E-Matching with Free Variables

Philipp Rümmer

Department of Information Technology, Uppsala University, Sweden

Abstract. E-matching is the most commonly used technique to handle quantifiers in SMT solvers. It works by identifying characteristic subexpressions of quantified formulae, named triggers, which are matched during proof search on ground terms to discover relevant instantiations of the quantified formula. E-matching has proven to be an efficient and practical approach to handle quantifiers, in particular because triggers can be provided by the user to guide proof search; however, as it is heuristic in nature, e-matching alone is typically insufficient to establish a complete proof procedure. In contrast, free variable methods in tableau-like calculi are more robust and give rise to complete procedures, e.g., for first-order logic, but are not comparable to e-matching in terms of scalability. This paper discusses how e-matching can be combined with free variable approaches, leading to calculi that enjoy similar completeness properties as pure free variable procedures, but in which it is still possible for a user to provide domain-specific triggers to improve performance.

1 Introduction

SAT and SMT solvers form the backbone of many of today's verification systems, responsible for discharging verification conditions that encode correctness properties of hardware or software designs. Such verification conditions are often generated in the context of intricate theories, including various kinds of arithmetic, uninterpreted functions and equality, the theory of arrays, or the theory of quantifiers. Despite much research over the past years, efficient and scalable reasoning in the combination of such theories remains challenging: in particular for handling quantifiers, most state-of-the-art SMT solvers have to resort to heuristic techniques like e-matching and triggers [7, 8]. E-matching is a popular method due to its simplicity and performance, but offers little completeness guarantees and is sensitive to syntactic manipulations of input formulae.

This paper takes the standpoint that heuristics like e-matching should be considered as *optimisations,* and triggers as *hints,* possibly affecting the performance, but not the completeness of an SMT solver. In other words, the set of formulae that a solver can prove should be independent from chosen triggers. Working towards this goal, the paper presents calculi integrating constraint-based free variable reasoning with e-matching, the individual contributions being (i) a free variable sequent calculus for first-order logic (Sect. 3), with support for e-matching and user-provided triggers to guide instantiation of quantified formulae, partly inspired by the positive unit hyper-resolution calculus [14, 15]; (ii) a similar calculus for first-order logic modulo linear integer arithmetic (Sect. 5), extending the calculus in [23]; (iii) as a component of both calculi, an approach to capture functions and congruence closure procedures (commonly used in SMT) as uninterpreted predicates (Sect. 4); (iv) a complete implementation of the calculus in (ii), called PRINCESS, and experimental evaluation against SMT solvers competing in the SMT competition 2011 (AUFLIA category) (Sect. 6).

The calculus in (i) is sound and complete for first-order logic, while (ii) is sound and complete for fragments such as Presburger arithmetic, the universal and the existential fragment of first-order logic modulo integers, and the languages accepted by related methods like $\mathcal{ME}$(LIA) [4] and the complete instantiation method in [9]. The completeness results are significantly stronger than those guaranteed by most SMT solvers, and hold *independently* from the application of e-matching or the choice of triggers in proofs.

1.1 Introductory Example

We start by illustrating e-matching and free variable methods using an example. The first-order theory of non-extensional arrays [16] is often encoded using uninterpreted function symbols *sel* and *sto* by means of the following axioms:

$$\forall x, y, z.\, \mathit{sel}(\underline{\mathit{sto}(x, y, z)}, y) \doteq z \tag{1}$$

$$\forall x, y_1, y_2, z.\, \big(y_1 \doteq y_2 \vee \underline{\mathit{sel}(\mathit{sto}(x, y_1, z), y_2)} \doteq \mathit{sel}(x, y_2)\big) \tag{2}$$

Intuitively, $\mathit{sel}(x, y)$ retrieves the element of array x stored at position y, while $\mathit{sto}(x, y, z)$ denotes the array that is identical to x, except that position y stores value z. In order to prove that some formula holds over the theory of arrays, the underlined expressions can be used as *triggers* that determine when and how the axioms should be instantiated. Generally, triggers consist of a single or multiple expressions (normally sub-expressions in the body of the quantified formula) that contain all quantified variables. For instance, to prove that the implication

$$b \doteq \mathit{sto}(a, 1, 2) \;\rightarrow\; \mathit{sel}(b, 2) \doteq \mathit{sel}(a, 2) \tag{3}$$

holds over the theory of arrays, we can observe that the term $\mathit{sel}(\mathit{sto}(a, 1, 2), 2)$ occurs in the implication, modulo some equational reasoning. This term matches the underlined pattern in (2), and suggests to instantiate (2) to obtain the instance $1 \doteq 2 \vee \underline{\mathit{sel}(\mathit{sto}(a, 1, 2), 2)} \doteq \mathit{sel}(a, 2)$. In fact, (3) follows for this instance of (2), when reasoning in the theories of uninterpreted functions and arithmetic, which allows us to conclude the validity of (3).

Axioms and triggers as shown above are commonly used in SMT solvers, and give rise to efficient decision procedures for ground problems over arrays.[1] However, in the presence of quantifiers, e-matching might be unable to determine the right instantiations, possibly because required instantiations do not exist as ground terms in the formula. For instance, variants of (3) might include:

$$b \doteq \mathit{sto}(a, 1, 2) \;\rightarrow\; \exists x.\, \mathit{sel}(b, x) \doteq \mathit{sel}(a, 2) \tag{4}$$

$$b \doteq \mathit{sto}(a, 1, 2) \;\rightarrow\; \exists x.\, \mathit{sel}(b, x+1) \doteq \mathit{sel}(a, 2) \tag{5}$$

$$b \doteq \mathit{sto}(a, 1, 2) \;\rightarrow\; \exists x.\, \mathit{sel}(b, x) \doteq \mathit{sel}(a, x) \tag{6}$$

Although the formulae are still valid, the match $sel(sto(a, 1, 2), 2)$ used previously has been eliminated, which makes proof search more intricate. The state-of-the-art e-matching-based SMT solver CVC3 ([3], version 2.4.1) is able to solve (3), but none of (4), (5), (6). A more realistic example, though similar in nature to the formulae shown here, was reported in [12, Sect. 3.3], where a simple modification (Skolemisation) of a small formula prevented Z3 [19] from finding a proof. The goal of the calculus developed in this paper (and of our implementation PRINCESS) is to obtain a system that is more robust against such modifications, by combining e-matching with constraint-based free variable reasoning, while retaining the scalability of SMT solvers.

