developed an extension of E-matching based on Huet's algorithm to higher-order matching [32]. In this extension, when given a match for a trigger whose head is a function variable, we obtain variations of the match based on permuting the arguments of the value of the head in the match. Considering again the above formula φ, first-order E-matching for the pair $\langle F(y,z), \mathsf{f}(\mathsf{a},\mathsf{a})\rangle$ would find the substitution $\{F \mapsto \mathsf{f},\ y \mapsto \mathsf{a},\ z \mapsto \mathsf{a}\}$. Our procedure may then generate the following instantiations for F:

$$F \mapsto \lambda w_1 w_2.\, \mathsf{f}(w_1, w_2) \quad (1)$$
$$F \mapsto \lambda w_1 w_2.\, \mathsf{f}(w_2, w_1) \quad (2)$$
$$F \mapsto \lambda w_1 w_2.\, \mathsf{f}(\mathsf{a}, w_1) \quad (3)$$
$$F \mapsto \lambda w_1 w_2.\, \mathsf{f}(w_1, \mathsf{a}) \quad (4)$$
$$F \mapsto \lambda w_1 w_2.\, \mathsf{f}(\mathsf{a}, w_2) \quad (5)$$
$$F \mapsto \lambda w_1 w_2.\, \mathsf{f}(w_2, \mathsf{a}) \quad (6)$$
$$F \mapsto \lambda w_1 w_2.\, \mathsf{f}(\mathsf{a}, \mathsf{a}) \quad (7)$$

in which $(2) - (7)$ are variations obtained by permuting the function arguments with constants according to the match that was found. Note that (3) is the instantiation for F we gave as example above to prove φ unsatisfiable.

We have yet to evaluate the effectiveness of this technique in our pragmatic approach.

4 Redesigning a solver for HOSMT

As we discussed earlier, the main difficulties in extending an SMT solver to higher-order logic come from partially applied functions and function symbols as arguments of other functions or as quantified variables. In the previous section we saw how to use the applicative encoding approach inside SMT solvers. An alternative is redesigning the SMT solver to cope with these HO features without explicitly using the applicative encoding. This approach however requires much more work in the core data structures and algorithms, and is better suited for lightweight solvers. We propose below such a redesign. We assume the solver operates on λ-free terms, which can be obtained via preprocessing as in Section 3.1, and, without loss of generality, that only uninterpreted functions are partially applied, following Section 3.2.

4.1 Redesigning the core ground solver for HOSMT

Efficient implementations of the congruence-closure decision procedure operate on UNION-FIND data structures and have asymptotic time complexity $\mathcal{O}(n \log n)$. To accommodate partial applications natively, we propose a simpler algorithm which operates straightforwardly on a graph where nodes are terms, and edges relations (equality, congruence, disequality) between them. An equivalence class is a connected component without disequality edges. All operations (incremental addition of new constraints, backtracking, conflict analysis, proof production) are straightforward to implement. This simpler implementation comes at the cost of complexity (the algorithm is

quadratic) but better integrates with various other features such as term addition, injective functions, rewriting or even computation, in particular for β- and η-conversion, which could be done during solving rather than preprocessing. In the redesigned approach the solver keeps two term representations, a curried and a regular one. In the regular term representation partial and total applications are distinguished by type information. The curried representation is used only by the congruence closure algorithm. It is integrated with the rest of the solver via an interface with translation functions between the two different representations, e.g. $\texttt{curry}(\mathsf{f}(\bar{t}_n)) = (\dots((\mathsf{f}, t_1), \dots), t_n)$ and $\texttt{uncurry}((\dots((\mathsf{f}, t_1), \dots), t_n)) = \mathsf{f}(\bar{t}_n)$, with the latter only being defined for curried applications whose counterpart is a total application.

Example 7. Given $\mathsf{f} : \tau \times \tau \to \tau$, $\mathsf{g}, \mathsf{h} : \tau \to \tau$ and $\mathsf{a} : \tau$, consider the constraints $\{\mathsf{f}(\mathsf{a}) \simeq \mathsf{g},\ \mathsf{f}(\mathsf{a},\mathsf{a}) \not\simeq \mathsf{g}(\mathsf{a}),\ \mathsf{g}(\mathsf{a}) \simeq \mathsf{h}(\mathsf{a})\}$. The congruence closure module will operate on $\{(\mathsf{f},\mathsf{a}) \simeq \mathsf{g},\ ((\mathsf{f},\mathsf{a}),\mathsf{a}) \not\simeq (\mathsf{g},\mathsf{a}),\ (\mathsf{g},\mathsf{a}) \simeq (\mathsf{h},\mathsf{a})\}$, thanks to the curry translation. •

The calculus from Figure 1 can be adapted to operate on curried terms only. Formally, given a set of constraints E and its set of terms $\mathbf{T}(E)$, we define the set of curried constraints E_{curr} as:

$$E_{\text{curr}} = \{\texttt{curry}(\mathsf{f}(\bar{t}_n)) \simeq \texttt{curry}(\mathsf{g}(\bar{s}_m)) \mid \mathsf{f}(\bar{t}_n) \simeq \mathsf{g}(\bar{s}_m) \in E\}$$

and the set of curried terms $\mathbf{T}^{\texttt{curr}}(E)$ as

$$\mathbf{T}^{\texttt{curr}}(E) = \left\{ \bigcup_{i \in [0,\, n]} \texttt{curry}(\mathsf{f}(\bar{t}_n),\, i) \mid \mathsf{f}(\bar{t}_n) \in \mathbf{T}(E) \right\}$$

in which we overload the informal definition above of curry now as a binary function such that

$$\texttt{curry}(\mathsf{f}(\bar{t}_n),\, k) = (\dots((\mathsf{f},\, \texttt{curry}(t_1 : \bar{\tau}_{n_1} \to \tau_1, n_1)), \dots),\, \texttt{curry}(t_k : \bar{\tau}_{n_k} \to \tau_k, n_k))$$

Note that if $k = 0$ the result is f itself.

