



Enhancing Superposition Reasoning in Linear Real Arithmetic through Selection and Simplification

Johannes Schoisswohl¹, Laura Kovács¹, Konstantin Korovin², and Andrei Voronkov^{2,3}

¹ TU Wien

² University of Manchester

³ EasyChair

Abstract

The calculus ALASCA extends superposition-based saturation proving with reasoning in Uninterpreted Functions and Linear Real Arithmetic (UFLRA). As efficiency of saturation algorithms depend on literal selection, simplification, and redundancy elimination, this paper improves reasoning in UFLRA by introducing theory-specific notions and methods for all of these techniques. The implementation of these methods in the first-order theorem prover VAMPIRE shows substantial gains in every tested benchmark category, solving a number of problems previously unsolved by any system.

1 Introduction

The increasing complexity of modern computing systems requires automated approaches to ensure functional correctness, reliability and security of such systems [32, 34]. Most of existing formal methods supporting such automation make use of automated reasoning engines under the hood. A powerful class of automated reasoners are ATPs, which are used to check validity of logical formulas in extensions of first-order logical theories. Despite significant advances [4, 10, 26], reasoning about the combination of quantifiers ($\forall, \exists$), arithmetic and uninterpreted functions still poses challenges. This paper addresses this problem and introduces techniques that vastly improve efficiency of automated reasoning for Uninterpreted Functions and Linear Real Arithmetic (UFLRA). Key to our work is saturation-based first-order theorem proving using the Abstracting Linear Arithmetic Superposition Calculus (ALASCA) [26], an extension of the superposition calculus [35, 2, 44] used in modern ATPs for first-order logic [6, 45, 41, 16, 12].

ALASCA extends the superposition calculus by adding dedicated inference rules for arithmetic. These rules replace the need of axiomatizing theory symbols, such as addition $+$ and order $>$, as done by early saturation-based approaches to arithmetic reasoning [38]. The calculus was shown to outcompete state-of-the-art SMT-solvers and first-order theorem provers [26]. Let us explain the main challenges one faces when applying ALASCA, and the saturation algorithm implementing it, to UFLRA. Solving these challenges is the main motivation for our work.

Saturation & ALASCA. Given an input problem F , saturation theorem provers first negate F and clausify the negated input $\neg F$, obtaining a set of clauses; this set of clauses forms the initial search space. After that, one repeatedly applies a set of inference rules to clauses in the search space, adding conclusions of these inferences to the search space. The goal of the proving process is to derive the empty clause, denoted as $\square$ which corresponds to a contradiction.

Saturation algorithms may build very large search spaces; in practice, often containing millions of clauses. Thus, methods for making the search space smaller, can vastly improve performance. For handling this issue a number of concepts have been developed for superposition. Important for our work are *literal selection*, *redundancy* and *simplification*. Literal selection restricts the number of applicable inferences, and thus limits growth of the search space, redundancy allows for removing unnecessary clauses from the search space, and using simplification one can replace formulas by simpler ones instead of adding new clauses to the search space.

Even though these techniques have been well-studied in the context of superposition [2, 35, 31, 22, 20, 44], they cannot be straightforwardly used in ALASCA. ALASCA uses, among other rules, extensive normalizations, in particular rewriting arithmetic literals to a normal form $t \diamond 0$, where t is an arithmetic term and $\diamond \in \{\approx, \neq, >, \geq\}$. These normal forms are required for ALASCA to work in a sound and complete manner, but they interfere with (i) literal selection, (ii) equality reasoning and rewriting, and (iii) the detection of tautological clauses. Steps (i)-(iii) are however indispensable to keep proof search tractable. Our paper proposes new solutions for (i)-(iii), as detailed next.

Enhancing ALASCA. *Literal Selection* guides proof search, by selecting a subset of the literals of a clause that needs to be considered for making inferences with. In superposition, given a clause $C = L_1 \vee \dots \vee L_n$, a *selection function* sel determines a subset $\text{sel}(C)$ of the *negative* literals of C which needs to be considered for inferences. Inference with literals not in $\text{sel}(C)$ can be skipped, thus limiting the growth of the search space without compromising the solver's completeness [2]. Due to normalizations like $\neg(t > 0) \Rightarrow -t \geq 0$ negative inequality literals do not exist in ALASCA, which prevents the use of standard literal selection. We address this by extending literal selection to a wider class of so-called *selectable* literals (Sect. 5). A peek at our illustrating example (Ex. 1, Sect. 2) shows how this can tremendously improve proof search, by yielding an exponentially smaller search space.

Demodulation is one of the most widely used simplification rules in superposition [31]. Given an equality clause $l \approx r$ where r is simpler than l ($r \prec l$), it allows to *replace* any clause $C[l]$ with $C[r]$, discarding $C[l]$ from the search space without considering it for any further inferences. In the presence of arithmetic however, demodulation is less effective because it requires the term l in $C[l]$ to coincide with l in $l \approx r$. This, for example, would prevent $p(a + b)$ to be rewritten by $b + a \approx c$ into $p(c)$, despite the fact that $a + b$ and $b + a$ are semantically equal. Even worse, real-valued equalities are normalized to the form $t \approx 0$ in ALASCA, for which applying demodulation mostly fails. We address these shortcomings by formulating *ALASCA Demodulation* (Dmd), a *new variant of demodulation* that works modulo linear arithmetic. Ex. 1 shows how Dmd improves proof search by achieving an exponential reduction of the search space size.

Inequality Merging is a new inference rule that removes arithmetical tautologies and simplifies clauses. In the pure propositional case, tautologies (e.g. $L \vee \neg L \vee C$) can be detected and deleted easily. When dealing with arithmetic, we might face more complex tautologies like $a \geq 0 \vee -a \neq 0$, which are harder to discover, and might pollute our search space with unnecessary clauses. ALASCA's normalizations introduce such tautologies, for example by turning $t > 0 \vee \neg(t > 0)$ into $t > 0 \vee -t \geq 0$. To address such and similar issues, we introduce the *new inference rule Inequality Merging* (IM), which generalizes the idea of detecting tautologies and allows to simultaneously perform simplifications like $t > 0 \vee -t \neq 0 \Rightarrow -t \neq 0$. Ex. 1 shows how IM can

drastically shrink proof length.

Our contributions are summarized as follows:

- We show how *Literal Selection* can be used in ALASCA by extending selection to so-called *selectable*, and not just negative, literals (Sect. 5). This results in fewer inferences and hence reduces the size of the search space.
- We introduce *ALASCA Demodulation (Dmd)*, which allows one to rewrite clauses by equalities modulo Linear Real Arithmetic (LRA) (Sect. 4). This drastically improves proof search by eliminating many inferences that would add new clauses to the search space.
- We introduce a new inference rule, called *Inequality Merging (IM)*, for simplifying clauses with multiple inequalities containing the same term and removing tautological clauses (Sect. 4). This speeds up proof search by removing tautologies and by avoiding repetitive work.
- We introduce $\text{ALASCA}_{\prec}^{\text{sel}}$, an enhanced variant of ALASCA, which integrates all of these techniques (Sect. 3.2).
- We implemented our work in the theorem prover VAMPIRE [31] and evaluated it against existing methods for automating reasoning in UFLRA (Sect. 6). Our experiments show the practical benefits of $\text{ALASCA}_{\prec}^{\text{sel}}$, solving 84 more problems than state-of-the-art systems in UFLRA reasoning.

Outline. This paper is structured as follows. Sect. 2 gives an example that illustrates the main new techniques introduced in this paper. Sect. 3 introduces a variant of the original ALASCA [26] which integrates with the methods introduced in the following sections. Sect. 4 introduces our new simplification and redundancy elimination rules, while Sect. 5 shows how literal selection can be done in ALASCA. The implementation of our work is evaluated in Sect. 6, showing that, our work improves automated reasoning in UFLRA. We show that, with the new extensions, ALASCA-based reasoning solves problems that could not be solved before by other state-of-the-art provers. In Sect. 7 the paper is positioned within relevant related work in automated reasoning in UFLRA, and finally Sect. 8 concludes the paper.

2 Illustrative Example

We showcase our reasoning enhancements upon ALASCA via Ex. 1.

Example 1 (Illustrative Example). *Consider the set of clauses $C = \{C_1, C_2, C_3\}$*

$$\forall x, y. g(x, y) + x - y \approx 0 \quad C_1$$

$$\forall x_1, \dots, x_n, y. \bigvee_{i=1}^n -f(x_i, a_i) + g(y, x_i) > 0 \vee y > 0 \quad C_2$$

$$\forall x, y. -f(x, y) + y \approx 0 \vee f(x, y) - y > 0 \vee h(x) - h(y) \not\approx 0 \quad C_3$$

where a_i are constants, and f, g, h are uninterpreted functions over $\mathbb{R}$.

Our enhanced ALASCA easily proves unsatisfiability of C for instances up to $n = 50$ within 10 seconds, generating only 208 clauses. Yet, ALASCA without our extensions times out for $n = 3$ after 10 seconds, while generating 49508 intermediate clauses. Fig. 1 shows the unsatisfiability proof of C , using our enhanced ALASCA approach. In the sequel, we detail the key steps of this proof and explain why disabling either of our new enhancements causes ALASCA to fail (dis)proving Ex. 1 for large n .