The general philosophy of free variable methods [11] is to delay the choice of instantiations for quantified formulae with the help of symbolic reasoning. For example, we could instantiate the formula $\exists x. sel(b, x + 1) \doteq sel(a, 2)$ using a free variable X, resulting in $sel(b, X + 1) \doteq sel(a, 2)$. Modulo equational reasoning, this creates the term $sel(sto(a, 1, 2), X+1)$, which can be unified with the trigger in (2) under the constraint $X \doteq 1$. It is then possible to proceed with the proof as described above. After closing the proof, we can conclude that (5) indeed holds, since the derived constraint $X \doteq 1$ is satisfiable: it is possible (retrospectively) to instantiate $\exists x. sel(b, x + 1) \doteq sel(a, 2)$ with the concrete term $X = 1$.

This example demonstrates that a free variable calculus can be used to compute answers to queries, in a manner similar to constraint logic programming. The system developed in this paper is more general than "ordinary" logic programming, however, since no restrictions on the use of quantifiers are imposed.

2 Background

2.1 Syntax and Semantics of Considered Logics

We assume familiarity with classical first-order logic (FOL, e.g., [11]). Let x range over an infinite set X of variables, c over an infinite set C of constant symbols, p over a set P of uninterpreted predicates with fixed arity, f over a set F of uninterpreted functions with fixed arity, and α over the set $\mathbb{Z}$ of integers. The syntax of the unityped logics in this paper is defined by the following grammar:

$$\begin{aligned} \phi \; &::= \; \phi \wedge \phi \;\big|\; \phi \vee \phi \;\big|\; \neg\phi \;\big|\; \forall x.\phi \;\big|\; \exists x.\phi \;\big|\; t \doteq 0 \;\big|\; t \leq 0 \;\big|\; p(t, \ldots, t) \\ t \; &::= \; \alpha \;\big|\; c \;\big|\; x \;\big|\; \alpha t + \cdots + \alpha t \;\big|\; f(t, \ldots, t) \end{aligned}$$

The symbol t denotes terms constructed using functions and arithmetic operations. A formula ϕ is called *closed* if all variables in ϕ are bound by quantifiers, and *ground* if it does not contain variables or quantifiers. A location within a formula ϕ is called *positive* if it is underneath an even number of negations $\neg$, otherwise *negative*. Simultaneous substitution of terms $\bar{t} = (t_1, \ldots, t_n)$ for variables $\bar{x} = (x_1, \ldots, x_n)$ in ϕ is denoted by $[\bar{x}/\bar{t}]\phi$; we assume that variable capture

[1] We are grateful to the anonymous referees pointing out that a further trigger (not shown here) is needed in (2) for a complete array procedure.

is avoided by renaming bound variables as necessary. For simplicity, we sometimes write $s \doteq t$ as a shorthand of $1 \cdot s + (-1) \cdot t \doteq 0$. The abbreviation *true* (*false*) stands for $0 \doteq 0$ ($1 \doteq 0$), and implication $\phi \to \psi$ for $\neg\phi \vee \psi$.

We consider fragments of the syntax shown above, including function-free first-order logic (Sect. 2.3, 3), full first-order logic (Sect. 4), and first-order logic with linear integer arithmetic (Sect. 5). Semantics of any such logic $\mathcal{L}$ is defined by identifying a class $\mathcal{S}_{\mathcal{L}}$ of structures (U, I), where U is a non-empty *universe,* and I is an *interpretation* that maps predicates $p \in P$ to relations over U, functions $f \in F$ to set-theoretic functions over U, and constants $c \in C$ to values in U. Given (U, I), the evaluation of terms and formulae is defined recursively as is common. A closed formula is called *valid* if it evaluates to *true* for all structures $(U, I) \in \mathcal{S}_{\mathcal{L}}$, and *satisfiable* if it evaluates to *true* for at least one structure.

2.2 Sequent Calculi with Constraints

Throughout the paper we will work with the *constraint sequent calculus* that is introduced in [23]. The calculus differs from normal Gentzen-style sequent calculi [11] in that every sequent $\Gamma \vdash \Delta$ is annotated with a constraint C (written $\Gamma \vdash \Delta \Downarrow C$) that captures unification conditions derived in a sub-proof. Such unification conditions come into play when free variables (which technically are treated as constants) are used to instantiate quantified formulae. All calculi in this paper are designed such that constraints cannot contain uninterpreted predicates or functions, so that validity/satisfiability of constraints is decidable. Proof procedures and refinements for the calculi are discussed in [23, 22].

More formally, if Γ, Δ are finite sets of closed formulae (the *antecedent* and *succedent*) and C is a closed formula, then $\Gamma \vdash \Delta \Downarrow C$ is called a *constrained sequent.* A sequent $\Gamma \vdash \Delta \Downarrow C$ is called *valid* if the formula $(\bigwedge \Gamma \wedge C) \to \bigvee \Delta$ is valid. A calculus rule is a binary relation between finite sets of sequents (the premises) and single sequents (the conclusion). Proof trees are defined as is common as trees growing upwards in which each node is labelled with a constrained sequent, and in which each node that is not a leaf is related with the nodes directly above through an instance of a calculus rule. A proof is closed if it is finite, and if all leaves are justified by a rule instance without premises.

2.3 The Basic Calculus for Function-Free First-Order Logic

At the core of all calculi introduced in this paper is a calculus for first-order logic with equality, at this point including uninterpreted predicates, but no functions:

$$\phi_{\mathrm{FOL}} ::= \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \forall x.\phi \mid \exists x.\phi \mid s \doteq s \mid p(\bar{s}) \qquad s ::= c \mid x$$

Since functions and arithmetic are not included in the logic, terms can only be (symbolic) constants or bound variables. Semantics is defined over the class $\mathcal{S}_{\mathrm{FOL}}$ of structures (U, I) with arbitrary non-empty universe U. The constraint calculus PredEq^C for the logic is shown in Fig. 1, with constraints consisting of (possibly negated) equalities, Boolean connectives, and quantifiers. The validity

$$\frac{\Gamma, \phi \vdash \Delta \Downarrow C \qquad \Gamma, \psi \vdash \Delta \Downarrow D}{\Gamma, \phi \vee \psi \vdash \Delta \Downarrow C \wedge D}\ \vee\textsc{l} \qquad \frac{\Gamma, \phi, \psi \vdash \Delta \Downarrow C}{\Gamma, \phi \wedge \psi \vdash \Delta \Downarrow C}\ \wedge\textsc{l} \qquad \frac{\Gamma \vdash \phi, \Delta \Downarrow C}{\Gamma, \neg\phi \vdash \Delta \Downarrow C}\ \neg\textsc{l}$$

$$\frac{\Gamma \vdash \phi, \Delta \Downarrow C \qquad \Gamma \vdash \psi, \Delta \Downarrow D}{\Gamma \vdash \phi \wedge \psi, \Delta \Downarrow C \wedge D}\ \wedge\textsc{r} \qquad \frac{\Gamma \vdash \phi, \psi, \Delta \Downarrow C}{\Gamma \vdash \phi \vee \psi, \Delta \Downarrow C}\ \vee\textsc{r} \qquad \frac{\Gamma, \phi \vdash \Delta \Downarrow C}{\Gamma \vdash \neg\phi, \Delta \Downarrow C}\ \neg\textsc{r}$$