Example 8. Consider a set of terms $\mathbf{T} = \{\mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{b},\mathsf{c})),\ \mathsf{g}(\mathsf{b},\mathsf{c}),\ \mathsf{a},\ \mathsf{b},\ \mathsf{c}\}$, with $\mathsf{a},\mathsf{b},\mathsf{c} : \tau$ and $\mathsf{f},\mathsf{g} : \tau \times \tau \to \tau$, then

$$\begin{aligned}\mathbf{T}^{\text{curr}} = \quad & \{\texttt{curry}(\mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{b},\mathsf{c})), i)\}_{i=0}^{2} \cup \{\texttt{curry}(\mathsf{g}(\mathsf{b},\mathsf{c}), i)\}_{i=0}^{2} \\ & \cup \{\texttt{curry}(\mathsf{a},0)\} \cup \{\texttt{curry}(\mathsf{b},0)\} \cup \{\texttt{curry}(\mathsf{c},0)\}\end{aligned}$$

with

$$\begin{aligned}\texttt{curry}(\mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{b},\mathsf{c})),2) &= ((\mathsf{f},\, \texttt{curry}(\mathsf{a},0)), \texttt{curry}(\mathsf{g}(\mathsf{b},\mathsf{c}),2)) \\ \texttt{curry}(\mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{b},\mathsf{c})),1) &= (\mathsf{f},\, \texttt{curry}(\mathsf{a},0)) \\ \texttt{curry}(\mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{b},\mathsf{c})),0) &= \mathsf{f} \\ \texttt{curry}(\mathsf{g}(\mathsf{b},\mathsf{c}),2) &= ((\mathsf{g},\, \texttt{curry}(\mathsf{b},0)), \texttt{curry}(\mathsf{c},0)) \\ \texttt{curry}(\mathsf{g}(\mathsf{b},\mathsf{c}),1) &= (\mathsf{g},\, \texttt{curry}(\mathsf{b},0)) \\ \texttt{curry}(\mathsf{g}(\mathsf{b},\mathsf{c}),0) &= \mathsf{g} \\ \texttt{curry}(\mathsf{a},0) &= \mathsf{a} \\ \texttt{curry}(\mathsf{b},0) &= \mathsf{b} \\ \texttt{curry}(\mathsf{c},0) &= \mathsf{c}\end{aligned}$$

$$\frac{t \in \mathbf{T}^{\text{curr}}(E)}{t \simeq t \in E_{\text{curr}}}\ \textsc{Refl}_{\textsc{curr}} \qquad \frac{t \simeq u}{u \simeq t \in E_{\text{curr}}}\ \textsc{Sym} \qquad \frac{s \simeq t,\ t \simeq u}{s \simeq u \in E_{\text{curr}}}\ \textsc{Trans}$$

$$\frac{t \simeq u \quad t' \simeq u' \quad (t,\ t'), (u,\ u') \in \mathbf{T}^{\text{curr}}(E)}{(t,\ t') \simeq (u,\ u') \in E_{\text{curr}}}\ \textsc{Cong}_{\textsc{curr}} \qquad \frac{t \simeq u,\ t \not\simeq u}{\bot}\ \textsc{Conflict}$$

$$\frac{t \simeq u \qquad \texttt{uncurry}(t) \text{ or } \texttt{uncurry}(u) \text{ is shared with theory } \mathcal{T}}{\texttt{uncurry}(t) \simeq \texttt{uncurry}(u) \in E}\ \textsc{Theory-prop}$$

$$\frac{t,\ u \in \mathbf{T}^{\text{curr}}(E) \qquad t,\ u : \bar{\tau}_n \to \tau,\ n > 0}{\forall F, G : \bar{\tau}_n \to \tau.\ F \not\simeq G \Rightarrow F(\mathsf{sk}_1,\ \ldots,\ \mathsf{sk}_n) \not\simeq G(\mathsf{sk}_1,\ \ldots,\ \mathsf{sk}_n) \in Q}\ \textsc{Ext-Ax}$$

where $\mathsf{sk}_1,\ \ldots,\ \mathsf{sk}_n$ are fresh symbols of respective sorts $\tau_1,\ \ldots,\ \tau_n$.

Fig. 2: Adapted derivation rules for EUF.

such that

$$\mathbf{T}^{\text{curr}} = \{((\mathsf{f},\ \mathsf{a}),\ ((\mathsf{g},\ \mathsf{b}),\ \mathsf{c})), (\mathsf{f},\ \mathsf{a}), (\mathsf{g},\ \mathsf{b}), ((\mathsf{g},\ \mathsf{b}),\ \mathsf{c}), \mathsf{f}, \mathsf{g}, \mathsf{a}, \mathsf{b}, \mathsf{c}\}$$

is the resulting set of subterms. •

We adapt the rules specified in Figure 1 by providing the following calculus in Figure 2. The calculus operates on the curried constraints E_{curr} and relies on the set of curried subterms $\mathbf{T}^{\text{curr}}(E)$. The APP-ENCODE rule is not necessary anymore since all deductions are done in the same (curried) representation.

