Higher-Order SMT Solving
(Work in Progress)

Haniel Barbosa[1], Andrew Reynolds[1], Pascal Fontaine[2], Daniel El Ouraoui[2], and
Cesare Tinelli[1]

[1] University of Iowa, Iowa City, USA
{haniel-barbosa,cesare-tinelli}@uiowa.edu,andrew.j.reynolds@gmail.com
[2] University of Lorraine, CNRS, Inria, and LORIA, Nancy, France
{daniel.el-ouraoui,pascal.fontaine}@inria.fr

Abstract

Satisfiability modulo theories (SMT) solvers have throughout the years been able to cope with increasingly expressive formulas, from ground logics to full first-order logic modulo theories. Nevertheless, higher-order logic within SMT (HOSMT) is still little explored. In this preliminary report we discuss how to extend SMT solvers to natively support higher-order reasoning without compromising their performances on FO problems. We present a pragmatic extension of the CVC4 solver in which we generalize existing data structures and algorithms to HOSMT, thus leveraging the extensive research and implementation efforts dedicated to efficient FO solving. Our evaluation shows that the initial implementation does not add significant overhead to FO problems and its performance is on par with the encoding-based approach for HOSMT. We also discuss an alternative extension being implemented in VERIT, in which new data structures and algorithms are being developed from scratch to best support HOSMT, thus avoiding the inherent difficulties of generalizing in a graceful way existing infrastructure not intended for higher-order reasoning.

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, machine-checkable formal 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, MizAℝ, Sledgehammer, and Why3, which provide a one-click connection from proof assistants to first-order provers, have led in recent years to considerable improvements in proof assistant automation [11]. Today, the leading automatic provers for first-order classical logic are based either on the superposition calculus [1, 25] or on CDCL($\mathcal{T}$) [24]. Those based on the latter are usually called satisfiability modulo theory (SMT) solvers [6] and are the focus of this paper. We present preliminary extensions of SMT solvers to natively support higher-order reasoning. Our goal is to avoid the completeness and performance issues associated with encoding higher-order problems into first-order ones.

The higher-order language we use is the syntax extension proposed by Barbosa et al. [2] to augment SMT-LIB [5], the standard first-order language recognized by SMT solvers, with partial applications, λ-abstractions, quantification on higher-order variables and application of variables. We adopt Henkin semantics [18, 9], in which the domain of function interpretations can be restricted to terms expressible in the language, so that we can search for refutations

relying on a natural lifting of the Herbrand theorem from first-order logic. Higher-order logic with Henkin semantics is thus semi-decidable, as is first-order logic. Part of our future work is to properly present the theoretical foundations and formal correctness of our techniques, but we are effectively developing sound but incomplete refutation calculi for extensional higher-order logic with Henkin semantics.

We present a pragmatic extension of the CVC4 [4] solver, which has two main components: extensions to the ground congruence closure procedure to account for partial applications and extensionality (Section 2.1); and extensions to the E-matching algorithm to account for partial applications and higher-order variables (Section 2.2). We assume without loss of generality that both components operate on λ-free terms, which we achieve by performing λ-lifting at preprocessing in a standard way [20]. Partial applications are supported by means of the *applicative encoding*, a complete approach for reducing HO proving to FO proving [21]: the problem signature is extended with a family of binary symbols $@_{\tau_1,\tau_2}$ to represent the (curried) application of terms of function type τ_1 to arguments of type τ_2. By considering function symbols as constants, partial applications become regular first-order applications. To avoid the performance issues of applying this reduction eagerly to regular applications (which is required for completeness), we apply this reduction lazily by means of our extension to the congruence closure procedure (Section 2.1.1). Techniques to account for extensionality and higher-order variables are described in Sections 2.1.2 and 2.2, respectively, and mostly involve engineering challenges. Finally, we evaluate the initial CVC4 implementation in Section 2.3, observing no significant overhead in FO problems due to the extended algorithms and that on HOSMT the new system performs similarly to the old one on the first-order-encoding-based equivalent problems.