C_1 :	$g(x, y) + x - y \approx 0$	input
C_2 :	$\bigvee_{i=1}^n -f(x_i, a_i) + g(y, x_i) > 0 \vee y > 0$	input
C_3 :	$-f(x, y) + y \approx 0 \vee f(x, y) - y > 0 \vee h(x) - h(y) \not\approx 0$	input
C_4^1 :	$-f(x_1, a_1) + x_1 - y > 0 \vee \bigvee_{i=2}^n -f(x_i, a_i) + g(y, x_i) > 0 \vee y > 0$	Dmd using C_1, C_2
C_4^k :	$\bigvee_{i=1}^k -f(x_i, a_i) + x_i - y > 0 \vee \bigvee_{i=k+1}^n -f(x_i, a_i) + g(y, x_i) > 0 \vee y > 0$	Dmd using C_1, C_4^{k-1}
C_4^n :	$\bigvee_{i=1}^n -f(x_i, a_i) + x_i - y > 0 \vee y > 0$	Dmd using C_1, C_4^{n-1}
C_5 :	$\bigvee_{i=1}^n -f(x_i, a_i) + x_i > 0$	VE using C_4^n
C_6 :	$f(x, y) - y \geq 0 \vee h(x) - h(y) \not\approx 0$	IM using C_3
C_7 :	$f(x, x) - x \geq 0$	TF using C_6
C_8^1 :	$\bigvee_{i=2}^n -f(x_i, a_i) + x_i > 0$	FM using C_5, C_7
C_8^k :	$\bigvee_{i=k+1}^n -f(x_i, a_i) + x_i > 0$	FM using C_8^{k-1}, C_7
C_8^n :	$\square$	FM using C_8^{n-1}, C_7

Figure 1: Proof of unsatisfiability of C from Ex. 1. Inference rules indicated on the side can be found in Figs. 4 and 5. Normalizations using inference rules $\text{NF} \cup \text{Ctd}$ are omitted.

Enhanced ALASCA proof. In the first steps of the proof of Fig. 1, we use *our new rule ALASCA Demodulation* (Dmd) with the clause C_1 to rewrite $g(y, x_1) \Rightarrow x_1 - y$ in C_2 deriving C_4^1 , and then rewrite the result again by C_1 until we arrive at the clause C_4^n . Dmd is a simplification rule, which means whenever we use it to rewrite a clause, the original clause becomes redundant, that is, the original clause can be discarded from the search space. C_4^n contains so-called unshielded variable y ; a variable that does not occur under any uninterpreted function. This unshielded variable will be processed by the rule Variable Elimination (VE)¹, turning C_4^n into the equivalent clause C_5 by substituting 0 for y .

For the input clause C_3 , we use *our new rule Inequality Merging* (IM) to simplify C_3 into the equivalent clause C_6 , and then use the rule Term Factoring (TF) to derive clause C_7 .

All literals $-f(x_i, a_i) + x_i > 0$ of C_5 are selectable² (see Sect. 5). This means we can select an arbitrary single literal from C_5 to apply inferences with, and do not need to make inferences with any other literal of C_5 . We apply *Literal Selection* choosing the first literal ($\text{sel}(C_5) = \{-f(x_1, a_1) + x_1 > 0\}$) and use the rule Fourier-Motzkin (FM) to resolve C_5 and C_7 deriving C_8^1 . Of the result we select the first literal again and resolve it with C_7 , until we finally arrive at C_8^n which is the empty clause $\square$. This step represents a contradiction and concludes our proof of unsatisfiability of C . We note that choosing different literals for selection would lead to deriving $\square$ in essentially the same way.

Need for ALASCA enhancements. Fig. 2 shows the results of running our enhanced ALASCA on Ex. 1. Our evaluation is conducted using our implementation in the VAMPIRE prover (see Sect. 6). Let us now explain why all three of our new reasoning techniques are necessary to solve Ex. 1 in reasonable time.

ALASCA Demodulation (Dmd). If Dmd is disabled (conf. $\{\mathbf{s}, \mathbf{i}\}$), the rewrites obtained using C_1 in order to obtain C_4^k are performed using the standard superposition (Sup), instead of Dmd. Sup is however a generating, not simplifying, inference rule: it generates and adds clauses to the search spaces, without checking whether premises become redundant (and could

¹ Inference rules can be found in Fig. 4 (Page 174) and Fig. 5 (Page 178).

² as their only $\prec$ -maximal term $f(x_i, a_i)$ occurs with a negative sign

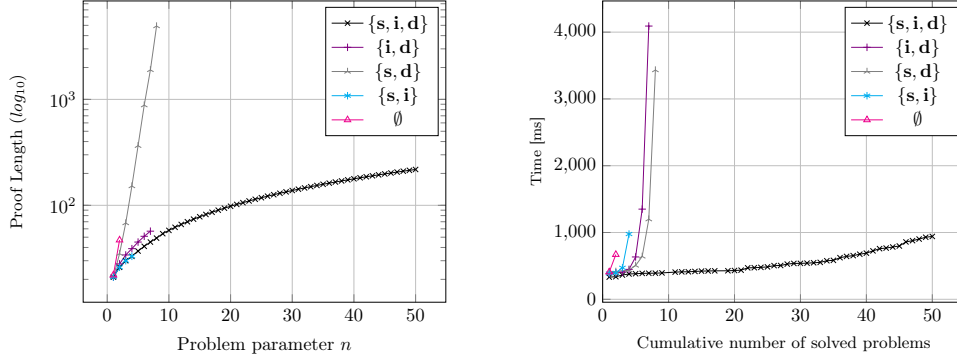


Figure 2: Results of running our enhanced ALASCA implementation (see Sect. 6) on Ex. 1 up to $n = 50$, with a timeout of 10 seconds. Configurations are denoted by the set of enabled reasoning features, where **s** means Literal Selection; **d** means ALASCA Demodulation (Dmd), and **i** means Inequality Merging (IM).

be removed). This means, we also obtain the clause C_4^2 , but derive exponentially many unnecessary consequences on the way: for arbitrary n we will derive $\bigvee_{i \in I} -f(x_i, a_i) + g(y, x_i) > 0 \vee \bigvee_{j \in \bar{I}} -f(x_j, a_j) + g(y, x_j) > 0 \vee y > 0$ for any $I \subseteq \{1 \dots n\}$. Fig. 2 shows that proof lengths with this configuration are the same as for $\{s, i, d\}$ since the same proofs can be found, but the solver times out for $n = 5$.

Inequality Merging (IM). If we disable IM (conf. $\{s, d\}$), the proof of Fig. 1 can be performed only up to step C_5 . However, we cannot use IM on C_3 to further derive C_6 , from which we would infer C_7 . Rather, we directly obtain the clause $D_7 : f(x, x) - x > 0 \vee -f(x, x) + x \approx 0$ from C_3 . Progressing as in Fig. 1, we finally get at $D_8^n : \bigvee_{i=1}^n -f(a_i, a_i) + a_i \approx 0$ instead of the empty clause C_8^n . Derivation of the empty clause from here comes at a massive cost in proof length, yielding a timeout after 10 seconds for $n = 9$ (see Fig. 2). Looking at the length of proofs (Fig. 2), we also observe an approximately exponential growth of the proof length in terms of n .

Literal Selection. Disabling literal selection (conf. $\{i, d\}$) implies that FM needs to be applied for every literal of C_5 instead of one single selected literal. This gives an exponential blow-up in search space as we can derive all clauses $\bigvee_{i \in I} -f(x_i, a_i) + x_i > 0$ for $I \subseteq \{1 \dots n\}$ from C_5, C_7 and their successors, which yields a timeout for $n = 8$ (Fig. 2).

In summary, we conclude that the combination of Literal Selection, ALASCA Demodulation, and Inequality Merging is essential for solving Ex. 1.

3 ALASCA Revised

In what follows, we use ALASCA_0 to refer to the calculus ALASCA from [26]. This section introduces $\text{ALASCA}_{\text{sel}}^{\text{el}}$, a variant of ALASCA_0 which allows integrating simplification and selection techniques we will introduce in the following sections. Changes to the old calculus were made in order to make the newly introduced simplification rules Inequality Merging (IM), and ALASCA

Demodulation (Dmd) compatible with $\text{ALASCA}_{\prec}^{\text{sel}}$ ³.

We introduce necessary background and fix notation (Sect. 3.1), before defining the calculus $\text{ALASCA}_{\prec}^{\text{sel}}$ and introducing conditions required for its parameters $\prec$, and sel (Sect. 3.2).

3.1 Notation and Background Knowledge

We consider a first-order signature of UFLRA, with a single sort⁴. This sort contains the function symbol $+$, the constant 1 , the unary predicate symbols (≈ 0) , (> 0) , (≥ 0) , and an infinite set of unary functions $\mathbb{Q}$ called the numeral multiplications, representing multiplications by a constant factor.