SMT solvers generally perform theory combination via equalities over terms shared between different theories. Given the different term representations kept between the congruence closure and the rest of the solver, to ensure that theory combination is done properly, the procedure keeps track of terms shared with other theory solvers. Therefore, whenever an equality is inferred on a term whose translation is shared with another theory, a shared equality is sent out in terms of the translation. This is modeled by the THEORY-PROP rule, which assumes an awareness of which original uncurried terms were shared with other theories.

Example 9. Consider the function symbol $\mathsf{f} : \mathsf{Int} \to \mathsf{Int},\ \mathsf{a}, \mathsf{b}, \mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \mathsf{c}_4 : \mathsf{Int}$, the predicate symbol $\mathsf{p} : \mathsf{Int} \to o$ and the set of arithmetic constraints $E = \{\mathsf{a} \leq \mathsf{b},\ \mathsf{b} \leq \mathsf{a},\ \mathsf{p}(\mathsf{f}(\mathsf{a}) - \mathsf{f}(\mathsf{b})),\ \neg\mathsf{p}(0),\ \mathsf{c}_1 \simeq \mathsf{c}_3 - \mathsf{c}_4,\ \mathsf{c}_2 \simeq 0\}$ and the set of equality constraints $E' = \{(\mathsf{p},\ \mathsf{c}_1),\ \neg(\mathsf{p},\ \mathsf{c}_2), \mathsf{c}_3 \simeq (\mathsf{f},\ \mathsf{a}),\ \mathsf{c}_4 \simeq (\mathsf{f},\ \mathsf{b})\}$ obtained after translation. First, the arithmetic module deduces $\mathsf{a} \simeq \mathsf{b}$ yielding the new constraint $E' = E' \cup \{\mathsf{a} \simeq \mathsf{b}\}$. By CONG it follows that $(\mathsf{f},\ \mathsf{a}) \simeq (\mathsf{f},\ \mathsf{b})$ and, as $\mathsf{c}_3 \simeq (\mathsf{f},\ \mathsf{a})$ and $\mathsf{c}_4 \simeq (\mathsf{f},\ \mathsf{b})$ are keeping track that they are shared, $\mathsf{c}_3 \simeq \mathsf{c}_4$ is propagated, where $\mathsf{c}_3,\ \mathsf{c}_4$ were the representatives of the former merged classes. Therefore, $E = E \cup \{\mathsf{c}_3 \simeq \mathsf{c}_4\}$ and $\mathsf{c}_1 \simeq \mathsf{c}_2$ is deduced producing the unsatisfiable constraint set $E' = E' \cup \{\mathsf{a} \simeq \mathsf{b},\ \mathsf{c}_1 \simeq \mathsf{c}_2\}$. •

Extensionality The pragmatic extension handles extensionality via the dedicated rule EXTENSIONALITY in Figure 1, which suffices for ground reasoning, but has shortcom-

ings when quantifiers, even only FO quantifiers, are considered, as shown in Example 6. An alternative is to handle extensionality via axioms, which we chose for the redesigned solver.

Example 10. Consider again the constraint set $\{\mathsf{h}(\mathsf{f}) \simeq \mathsf{b},\ \mathsf{h}(\mathsf{g}) \not\simeq \mathsf{b},\ \forall x.\ \mathsf{f}(x) \simeq \mathsf{a},\ \forall x.\ \mathsf{g}(x) \simeq \mathsf{a}\}$, with $\mathsf{h} : \tau \to \tau \to \tau$, $\mathsf{f}, \mathsf{g} : \tau \to \tau$, $\mathsf{a}, \mathsf{b} : \tau$. The pragmatic solver can prove this problem unsatisfiable only with a ground decision procedure closed under entailment w.r.t. disequalities, as $\mathsf{f} \not\simeq \mathsf{g}$ is necessary to derive $\mathsf{f}(\mathsf{sk}) \not\simeq \mathsf{g}(\mathsf{sk})$, via extensionality, and the subsequent instantiations that would lead to a conflict. But SMT solvers are well known not to propagate all disequalities. However, with the axiom $\forall F, G : \bar{\tau}_n \to \tau.\ F \not\simeq G \Rightarrow F(\mathsf{sk}_1, \ldots, \mathsf{sk}_n) \not\simeq G(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, the instantiation $\{F \mapsto \mathsf{f},\ G \mapsto \mathsf{g}\}$, which may be derived e.g. via enumerative instantiation as both $\mathsf{f}, \mathsf{g} \in \mathbf{T}(E)$, provides the splitting lemma $\mathsf{f} \simeq \mathsf{g} \vee \mathsf{f}(\mathsf{sk}) \not\simeq \mathsf{g}(\mathsf{sk})$. The case $E \cup \{\mathsf{f} \simeq \mathsf{g}\}$ leads to a conflict via pure ground reasoning, while the case $E \cup \{(\mathsf{f},\ \mathsf{sk}) \not\simeq (\mathsf{g},\ \mathsf{sk})\}$ leads to a conflict via the aforementioned instances $\mathsf{f}(\mathsf{sk}) \simeq \mathsf{a},\ \mathsf{g}(\mathsf{sk}) \simeq \mathsf{a}$. •

4.2 Instantiation module

The challenge for E-matching here lies in the different term representations within the solver, particularly between the E-graph and the instantiation module. Extending the term index to queries on type, to obtain the correct corresponding term translations, between curried and regular representations, allowing us to apply the algorithm as expected.