We also discuss an alternative extension implemented in VERIT [13], in which new data structures and algorithms are being developed from scratch to best support HOSMT. This will provide more flexibility to later develop new techniques specially suited for higher-order reasoning. As a starting point, we reimplemented a ground decision procedure with native support for higher-order terms. It is based on congruence closure, but with a focus on simplicity rather than optimal complexity. Indeed simplicity of the algorithm will be instrumental to extend its expressivity in the future. Even so, we show the algorithm is quite efficient on QF_UF, the SMT-LIB category which is most sensitive to the efficiency of the congruence closure algorithm. We are currently proceeding to extend the instantiation infrastructure to higher-order, and in particular, the CCFV [3] framework which actually interacts heavily with the congruence closure algorithm.

2   A pragmatic extension for HOSMT

For simplicity, and without loss of generality, we consider λ-free formulas in Skolem form, with all quantified subformulas being quantified clauses; we also assume all atomic formulas are equalities. CDCL($\mathcal{T}$) solvers proceed by enumerating assignments for the propositional abstraction of the input formula, i.e. the formula obtained by replacing every atom and quantified subformula by a proposition. Such an assignment corresponds to a set $E \cup Q$, in which E and Q are conjunctive sets of ground literals and quantified formulas, respectively. If $E \cup Q$ is consistent, all of its models also satisfy the input formula; if not, a new assignment is derived. A ground solver first checks the satisfiability of E, and, if it is satisfiable, an instantiation module derives ground instances I from Q such that the satisfiability of $E \cup I$ is checked. This is repeated until either a conflict is found, and a new assignment for the propositional abstraction must be produced, or no more instantiations are possible. Of course, the whole process might

not terminate and the solver might loop indefinitely.

We discuss below how CVC4 has been extended to check the satisfiability of E in a setting with partial applications and to instantiate Q when functional variables are present.

2.1   Extending the ground solver

2.1.1   Lazy applicative encoding

We extend the ground decision procedure in CVC4 by introducing equalities between first-order terms (e.g. $f(a)$) and their equivalent representation in the applicative encoding (e.g. $@(f, a)$).

Inputs in the higher-order extension of SMT LIB [2] may contain partial applications of function symbols. For example, if f is of type $\mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$ and a is an integer, then we say the term $f(a)$ is a *partial application* of f; we understand type $\mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$ as type $\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ when the function f is partially applied. Notice this term has function type $\mathsf{Int} \to \mathsf{Int}$. As a preprocessing step, all partial applications are turned into total applications by using the binary @ symbols and considering function symbols as constants. Above, $f(a)$ would be replaced by $@(f, a)$. Once 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. For the sake of performance, we only introduce these equalities for regular terms which interact with partially applied ones. In particular, we introduce equalities only for function symbols that appear as members of congruence classes in the E-graph (i.e. the congruence closure of E built by the ground 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 is kept internal to the E-graph, therefore avoiding that it affects other parts of the ground decision procedure.

For example, consider the constraints $\{@(f, a) \simeq g,\ f(a, a) \not\simeq g(a),\ g(a) \simeq h(a)\} \subseteq E$ where g has type $\mathsf{Int} \to \mathsf{Int}$. We have that E is initially found to be satisfiable. Equalities are added to E since f and g appear in the E-graph, resulting in the set

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

When determining the satisfiability of E' the equality $@(@(f, a), a) \simeq @(g, a)$ will be derived by congruence and hence $f(a, a) \simeq g(a)$ will be derived by transistivity, which leads to a conflict that would otherwise not have been detected without the additional equalities. 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 $h(a) \simeq @(h, a)$ is not introduced in this example.