Clauses are multisets of literals $L_1 \vee \dots \vee L_n$. We call a clause unit if $n = 1$. The empty clause $n = 0$ is denoted by $\square$. By $\mathbf{F}$, $\mathbf{V}$, Trm , Lit , and Cls we denote the set of all first-order function symbols, variables, terms, literals, and clauses respectively. We use gTrm , gLit for the respective ground (variable-free) subsets.

We say f is *uninterpreted* if $f \notin \{+\} \cup \mathbb{Q}$ and a term t is *atomic* if it is a variable or its top-level symbol $\text{sym}(t)$ is uninterpreted. In the sequel, let $x \in \mathbf{V}$, $f \in \mathbf{F}$, $k \in \mathbb{Q}$, $\diamond \in \{>, \geq, \approx, \not\approx\}$, $s, t \in \text{Trm}$, $L \in \text{Lit}$, $C \in \text{Cls}$ and $\prec$ be a preorder.

For the sake of readability we sometimes abbreviate numeral terms, e.g. 2 for $2(1)$. Literals are considered to be normalized to $s \diamond 0$ implicitly. $\mathbf{AC}$ are the axioms of associativity and commutativity of $+$. We define the function $\circ^-$ which negates all the coefficients of a term (e.g., $(a + -2b - c)^- = -a + 2b + c$), and we use $\circ^+$ for the identity function ($t^+ = t$). By $\equiv_T$ we denote equivalence modulo T . We write $s \triangleleft_T t$ ($s \triangleleft_T t$) for s being a (strict) subterm of t modulo T .

We define $\langle \text{coeff}(kt), \text{trm}(kt) \rangle = \langle k, t \rangle$, and $\langle \text{coeff}(t), \text{trm}(t) \rangle = \langle 1, t \rangle$ if $\text{sym}(t) \notin \mathbb{Q}$. We call the multiset $\{t_i \mid i \in I\} = \text{smnds}(\sum_{i \in I} t_i)$ the *summands* of $\sum_i t_i$.

We call $\text{smnds}^{\prec}(t)$ the weakly maximal summands of t , defined as $\text{smnds}^{\prec}(t) = \{t_0 \in \text{smnds}(t) \mid \forall t_1 \in \text{smnds}(t) \setminus \{t_0\}. \text{trm}(t_0) \not\prec \text{trm}(t_1)\}$, and define the strictly maximal summands $\underline{\text{smnds}}^{\prec}$ (up to $\equiv_{\mathbf{AC}}$) analogously, but replacing $\not\prec$ by $\not\triangleleft$. We use double underlining to mark strictly maximal summands of a term ($k \underline{\text{smnds}}^{\prec} + t$), single underlining for weakly maximal summands ($k \underline{\text{smnds}} + t$), and dots for arbitrary summands ($k \underline{\text{smnds}} + t$). That is $k \underline{\text{smnds}}^{\prec} + t$ represents some term containing a strictly maximal summand s' with $\text{coeff}(s') = k$, and $\text{trm}(s') = t$. We drop the relation $\prec$ from the notation where there is no ambiguity. We define $\text{trms}(s) = \{\text{trm}(t) \mid t \in \text{smnds}(s)\}$ and define $\underline{\text{trms}}^{\prec}$, $\underline{\text{trms}}$ analogously.

We call a literal $t \diamond 0$ *stably positive* ($t \diamond 0 \in \text{Lit}^+$) iff it is an equality, or it is an inequality and $\forall t_i \in \text{smnds}(t). \text{coeff}(t_i) > 0 \ \& \ \text{trm}(t_i) \notin \mathbf{V}$. We call it *stably negative* ($t \diamond 0 \in \text{Lit}^-$) if it is a disequality, or it is an inequality and $\forall t_i \in \text{smnds}(t). \text{coeff}(t_i) < 0 \ \& \ \text{trm}(t_i) \notin \mathbf{V}$. We call it *undecided* ($t \diamond 0 \in \text{Lit}^?$) otherwise.

A selection function sel is a function that maps a clause to a subset of its literals. We define the *leading literals* $\text{leadLit}(C)$, and *leading terms* $\text{lead}_{\prec}^{\text{sel}}(C)$ of C to be $\text{leadLit}_{\prec}^{\text{sel}}(C) = \text{sel}(C)$ if $\text{sel}(C) \neq \emptyset$, and otherwise $\text{leadLit}_{\prec}^{\text{sel}}(C) = (\underline{\text{lit}}(C) \cap \text{Lit}^+) \cup (\underline{\text{lit}}(C) \cap (\text{Lit} \setminus \text{Lit}^+))$. Further $\text{lead}_{\prec}^{\text{sel}}(C) = \{\sum \text{smnds}(L) \mid L \in \text{leadLit}_{\prec}^{\text{sel}}(C)\}$, where $\underline{\text{lit}}(C)$ and $\underline{\text{lit}}(C)$ are the weakly and strictly maximal literals in C . We leave away $\prec$ and sel if there is no ambiguity.

We say x is a *top level variable* of any $k \underline{\text{smnds}}^{\prec} + t$. x is *shielded* in t iff there is a t' such that $x \triangleleft t' \triangleleft t$ where $\text{sym}(t')$ is uninterpreted, or unshielded otherwise.

³For a detailed comparison between ALASCA_0 and $\text{ALASCA}_{\prec}^{\text{sel}}$, we refer to Sect. A in the extended version of this paper [28].

⁴Our work can be easily extended to additional uninterpreted sorts.

By $s[\circ]$ we denote a term (literal, clause) with a single occurrence of a designated variable $\circ \notin \mathbf{V}$. By $s[t]$ we denote the term obtained from replacing t for $\circ$. We call $s[\circ]$ a *term (literal, clause) context*. We call a context $C[\circ]$ *uninterpreted*, iff $f(\dots, \circ, \dots) \trianglelefteq C[\circ] \implies f$ is an uninterpreted function.

We say $\prec$ is T -compatible iff $s \equiv_T s' \prec t' \equiv_T t \implies s \prec t$. A simplification ordering modulo T is a binary relation $\prec$ on Trm that is closed under ground substitutions and, for $\mathbf{gTrm}$, it is a T -compatible well-founded monotonic strict partial ordering that is total modulo T and fulfils the subterm property.

As shown in [30] there is no complete calculus of UFLRA wrt. the standard model $\mathbb{R}$. Therefore in [26] completeness of the original ALASCA is shown wrt. to a wider class of models which are defined by the axiomatisation $\mathcal{A}$ given in Fig. 3.

$$\begin{array}{ll}
\mathcal{A} = \mathcal{A}_{\text{eq}} \cup \mathcal{A}_{\text{ineq}} & \mathbf{AC} = \{x + y \approx y + x\} \\
\mathcal{A}_{\text{ineq}} = \{x < y \wedge y < z \rightarrow x < z\} & \cup \{x + (y + z) \approx (x + y) + z\} \\
& \cup \{x < y \rightarrow x + z < y + z\} \\
& \cup \{x < y \vee x > y \vee x \approx y\} \\
& \cup \{\neg(x < x)\} \\
& \cup \{x \leq y \leftrightarrow (x < y \vee x \approx y)\} \\
& \cup \{x > 0 \rightarrow kx > 0 \mid k \in \mathbb{Q}, k > 0\} \\
& \cup \{1 > 0\} \\
\mathcal{A}_{\text{eq}} = \mathbf{AC} & \cup \{jx + kx \approx (j + k)x \mid j, k \in \mathbb{Q}\} \\
& \cup \{j(k(x)) \approx (jk)x \mid j, k \in \mathbb{Q}\} \\
& \cup \{1(x) \approx x\} \\
& \cup \{k(x + y) \approx kx + ky \mid k \in \mathbb{Q}\} \\
& \cup \{x + 0 \approx x, 0x \approx 0\}
\end{array}$$

Figure 3: The axiom set $\mathcal{A}$ of the ALASCA calculus from [26]. All axioms are implicitly universally quantified. We note that this axiomatisation has both $\mathbb{R}$ and $\mathbb{Q}$ as models.

3.2 The Calculus $\text{ALASCA}_{\prec}^{\text{sel}}$

The base inference system of $\text{ALASCA}_{\prec}^{\text{sel}}$ is given in Fig. 4. It lists all its inferences except for the new simplification rules ALASCA Demodulation (Dmd) and Inequality Merging (IM), which are presented in Fig. 5 and discussed in Sect. 4.

We point out that inferences make use of an ordering $\prec$, Unification With Abstraction (UWA), and a selection function sel . The selection function will be discussed in Sect. 5; thus for this section it is sufficient to consider a trivial empty selection function $\forall C. \text{sel}(C) = \emptyset$.