Example 11. Consider the symbols $\mathsf{g} : \tau \to \tau$, $\mathsf{f} : \tau \times \tau \to \tau$, $\mathsf{a}, \mathsf{b} : \tau$, and the set of constraints $E = \{\mathsf{f}(\mathsf{a}, \mathsf{b}) \not\simeq \mathsf{g}(\mathsf{b})\}$ and $Q = \{\forall F.\ F(\mathsf{a}) \simeq \mathsf{g}\}$. The original $\mathbf{T}(E)$ does not contain $\mathsf{f}(\mathsf{a})$, but it belongs in the curried set of subterms of the congruence closure since is it understood as $E' = \{((\mathsf{f}, \mathsf{a}), \mathsf{b}) \not\simeq (\mathsf{g}, \mathsf{b})\}$. Then the instantiation module get the following set of terms $\{(\mathsf{f}, \mathsf{a}),\ \mathsf{g}\}$ by querying the sort $\tau \to \tau$ to the congruence closure module in order to match the trigger $F(\mathsf{a})$. It follows by applying E-matching the unsatisfiable set of constraints $E' = \{((\mathsf{f},\ \mathsf{a}),\ \mathsf{b}) \not\simeq (\mathsf{g},\ \mathsf{b}),\ (\mathsf{f}, \mathsf{a}) \simeq \mathsf{g}\}$. •

5 Evaluation

We have implemented the above techniques in the state-of-the-art CVC4 solver and in the lightweight veriT solvers. We distinguish between two main versions of each solver: one that performs a full applicative encoding (Section 2) into FOL a priori, denoted @cvc and @vt, and another that implements the pragmatic (Sections 3) or redesigned (Section 4) extensions to HOL within the solvers, denoted cvc and vt. Both CVC4 modes eliminate λ-expressions via λ-lifting. Neither veriT configuration supports benchmarks with λ-expressions. The CVC4 configurations that employ the "store axiom" (Section 3.4) are denoted by having the suffix -sax.

We use the state-of-the-art HO provers Leo-III [51], Satallax [20,29] and Ehoh [49, 57] as baselines in our evaluation. The first two have refutationally complete calculi for extensional HOL with Henkin semantics, while the latter only supports λ-free HOL without first-class Booleans. For Leo-III and Satallax we use their configurations from

the CASC competition [55], while for Ehoh we report on their best non-portfolio configuration from Vukmirović et al, [57], Ehoh hb.

We split our report between proving HO theorems and reporting countermodels for HO conjectures, which require different strengths from the considered solvers. Only a handful of the above solvers are considered for the second evaluation: Leo-III and veriT do not provide models and Ehoh is not model-sound w.r.t. Henkin semantics, only w.r.t. λ-free Henkin semantics. Therefore, we only consider CVC4 and Satallax for the countermodels evaluation. We ran our experiments on a cluster equipped with Intel E5-2637 v4 CPUs running Ubuntu 16.04, providing one core, 60 seconds, and 8GB RAM for each job. The full experimental data is publicly available.[1]

We consider the following sets[2] of HO benchmarks: the 3188 monomorphic HO benchmarks in TPTP [54], split into three subsets: the 530 problems which are both λ-free and without first-class Booleans (TH0); the 743 which are only λ-free (oTH0); and the 1915 that are neither (λoTH0). The next sets are Sledghammer (SH) benchmarks from the "Judgment Day" test harness [18], consisting of 1253 provable goals *manually* chosen from different Isabelle theories [41] and encoded into λ-free monomorphic HOL problems without first-class Booleans. The encoded problems are such that if they are provable so is the original goal. These problems are split into four subsets, $\text{JD}^{32}_{\text{lift}}$, $\text{JD}^{32}_{\text{combs}}$, $\text{JD}^{512}_{\text{lift}}$, and $\text{JD}^{512}_{\text{combs}}$, depending, respectively on whether they have 32 or 512 Isabelle lemmas, or facts, and whether λ-abstractions are removed via λ-lifting or via SK-style combinators. The last set, $\lambda o\text{SH}^{1024}$, has 832 SH benchmarks from 832 provable goals *randomly* selected from different Isabelle theories, encoded with 1024 facts and preserving λs and first-class Booleans. Considering a varying number of facts in the SH benchmarks emulates the needs of increasingly larger problems in interactive verification, while different λ handling schemes allow us to measure from which alternative each particular solver benefits more.

We point out that our extensions to CVC4 and veriT do not compromise their FO solving performance. The pragmatic extension of CVC4 has virtually the same performance as the original solver on SMT-LIB [9], the standard SMT test suite. The redesigned veriT does suffer a significant impact on FO performance. While it is, for example, three times slower on the QF_UF category of SMT-LIB due to its simpler ground solver, it still performs better on this category than CVC4, which shows its flexibility cost does not prevent the solver from being a suitable basis for handling HO formulas.

5.1 Proving HO theorems

The number of theorems proved by each solver configuration per benchmark set is given in Table 1. Grayed out cells represent unsupported benchmark sets. Figure 3 compares benchmarks solved per time. It only includes benchmark sets supported by all solvers (namely TH0 and the JD benchmarks).

[1] http://homepage.divms.uiowa.edu/~hbarbosa/papers/hosmt/

[2] Since veriT does not parse TPTP, its reported results are on the equivalent benchmarks as translated by CVC4 into the HOSMT language [6].