$$\frac{\Gamma, [x/c]\phi, \forall x.\phi \vdash \Delta \Downarrow [x/c]C}{\Gamma, \forall x.\phi \vdash \Delta \Downarrow \exists x.C}\ \forall\textsc{l} \qquad \frac{\Gamma, [x/c]\phi \vdash \Delta \Downarrow [x/c]C}{\Gamma, \exists x.\phi \vdash \Delta \Downarrow \forall x.C}\ \exists\textsc{l} \qquad \frac{\Gamma \vdash \Delta \Downarrow C}{\Gamma, s \doteq t \vdash \Delta \Downarrow s \not\doteq t \vee C}\ {=}\textsc{l}$$

$$\frac{\Gamma \vdash [x/c]\phi, \exists x.\phi, \Delta \Downarrow [x/c]C}{\Gamma \vdash \exists x.\phi, \Delta \Downarrow \exists x.C}\ \exists\textsc{r} \qquad \frac{\Gamma \vdash [x/c]\phi, \Delta \Downarrow [x/c]C}{\Gamma \vdash \forall x.\phi, \Delta \Downarrow \forall x.C}\ \forall\textsc{r} \qquad \frac{*}{\Gamma \vdash s \doteq t, \Delta \Downarrow s \doteq t}\ {=}\textsc{r}$$

$$\frac{*}{\Gamma, p(s_1, \ldots, s_n) \vdash p(t_1, \ldots, t_n), \Delta \Downarrow \bigwedge_i s_i \doteq t_i}\ \textsc{pc} \qquad \frac{[s/t]\Gamma, s \doteq t \ \vdash\ [s/t]\Delta \Downarrow C}{\Gamma, s \doteq t \ \vdash\ \Delta \Downarrow C}\ {=}\textsc{red}$$

Fig. 1. The rules of the calculus PredEq^C for first-order predicate logic. In all rules, c is a constant that does not occur in the conclusion: in contrast to the use of Skolem functions and free variables in tableaux, the same kinds of symbols (constants) are used to handle both existential and universal quantifiers. Arbitrary renaming of bound variables is allowed in the constraints when necessary to avoid variable capture.

of formulae of this kind is decidable by quantifier elimination [11]. The calculus is analytic and contains two rules for each formula constructor, as well as a closure rule PC to unify complementary literals. As an optimisation, the rule =RED can be used to destructively apply equations; the rule is not necessary to establish completeness, but relevant (together with further refinements) to turn PredEq^C into a practical calculus [23, 22].

Lemma 1 (Soundness [22]). *If a sequent $\Gamma \vdash \Delta \Downarrow C$ is provable in PredEq^C, then it is valid (holds in all $\mathcal{S}_{FOL}$-structures).*

In particular, proving a sequent $\Gamma \vdash \Delta \Downarrow C$ with a valid constraint C implies that also the implication $\bigwedge \Gamma \to \bigvee \Delta$ is valid. This gives rise to a constraint-based proof procedure that iteratively constructs proof trees for an input sequent $\Gamma \vdash \Delta \Downarrow ?$ with a yet unknown constraint. The constraints in a proof can be filled in once all proof branches have been closed. In each iteration, the procedure checks whether the constraint generated by the current proof is valid, in which case the procedure can terminate with the result that the input problem has been proven; otherwise, the current proof has to be unfolded further. Strategies for generating proofs (without the need for backtracking, i.e., undoing previous proof steps) are discussed in [23].

Example 2. We show how to prove $\neg\forall x.(\neg p(x) \vee x \doteq c) \vee \neg p(d) \vee p(c)$, in which $p \in P$ is a unary predicate and $c, d \in C$ are constants:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{p(d) \ \vdash\ p(a) \Downarrow d \doteq a}\ \textsc{pc}}{\neg p(a), p(d) \ \vdash\ \ldots \Downarrow d \doteq a}\ \neg\textsc{l} \qquad \dfrac{\dfrac{*}{a \doteq c, p(d) \ \vdash\ p(c) \Downarrow d \doteq c}\ \textsc{pc}}{a \doteq c, p(d) \ \vdash\ p(c) \Downarrow a \not\doteq c \vee d \doteq c}\ {=}\textsc{l}}{\ldots, \neg p(a) \vee a \doteq c, p(d) \ \vdash\ p(c) \Downarrow d \doteq a \wedge (a \not\doteq c \vee d \doteq c)}\ \vee\textsc{l}}{\forall x.(\neg p(x) \vee x \doteq c), p(d) \ \vdash\ p(c) \Downarrow R}\ \forall\textsc{l}}{\vdash\ \neg\forall x.(\neg p(x) \vee x \doteq c) \vee \neg p(d) \vee p(c) \Downarrow R}\ \vee\textsc{r}*, \neg\textsc{r}*$$

In order to instantiate the universal quantifier, the fresh constant a is introduced; the constant is quantified existentially in the derived constraints, and therefore can be seen as a "free variable." The constraints on the right-hand side of $\Downarrow$ are practically filled in *after* closing the proof using PC. The validity of the original formula follows from the validity of $R = \exists x.(d \doteq x \wedge (x \not\doteq c \vee d \doteq c))$.

Lemma 3 (Completeness [24]). *Suppose ϕ is closed and valid. Then there is a valid constraint C such that $\vdash \phi \Downarrow C$ is provable in $PredEq^C$.*

3 Positive Unit Hyper-Resolution

As argued in Sect. 1.1, axioms and quantified formulae (in particular in verification problems) are often manually formulated with a clear, directed application strategy in mind. This makes it possible to systematically instantiate axioms in a manner that more resembles the execution of a functional or logic program than the search for a proof. From a practical point of view, providing support for this style of reasoning (even if it is only applicable to a subset of input problems) is crucial to achieve the scalability needed for applications. We integrate such user-guided reasoning into our calculus with the help of concepts from the *positive unit hyper-resolution* (PUHR) calculus, an approach first used in the SATCHMO theorem prover [14, 15]. PUHR will be used in Sect. 4 to simulate the e-matching method common in SMT solvers.