2.1.2   Extensionality

When reasoning about equality between terms of function types, we must ensure all models satisfy the extensionality property, characterized by the axiom $\forall \bar{x}.\ f(\bar{x}) \simeq g(\bar{x}) \leftrightarrow f \simeq g$ for all functions f and g of same type. Notice that the "$\leftarrow$" direction comes as a consequence of congruence in the above scheme. If e.g. both $f \simeq g$ and $f(a) \not\simeq g(a)$ are asserted for some a, a conflict will be derived. To ensure the "$\rightarrow$" direction, first, for each disequality between functions $f \not\simeq g$, we infer the disequality $f(\bar{k}) \not\simeq g(\bar{k})$ for some fresh $\bar{k}$, which witnesses the contrapositive of the above extensionality property. We also apply this lemma for certain pairs of functions f and g that are neither asserted to be equal nor disequal in the current assignment,

which is based on the cardinality of their types. If f and g are of some type that has infinite cardinality, it is always possible to satisfy the disequality $f(\bar{k}) \not\simeq g(\bar{k})$ for fresh $\bar{k}$. On the other hand, if f and g have function types of finite cardinality, the above disequality may lead to a conflict.

Consider for example the constraint $distinct(p_1, p_2, p_3)$, in which p_1, p_2, p_3 are predicates over the same type U. The $distinct$ predicate holds if and only if all of its arguments are interpreted differently. If U has cardinality one, this constraint is unsatisfiable since it enforces that there must exist elements k_1, k_2, k_3 of type U such that $p_1(k_1) \not\simeq p_2(k_1)$, $p_1(k_2) \not\simeq p_3(k_2)$, and $p_2(k_3) \not\simeq p_3(k_3)$. Since U has cardinality one, we have that $k_1 \simeq k_2 \simeq k_3$, and thus the three disequalities are unsatisfiable since the range of p_1, p_2, p_3 is a finite set with cardinality 2. Therefore in this example, we would introduce the lemmas:

$$p_1 \simeq p_2 \vee p_1(k_1) \not\simeq p_2(k_1)$$
$$p_1 \simeq p_3 \vee p_1(k_3) \not\simeq p_3(k_3)$$
$$p_2 \simeq p_3 \vee p_2(k_2) \not\simeq p_3(k_2)$$

where k_1, k_2, k_3 are fresh Skolem constants to ensure that the decision procedure detects the inconsistency. We add lemmas of this form for each pair of functions f and g having the same (finite) type where we have yet to infer $f \simeq g$ or $f \not\simeq g$.

2.1.3 Model generation for ground formulas

When CVC4 determines that a problem is satisfiable it can produce a first-order model M as a witness. The models generated by SMT solvers like CVC4 map functions f to a function, denoted $M(f)$, of the form $\lambda x\, \mathsf{ite}(x \simeq t_1, s_1, \dots \mathsf{ite}(x \simeq t_{n-1}, s_{n-1}, s_n) \dots)$, where ite denotes if-then-else. 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

$$f_1(0) \simeq f_1(1) \wedge f_1(1) \simeq f_2$$
$$f_2(0) \simeq f_2(1) \wedge f_2(1) \simeq f_3$$
$$f_3(0) \simeq f_3(1) \wedge f_3(1) \simeq 2$$

in which $f_1 : \mathsf{Int} \times \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$, $f_2 : \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$, and $f_3 : \mathsf{Int} \to \mathsf{Int}$. To produce the model values of f_1 as a list of total applications with three arguments into an integer, we would need to account for 8 cases. In other words, we require 8 ite cases to indicate $f_1(x, y, z) \simeq 2$ for all inputs where $x, y, z \in \{0, 1\}$. 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.

To avoid such an exponential behavior, model building for higher-order functions in CVC4 assigns values for functions in terms of the other functions that their partial applications are equated to. This way in the above example f_1 would have only two model values, depending on whether the first argument of its application is 0 or 1, by using the model values of f_2 applied on its two other arguments. In other words, we construct $M(f_1)$ as the term:

$$\lambda xyz\, \mathsf{ite}(x \simeq 0, M(f_2)(y, z), \mathsf{ite}(x \simeq 1, M(f_2)(y, z), \_))$$

where $M(f_2)$ 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.