Solver	Total	TH0	oTH0	λoTH0	$\text{JD}_{\text{lift}}^{32}$	$\text{JD}_{\text{combs}}^{32}$	$\text{JD}_{\text{lift}}^{512}$	$\text{JD}_{\text{combs}}^{512}$	$\lambda o\text{SH}^{1024}$
#	9032	530	743	1915	1253	1253	1253	1253	832
@cvc	4318	384	344	940	457	459	655	667	**412**
@cvc-sax	4348	390	373	937	456	457	655	**668**	**412**
cvc	4232	389	342	865	463	447	**667**	654	405
cvc-sax	4275	389	376	883	458	443	**667**	654	405
@vt	2556	370	332		404	396	525	529	
vt	2671	369	346		426	424	550	556	
Ehoh	2631	394			489	481	637	630	
Leo-III	**4410**	**402**	452	1178	**491**	**482**	609	565	231
Satallax	3961	392	**457**	**1215**	394	390	407	404	302

Table 1: Proved theorems per benchmark set. Best results are in **bold**.

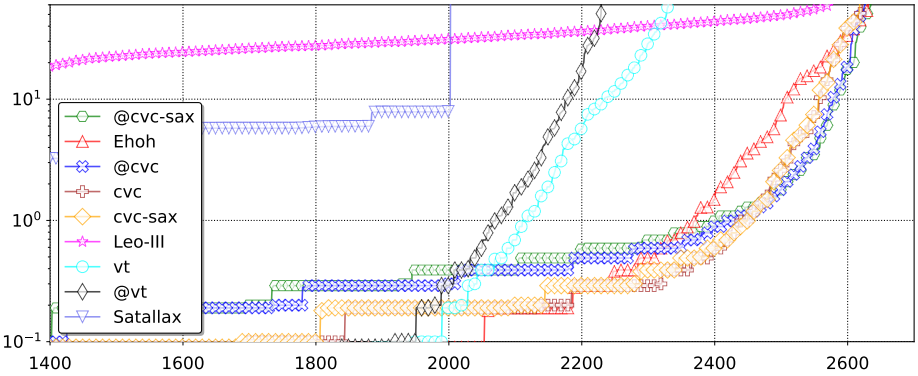

Fig. 3: Comparison on 5543 benchmarks, from TH0 and JD, supported by all solvers.

As expected, the results vary significantly between benchmark sets. Leo-III and Satallax have a clear advantage on TPTP, which contains a significant number of small logical problems meant to exercise the HO features of a prover. Considering the TPTP benchmarks from less to more expressive, i.e., including first-class Booleans and then λs, we see the advantages of these systems only increase. We also observe that both @cvc and cvc, but specially the latter, benefit from -sax as more complex benchmarks are considered in TPTP, showing that the disadvantage of not synthesizing λs can sometimes be offset by well-chosen axioms. Nevertheless, the results on $\lambda o\text{TH0}$ show that only this axiom is far from enough to offset the gap between @cvc and cvc, with cvc giving up more often from lack of instantiations to perform.

Sledghammer-generated problems stem from formalization efforts across different applications. As others note [53, 57], the bottleneck in solving these problems is often scalability and efficient FO reasoning, rather than a refined handling of HO constructs, specially as more facts are considered. Thus, the advantages gained by synthesizing λs are not sufficient to offset the scalability issues as more facts are considered, and Ehoh and CVC4 extensions eventually surpass the native HO provers. In particular, in the largest set we considered, $\lambda o\text{SH}^{1024}$, both @cvc and cvc have significant advantages.

Solver	Total	TH0	oTH0	λoTH0	$\text{JD}^{32}_{\text{lift}}$	$\text{JD}^{32}_{\text{combs}}$	$\text{JD}^{512}_{\text{lift}}$	$\text{JD}^{512}_{\text{combs}}$	$\lambda o\text{SH}^{1024}$
#	9032	530	743	1915	1253	1253	1253	1253	832
all-cvc-port	**4616**	**408** (8)	385 (2)	1001 (26)	482	482 (5)	**703** (38)	**702** (38)	**453** (137)
vt-port	2746	376 (1)	351 (3)		444 (3)	441 (3)	565	569	
Ehoh-port	2749	399 (1)			**494** (2)	**485** (1)	690 (31)	681 (31)	
Leo-III	4410	402 (1)	452 (21)	1178 (53)	491 (5)	482 (3)	609 (1)	565 (2)	231 (5)
Satallax	3961	392	**457** (18)	**1215** (101)	394	390	407 (8)	404 (3)	302 (24)

Table 2: Proved theorems by portfolio configurations of $[@]\text{cvc}[-\text{ui}][-\text{sax}]$, $[@]\text{veriT}$ and $\text{Ehoh}\,[\text{a}|\text{as}|\text{b}|\text{hb}]$, per benchmark set. Best results are in **bold**. Number of benchmarks solved uniquely are between parenthesis.

Solver	Total	TH0	oTH0	λoTH0	$\text{JD}^{32}_{\text{lift}}$	$\text{JD}^{32}_{\text{combs}}$	$\text{JD}^{512}_{\text{lift}}$	$\text{JD}^{512}_{\text{combs}}$	$\lambda o\text{SH}^{1024}$
#	9032	530	743	1915	1253	1253	1253	1253	832
@cvc-port	**4476**	**407** (7)	377 (6)	**965** (92)	**474** (12)	**476** (26)	670 (24)	**682** (29)	**425** (34)
cvc-port	4386	401 (1)	**379** (8)	909 (36)	470 (8)	456 (6)	**679** (33)	673 (20)	419 (28)

Table 3: Proved theorems by portfolio configurations of $@\text{cvc}[-\text{ui}][-\text{sax}]$, and $\text{cvc}[-\text{ui}][-\text{sax}]$. Best results are in **bold**. Number of benchmarks solved uniquely are between parenthesis.

As in λoTH0, @cvc also solves more problems than cvc in $\lambda o\text{SH}^{1024}$, which we attribute again to the expressivity difference, as otherwise cvc is often faster than @cvc, albeit by a small margin.