PUHR is a tableau procedure in which clauses are instantiated by matching negative literals on (ground) literals already present on a proof branch. Starting from the calculus PredEq^C defined in the last section, we introduce a similar rule in our hyper-resolution sequent calculus PredEqHR^C, instantiating quantified formulae that are "guarded" by negative literals $\neg p_1(\bar{t_1}), \ldots, \neg p_n(\bar{t_n})$ using symbols from matching literals $p_1(\bar{s_1}), \ldots, p_n(\bar{s_n})$ in the antecedent of a sequent:

$$\frac{\varGamma, \{p_i(\bar{s_i})\}_{i=1}^n,\ \forall\bar{x}.\left(\bigvee_{i=1}^n \neg p_i(\bar{t_i}) \vee \phi\right),\ \mathit{simp}\left(\forall\bar{x}.\left(\bigvee_{i=1}^n \bar{s_i} \not\doteq \bar{t_i} \vee \phi\right)\right) \vdash \varDelta \Downarrow C}{\varGamma, \{p_i(\bar{s_i})\}_{i=1}^n,\ \forall\bar{x}.\left(\bigvee_{i=1}^n \neg p_i(\bar{t_i}) \vee \phi\right) \vdash \varDelta \Downarrow C}\ \forall\text{L-M}$$

Given literals $\{p_i(\bar{s_i})\}_{i=1}^n$ in a sequent, a quantified formula $\forall\bar{x}.\left(\bigvee_{i=1}^n \neg p_i(\bar{t_i}) \vee \phi\right)$ can be instantiated using the argument terms $\bar{s_i}$ by simultaneously solving the systems $\bar{s_i} \doteq \bar{t_i}$ of equalities. In contrast to the original PUHR [14], we do not require formulae to be range restricted. Note that the formula ϕ might be *false* and disappear, and that the literals $\{p_i(\bar{s_i})\}_{i=1}^n$ are not necessarily distinct. The solving of equalities is formulated using a recursive simplification function *simp*:

$$\begin{array}{rcll}
\mathit{simp}(\forall\bar{x}.(t \not\doteq t \vee \phi)) & = & \mathit{simp}(\forall\bar{x}.\phi) & \\
\mathit{simp}(\forall\bar{x}.(x_i \not\doteq t \vee \phi)) & = & \mathit{simp}(\forall\bar{x}.[x_i/t]\phi) & (x_i \neq t) \\
\mathit{simp}(\forall\bar{x}.(t \not\doteq x_i \vee \phi)) & = & \mathit{simp}(\forall\bar{x}.[x_i/t]\phi) & (x_i \neq t) \\
\mathit{simp}(\forall\bar{x}.(s \not\doteq t \vee \phi)) & = & s \not\doteq t \vee \mathit{simp}(\forall\bar{x}.\phi) & (s, t \notin \bar{x}) \\
\mathit{simp}(\forall\bar{x}.\phi) & = & \forall(\bar{x} \cap \mathit{fv}(\phi)).\, \phi & (\text{otherwise})
\end{array}$$

A rule $\exists$R-M similar to $\forall$L-M is introduced for existentially quantified formulae $\exists\bar{x}.\left(\bigwedge_{i=1}^n p_i(\bar{t_i}) \wedge \phi\right)$ in the succedent. The soundness of the new rules is immediate, since the rules only introduce instances of quantified formulae already present in a sequent. After adding $\forall$L-M and $\exists$R-M, it is possible to impose the side-condition that the rule $\forall$L is no longer allowed to be applied to formulae $\forall \bar{x}.\, (\bigvee_{i=1}^{n} \neg p_i(\bar{t_i}) \vee \phi)$; similarly for $\exists$R. In other words, the ordinary rules $\forall$L and $\exists$R may only be applied to formulae that do not start with negative literals. We denote the resulting calculus by PredEqHR^C.

Example 4. We show how the proof from Example 2 can be carried over to PredEqHR^C. To this end, observe that the formula $\forall x.(\neg p(x) \vee x \doteq c)$ in the antecedent is amenable to hyper-resolution, so that it is no longer necessary to introduce the constant a in the proof. Also proof splitting can now be avoided:

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{d \doteq c, p(d) \ \vdash\ p(c) \ \Downarrow d \doteq c}\ \textsc{pc}}{\ldots, d \doteq c, p(d) \ \vdash\ p(c) \ \Downarrow d \not\doteq c \vee d \doteq c}\ {=}\textsc{l}}{\forall x.(\neg p(x) \vee x \doteq c), p(d) \ \vdash\ p(c) \ \Downarrow \mathit{true}}\ \forall\textsc{l-m}}{\vdash\ \neg \forall x.(\neg p(x) \vee x \doteq c) \vee \neg p(d) \vee p(c) \ \Downarrow \mathit{true}}\ \vee\textsc{r}{*}, \neg\textsc{r}{*}$$

$\forall$L-M introduces the formula $\mathit{simp}(\forall x.(d \not\doteq x \vee x \doteq c))$, which can be simplified to $d \doteq c$. A further optimisation is the use of =RED to minimise constraints.

Lemma 5 (Completeness [24]). *Suppose ϕ is closed and valid. Then there is a valid constraint C such that $\vdash \phi \Downarrow C$ is provable in PredEqHR^C.*

Importantly for efficiency, a variety of refinements [22] restricting applications of $\exists$R-M, $\forall$L-M can be imposed, without losing this completeness result.

4 E-Matching through Relational Encoding

For practical applications, uninterpreted functions are more common and often more important than uninterpreted predicates. Uninterpreted functions and equalities are in SMT solvers normally represented using congruence closure methods [21], which build a *congruence graph* (also called *e-graph*) containing nodes for all function terms present in a problem, with edges representing asserted equalities. More formally, given a finite subterm-closed set T of terms and a finite set E of equalities, the congruence graph is the undirected graph (T, E'), where $E' \supseteq E$ is the smallest transitive and reflexive set of edges satisfying:

> if $f(s_1, \ldots, s_n), f(t_1, \ldots, t_n) \in T$ are nodes with $\{(s_1, t_1), \ldots, (s_n, t_n)\} \subseteq E'$, then also $(f(s_1, \ldots, s_n), f(t_1, \ldots, t_n)) \in E'$.

The relation E' can be constructed by fixed-point iteration, starting from the given equalities E. Congruence graphs can be used to efficiently decide whether an equality $s \doteq t$ follows from the set E of equalities. The congruence graph is also used as the underlying datastructure for e-matching, since matching terms (modulo equations) can efficiently be found using the congruence graph. We discuss in this section how both congruence closure and e-matching can be understood as an encoding of functions as uninterpreted predicates, enabling the integration of e-matching with free variables, without preventing the implementation of congruence closure with the help of efficient native datastructures.