UWA, as used in side conditions like $\langle \sigma, \mathcal{E} \rangle = \text{UWA}(s, t)$, is an alternative to Unification modulo a background theory – in our case modulo $\mathbf{AC}$ – which is defined in terms of an abstraction oracle ao [8]. It returns $\langle \sigma, \mathcal{E} \rangle$, a set of literals $\mathcal{E}$ and a substitution σ , such that $\neg \mathcal{E} \implies s\sigma \equiv_{\mathbf{AC}} t\sigma$, or it returns $\perp$ if s, t are not unifiable modulo $\mathbf{AC}$. An example could be applying FM to $-g(x, y + z) + c > 0$ and $g(a, b + c) > 0$, resulting in $\sigma = \{x \mapsto a\}$, $\mathcal{E} = \{y + z \not\approx b + c\}$, and thus in the conclusion $c > 0 \vee y + z \not\approx b + c$. We can read this as “if $y + z$ and $b + c$ are equal (unify), then $c > 0$ ”.⁵

Similarly to literal selection, orderings serve to guide proof search. They help limiting the number of applicable inferences, by restricting the literals and terms to use for inferences. For example, the rule FM only resolves maximal terms s (as defined by the syntax $k \underline{s} + t$). Thus,

⁵Sect. B in the extended version of this paper [28] gives the concrete ao we use in our implementation which fulfils the properties required for abstraction oracles from [8].

Fourier-Motzkin

$$\frac{C_1 \vee j \underline{s} + t_1 \gtrsim_1 0 \quad C_2 \vee -k \underline{s}' + t_2 \gtrsim_2 0}{(C_1 \vee C_2 \vee kt_1 + jt_2 > 0) \sigma \vee \mathcal{E}} \text{ (FM)}$$

where

- $\langle \sigma, \mathcal{E} \rangle = \text{UWA}(s, s')$
- $js + t_1 \gtrsim_1 0 \not\vdash_{\text{sel}}^\sigma C_1$
- $-ks' + t_2 \gtrsim_2 0 \not\vdash_{\text{sel}}^\sigma C_2$
- $\{>\} \subseteq \{\gtrsim_1, \gtrsim_2\} \subseteq \{>, \geq\}$
- $j, k > 0$

Superposition

$$\frac{C_1 \vee k \underline{s} + t \approx 0 \quad C_2 \vee L[s']}{(C_1 \vee C_2 \vee L[-\frac{1}{k}t]) \sigma \vee \mathcal{E}} \text{ (Sup)}$$

where

- $\langle \sigma, \mathcal{E} \rangle = \text{UWA}(s, s')$
- $k \underline{s} + t \approx 0 \not\vdash_{\text{sel}}^\sigma C_1$
- $(s') \sigma \triangleleft u \in \text{lead}((L[s'] \vee C_2) \sigma)$
- $(k \underline{s} + t \approx 0) \sigma \not\vdash L[s'] \sigma$
- s is atomic

Equality Factoring

$$\frac{C \vee j \underline{s} + t_1 \approx 0 \vee k \underline{s}' + t_2 \approx 0}{(C \vee kt_1 - jt_2 \approx 0 \vee js + t_1 \approx 0) \sigma \vee \mathcal{E}} \text{ (EF)}$$

where

- $\langle \sigma, \mathcal{E} \rangle = \text{UWA}(s, s')$
- $ks' + t_2 \approx 0 \not\vdash_{\text{sel}}^\sigma C \vee js + t_1 \approx 0$

Normalization

$$\frac{C \vee L[t_1]}{C \vee L[t_2]} \text{ (NF)}$$

where

- $t_1 \triangleleft u \in \text{lead}(L[t_1] \vee C)$
- $t_1 \approx t_2 \in \left\{ \begin{array}{l} 0u \approx 0, \quad 0 \cdot 1 + u \approx u, \\ j(ku) \approx (jk)u, \quad 1(u) \approx u, \\ k(u+v) \approx ku + kv, \\ j \cdot u + k \cdot u + v \approx (j+k)u + v, \end{array} \middle| \begin{array}{l} u, v \in \text{Trm} \\ j, k \in \mathbb{Q} \end{array} \right\}$

Abstraction

$$\frac{C[k \underline{x} + t]}{C[y] \vee kx + t \not\approx y} \text{ (Abs)}$$

where

- $x \in \mathbf{V}$
- y is a fresh variable
- $C[o]$ is an uninterpreted clause context

Tight Fourier-Motzkin

$$\frac{C_1 \vee j \underline{s} + t_1 \geq 0 \quad C_2 \vee -k \underline{s}' + t_2 \geq 0}{(C_1 \vee C_2 \vee kt_1 + jt_2 > 0 \vee -ks' + t_2 \approx 0) \sigma \vee \mathcal{E}} \text{ (TFM)}$$

where

- $\langle \sigma, \mathcal{E} \rangle = \text{UWA}(s, s')$
- $js + t_1 \geq 0 \not\vdash_{\text{sel}}^\sigma C_1$
- $-ks' + t_2 \geq 0 \not\vdash_{\text{sel}}^\sigma C_2$
- $j, k > 0$

Inequality Factoring

$$\frac{C \vee j \underline{s} + t_1 \gtrsim_1 0 \vee k \underline{s}' + t_2 \gtrsim_2 0}{(C \vee kt_1 - jt_2 \gtrsim_3 0 \vee k \underline{s}' + t_2 \gtrsim_2 0) \sigma \vee \mathcal{E}} \text{ (IF)}$$

where

- $\langle \sigma, \mathcal{E} \rangle = \text{UWA}(s, s')$
- $ks' + t_2 \gtrsim_2 0 \not\vdash_{\text{sel}}^\sigma C \vee js + t_1 \gtrsim_1 0$
- or $js + t_1 \gtrsim_1 0 \not\vdash_{\text{sel}}^\sigma C \vee ks' + t_2 \gtrsim_2 0$
- $\gtrsim_1, \gtrsim_2 \in \{>, \geq\}$
- $\gtrsim_3 = \begin{cases} \geq & \text{if } \gtrsim_1 = \geq, \text{ and } \gtrsim_2 = > \\ > & \text{else} \end{cases}$
- $j, k > 0$

Term Factoring

$$\frac{L[j \underline{s} + k \underline{s}' + t] \vee C}{(L[(j+k)s + t] \vee C) \sigma \vee \mathcal{E}} \text{ (TF)}$$

where

- $\langle \sigma, \mathcal{E} \rangle = \text{UWA}(s, s')$
- $(j \underline{s} + k \underline{s}' + t) \sigma \triangleleft u \in \text{lead}((L[j \underline{s} + k \underline{s}' + t] \vee C) \sigma)$

Variable Elimination

$$\frac{C \vee \bigvee_{i \in I} (k_{i \cdot \underline{x}} + t_i \diamond_i 0)}{\bigwedge_{j \in J} (C \vee C_j)} \text{ (VE)}$$

where

- $\forall i. \diamond_i \in \{\approx, \not\approx, >, \geq\}$
- $\{C_j \mid j \in J\} = \text{QE}_x(\bigvee_{i \in I} (k_{i \cdot \underline{x}} + t_i \diamond_i 0))$
- QE_x is the universal quantifier elimination of the variable x , for details of this rule we refer to [26]
- $x \in \mathbf{V}$ is unshielded in the premise

Contradiction

$$\frac{C \vee q \diamond 0}{C} \text{ (Ctd)}$$

where

- $q \in \mathbb{Q}$
- $\diamond \in \{>, \geq, \approx, \not\approx\}$
- $\mathbb{Q} \not\models q \diamond 0$

Figure 4: Inference rules of our revised calculus $\text{ALASCA}_{\gtrsim}^{\text{sel}}$ parametrized by an ALASCA-ordering $\prec$ (Def. 2) and an ALASCA-compatible selection function sel (Def. 6 in Sect. 5). Notation $L \diamond_{\text{sel}}^\sigma C$ stands for “Either $\forall L' \in C. L \sigma \diamond L' \sigma$ and $\emptyset = \text{sel}(L \sigma \vee C \sigma)$, or $L \sigma \in \text{sel}(L \sigma \vee C \sigma)$ ”. We consider arbitrary $C, C_i \in \text{Cls}, L \in \text{Lit}, j, k, q \in \mathbb{Q}, s, s', s_i \in \text{Trm} \setminus \mathbf{V}, t, t_i \in \text{Trm}, \diamond, \diamond_i \in \{>, \geq, \approx, \not\approx\}, \gtrsim, \gtrsim_i \in \{>, \geq\}$.

if the ordering asserts that $a \succ b$, then a FM inference can be performed with $a + b > 0$ and $-a > 0$, but not with $a + b > 0$ and $-b > 0$. We require the term ordering to fulfil Def. 1 and 2⁶, ensuring behaviour analogous to the completeness proof in [26], while making our new simplification rules compatible with the calculus (see Lem. 1). The concrete orderings used in our implementation are given in Sect. 3.3.