Both CVC4 configurations dominate JD^{512}, independent of the λ-encoding used, significantly ahead of Ehoh and Leo-III. Comparing the results between using λ-lifting or combinators, the former favors cvc and the latter, @cvc. These results, as well as the previously discussed ones, indicate that the pragmatic extension of CVC4 should not, in its current state, when it comes to refutations, substitute an encoding based approach, but complement it. In fact, a virtual best solver of all the CVC4 configurations, as well as others employing interleaved enumerative instantiation [44] (identified by -ui), in portfolio, would solve 703 problems in $\text{JD}^{512}_{\text{lift}}$, 702 in $\text{JD}^{512}_{\text{combs}}$, 453 in $\lambda o\text{SH}^{1024}$, and 408 in TH0, the most in these categories, even also considering a virtual best solver of all Ehoh configurations from [57]. The CVC4 portfolio would also solve 482 problems in $\text{JD}^{32}_{\text{lift}}$, and 482 in $\text{JD}^{32}_{\text{combs}}$, almost as good as Leo-III, and 1001 problems in λoTH0, Overall the virtual best CVC4 has a success rate 3 percentage points higher than @cvc on Sledghammer benchmarks, as well as overall, which represents a significant improvement when considering the usage of these solvers as backends for interactive theorem provers. Tables 2 and 3 summarize the results of the portfolio configurations, including the number of problems solved uniquely by each entry per benchmark set.

Differently from the pragmatic extension in CVC4, which provides more of an alternative to the full applicative encoding, the redesigned veriT is an outright improvement,

Solver	Total	TH0	oTH0	λoTH0	$\text{JD}^{32}_{\text{lift}}$	$\text{JD}^{32}_{\text{combs}}$	$\text{JD}^{512}_{\text{lift}}$	$\text{JD}^{512}_{\text{combs}}$	$\lambda o\text{SH}^{1024}$
#	9032	530	743	1915	1253	1253	1253	1253	832
@cvc-fmf-sax	224	58	**43**	80	20	18	**1**	**1**	**3**
cvc-fmf	**482**	**90**	17	**205**	**93**	**73**	**1**	**1**	2
Satallax	186	72	15	98	0	0	0	0	1

Table 4: Conjectures found countersatisfiable per benchmark set. Best results in **bold**.

with vt consistently solving more problems and with better solving times than @vt, specially on harder problems, as seen by the wider separation between them after 10s in Figure 3. Overall, veriT's performance, consistently with it being a lightweight solver, lags behind CVC4 and Ehoh as bigger benchmarks are considered, but it is comparable with the less optimized Leo-III and ahead of Satallax, thus validating the effort of redesigning the solver for a more refined handling of higher-order constructs and indicating that further extensions should be beneficial.

5.2 Providing countermodels to HO conjectures

The number of countermodels found by each solver configuration per benchmark set is given in Table 4. We consider the two CVC4 modes, @cvc and cvc, while performing finite-model-finding (-fmf) [46]. The builtin HO support in cvc is vastly superior to @cvc when it comes to model finding, as cvc-fmf greatly outperforms @cvc-fmf-sax. We note that @cvc-fmf is only model-sound if combined with -sax. Differently from cvc-fmf, which fails to provide a model as soon as it is faced with quantification over a functional sort, in @cvc-fmf functional sorts are encoded as atomic sorts. Thus it needs the extra axiom to ensure model soundness. For example, @cvc-fmf considers Example 5 satisfiable while @cvc-fmf-sax properly reports it unsatisfiable.

The high number of countermodels in JD^{32} indicates, not surprisingly, that providing few facts makes several SH goals unprovable. Note how JD^{512} and $\lambda o\text{SH}^{1024}$ are virtually devoid of countermodels. Albeit expected from using only 32 facts it is still useful to know where exactly the Sledghammer generation is being "incomplete" (i.e., making originally provable goals unprovable), which is difficult to determine without effective model finding procedures. We have also encountered a disagreement between cvc-fmf and Leo-III on exactly one JD^{32} benchmark, the only such disagreement across all of our experiments, with the former reporting a countermodel and the former a validity proof, and have yet to determine which of the systems is wrong.

6 Related work

Since the dawn of automated reasoning, mathematicians and ATPs developers have been actively working to improve automation on higher-order theorem proving. The pioneering work of Robinson [48] proposed to do it using a translation to reduce higher-order reasoning to first-order logic. Tools such as Sledgehammer [43], Miz$\mathbb{AR}$ [56],

HOLyHammer [35], and CoqHammer [22] build on this idea by automating HO reasoning via automatic FO provers. Earlier work as well related to native HO proving are Andrews's higher-order resolution [1], Huet's constrained resolution [32], Jensen and Pietrzykowski's ω-resolution [34], Snyder's higher-order E-resolution [50], Kohlhase's higher-order tableau [36], Backes and Brown's higher-order analytic tableaux [5] or Andrews's connections [2]. Modern HO provers are e.g. LEO-II [14] and Leo-III [51], respectively implementing HO resolution and HO paramodulation, and Satallax [20], based on a HO tableau calculus guided by a SAT solver. Another effective prover based on a focused sequent calculus is the Lindblad's AgsyHOL [37] prover guided by narrowing. A survey by Andrews [3] and one by Benzmuller and Miller [12] provide together an extensive overview of higher-order theorem proving techniques. However such systems are often not effective on first-order problems since they have been built primarily to solve HO problems. Our approach shares the same goal as recent work by Blanchette et al. [11, 57] on *gracefully* generalizing the superposition calculus [4, 40] to support higher-order reasoning, such that superposition provers can solve higher-order problems effectively while maintaining their efficiency at first-order ones. Unlike instantiation-based SMT solvers, however, superposition provers are much more sensitive to the applicative encoding, which can significantly decrease their performance [11]. Therefore, much of their work consists of extending the theoretical grounds on which a new generation of superposition provers that avoid the applicative encoding can be based on. Extending superposition involves completeness issue and is hard for full higher-order logic with Boolean subterms. As a preliminary step, Blanchette et al. have extended superposition calculus for the λ-free fragment of higher-order logic with completeness results [11]. First, implemented in the Zipperposition prover [21] and recently, integrated into the E-prover [49] by Vukmirovic et al. [57] demonstrating competitive results against state-of-the-art HO provers. As a second step, they are working on extending their calculus to λ-terms while preserving the completeness properties. A pragmatic extension of a superposition prover is being done by Vampire's developers [15], in which they lift-up their prover to higher-order reasoning by using full applicative translation, besides encoding Booleans and introducing axioms for Turner combinators to simulate higher-order unification [26]. The main challenge of their work, which is still in progress, resides in taming the combinator axioms.