4.1 Relational Encoding of Functions

We consider first-order logic including function symbols, which means that the grammar for terms shown in the beginning of Sect. 2.3 is extended to:

$$s \ ::= \ c \mid x \mid f(s, \ldots, s)$$

where $f \in F$ ranges over function symbols. For the purpose of the encoding of functions into relations, we assume that a fresh $(n+1)$-ary uninterpreted predicate $f_p \in P$ exists for every n-ary uninterpreted function $f \in F$, representing the graph of f. The relation f_p satisfies two axioms, *functionality* and *totality:*

$$\mathit{Fun}_f = \forall \bar{x}, y_1, y_2.\ \big(\neg f_p(\bar{x}, y_1) \vee \neg f_p(\bar{x}, y_2) \vee y_1 \doteq y_2\big), \quad \mathit{Tot}_f = \forall \bar{x}. \exists y.\ f_p(\bar{x}, y)\ .$$

We can then translate from formulae ϕ over the functional vocabulary F (and relational vocabulary P) to formulae ϕ_{Rel} purely over the relational vocabulary P. This can be done by means of the following rewriting rules:

$$\begin{array}{ll} \exists\text{-}\mathit{enc}: & \psi[f(\bar{t})] \ \rightsquigarrow\ \exists x.\ (f_p(\bar{t}, x) \wedge \psi[x]) \\ \forall\text{-}\mathit{enc}: & \psi[f(\bar{t})] \ \rightsquigarrow\ \forall x.\ (\neg f_p(\bar{t}, x) \vee \psi[x]) \end{array}$$

Both rules have the side condition that rewritten occurrences of $f(\bar{t})$ must not be in the scope of quantifiers binding variables in the terms $\bar{t}$; furthermore, the variable x must be fresh in $\psi[f(\bar{t})]$. It is possible, however, to apply the rewriting rules to arbitrary sub-formulae of a given formula ϕ; in other words, the predicate and quantifier that encode a function application $f(\bar{t})$ can be placed arbitrarily in the rewritten formula, as long as the function application remains in the scope of the quantifier. Rewriting strategies are discussed later in this section.

Lemma 6. *Suppose ϕ is a closed formula over the vocabulary F, and ϕ_{Rel} is a function-free formula obtained from ϕ by application of the rewriting rules $\exists$-enc and $\forall$-enc. Then ϕ is valid iff $\bigwedge_{f \in F} \big(\mathit{Fun}_f \wedge \mathit{Tot}_f\big) \to \phi_{Rel}$ is valid.*

Since the calculi PredEq^C and $\mathrm{PredEqHR}^C$ are sound and complete for first-order logic without function symbols, we can therefore construct calculi for first-order logic including functions by first encoding functions as relations.

4.2 Ground Reasoning and Congruence Closure

We first concentrate on quantifier-free first-order formulae with functions. In this setting, it is easy to see that the hyper-resolution calculus $\mathrm{PredEqHR}^C$, in combination with the functionality axioms Fun_f for functions f, is able to simulate congruence closure procedures. This is supported by the following strengthened version of Lem. 6, which observes that totality axioms are not necessary when solving essentially ground formulae:

Lemma 7. *Suppose ϕ is a closed formula over the vocabulary F, and ϕ_{Rel} a function-free formula obtained from ϕ by application of the rewriting rules $\exists$-enc and $\forall$-enc that contains $\forall$-quantifiers only in positive positions, and $\exists$-quantifiers only in negative positions. Then ϕ is valid iff $\bigwedge_{f \in F} \mathit{Fun}_f \to \phi_{Rel}$ is valid.*

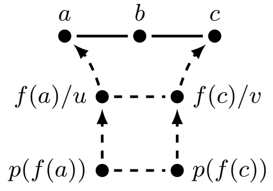
The assumptions of the lemma require that the rewriting rule $\forall$-*enc* is only applied in positive, and $\exists$-*enc* only in negative positions when deriving ϕ_{Rel} from ϕ. As a result, there are only two kinds of quantifiers in the last formula in Lem. 7: quantifiers in ϕ_{Rel} that can be eliminated with the help of the rules $\exists$L and $\forall$R by means of Skolem symbols, and the quantifiers in the axioms Fun_f. Since the latter can be handled using $\forall$L-M, formulae $\bigwedge_{f \in F} \mathit{Fun}_f \to \phi_{Rel}$ can be proven in the calculus PredEqHR^C purely through ground reasoning, without ever resorting to the rules $\forall$L/$\exists$R that introduce existentially quantified constants. This style of reasoning closely corresponds to congruence closure, with literals $f_p(\bar{t}, s)$ in the antecedent of sequents representing equivalence classes of nodes of the congruence graph (T, E'), and instantiation of axioms Fun_f simulating the addition of further edges to the congruence relation E'.

Example 8. We show how $\phi = (p(f(a)) \wedge a \doteq b \wedge b \doteq c \to p(f(c)))$ is proven using the relational encoding. The corresponding formula ϕ_{Rel} is obtained by replacing the function terms $f(a), f(b)$ with fresh quantified variables x, y:

$$\phi_{Rel} = \quad \forall x, y. \big(f_p(a, x) \wedge f_p(c, y) \wedge p(x) \wedge a \doteq b \wedge b \doteq c \;\to\; p(y)\big)$$

We can then construct a proof of $\mathit{Fun}_f \to \phi_{Rel}$ using the rules =RED and $\forall$L-M. The central step in the proof is to conclude $u \doteq v$ by instantiating the axiom Fun_f using the symbols occurring in the literals $f_p(c, u)$ and $f_p(c, v)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\mathit{Fun}_f, u \doteq v, f_p(c, v), p(v) \;\vdash\; p(v)}\;\textsc{pc}}{\mathit{Fun}_f, u \doteq v, f_p(c, u), f_p(c, v), p(u), \ldots \;\vdash\; p(v)}\;{=}\textsc{red}}{\mathit{Fun}_f, f_p(c, u), f_p(c, v), p(u), a \doteq c, b \doteq c \;\vdash\; p(v)}\;\forall\textsc{l-m}}{\mathit{Fun}_f, f_p(b, u), f_p(c, v), p(u), a \doteq b, b \doteq c \;\vdash\; p(v)}\;{=}\textsc{red}}{\mathit{Fun}_f, f_p(a, u), f_p(c, v), p(u), a \doteq b, b \doteq c \;\vdash\; p(v)}\;{=}\textsc{red}}{\vdash\; \mathit{Fun}_f \to \phi_{Rel}}\;\forall\textsc{r}, \ldots$$



The constraint $\Downarrow$ *true* of each of the sequents has been left out. The proof can also be visualised using the congruence graph shown on the right.

4.3 Relational E-Matching and Free Variables to Handle Quantifiers

E-matching instantiates quantified formulae $\forall x.\phi$ by means of pattern matching: triggers are identified in the matrix ϕ, and are compared with the expressions occurring in the congruence graph to determine relevant instances of the formula. This process can be simulated using the relational function encoding, in combination with the hyper-resolution calculus PredEqHR^C, by deliberately choosing whether literals $f_p(\bar{t}, x)$ in the relational formula $\forall x.\phi_{Rel}$ are introduced with positive or negative sign: since the unit-hyper-resolution rule $\forall$L-M only considers *negative* literals in the matrix ϕ_{Rel} of $\forall x.\phi_{Rel}$ for matching, it is possible to encode triggers by negating the respective literals $f_p(\bar{t}, x)$ (i.e., by using the rewriting rule $\forall$-*enc* to generate such literals), and keeping all other literals positive using the rule $\exists$-*enc*.