Definition 1 (ALASCA Term Ordering). *We say a relation $\prec$ is an ALASCA term ordering iff*

1. *it is a simplification ordering modulo $\mathbf{AC}$,*
2. *for any $k \underline{s} + t \in \mathbf{gTrm}$, where s is atomic, we have $s \succ -\frac{1}{k}t$, and*
3. *for any $s \approx t \in \mathcal{A}_{\text{eq}}^{\overline{\mathbf{AC}}}$, and ground substitution θ we have $s\theta \succ t\theta$, where $\mathcal{A}_{\text{eq}}^{\overline{\mathbf{AC}}} = \mathcal{A}_{\text{eq}} \setminus \mathbf{AC}$.*

Definition 2 (ALASCA Ordering). *We say $\prec$ is an ALASCA ordering iff for any $s, s_i, t, t_i \in \mathbf{gTrm}$*

1. *$\prec$ is an ALASCA term ordering*
2. *$s \prec t \implies s \diamond 0 \prec t \diamond 0$*
3. *$\prec$ is closed under ground substitutions, and for $\mathbf{gLit}$ it is an $\mathbf{AC}$ -compatible well-founded strict partial ordering that is total modulo $\mathbf{AC}$.*
4. *$s_1 \prec s_2 \implies j \underline{s_1} + t \diamond_1 0 \prec k \underline{s_2} + u \diamond_2 0$ where $\diamond_i \in \{>, \geq, \approx, \not\approx\}$*
5. *$\forall i. t_i \prec s \implies k_1 t_1 + \dots k_n t_n \diamond_1 0 \prec k \underline{s} + t \diamond_2 0$ where $\diamond_i \in \{>, \geq, \approx, \not\approx\}$*
6. *$k \underline{s} + t \approx 0 \prec j \underline{s} + u \diamond 0$ where $\diamond \in \{\not\approx, >, \geq\}$*
7. *$|k| \underline{s} + t \gtrsim_1 0 \prec -|j| \underline{s} + u \gtrsim_2 0$ where $\gtrsim_i \in \{>, \geq\}$*
8. *$k \underline{s} + t \not\approx 0 \prec j \underline{s} + u \diamond 0$ where $\diamond \in \{>, \geq\}$*
9. *$t \geq 0 \prec t > 0$*

3.3 The Skeleton Ordering

Our term ordering is inspired by QKBO from [26], an extension of the Knuth-Bendix Ordering (**KBO**) [25]. In contrast to QKBO, our new $\text{ALASCA}_{\prec}^{\text{sel}}$ ordering can not only be parametrized by a KBO, but by an *arbitrary* simplification ordering, (including e.g., LPO [24]).

We define our new ordering as a comparison function cmp that maps two terms to *less*, *greater*, *equivalent*, or *incomparable*. For combining the results of comparison functions, we introduce the operator $\oplus$ defined as ($\text{equivalent} \oplus y$) = y , and $(x \oplus y) = x$ if $x \neq \text{equivalent}$. For any comparison function cmp of a relation $\prec$, we introduce its multiset extension cmp^{mul} which corresponds to the comparison function of the multiset extension $\prec^{\text{mul}}$ wrt equality up to $\mathbf{AC}$ [35], and cmp^{lex} corresponding to the lexicographical extension respectively. We compare natural numbers $\text{cmp}_{\mathbb{N}}$ by the $<$ relation, and signs $\pm \in \{0, +, -\}$ by the comparison function cmp_{sgn} of the total ordering $0 \prec + \prec -$. Rationals j, k are compared by the comparison function

$$\text{cmp}_{\mathbb{Q}}(j, k) = \text{cmp}_{\text{sgn}}(\text{sgn}(j), \text{sgn}(k)) \oplus \text{cmp}_{\mathbb{N}}(\text{den}(j), \text{den}(k)) \oplus \text{cmp}_{\mathbb{N}}(\text{num}(j), \text{num}(k)),$$

where sgn is the sign, den is the denominator, and num is the numerator of a number.

⁶Proof in Sect. C in the extended version of this paper [28]

Our new $\text{ALASCA}_{\prec}^{\text{sel}}$ ordering (Def. 4 and 5) extends a simplification ordering $\prec$ for uninterpreted terms to terms, and literals containing interpreted symbols. In the first steps, the two terms s, t which should be compared are transformed into terms that do not contain interpreted symbols using the function skel . Then, $\text{skel}(s)$ and $\text{skel}(t)$ are compared using the simplification ordering, and only if that comparison returns equal, we take into account the full terms s, t with their interpreted content.

Definition 3 (Skeleton). *Let $t_i, t, s \in \text{Trm}, k \in \mathbb{Q}, \prec$ some preorder, and f an arbitrary uninterpreted function. We call $\text{skel}(t)$ the (uninterpreted) skeleton of t wrt $\prec$:*

$$\begin{aligned} \text{skel}(f(t_1 \dots t_n)) &= f(\text{skel}(t_1) \dots \text{skel}(t_n)) \\ \text{skel}(kt) &= \text{skel}(t) & \text{skel}(s + t) &= \begin{cases} \text{skel}(t) & \text{if } \text{skel}(s) \prec \text{skel}(t) \\ \text{skel}(s) & \text{if } \text{skel}(t) \prec \text{skel}(s) \\ \text{undefined} & \text{else} \end{cases} \\ f(\dots \text{undefined} \dots) &= \text{undefined} \end{aligned}$$

Definition 4 (Skeleton Term Ordering). *Let $\prec_0$ be some simplification ordering and s, t be terms. We define the skeleton extension $\prec_0^{\text{sk}}$ of $\prec_0$ for terms to be $s \prec_0^{\text{sk}} t$ iff $\text{cmp}_{\text{sk}}(s, t) = \text{less}$. Here:*

$$\text{cmp}_{\text{sk}}(s, t) = \begin{cases} \text{cmp}_{\prec_0}(\text{skel}(s), \text{skel}(t)) \oplus \text{cmp}_{\text{seq}}(s, t) & \text{if } \text{skel}(s) \neq \text{undefined} \neq \text{skel}(t) \\ \text{incomparable} & \text{else} \end{cases}$$

where

$$\begin{aligned} \text{cmp}_{\text{seq}}(s, s) &= \text{equivalent} & (\text{trm}_1) \\ \text{cmp}_{\text{seq}}(s, t) &= \text{incomparable} & \text{if } \{v \in \text{smnds}(s) \mid v \in \mathbf{V}\} \neq \{v \in \text{smnds}(t) \mid v \in \mathbf{V}\} \quad (\text{trm}_2) \\ \text{cmp}_{\text{seq}}(1, t) &= \text{less} & \text{else } (\text{trm}_3) \\ \text{cmp}_{\text{seq}}(t, 1) &= \text{greater} & \text{else } (\text{trm}_4) \\ \text{cmp}_{\text{seq}}(\underbrace{s_1 + s_2}_s, \underbrace{t_1 + t_2}_t) &= \text{cmp}_{\text{seq}}(s, \underbrace{t_1 + t_2}_t) = \text{cmp}_{\text{sk}}^{\text{mul}}(\text{trms}(s), \text{trms}(t)) \oplus \text{cmp}_{\text{sk}}^{\text{mul}}(\text{smnds}(s), \text{smnds}(t)) & \text{else } (\text{trm}_5) \\ \text{cmp}_{\text{seq}}(js, kt) &= \text{cmp}_{\text{seq}}(s, t) \oplus \text{cmp}_{\mathbb{Q}}(j, k) & \text{else } (\text{trm}_6) \\ \text{cmp}_{\text{seq}}(js, t) &= \text{cmp}_{\text{seq}}(s, t) \oplus \text{greater} & \text{else } (\text{trm}_7) \\ \text{cmp}_{\text{seq}}(s, kt) &= \text{cmp}_{\text{seq}}(s, t) \oplus \text{less} & \text{else } (\text{trm}_8) \\ \text{cmp}_{\text{seq}}(f(s_1 \dots s_n), f(t_1 \dots t_n)) &= \text{cmp}_{\text{seq}}(s_1, t_1) \oplus \dots \oplus \text{cmp}_{\text{seq}}(s_n, t_n) & \text{else } (\text{trm}_9) \end{aligned}$$