7 Conclusions and future directions

We have presented extensions for SMT solvers to handle HOSMT problems. The pragmatic extension of CVC4, which can be implemented in any state-of-the-art SMT solver without significant effort, performs similarly to the standard encoding-based approach despite our limited support for HO instantiation techniques. Moreover, it allows numerous new problems to be solved by CVC4, with a portfolio approach performing very competitively and often ahead of state-of-the-art HO provers. The redesigned veriT on the other hand consistently outperforms its standard encoding-based counterpart, showing it can be the basis for future advancements towards stronger HO automation.

The natural extension for both approaches is integrating HO E-matching and E-unification into the core instantiation algorithms, thus allowing techniques such as

conflict-based and enumerative instantiation to synthesize λ-terms and allow SMT solvers to compete with full HO provers in problems containing complex HO constraints.

Acknowledgments We are grateful to Jasmin Blanchette and Pascal Fontaine for numerous discussions throughout the development of this work, for providing funding for research visits and for suggesting many improvements. We also thank Jasmin for generating several of the benchmarks with which we evaluate our approach; Simon Cruanes and Martin Riener for many fruitful discussions on the intricacies of HOL; Andres Nötzli for help with the table and plot scripts; Mathias Fleury, Hans-Jörg Schurr and Sophie Tourret for suggesting many improvements. This work has been partially supported by the National Science Foundation under Award 1656926 and by the European Research Council (ERC) starting grant Matryoshka (713999).

References

1. Peter B. Andrews. Resolution in type theory. *J. Symb. Log.*, 36(3):414–432, 1971.
2. Peter B. Andrews. On connections and higher-order logic. *J. Autom. Reason.*, 5(3):257–291, 1989.
3. Peter B. Andrews. Classical type theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume II, pages 965–1007. Elsevier and MIT Press, 2001.
4. Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
5. Julian Backes and Chad E. Brown. Analytic tableaux for higher-order logic with choice. *J. Autom. Reason.*, 47(4):451–479, 2011.
6. Haniel Barbosa, Jasmin Christian Blanchette, Simon Cruanes, Daniel El Ouraoui, and Pascal Fontaine. Language and proofs for higher-order SMT (work in progress). In Catherine Dubois and Bruno Woltzenlogel Paleo, editors, *PXTP 2017*, volume 262 of *EPTCS*, pages 15–22, 2017.
7. Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. Congruence closure with free variables. In Axel Legay and Tiziana Margaria, editors, *TACAS 2017*, volume 10206 of *LNCS*, pages 214–230. Springer, 2017.
8. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV 2011*, pages 171–177. Springer, 2011.
9. Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The smt-lib standard: version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017.
10. Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *FAIA*, chapter 26, pages 825–885. IOS Press, 2009.
11. Alexander Bentkamp, Jasmin Christian Blanchette, Simon Cruanes, and Uwe Waldmann. Superposition for lambda-free higher-order logic. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *IJCAR 2018*, volume 10900 of *LNCS*, pages 28–46. Springer, 2018.
12. Christoph Benzmüller and Dale Miller. Automation of higher-order logic. In Jörg H. Siekmann, editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 215–254. Elsevier, 2014.