Example 9. Consider the quantified formula $\forall x. f(x) \doteq g(x)$. Four possible ways of encoding the formula using relations, corresponding to different strategies when applying the rules $\forall$-*enc* and $\exists$-*enc*, are:

$$\forall x. \exists y, z. \big(f_p(x, y) \wedge g_p(x, z) \wedge y \doteq z\big) \tag{7}$$

$$\forall x, y. \big(\neg f_p(x, y) \vee \exists z. (g_p(x, z) \wedge y \doteq z)\big) \tag{8}$$

$$\forall x, z. \big(\neg g_p(x, z) \vee \exists y. (f_p(x, y) \wedge y \doteq z)\big) \tag{9}$$

$$\forall x, y, z. \big(\neg f_p(x, y) \vee \neg g_p(x, z) \vee y \doteq z\big) \tag{10}$$

Each of the relational formulae corresponds to a particular selection of triggers in $\forall x. f(x) \doteq g(x)$:

- in (7), no triggers have been chosen, with the result that the hyper-resolution rule $\forall$L-M is not applicable. Instantiation of (7) is only possible using the rule $\forall$L, replacing the bound variable x with an existentially quantified constant that can later unified with some term.
- in (8), the term $f(x)$ (corresponding to the negative literal $f_p(x, y)$) has been selected as trigger. In the calculus PredEqHR^C, (8) can only be instantiated using the rule $\forall$L-M, and only in case a literal $f_p(s, t)$ occurs in the antecedent of a sequent, substituting the terms s, t for the variables x, y. This corresponds to e-matching the expression $f(x)$ on a node $f(t)$ of a congruence graph. No free variables are needed to instantiate (8).
- similarly, in (9) the term $g(x)$ is trigger.
- in (10), both $f(x)$ and $g(x)$ have been chosen as a *multi-trigger*, which means that (10) only can be instantiated if literals $f_p(s, t)$ and $g_p(s', t')$ occur in an antecedent. In this case, the instance $s \not\doteq s' \vee t \doteq t'$ will be generated, expressing that the equality $t \doteq t'$ can be assumed if s and s' are unifiable. In terms of e-graphs, the formula would only be instantiated if the e-graph contains nodes $f(s), g(s')$ such that s, s' are in the same equivalence class.

The following proof fragment illustrates how (9) can be instantiated referring to a literal $g_p(a, b)$ in the antecedent, effectively adding $f_p(a, b)$ to the sequent:

$$\cfrac{\cfrac{\cfrac{g_p(a,b), (9), f_p(a,b), b \doteq u \ \vdash\ \Downarrow [y/u]C}{g_p(a,b), (9), f_p(a,u), u \doteq b \ \vdash\ \Downarrow [y/u]C}\ {=}\textsc{red}}{g_p(a,b), (9), \exists y. (f_p(a,y) \wedge y \doteq b) \ \vdash\ \Downarrow \forall y. C}\ \exists\textsc{l}, \wedge\textsc{l}}{g_p(a,b), (9) \ \vdash\ \Downarrow \forall y. C}\ \forall\textsc{l-m} \tag{11}$$

The way in which a formula ϕ is translated to ϕ_{Rel} determines how quantified sub-formulae are instantiated, in the same way as SMT solvers can be guided by specifying triggers (Alg. 1 shows how the translation can be done systematically, for a given set of triggers). However, it can be observed that the four encodings (7)–(10) are all equivalent w.r.t. provability of theorems: in combination with the axioms Fun_f, Fun_g, Tot_f, Tot_g each of the formulae can simulate each other formula. *The choice of triggers in formulae therefore only influences efficiency, not completeness.* For instance, formula (9) in (11) can be replaced

Algorithm 1: EncodeTrigger: relational encoding of a quantified formula for a specific set of triggers

```
Input: Formula $\forall \bar{x}.\phi$, set $T$ of trigger terms with variables from $\bar{x}$
Output: Relational formula $\phi_{Rel}$
$qvars \leftarrow \{x \mid x \in \bar{x}\}$;
$premises \leftarrow \emptyset$;
while $T$ contains function terms do
    pick (sub)term $f(\bar{t})$ in $T$ s.t. $\bar{t}$ does not contain functions;
    pick fresh variable $y$;
    $qvars \leftarrow qvars \cup \{y\}$;
    $premises \leftarrow premises \cup \{f_p(\bar{t}, y)\}$;
    substitute $y$ for $f(\bar{t})$ everywhere in $T$ and $\phi$;
end
apply $\exists$-enc exhaustively to $\phi$;
return $\forall_{x \in qvars}.\big(\bigvee_{p \in premises} \neg p \lor \phi\big)$;
```

with (8) in the following way (the constraints of the sequents have been left out for sake of brevity):

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{
          \dfrac{
            \dfrac{*}{\ldots \vdash x \doteq a}\;{=}\mathrm{R}
            \qquad
            \dfrac{f_p(x,b), g_p(x,b), g_p(a,b), v \doteq b \vdash}{f_p(x,v), g_p(x,v), g_p(a,b), v \doteq b \vdash}\;{=}\mathrm{RED}
          }{\ldots, f_p(x,v), g_p(x,v), g_p(a,b), x \not\doteq a \lor v \doteq b \vdash}\;\lor\mathrm{L}, \neg\mathrm{L}
        }{Fun_g, f_p(x,v), g_p(x,v), v \doteq u, g_p(a,b) \vdash}\;\forall\mathrm{L\text{-}M}
      }{Fun_g, f_p(x,u), g_p(x,v), u \doteq v, g_p(a,b) \vdash}\;{=}\mathrm{RED}
    }{Fun_g, \ldots, f_p(x,u), \exists z.(g_p(x,z) \land u \doteq z), g_p(a,b) \vdash}\;\exists\mathrm{L}, \land\mathrm{L}
  }{Fun_g, Tot_f, f_p(x,u), g_p(a,b), (8) \vdash}\;\forall\mathrm{L\text{-}M}
}{Fun_g, Tot_f, g_p(a,b), (8) \vdash}\;\forall\mathrm{L}, \exists\mathrm{L}
$$

This illustrates that PUHR/e-matching-based reasoning (through ∀L-M and ∃R-M) can be mixed freely with free variable reasoning (through ∀L and ∃R). Proofs constructed without applying the rules ∀L and ∃R closely correspond to the ground reasoning in an SMT solver, while each application of ∀L or ∃R conceptually introduces a free variable that, at a later point during proof construction, can be unified with other terms, extracting unification conditions in the form of constraints.