Respecting extensionality   In order to be consistent with the extensionality axiom, model construction explicitly makes functions f and g which are not asserted equal to be disequal when building a model. This is accomplished by adding disequalities of the form $f(\bar{k}) \not\simeq g(\bar{k})$ for fresh $\bar{k}$. Such disequalities can always be satisfied because functions ranging over finite types would already be part of such an inference added by the decision procedure as described in the previous section. Hence, f and g must be of infinite type, and as noted earlier, we can always create a new element of the type to witness their disequality.

2.2   Extending the instantiation module

Currently CVC4 only applies trigger-based instantiation, or E-matching [16] for solving problems that involve higher-order constraints. These techniques are based on choosing trigger terms t containing free variables $\bar{x}$, and deriving substitutions $\{\bar{x} \mapsto \bar{s}\}$ such that E entails that $t\{\bar{x} \mapsto \bar{s}\}$ is equivalent to some term g in E. In this case, we say that g E-matches t under the substitution $\{\bar{x} \mapsto \bar{s}\}$. For example, if E is $\{f(a) \simeq g(b),\ a \simeq g(b)\}$, then $f(a)$ E-matches $f(g(x))$ under the substitution $\{x \mapsto b\}$. To accomodate function variable applications and partial applications, the standard algorithm for E-matching must be adapted in several ways.

   One of the challenges when using the applicative encoding is that regular indexing techniques, which are paramount for efficient E-matching, are dependent on looking at the head of the term. However, in the presence of equalities between functions, it may be the case that a match involves a trigger term and ground term with different head symbols. For example, if E contains the equality $@(f, a) \simeq g$ and the term $f(a, b)$ where $f : \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$ and $g : \mathsf{Int} \to \mathsf{Int}$, then notice that $g(x)$ is equivalent modulo E to the term $f(a, b)$ under the substitution $x \mapsto b$. Such a match is found by indexing all terms that are applications of *either* $@(f, a)$ or g in a common term index. This ensures that when we find matches for $g(x)$, the application $f(a, b)$ (whose applicative counterpart is $@(@(f, a), b)$) is considered.

   With this approach the regular first-order E-matching algorithm of CVC4 can be used for trigger-based instantiation. Our evaluation shows this performs on par with the first-order encoded benchmarks, even though the remaining instantiation techniques of CVC4 are not yet available for higher-order instantiation.

Towards higher-order E-matching.   Let $p : \mathsf{Int} \to \mathsf{Bool}$, $q : \mathsf{Int} \to \mathsf{Bool}$, and $k : \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$ and consider the formula

$$\varphi = q(k(0, 1)) \wedge \neg p(k(0, 0)) \wedge \forall (f : \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int})\,(y, z : \mathsf{Int}).\, p(f(y, z)) \vee \neg q(f(1, y))$$

The satisfiability of φ depends on which semantics we use. We distinguish between two semantics, as defined by Bentkamp et al. [8]: "λ-free Henkin semantics" and regular Henkin semantics. In Henkin semantics the universes of interpretation for functions need only contain the functions that are expressible as terms, so whether we allow or not λ-abstractions as valid terms in our logic determines which semantics we want to use. In λ-free Henkin semantics the above problem is satisfiable, as there is no instantiation for the functional variable f into a lambda-free term that leads to a refutation. However if we allow λ-abstractions, and therefore use regular Henkin semantics, the above problem can be found unsatisfiable with e.g. the instantiation $\{f \mapsto \lambda w_1 w_2.\, k(0, w_1),\ y \mapsto 0,\ z \mapsto 0\}$.