13. Christoph Benzmüller, Florian Rabe, and Geoff Sutcliffe. THF0 - the core of the TPTP language for higher-order logic. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR 2008*, volume 5195 of *LNCS*, pages 491–506. Springer, 2008.
14. Christoph Benzmüller, Nik Sultana, Lawrence C. Paulson, and Frank Theiss. The higher-order prover LEO-II. *J. Autom. Reason.*, 55:389–404, 2015.
15. Ahmed Bhayat and Giles Reger. Set of support for higher-order reasoning. In Boris Konev, Josef Urban, and Philipp Rümmer, editors, *PAAR-2018*, volume 2162 of *CEUR Workshop Proceedings*, pages 2–16. CEUR-WS.org, 2018.
16. Jasmin Christian Blanchette. *Automatic proofs and refutations for higher-order logic*. PhD thesis, Technical University Munich, 2012.
17. Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formaliz. Reas.*, 9(1):101–148, 2016.
18. Sascha Böhme and Tobias Nipkow. Sledgehammer: judgement day. In Jürgen Giesl and Reiner Hähnle, editors, *IJCAR 2010*, volume 6173 of *LNCS*, pages 107–121. Springer, 2010.
19. Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. veriT: an open, trustable and efficient SMT-solver. In Renate A. Schmidt, editor, *CADE–22*, volume 5663 of *LNCS*, pages 151–156. Springer, 2009.
20. Chad E. Brown. Satallax: an automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR 2012*, volume 7364 of *LNCS*, pages 111–117. Springer, 2012.
21. Simon Cruanes. Superposition with structural induction. In Clare Dixon and Marcelo Finger, editors, *FroCoS 2017*, volume 10483 of *LNCS*, pages 172–188. Springer, 2017.
22. Łukasz Czajka and Cezary Kaliszyk. Hammer for Coq: automation for dependent type theory, 2018.
23. Leonardo de Moura and Nikolaj Bjørner. Efficient e-matching for smt solvers. In Frank Pfenning, editor, *CADE–21*, volume 4603 of *LNCS*, pages 183–198. Springer, 2007.
24. Leonardo de Moura and Nikolaj Bjørner. Generalized, efficient array decision procedures. In *FMCAD 2009*, pages 45–52. IEEE, 2009.
25. David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52:365–473, 2005.
26. Daniel J. Dougherty. Higher-order unification via combinators. *Theoretical Computer Science*, 114(2):273 – 298, 1993.
27. Gilles Dowek. Higher-order unification and matching. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume II, pages 1009–1062. Elsevier and MIT Press, 2001.
28. Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *J. ACM*, 27:758–771, 1980.
29. Michael Färber and Chad E. Brown. Internal guidance for Satallax. In Nicola Olivetti and Ashish Tiwari, editors, *IJCAR 2016*, volume 9706 of *LNCS*, pages 349–361. Springer, 2016.
30. Yeting Ge and Leonardo de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In Ahmed Bouajjani and Oded Maler, editors, *CAV 2009*, volume 5643 of *LNCS*, pages 306–320. Springer, 2009.
31. Leon Henkin. Completeness in the theory of types. *J. Symb. Log.*, 15(2):81–91, 1950.
32. Gerard P. Huet. A mechanization of type theory. In *IJICAI 1973*, pages 139–146. Morgan Kaufmann Publishers Inc., 1973.
33. R. J. M. Hughes. Super combinators: a new implementation method for applicative languages. In *Symposium on LISP and Functional Programming*, pages 1–10, 1982.
34. D.C. Jensen and T. Pietrzykowski. Mechanizing Omega-order type theory through unification. *Theoretical Computer Science*, 3:123 – 171, 1976.
35. Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: online ATP service for HOL Light. *Math. Comput. Sci.*, 9(1):5–22, 2015.

36. Michael Kohlhase. Higher-order tableaux. In Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors, *TABLEAUX '95*, volume 918 of *LNCS*, pages 294–309. Springer, 1995.
37. Fredrik Lindblad. A focused sequent calculus for higher-order logic. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *IJCAR 2014*, volume 8562 of *LNCS*, pages 61–75. Springer, 2014.
38. Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reas.*, 40(1):35–60, 2008.
39. Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27:356–364, 1980.
40. Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of automated reasoning*, volume 1, pages 371–443. 2001.
41. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: a proof Assistant for higher-order logic*, volume 2283 of *LNCS*. Springer, 2002.
42. Kohei Noshita. Translation of turner combinators in O(n log n) space. *IPL*, 20:71 – 74, 1985.
43. Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *IWIL-2010*, volume 2 of *EPiC*, pages 1–11. EasyChair, 2012.
44. Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In Dirk Beyer and Marieke Huisman, editors, *TACAS 2018*, volume 10806 of *LNCS*, pages 112–131. Springer, 2018.
45. Andrew Reynolds, Cesare Tinelli, and Leonardo de Moura. Finding conflicting instances of quantified formulas in SMT. In *FMCAD 2014*, pages 195–202. IEEE, 2014.
46. Andrew Reynolds, Cesare Tinelli, Amit Goel, and Sava Krstić. Finite model finding in SMT. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *LNCS*, pages 640–655. Springer, 2013.
47. Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstić, Morgan Deters, and Clark Barrett. Quantifier instantiation techniques for finite model finding in smt. In Maria Paola Bonacina, editor, *CADE–24*, volume 7898 of *LNCS*, pages 377–391. Springer, 2013.
48. John Alan Robinson. Mechanizing higher order logic. *Machine Intelligence*, 4:151–170, 1969.
49. Stephan Schulz. E - a brainiac theorem prover. *AI Commun.*, 15:111–126, 2002.
50. Wayne Snyder. Higher order E-unification. In Mark E. Stickel, editor, *CADE–10*, volume 449 of *LNCS*, pages 573–587. Springer, 1990.
51. Alexander Steen and Christoph Benzmüller. The higher-order prover Leo-III. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *IJCAR 2018*, volume 10900 of *LNCS*, pages 108–116. Springer, 2018.
52. Aaron Stump, Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. A decision procedure for an extensional theory of arrays. In *LICS 2001*, pages 29–37. IEEE Computer Society, 2001.
53. Nik Sultana, Jasmin Christian Blanchette, and Lawrence C. Paulson. LEO-II and Satallax on the Sledgehammer test bench. *J. Applied Logic*, 11:91–102, 2013.
54. Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *J. Autom. Reason.*, 43:337–362, 2009.
55. Geoff Sutcliffe. The CADE ATP system competition - CASC. *AI Magazine*, 37:99–101, 2016.
56. Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *J. Autom. Reason.*, 50(2):229–241, 2013.

57. Petar Vukmirović, Jasmin Christian Blanchette, Simon Cruanes, and Stephan Schulz. Extending a brainiac prover to lambda-free higher-order logic. In Tomas Vojnar and Lijun Zhang, editors, *TACAS 2019*, LNCS. Springer, 2019.