5 Extension to Linear Integer Arithmetic

All techniques discussed so far carry over to first-order logic modulo the theory of linear integer arithmetic (FOL(LIA)), via integration into the calculus defined in [23]. The syntax of FOL(LIA) is defined by the grammar in the beginning of Sect. 2.1 and combines first-order logic (with uninterpreted predicates and functions) with arithmetic terms and predicates. Semantics is defined over structures $(\mathbb{Z}, I)$ with the set of integers as universe.

$$\frac{\Gamma, t \doteq 0 \;\vdash\; \phi[s + \alpha \cdot t], \Delta \;\Downarrow C}{\Gamma, t \doteq 0 \;\vdash\; \phi[s], \Delta \;\Downarrow C}\ {=}\textsc{red-}\mathbb{Z} \qquad \frac{\Gamma, s \leq 0, t \leq 0, \alpha s + \beta t \leq 0 \;\vdash\; \Delta \;\Downarrow C}{\Gamma, s \leq 0, t \leq 0 \;\vdash\; \Delta \;\Downarrow C}\ {\leq}\textsc{l-}\mathbb{Z}$$

$$\frac{\Gamma, p(s_1, \ldots, s_n) \;\vdash\; p(t_1, \ldots, t_n), \bigwedge_i s_i - t_i \doteq 0, \Delta \;\Downarrow C}{\Gamma, p(s_1, \ldots, s_n) \;\vdash\; p(t_1, \ldots, t_n), \Delta \;\Downarrow C}\ \textsc{pu-}\mathbb{Z}$$

$$\frac{*}{\Gamma, \phi_1, \ldots, \phi_n \;\vdash\; \psi_1, \ldots, \psi_m, \Delta \;\Downarrow \neg\phi_1 \vee \cdots \vee \psi_1 \vee \cdots}\ \textsc{close}$$

Fig. 2. A selection of rules of the calculus PresPred^C; for a complete list see [23]. In =RED-$\mathbb{Z}$, α is a literal; we write $\phi[s]$ in the succedent to denote that s occurs in an arbitrary formula in the sequent, which can in particular also be in the antecedent. In $\leq$L-$\mathbb{Z}$, $\alpha, \beta > 0$ are positive literals. In CLOSE-$\mathbb{Z}$, the formulae $\phi_1, \ldots, \phi_n, \psi_1, \ldots, \psi_m$ do not contain uninterpreted predicates.

As for FOL, we first introduce a calculus for the function-free fragment of FOL(LIA). The integration of functions is then done in the same way as in Sect. 3, 4 with the help of a relational encoding. The calculus PresPred^C for the function-free fragment consists of the rules in Fig. 1, together with a number of rules specific for linear integer arithmetic, a selection of which are shown in Fig. 2 (as a result, the rules =L, =R, =RED, and PC of the first-order calculus can be removed); in the full calculus, also simplification and splitting rules are needed [23]. A more general closure rule CLOSE has to be used than in PredEq^C to support disjunctive constraints. Constraints in PresPred^C are always formulae in Presburger arithmetic (PA), i.e., do not contain uninterpreted predicates.

Lemma 10 (Soundness [23]). *If a sequent $\Gamma \vdash \Delta \Downarrow C$ can be proven in PresPred^C, then it is valid.*

The logic FOL(LIA) subsumes Presburger arithmetic. Since the logic of quantified Presburger arithmetic with predicates is Π^1_1-complete [10], no complete calculi can exist for FOL(LIA); however, it can be shown that the calculi introduced in this section are complete for relevant and non-trivial fragments:

Lemma 11 (Completeness [23]). *Suppose ϕ is a closed formula without functions or constants in one of the following fragments:*

(i) ϕ does not contain uninterpreted predicates (i.e., in Presburger arithmetic);
(ii) ϕ contains universal (exist.) quantifiers only in positive (negative) positions;
(iii) ϕ contains universal (exist.) quantifiers only in negative (positive) positions;
(iv) ϕ is of the form $\forall \bar{x}.(\sigma \to \psi)$, where σ is a formula in Presburger arithmetic (without uninterpreted predicates) that has only finitely many solutions in $\bar{x}$, and ψ contains universal (existential) quantifiers only in negative (positive) positions (i.e., a formula accepted by the $\mathcal{ME}$(LIA) calculus [4]).

Then there is a valid constraint C such that $\vdash \phi \Downarrow C$ is provable in PresPred^C.

Practically, it can be observed that PresPred^C can often also be applied successfully to formulae outside of those fragments.

5.1 Hyper-Resolution and E-Matching for FOL(LIA)

The unit hyper-resolution rule ∀L-M (and similarly the rule ∃R-M) defined in Sect. 3 can be integrated in the calculus PresPred^C in the same way as in the earlier first-order calculi, in order to instantiate formulae $\forall \bar{x}.\, (\bigvee_{i=1}^{n} \neg p_i(\bar{t_i}) \vee \phi)$ by matching. In this context, the simplification function *simp* can be (but does not have to be) replaced with a function tailored to integer arithmetic, i.e., a function that is able to solve the system $\bigvee_{i=1}^{n} \bar{s}_i \not\doteq \bar{t}_i$ modulo integer arithmetic.

The calculus PresPredHR^C is derived from PresPred^C by adding the rules ∀L-M and ∃R-M, and by imposing the side condition that the rule ∀L is no longer applied to formulae of the shape $\forall \bar{x}.\, (\bigvee_{i=1}^{n} \neg p_i(\bar{t_i}) \vee \phi)$; similarly for the rule ∃R. As before, the soundness of the rules ∀L-M and ∃R-M is immediate. We can also observe that PresPredHR^C is relatively complete, in the sense that formulae that are provable in PresPred^C can also be proven using PresPredHR^C:

Lemma 12. *Suppose $\Gamma \vdash \Delta \Downarrow C$ is provable in PresPred^C, where C is valid. Then there is a valid constraint C' so that PresPredHR^C can prove $\Gamma \vdash \Delta \Downarrow C'$.*

Encoding of functions. The relational encoding of functions from Sect. 4 can be used to obtain a calculus for the full logic FOL(LIA) with functions. Although there are no complete calculi for the full logic, we can observe that PresPred^C (and therefore, by Lem. 12, PresPredHR^C) can handle at least all formulae that can be proven by considering a finite set of ground instances:

Lemma 13. *Suppose $\exists \bar{x}.\phi$ is a closed formula in FOL(LIA), with functions taken from a finite set F, such that ϕ is quantifier-free. If there is a valid disjunction $\bigvee_{i=1}^{n} [\bar{x}/\bar{t}_i]\phi$ of ground instances of $\exists \bar{x}.\phi$, then there is a valid constraint C such that $\{Fun_f,\, Tot_f\}_{f \in F} \vdash (\exists \bar{x}.\phi)_{Rel} \Downarrow C$ is provable in PresPred^C.*

The lemma directly generalises to disjunctions of existentially quantified formulae, which in particular entails that PresPred^C is complete for the class of *essentially uninterpreted formulae* F (modulo linear integer arithmetic) with finite ground instantiation $F*$ defined in [9], and thus also for the array property fragment [6] (PresPred^C cannot easily be turned into a decision procedure, however, since it would be unclear how to ensure termination on invalid problems).