   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 [19]. 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 $\langle f(y, z), k(0,0) \rangle$ would find the substitution $\{f \mapsto k, y \mapsto 0, z \mapsto 0\}$. Our procedure may then generate the following instantiations for f:

$$f \mapsto \lambda w_1 w_2. \, k(w_1, w_2) \tag{1}$$
$$f \mapsto \lambda w_1 w_2. \, k(w_2, w_1) \tag{2}$$
$$f \mapsto \lambda w_1 w_2. \, k(0, w_1) \tag{3}$$
$$f \mapsto \lambda w_1 w_2. \, k(w_1, 0) \tag{4}$$
$$f \mapsto \lambda w_1 w_2. \, k(0, w_2) \tag{5}$$
$$f \mapsto \lambda w_1 w_2. \, k(w_2, 0) \tag{6}$$
$$f \mapsto \lambda w_1 w_2. \, k(0, 0) \tag{7}$$

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 show that φ unsatisfiable with Henkin semantics.

We have yet to evaluate the effectiveness of this technique in CVC4, but once we consider benchmarks which are only unsatisfiable w.r.t. regular Henkin semantics such techniques based on higher-order unification will be paramount. Ideally the benchmarks themselves should specify w.r.t. which semantics their satisfiability should be checked, so that the appropriate solving technique can be chosen.

2.3   Evaluation

We have evaluated our pragmatic extension of CVC4 on a benchmark set derived from the "Judgment Day" test harness [12]. It consists of 1240 provable goals originating from different Isabelle [26] formalizations such that each one is encoded into a first-order problem in SMT-LIB [5] and into an HOSMT problem. The encodings maintain the invariant that the SMT-LIB problem is unsatisfiable if and only if the HOSMT problem is unsatisfiable in the λ-free Henkin semantics. Moreover, the encoded HOSMT problems are λ-free.

We present the results of two configurations of CVC4, with and without support for HOSMT, i.e. to partial applications, application of variables and higher-order quantification. We denote the version with the techniques described in this paper by CVC4-HO. Our goal is to measure whether CVC4-HO has a significant decrease in performance on the first-order problems w.r.t. CVC4 and how CVC4-HO performs on HOSMT problems in comparison with CVC4 on their first-order counterpart.

	hosmt		smt-lib	
	#unsat	avg time (s)	#unsat	avg time (s)
CVC4-HO	648	1.08	662	1.02
CVC4	4	0.06	662	1.01

Table 1: Summary of CVC4 configurations on "Judgement day" benchmarks with 60s timeout.

Our experiments were performed on the StarExec logic solving service [32]. The timeout was set for 60 seconds, since our goal is evaluating SMT solvers as back-ends of verification and ITP platforms, which require fast answers. The results for problems solved within the
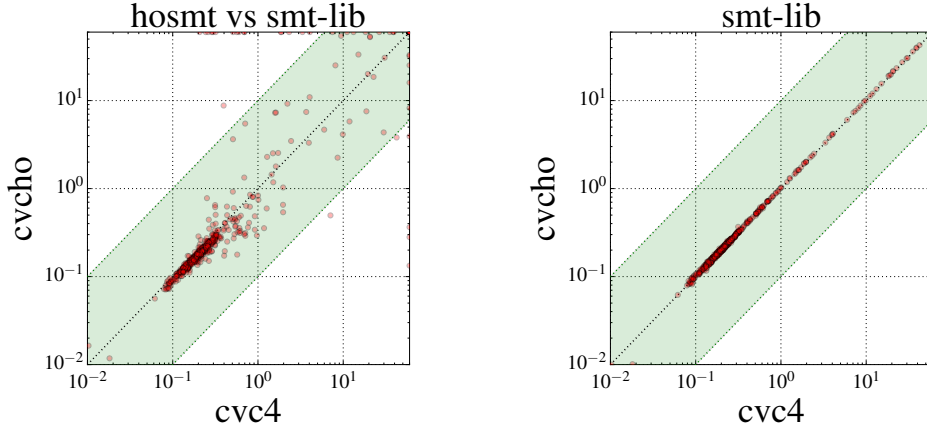
Figure 1: Time comparison of CVC4 configurations on "Judgement day" benchmarks.
The left figure compares CVC4-HO on "Judgement day" higher-order problems with CVC4 on the translation to first-order of the same problems. The right figure evaluates both CVC4 and CVC4-HO on the same first-order problems.

timeout are summarized in Table 1 and Figure 1. Note that CVC4 solves 4 HOSMT problems, which indicates that these problems contained no partial applications and that proving their unsatisfiability did not depend on high-order instantiation.