Definition 5 (Literal Extension). *Let $\prec_0$ be an ALASCA term ordering, and $L, M \in \text{Lit}$. We define the literal extension $\prec_0^{\text{lit}}$ of $\prec_0$ for literals to be $L \prec_0^{\text{lit}} M$ iff $\text{cmp}_{\text{lit}}(L, M) = \text{less}$:*

$$\begin{aligned} \text{cmp}_{\text{lit}}(s \diamond_s 0, t \diamond_t 0) &= \text{incomparable} & \text{if } \text{smnds}(s) \cap \mathbf{V} \neq \text{smnds}(t) \cap \mathbf{V} & (\text{lit}_0) \\ \text{cmp}_{\text{lit}}(s \diamond_s 0, t \diamond_t 0) &= \text{cmp}_{\text{at}}^{\text{mul}}(\text{atms}_{\text{lit}}(s \diamond_s 0), \text{atms}_{\text{lit}}(t \diamond_t 0)) & (\text{lit}_1) \\ &\oplus \text{cmp}_{\prec_0}(s, t) & (\text{lit}_2) \\ \text{atms}_{\text{lit}}(l + r \diamond 0) &= \text{atms}_{\text{lit}}(l \diamond 0) \cup \text{atms}_{\text{lit}}(r \diamond 0) \\ \text{atms}_{\text{lit}}(0t \diamond 0) &= \emptyset \\ \text{atms}_{\text{lit}}(kt \sim 0) &= \{\langle t, (\sim) \rangle\} & \text{if } \sim \in \{\approx, \neq\} \\ \text{atms}_{\text{lit}}(kt \gtrsim 0) &= \{\langle t, (\gtrsim) \rangle\} & \text{if } k > 0 \ \& \gtrsim \in \{>, \geq\} \\ \text{atms}_{\text{lit}}(kt \gtrsim 0) &= \{\langle t, (\lesssim) \rangle\} & \text{if } k < 0 \ \& \gtrsim \in \{>, \geq\} \\ \text{atms}_{\text{lit}}(t \diamond 0) &= \{\langle t, (\diamond) \rangle\} \end{aligned}$$

$$\begin{aligned} \text{cmp}_{\text{at}}(\langle s, l \rangle, \langle t, m \rangle) &= \text{cmp}_{\prec_0}(s, t) \oplus \text{cmp}_{\text{sym}}(l, m) \\ (\approx) \prec_{\text{sym}} (\not\approx) \prec_{\text{sym}} (\geq) \prec_{\text{sym}} (>) \prec_{\text{sym}} (\leq) \prec_{\text{sym}} (<) \end{aligned}$$

Example 2. To illustrate Def. 4 and 5 we give some examples of how literals and terms are being oriented using the literal extension of the skeleton extension of a KBO ordering with weight function $w(s) = 1$ for every function symbol and every variable using an arbitrary precedence relation. In what follows f is a function, a is a constant, x, y are variables and $\succ, \succ' \in \{>, \geq\}$.

$$\begin{array}{ll} -f(x) \succ 0 \succ 5f(x) + a \succ' 0 & 3f(x) \succ f(x) \\ -f(x) \approx 0 \prec 5f(x) + a \succ 0 & 3f(x) \prec f(x) + 3 \\ f(x) \succ 0 \succ f(x) + x \approx 0 & 3f(x) + 1 \succ f(x) + 3 \\ f(x) \approx 0 \prec f(x) + x \approx 0 & x + y \prec x + y + y \\ -f(x) \approx 0 \prec f(x) + x \approx 0 & x + 2y \prec x + y + y \\ f(x) \not\approx 0 \succ -f(x) + x \approx 0 & x + y \text{ incomparable } x + a \\ -f(x) > 0 \succ -7f(x) \geq 0 & f(x) \text{ incomparable } f(y) \\ f(x) > 0 \succ 7f(x) \geq 0 & f(x) + f(y) \text{ incomparable to any term} \end{array}$$

4 Simplification and Redundancy in ALASCA

Simplification and redundancy elimination are essential techniques for improving reasoning in saturation-based theorem provers. Redundancy elimination is used to delete redundant clauses from the search space, while simplification rules are inferences that replace some clause C in the search space by a set of “simpler” clauses D : by clauses D that make C redundant. The aim of both of these techniques is to keep the search space small and tractable in order to speed up proof search. In this section, we introduce our new inferences rules ALASCA Demodulation (Dmd) and Inequality Merging (IM) (Fig. 5), handling redundancy elimination and simplification in $\text{ALASCA}_{\prec}^{\text{sel}}$. Dmd is a variant of the demodulation inference rule [31], adjusted for UFLRA. IM is a combination of a simplification rule and a redundancy elimination technique that extends classical tautology deletion to work when reasoning in UFLRA.

Simplification rules must be compatible with the calculus, meaning that newly derived clauses and side premises make the main premise redundant. Speaking more formally we say a simplification rule $\langle C_1 \dots C_n, D \rangle$ is *compatible with* $\text{ALASCA}_{\prec}^{\text{sel}}$ iff for any ground substitution θ it is the case that $\forall C' \in Cs. C'\theta \prec C_n\theta$ and $Cs\theta \models C_n\theta$ where $Cs = \{C_1, \dots, C_{n-1}, D\} \setminus \{\top\}$.

ALASCA Demodulation (Dmd). In the standard superposition calculus, demodulation is an inference rule that rewrites clauses by unit equalities $s \approx t$, such that the rewritten clause becomes redundant [31]. For example, the unit equality $f(x) \approx x$, where $x \in \mathbf{V}$, can be used to rewrite $g(f(a)) > 0$ to $g(a) > 0$. It is one of the most used simplification rules in provers like VAMPIRE, thus proof search in $\text{ALASCA}_{\prec}^{\text{sel}}$ should clearly benefit from enhancing it with demodulation.

In $\text{ALASCA}_{\prec}^{\text{sel}}$ unit equalities $s \approx t$ that would usually be used by demodulation, are normalized to $s - t \approx 0$, which in general will prevent the application of classic demodulation. Take for example the unit equality $f(x) \approx x$ from above. In our signature this literal will only be present as $f(x) - x \approx 0$, which will block the application of demodulation to rewrite the clause $g(f(a)) > 0$, as none of the subterms can be matched with $f(x) - x$.

As a remedy, we introduce our inference rule ALASCA Demodulation (Dmd) (Fig. 5). Dmd differs from standard demodulation: it uses unit equalities $k\underline{s} + t \approx 0$ to perform rewrites $s \Rightarrow -\frac{1}{k}t$ whenever s is the $\prec$ -maximal summand of $ks + t$, instead of performing rewrites $ks + t \Rightarrow 0$. This also allows not only for rewriting atomic terms, but automatically also rewriting sums of multiple terms. Consider for example $a + b \approx c$, with $a \succ b \succ c$, and the clause $f(a + b) > 0$. The equality will be normalized to $a + b - c \approx 0$ and then Dmd will rewrite $a \Rightarrow c - b$ in $f(a + b) > 0$, deriving $f((c - b) + b) > 0$ which is next normalized to $f(c) > 0$.

To save computation time, Dmd can also be used in a *preordered* mode: replacing the rule's condition $\forall t_0 \in \text{trms}(t). t_0 \sigma \prec s \sigma$ by $\forall t_0 \in \text{trms}(t). t_0 \prec s$. The advantage is that this condition does only depend on the left premise of the rule, thus it can be evaluated once per clause, and does not have to be evaluated for every pair of clauses the rule could be applied to. Nevertheless, this change makes Dmd less applicable. Our experiments allow for both of these Dmd variants (Sect. 6).

Inequality Merging (IM). The other simplification rule we introduce is Inequality Merging (IM), as introduced in Fig. 5. IM is inspired by tautology deletions to delete trivially redundant clauses of the form $C \vee L \vee \neg L$. However, negative inequality literals, like $\neg(t > 0)$, are normalized in ALASCA_0 to positive ones $-t \geq 0$, and hence tautology deletions cannot be applied. To overcome this limitation, IM combines any two literals $t_1 \diamond_1 0$ and $t_2 \diamond_2 0$ that define a bound on the same term $t = t_1 = t_2^\pm$, by merging them into one equivalent literal $t \diamond_3 0$. For combining the inequality symbols $\diamond_1, \diamond_2$ to get $\diamond_3$, we use a *join operation* $\sqcup$ defined by semantic set union of the respective relations.

For example, $-a + b > 0 \vee a - b \approx 0$ is rewritten to $-a + b \geq 0$. This merging of inequalities can detect tautologies, like $-t \geq 0 \vee t \not\approx 0 \Rightarrow \top$, in which case the clause will be deleted from the search space.

Lemma 1. IM and Dmd are compatible with $\text{ALASCA}_{\prec}^{\text{sel}}$.⁷

Inequality Merging	ALASCA Demodulation
$\frac{C \vee s \diamond_1 0 \vee s' \diamond_2 0}{C \vee s \diamond_3 0} \text{ (IM)}$	$\frac{k\underline{s} + t \approx 0 \quad \cancel{C[s\sigma]}}{C[-\frac{1}{k}t\sigma]} \text{ (Dmd)}$
where • $s \equiv_{\text{AC}} (s')^\pm$ • $\diamond_2 \in \{>, \geq\} \implies \diamond_1 \in \{>, \geq\}$ • $\diamond_3 = \begin{cases} \diamond_1 \sqcup \lesssim & \text{if } \pm = -, \diamond_2 = \gtrsim \in \{>, \geq\} \\ \diamond_1 \sqcup \diamond_2 & \text{else} \end{cases}$ • $L \sqcup R = \text{set}^{-1}(\text{set}(L) \cup \text{set}(R))$, • $\text{set}(\top) = \mathbb{Q}$ • $\text{set}(\diamond) = \{q \in \mathbb{Q} \mid q \diamond 0\}$ - for $\diamond \in \{>, \geq, <, \leq, \approx, \not\approx\}$ • the conclusion is $\top$ if $\diamond_3 = \top$	where • $(k\underline{s} + t \approx 0)\sigma \prec C[s\sigma]$ • $\forall t_0 \in \text{trms}(t). t_0 \sigma \prec s \sigma$ • s is atomic • $s \notin \mathbf{V}$

Figure 5: Simplification rules of $\text{ALASCA}_{\prec}^{\text{sel}}$. $\cancel{C}$ denotes C is deleted from the search space.