6 Experiments and Related Work

We have implemented the described calculus PresPredHR^C for FOL(LIA) in the theorem prover Princess,[2] and are in the process of adding further optimisations. Princess uses the relational encoding from Sect. 4 to represent functions, and heuristics similar to the ones in Simplify [7] to automatically identify triggers in quantified formulae; redundancy criteria [22] and theory propagation help to reduce the number of instances generated from quantified formulae. Princess is able to handle all of the examples discussed in Sect. 1.1.

[2] `http://www.philipp.ruemmer.org/princess.shtml`

	AUFLIA+p (193)	**AUFLIA-p** (193)
Z3	191	191
Princess	**145**	**137**
CVC3	132	128

Fig. 3. Number of solved benchmarks, out of 2 × 193 unsatisfiable (scrambled) AUFLIA benchmarks selected in the SMT competition 2011. Experiments with Princess were done on an Intel Core i5 2-core machine with 3.2GHz, with a timeout of 1200s, heap-space limited to 4Gb. The benchmarks in AUFLIA+p contain hand-written triggers for most of the quantified formulae, while all triggers have been removed in AUFLIA-p. The corresponding figures for Z3 and CVC3 are the results obtained during the SMT competition 2011 (`http://www.smtexec.org/exec/?jobs=856`).

To evaluate the overhead of the relational function encoding, we compared the performance of Princess with the SMT solvers CVC3 [3] and Z3 [19], using benchmarks selected in the SMT competition 2011. Since our work concentrates on proof construction, we only considered unsatisfiable benchmarks, removing 13 satisfiable AUFLIA problems in each category. The results show that Princess, while currently not being able to compete with the fastest SMT solver Z3, performs better than the (state-of-the-art) e-matching-based CVC3. This is promising, since Princess does not (yet) use SMT techniques like lemma learning, which are important for large or propositionally complex problems. Princess can solve most benchmarks using e-matching alone, but uses free variables in 17 of the (solved) benchmarks, typically in smaller (but harder) instances.

Related Work E-matching is today used in most SMT solvers, based on techniques that go back to the Simplify prover [7] and The Stanford Pascal Verifier [20]; since then, various refinements of the e-matching approach have been published, for instance [8, 18]. To the best of our knowledge, e-matching has not previously been combined with free variable methods. An instantiation method similar to e-matching, but with much stronger completeness results, has been published in [9] and is used in Z3; a comparison with our method is in Sect. 5.1.

There is a large body of work on integrating theories into resolution and superposition calculi (e.g., [25, 2, 13, 1]), as well as on the integration of resolution into SMT [17]. These approaches completely avoid e-matching, offering stronger completeness guarantees but limiting the possibility of user-provided guidance.

The model evolution calculus has been extended to theories, including integer arithmetic [4, 5]. Our approach resembles model evolution in that it also uses free variables in a tableaux setting, albeit in a more “rigid”/global manner. Further differences are that $\mathcal{ME}$(LIA) works on clauses, only supports a restricted form of existential quantification, and has a more explicit representation of models.

References

1. Althaus, E., Kruglov, E., Weidenbach, C.: Superposition modulo linear arithmetic sup(la). In: FroCos. LNCS, vol. 5749, pp. 84–99. Springer (2009)

2. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. Appl. Algebra Eng. Commun. Comput. 5 (1994)
3. Barrett, C., Tinelli, C.: CVC3. In: Proceedings, CAV. LNCS, vol. 4590, pp. 298–302. Springer (Jul 2007)
4. Baumgartner, P., Fuchs, A., Tinelli, C.: ME(LIA) – Model Evolution With Linear Integer Arithmetic Constraints. In: LPAR. LNCS, vol. 5330. Springer (2008)
5. Baumgartner, P., Tinelli, C.: Model evolution with equality modulo built-in theories. In: CADE. LNCS, vol. 6803, pp. 85–100. Springer (2011)
6. Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: VMCAI. LNCS, vol. 3855, pp. 427–442. Springer (2006)
7. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: A theorem prover for program checking. Journal of the ACM 52(3) (2005)
8. Ge, Y., Barrett, C., Tinelli, C.: Solving quantified verification conditions using satisfiability modulo theories. In: CADE. LNCS, vol. 4603. Springer (2007)
9. Ge, Y., de Moura, L.M.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: CAV. pp. 306–320 (2009)
10. Halpern, J.Y.: Presburger arithmetic with unary predicates is Π^1_1 complete. Journal of Symbolic Logic 56 (1991)
11. Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press (2009)
12. Klebanov, V., Müller, P., Shankar, N., Leavens, G.T., Wüstholz, V., Alkassar, E., Arthan, R., Bronish, D., Chapman, R., Cohen, E., Hillebrand, M., Jacobs, B., Leino, K.R.M., Monahan, R., Piessens, F., Polikarpova, N., Ridge, T., Smans, J., Tobies, S., Tuerk, T., Ulbrich, M., Weiß, B.: The 1st Verified Software Competition: Extended experience report (2011)
13. Korovin, K., Voronkov, A.: Integrating linear arithmetic into superposition calculus. In: CSL. LNCS, vol. 4646, pp. 223–237. Springer (2007)
14. Manthey, R., Bry, F.: A hyperresolution-based proof procedure and its implementation in Prolog. In: GWAI. pp. 221–230. Springer (1987)
15. Manthey, R., Bry, F.: SATCHMO: A theorem prover implemented in Prolog. In: Proceedings, CADE. pp. 415–434. LNCS, Springer (1988)
16. McCarthy, J.: Towards a mathematical science of computation. In: Popplewell, C.M. (ed.) Information Processing 1962. pp. 21–28. North Holland (1963)
17. de Moura, L., Bjørner, N.: Engineering DPLL(T) + saturation. In: Proceedings, IJCAR. LNCS, vol. 5195. Springer (2008)
18. de Moura, L.M., Bjørner, N.: Efficient e-matching for SMT solvers. In: Proceedings, CADE. pp. 183–198. LNCS, Springer (2007)
19. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
20. Nelson, G.: Techniques for program verification. Tech. Rep. CSL-81-10, Xerox Palo Alto Research Center (1981)
21. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM 27, 356–364 (April 1980)
22. Rümmer, P.: Calculi for Program Incorrectness and Arithmetic. Ph.D. thesis, University of Gothenburg (2008)
23. Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. In: LPAR. LNCS, Springer (2008)
24. Rümmer, P.: E-matching with free variables. Tech. rep. (2012), to appear
25. Stickel, M.E.: Automated deduction by theory resolution. Journal of Automated Reasoning 1(4), 333–355 (1985)