In the comparison on the effectiveness of solving HOSMT rather than the encoded first-order counterpart, we have that CVC4-HO solves slightly less problems than CVC4 while being slightly slower on average. The "hosmt vs smt-lib" scatter plot in Figure 1 shows that there is a gain/loss behavior as well, which we can attribute to the heuristic nature of instantiation in SMT solvers. In total, CVC4-HO solves 10 more and 24 less problems than CVC4. The "smt-lib" scatter plot shows that CVC4-HO and CVC4 have virtually the same performance on first-order problem and that they solve the same number of problems, indicating that our pragmatic extension costs no significant overhead for pure first-order solving.

Given the preliminary state of our implementation we consider these results promising. There are important instantiation techniques, such as finding conflicting instances [28], available for first-order problems that we have not yet generalized to the higher-order case. This indicates to us that once a more mature implementation of our pragmatic extension is in place we can outperform CVC4 w.r.t. the first-order encoded problems.

3   Redesigning a solver for HOSMT

As seen previously, the main difficulties in extending an SMT solver to higher-order logic come from partially applied functions and functions symbols as arguments of other functions or as quantified variables. There already exist some automatic provers with native support for higher-order logic, e.g. Satalax [14] and LEO-II [10], LEO-III [31], respectively extending the tableau calculus and resolution to higher-order reasoning. It seems natural to also lift-up SMT solving techniques to higher-order, and not only use encodings, even if it is an on-the-fly encoding as shown in Section 2. This approach requires however much more work in the core data-structures and algorithms, and can only be done within a lightweight solver. The VERIT solver seems an
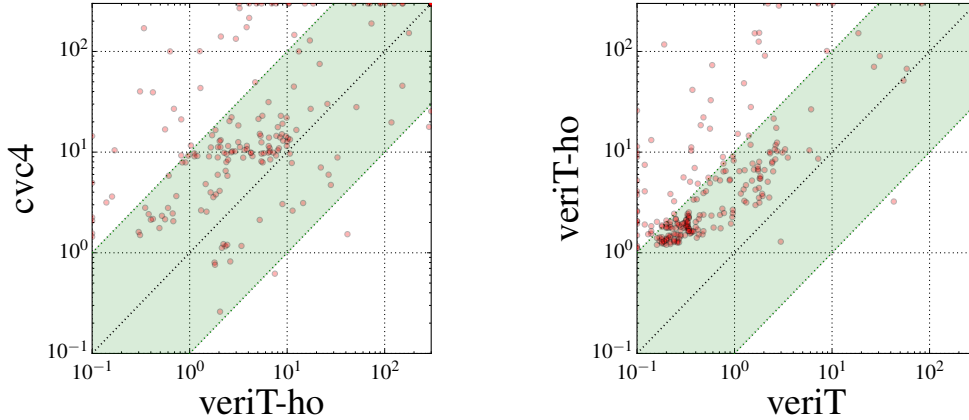
Figure 2: Time comparison of CVC4 VERIT and VERIT-Ho on QFUF benchmarks.

appropriate basis.