⁷Proof in Sect. D in the extended version of this paper [28]

5 Literal Selection in ALASCA

Literal selection is a mechanism to guide proof search: it selects subsets of the literals of a clause and restricts the application of inferences to only selected literals. This refines the idea of standard superposition to restrict inferences to only $\prec$ -maximal literals.

Example 3 (Superposition and Maximal Literals). *Consider the superposition inference rule $C \vee s \approx t, D \vee L[s] \vdash C \vee D \vee L[t]$. Take the set of clauses $c \not\approx b \vee d \approx c \vee c \approx a$ and $d \not\approx c$, with term ordering $d \succ c \succ b \succ a$. A side condition of superposition ensures that the rule is applied only when $L[s]$ is maximal wrt $\prec$, that is $L[s] \succ D$. Applied to our clause set, this means we only apply rewriting $d \rightarrow c$, but not rewriting $c \rightarrow a$, in $d \not\approx c$, deriving only the conclusion $c \not\approx b \vee c \approx a \vee c \not\approx c$.*

Selection is implemented by using a selection function sel that maps a clause to a subset of its literals ($\text{sel}(C) \subseteq C$). The completeness proof of superposition [2] shows that whenever a clause contains at least one negative literal, we can *select some of these negative literals* and only make inferences with these selected literals, skipping all other inferences involving this clause. Based on this observation any superposition inference that has a side condition of the form “ L is maximal in C ” is then replaced by the condition “either $L \in \text{sel}(C)$, or $\text{sel}(C) = \emptyset$ and L is maximal in C ”.

Example 4 (Superposition and Selection). *In Ex. 3, we can select the negative literal $c \not\approx b$ in the first clause; that is, $\text{sel}(c \not\approx b \vee d \approx c \vee c \approx a) = \{c \not\approx b\}$. No inferences can be performed with the selected literal $c \not\approx b$ and the other clause $d \not\approx c$, thus the set of clauses is saturated without computing any new inferences.*

Selection of negative literals cannot be however easily used in $\text{ALASCA}_{\prec}^{\text{sel}}$, as negative inequalities $\neg(s > 0)$ are normalized to positive ones $-s \geq 0$. As such, selecting arbitrary negative literals in $\text{ALASCA}_{\prec}^{\text{sel}}$ would not work with inequality literals. Our work therefore introduces the concept of *selectable* literals (Def. 6), allowing to extend selection functions also to non-negative literals.

In order to see which literals can be selected we need to understand key concepts of the completeness proof⁸. As in classic superposition [2], the completeness proof of $\text{ALASCA}_{\prec}^{\text{sel}}$ distinguishes productive literals. A key insight for selection is that any literal that is not a productive literal can be selected. As in classic superposition every equality might be productive and every disequality is unproductive, thus disequalities can be selected. In $\text{ALASCA}_{\prec}^{\text{sel}}$ the same holds for (dis)equalities. Inequalities in $\text{ALASCA}_{\prec}^{\text{sel}}$ may be productive iff their maximal term occurs with a positive coefficient (e.g. $2a - b > 0$ if $a \succ b$), but are unproductive if it occurs with a negative coefficient (e.g. $-2a - b > 0$). Taking into account that variables might contain terms with positive or negative coefficients, this leads us to the following definition of selectable literals.

Definition 6 (Selectable Literal). *A literal L is selectable iff it is stably negative $L \in \text{Lit}^-$. A selection function sel is ALASCA-compatible if, for any clause C , every $L \in \text{sel}(C)$ is selectable.*

Selection Strategies. The notion of selectable literals does not yield a unique selection function. Thus as there is no single best selection strategy for how many and/or which selectable literals should be selected, we implemented the following selection mechanisms:

- *total*: selecting all literals (including both selectable and non-selectable ones);

⁸We refer to Sect. 5 in the extended version of this paper [28] for a detailed discussion.

- *maximal*: selecting no literals, thus always using maximal literals only for rule applications;
- *random*: choosing a random selectable literal;
- *best*: choosing the “best” selectable literal, where best means it is maximal wrt some measure $\ll$, as defined in Def. 7;
- *lookahead*: choosing the selectable literal which unifies with the least number of terms that are currently present in the search space, adjusting results of [22];
- *incomplete*: for every strategy which chooses from the set of selectable literals, there is also an incomplete version that chooses from the set of all literals of the clause instead.

Def. 7 gives the ordering $\ll$ for the selection strategy *best*. Using $\ll$, we select literals to yield as few rule applications as possible, but, in contrast to *lookahead* selection, without looking at the entire search space. The measure $\ll$ of the selection strategy *best* is defined lexicographically in Def. 7, by comparing tuples of multiple values and using the following functions:

- **prob**: is a measure that reflects how likely it is for a term to unify with a ground term, assuming that the probability distribution of terms is $P(X = f(t_1 \dots t_n)) = \frac{1}{|\mathbf{F}|} (P(X = t_1) \cdot \dots \cdot P(X = t_n))$. As a greater **prob** will potentially yield more unifiers, we prefer terms that minimize **prob**.
- **cnt₊**: is the number of + symbols occurring in a literal. Due to **AC**, more occurrences of + will potentially yield more unifiers, thus we prefer terms that minimize **cnt₊**.
- **negEq**: is 1, if the literal is a negative equality, and 0 otherwise. The idea of this measure is that negative equalities are mostly introduced by UWA, and thus represent unifiability constraints. It is unclear whether terms with **negEq** being 0 or 1 should be preferred in general. Thus our portfolio approach (Sect. 6) uses both these options.

Finally, in every literal $t \diamond 0$, only the terms $\underline{\text{trms}}(t)$ are used to compute inferences, and the rest of the terms of $\underline{\text{smnds}}(t)$ are used only after the maximal ones have been resolved. We thus first compute the measures $m \in \{\text{cnt}_+, \text{prob}\}$ for the maximal terms m^{max} , and then for all terms m^{all} .

Definition 7 (Literal Selection Measure). *Our measure $\ll$ used within the (best) literal selection mechanism is parametrized by the option $\pm$ and defined below:*

$$\begin{aligned}
 L_1 \ll L_2 &\iff \text{measure}(L_1) <^{lex} \text{measure}(L_2) \\
 \text{measure}(L) &= \langle \pm \text{negEq}(L), -\text{prob}^{\text{max}}(L), -\text{prob}^{\text{all}}(L), -\text{cnt}_+^{\text{max}}(L), -\text{cnt}_+^{\text{all}}(L) \rangle \\
 \text{cnt}_+(t_1 + \dots + t_n) &= 1 + \sum_{i=1}^n \text{cnt}_+(t_i) & \text{prob}(t) &= \left(\frac{1}{|\mathbf{F}|}\right)^{\text{sz}(t) - |\{v \in \mathbf{V} \mid v \leq t\}|} \\
 \text{cnt}_+(f(t_1 \dots t_n)) &= \sum_{i=1}^n \text{cnt}_+(t_i) & \text{negEq}(t \diamond 0) &= \begin{cases} 1 & \text{if } (\diamond) = (\neq) \\ 0 & \text{else} \end{cases} \\
 \text{cnt}_+(x) &= 0 \\
 m^{\text{max}}(t \diamond 0) &= \Sigma \{m(t) \mid t \in \underline{\text{trms}}(t)\} & \text{sz}(f(t_1 \dots t_n)) &= 1 + \sum_{i=1}^n \text{sz}(t_i) \\
 m^{\text{all}}(t \diamond 0) &= \Sigma \{m(t) \mid t \in \underline{\text{trms}}(t)\} & \text{sz}(t_1 + t_2) &= 1 + \text{sz}(t_1) + \text{sz}(t_2) \\
 & & \text{sz}(v) &= 1
 \end{aligned}$$

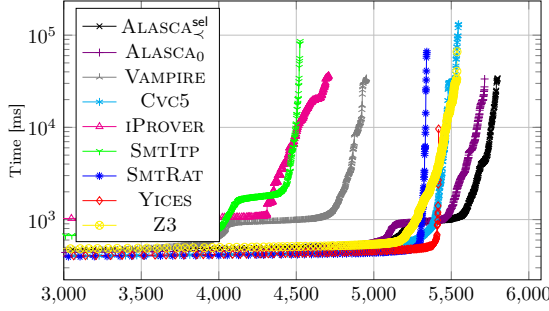


Figure 6: The cumulative number of benchmarks solved by each solver.

Benchmarks	ALASCA _{sel}	CVC5	Z3	YICES	SMTSAT	VAMPIRE	iPROVER	SMTITP
all	42	30	2	9	-	-	1	1
LRA	-	2	1	9	-	-	-	-
NRA	-	-	-	-	-	-	-	-
UFLRA	-	-	-	-	-	-	-	-
TRIANGULAR	13	-	-	-	-	-	-	-
LIMIT	24	-	-	-	-	-	-	-
SH	5	28	1	-	-	-	1	1

Figure 7: Number of problems solved uniquely by each solver.