As above, the solver should be adapted in two ways. First, since SMT reasoning is mostly based on ground solving, the ground decision procedure for uninterpreted symbols should first handle natively both partial functions and functions as arguments of other functions. Second, instantiation techniques should also be adapted and complemented for higher-order reasoning. The decision procedure for uninterpreted symbols in the VERIT solver is essentially based on the classical congruence closure algorithm developed by Downey et al. [17] and Nelson and Oppen [22], whose asymptotic time complexity is $\mathcal{O}(n \log n)$. The congruence closure module in VERIT more precisely follows Nieuwenhuis et al. [23] which is more appropriate to SMT with an underlying SAT solver. As good as it is, this algorithm is however quite complex and not really flexible, especially considering the expected demands of our future higher-order SMT solver: data structures are strongly nested together and extensions are hard. We thus reimplemented a new, simpler module, suitable for the changes we envision. This is at the cost of complexity (the algorithm is quadratic and not $\mathcal{O}(n \log n)$ anymore) but better integrates with various other features such as term addition, injective functions, rewriting or even to do computation, in particular for β and η transformations. We also aim at a better integration with the CCFV [3] framework, which we intend to extend to HO unification and thus fully lift-up VERIT to HOSMT. Instead of union-find data-structures, this simple algorithm operates straightforwardly on a graph where nodes are the terms, and edges are 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. The algorithm uses a curried representation: the term $f(a, g(b, c))$ is understood as $((f\,a)\,((g\,b)\,c))$, much like in the original presentation by Downey et al. [17]. Partial functions handled straightforwardly, as well as functions as arguments of other functions. The congruence closure algorithm operates on pairs, left and right being handled totally symmetrically in contrast to FO congruence closure.

All this can be considered as folklore (see e.g. [15, 30]) but an important question is: how much efficiency do we pay for simplicity and higher-order? Figure 2 shows that VERIT with the simpler congruence closure is around 3 times slower on the QF_UF category of the SMT-LIB than the standard version of VERIT. The VERIT solver was second in this category in the SMT-COMP 2017, behind Yices and in front of CVC4. The simpler solver is still better

than CVC4, which means using the simpler congruence closure would not have changed the ranking in the SMT-COMP 2017. This ground solver is thus a good basis to handle higher-order formulas. We are now focusing on higher-order instantiation to make the solver ready to deal with higher-order formulas natively.

4 Related work

Since the dawn of automated reasoning much has been done on automating higher-order theorem proving, from the pioneering work of Robinson [29] and Huet [19] to more recent approaches such as in the higher-order ATPs LEO-II [10], Leo-III [31] and Satallax [14]. However such systems are often not effective on first-order problem, since they have been built primarily to solve HO problems. Nevertheless, we intend to investigate HO provers based on instantiation, such as Satallax, to determine whether we can adapt some of their successful techniques for HO solving to HOSMT without detriment to performance on FO solving.

Our approach shares the same goal of recent work by Blanchette et al. [7, 8, 33] on *gracefully* generalizing the superposition calculus [1, 25] to handle higher-order reasoning, such that superposition provers can solve higher-order problems effectively while maintaining their efficiency at first-order ones. Differently from instantiation-based SMT solvers, however, superposition provers are much more sensitive to the applicative encoding, which can significantly decrease their performance on first-order problems [8]. 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. Blanchette et al. have promising results [8, 33] with such extensions for the λ-free fragment of higher-order logic, but they have yet to extend their work to tackle full higher-order problems.

5 Future directions

We have presented extensions for SMT solvers to handle HOSMT problems. The pragmatic extension of CVC4 performs similarly to the standard encoding-based approach despite our limited support for instantiation techniques. We plan to continue our extension to perform higher-order E-matching as described in Section 2.2 and lift complete techniques such as enumerative instantiation [27] in order to obtain effective and refutationally complete calculi w.r.t. Henkin semantics. We also plan to lift conflict-based instantiation [28] to HOSMT by extending the instantiation module of CVC4 to perform some limited form of higher-order E-unification.

In ongoing work we are improving the implementation in veriT to combine the new ground decision procedure with the instantiation module and evaluate the new framework in a similar manner as done for CVC4 in Section 2.3. Extending the instantiation module to perform higher-order E-matching and E-unification will also allow performing both trigger-based and conflict-based instantiation in HOSMT.

References

[1] Leo Bachmair and Harald Ganzinger. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[2] 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, *Workshop on Proof eXchange for Theorem Proving (PxTP)*, volume 262 of *EPTCS*, pages 15–22, 2017.

[3] Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. Congruence closure with free variables. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 10206 of *Lecture Notes in Computer Science*, pages 214–230, 2017.

[4] 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, *Computer Aided Verification (CAV)*, pages 171–177. Springer, 2011.

[5] 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. Available at www.SMT-LIB.org.

[6] 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 *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, 2009.

[7] Heiko Becker, Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand. A transfinite knuth-bendix order for lambda-free higher-order terms. In Leonardo de Moura, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 10395 of *Lecture Notes in Computer Science*, pages 432–453. Springer, 2017.

[8] Alexander Bentkamp, Jasmin Christian Blanchette, Simon Cruanes, and Uwe Waldmann. Superposition for lambda-free higher-order logic. IJCAR 2018.

[9] 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.

[10] Christoph Benzmüller, Nik Sultana, Lawrence C. Paulson, and Frank Theiss. The higher-order prover leo-ii. *J. Autom. Reasoning*, 55(4):389–404, 2015.

[11] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.

[12] Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement day. In Jürgen Giesl and Reiner Hähnle, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 6173 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2010.

[13] 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, *Proc. Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156. Springer, 2009.

[14] ChadE. Brown. Satallax: An automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning*, volume 7364 of *Lecture Notes in Computer Science*, pages 111–117. Springer Berlin Heidelberg, 2012.

[15] Pierre Corbineau. Deciding equality in the constructor theory. In Thorsten Altenkirch and Conor McBride, editors, *Types for Proofs and Programs*, pages 78–92, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[16] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A Theorem Prover for Program Checking. *J. ACM*, 52(3):365–473, 2005.

[17] Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression

problem. *J. ACM*, 27(4):758–771, October 1980.

[18] Leon Henkin. Completeness in the theory of types. *J. Symb. Log.*, 15(2):81–91, 1950.

[19] Gerard P. Huet. A mechanization of type theory. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 139–146, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[20] Thomas Johnsson. Lambda lifting: Transforming programs to recursive equations. In *Proc. Of a Conference on Functional Programming Languages and Computer Architecture*, pages 190–203, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[21] Manfred Kerber. How to prove higher order theorems in first order logic. In John Mylopoulos and Raymond Reiter, editors, *IJCAI-91*, pages 137–142. Morgan Kaufmann, 1991.

[22] Greg Nelson and Derek C. Oppen. Fast Decision Procedures Based on Congruence Closure. *J. ACM*, 27(2):356–364, 1980.

[23] Robert Nieuwenhuis and Albert Oliveras. Proof-producing congruence closure. In Jürgen Giesl, editor, *Rewriting Techniques and Applications (RTA)*, volume 3467 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 2005.

[24] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). *J. ACM*, 53(6):937–977, November 2006.

[25] 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. Elsevier and MIT Press, 2001.

[26] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[27] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 10806 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 2018.

[28] Andrew Reynolds, Cesare Tinelli, and Leonardo Mendonça de Moura. Finding conflicting instances of quantified formulas in SMT. In *Formal Methods In Computer-Aided Design (FMCAD)*, pages 195–202. IEEE, 2014.

[29] John Alan Robinson. Mechanizing higher order logic. *Machine Intelligence*, 4:151–170, 1969.

[30] Daniel Selsam and Leonardo de Moura. Congruence closure in intensional type theory. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning: 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 – July 2, 2016, Proceedings*, pages 99–115, Cham, 2016. Springer International Publishing.

[31] Alexander Steen and Christoph Benzmüller. The higher-order prover leo-iii. IJCAR 2018.

[32] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. Starexec: A cross-community infrastructure for logic solving. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 8562 of *Lecture Notes in Computer Science*, pages 367–373. Springer, 2014.

[33] Petar Vukmirović. *Implementation of Lambda-Free Higher-Order Superposition*. PhD thesis, Vrije Universiteit Amsterdam, 2018.