Benchmarks (#)	ALASCA _{sel}	ALASCA ₀	CVC5	Z3	YICES	SMTSAT	VAMPIRE	iPROVER	SMTITP	solved
all (6291)	5799 (19)	5715 (1)	5547 (29)	5537 (2)	5418 (8)	5342 (0)	4953 (0)	4709 (1)	4525 (1)	5882
LRA (1629)	1594 (0)	1571 (0)	1346 (2)	1412 (1)	1611 (8)	1530 (0)	856 (0)	624 (0)	619 (0)	1616
NRA (3814)	3801 (0)	3797 (0)	3803 (0)	3812 (0)	3807 (0)	3812 (0)	3802 (0)	3782 (0)	3718 (0)	3812
UFLRA (13)	10 (0)	10 (0)	10 (0)	10 (0)	0 (0)	0 (0)	10 (0)	10 (0)	10 (0)	10
TRIANGULAR (34)	25 (0)	24 (0)	7 (0)	1 (0)	0 (0)	0 (0)	8 (0)	5 (0)	0 (0)	25
LIMIT (287)	116 (14)	102 (0)	91 (0)	81 (0)	0 (0)	0 (0)	82 (0)	82 (0)	0 (0)	116
SH (514)	253 (5)	211 (1)	290 (27)	221 (1)	0 (0)	0 (0)	195 (0)	206 (1)	178 (1)	303

Figure 8: Number of problems solved by each solver in each problem group. Column 1 lists the names and the number of problems of each group. In columns 2–8, a cell “ s (u)” for a solver and a benchmark group denotes that the solver solved s benchmarks, of which u benchmarks were uniquely solved by that solver.

Delayed VE. The rule Variable Elimination (VE) removes literals $kx + t \diamond 0$, since inferences involving unshielded variables x cannot be lifted easily. If a clause contains a selected literal, though there are no inferences involving x , that would need to be lifted. Thus the rule VE needs to only be applied if there are no selected literals in a clause, and can be skipped if there are selected ones. We call this skipping of VE inferences using *delayed* variable elimination, and implemented an option to enable or disable this behaviour.

6 Implementation and Experiments

We implemented⁹ ALASCA_{sel} with all additions described in the paper in the theorem prover VAMPIRE [6]. Our implementation handles a vast option space of various literal selection strategies (Sect. 5), enabling/disabling IM (Sect. 4), Dmd (Sect. 4), delayed variable elimination (Sect. 5), and running Dmd in preordered mode.

To accommodate the many options and strategies in ALASCA_{sel}, we created a portfolio of randomized option combinations. For this we chose a randomized subset B of 50% of our full benchmark set. Then, the following was repeated: (i) select a timeout within $[1, 10]$ seconds; (ii) generate 80 random option combinations; (iii) run options on the current set B . We chose the strategy that solved most problems of B and added it to the portfolio, after which we removed all solved problems from B . We repeated this process for 12 hours. In our experiments, ALASCA_{sel} refers to the resulting portfolio¹⁰ setting.

⁹ Available at <https://github.com/vprover/vampire/tree/alasca-lpar-2026>.

We use the state-of-the-art benchmark set of [26]¹⁰. We only consider unsatisfiable problems, as these are the only relevant ones for $\text{ALASCA}_{\prec}^{\text{sel}}$. Our benchmark set is split into multiple groups, as summarized in Fig. 8 and detailed next: NRA (non-linear real arithmetic), LRA and UFLRA represent the respective SMT-LIB logics [5]; TRIANGULAR and LIMIT contain mathematical properties from [26]; SH contains Sledgehammer proving problems from [14].

We compare $\text{ALASCA}_{\prec}^{\text{sel}}$ against all solvers that entered the **Arith** division of the SMT-COMP 2025 system competition¹¹. We also compare $\text{ALASCA}_{\prec}^{\text{sel}}$ to the original calculus ALASCA_0 from [26], and VAMPIRE without ALASCA enabled.

Our experiments were conducted on an AMD EPYC 7252 8-Core Processor with 16 cores and 2 threads per core and a RAM of 125Gi. Benchmarks were run for 120 seconds on 4 cores in parallel. Experimental results are summarized in Fig. 6–Fig. 7. As evidenced in Fig. 6, our implementation $\text{ALASCA}_{\prec}^{\text{sel}}$ vastly outperforms all other solvers: already after about 1 second, $\text{ALASCA}_{\prec}^{\text{sel}}$ solves substantially more benchmarks than all other solvers at that time. Looking at the raw data reveals that $\text{ALASCA}_{\prec}^{\text{sel}}$ solves 5716 problems within only 4330 milliseconds, which is more than any other prover solves in the full timeout of 120 seconds. Fig. 8 shows that $\text{ALASCA}_{\prec}^{\text{sel}}$ outperforms ALASCA_0 by 84 (1.46 %) more solved problems and dominates ALASCA_0 in every problem category. $\text{ALASCA}_{\prec}^{\text{sel}}$ also outperforms VAMPIRE in every problem category. Fig. 7 lists the number of uniquely solved problems by $\text{ALASCA}_{\prec}^{\text{sel}}$, demonstrating ALASCA-based reasoning solves problems no other solver is able to prove.

7 Related Work

The most prominent approaches for (restrictions of) UFLRA are SMT solving [13, 4, 10] and first-order theorem proving [17, 3, 43, 30, 26].

SMT solvers naturally support reasoning in quantifier-free UFLRA using dedicated decision procedures. As quantified UFLRA is undecidable [30], incomplete quantifier instantiations are used to instantiate universally quantified variables with pragmatically chosen terms, trying to reduce the problem of solving quantified UFLRA problems to quantifier-free UFLRA instances. Various quantifier instantiation heuristics emerged, notably E-matching [15] and model-based quantifier instantiation [19]. Further, [40, 23] introduce strategies for the order of enumerating all potentially useful instances. These approaches are restricted to heuristic-driven instantiations, limiting their applicability to quantified UFLRA in contrast to $\text{ALASCA}_{\prec}^{\text{sel}}$.

When only considering quantified linear real arithmetic (LRA), various Quantifier Elimination (QE) methods like [18, 33, 29, 1, 11, 42, 27] can be used to decide LRA. A model-driven approach to QE has been introduced in [9]. $\text{ALASCA}_{\prec}^{\text{sel}}$ uses a custom QE method introduced in [26] for LRA in its rule VE.

Using the AVATAR architecture [36] and theory instantiation [39], SMT solving can be used within saturation algorithms of first-order theorem provers. As shown in [16] SMT-solvers can be used to detect **AC** functions in input problems, enabling specialized reasoning techniques for these functions. Further theory reasoning with axioms in saturation provers, clause selection [21], and simplification rules have been introduced in [37].

Inference systems for first-order theorem provers were improved by introducing chaining calculi [3] for transitive relations, combining them with **AC**-reasoning in [43], and introducing a ground calculus for UFLRA in [30]. Fusing ideas of [3, 43, 30], the calculus ALASCA_0 has been proposed in [26] in combination with UWA [8]. Our work enhances ALASCA_0 by introducing selection, simplification and redundancy elimination. Another notable saturation-based

¹⁰Available at https://github.com/vprover/vampire_benchmarks/tree/master/alasca-lpar-2026.

¹¹See <https://smt-comp.github.io/2025>

approach is hierarchic superposition [7], which uses abstraction to separate interpreted from uninterpreted symbols. Superposition inferences are used to resolve uninterpreted symbols until a decision procedure is used to resolve the remaining purely interpreted clauses. In contrast ALASCA reasons about uninterpreted and interpreted symbols at the same time.

8 Conclusions

We introduced the calculus $\text{ALASCA}_{\prec}^{\text{sel}}$ enhancing ALASCA_0 with Literal Selection and the new simplification rules ALASCA Demodulation and Inequality Merging. We implemented $\text{ALASCA}_{\prec}^{\text{sel}}$ in a standard saturation loop and demonstrated experimentally that our work outperforms the state-of-the-art. We aim to further improve literal selection by finding an accurate probability of a term unifying with other terms modulo **AC** which could be a promising heuristic for literal selection. We are also interested in instantiations and generalizations modulo **AC**, increasing applicability of demodulation.

Acknowledgements. This research was funded in whole or in part by the Austrian Science Fund – FWF Grant ID 10.55776/EFP2661725 and SpyCoDe Grant 10.55776/F85, the ERC Proof of Concept Grant LEARN 101213411, the TU Wien Doctoral College SecInt, and the WWTF grant [ForSmart Grant ID: 10.47379/ICT22007].

References

- [1] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical algebraic decomposition I: the basic algorithm. *SIAM J. Comput.*, 13(4):865–877, 1984.
- [2] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.*, 4(3):217–247, 1994.
- [3] Leo Bachmair and Harald Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *J. ACM*, 45(6):1007–1049, 1998.
- [4] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS*, volume 13243 of *LNCS*, pages 415–442. Springer, 2022.
- [5] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The smt-lib standard: Version 2.7. Technical report, Department of Computer Science, The University of Iowa, 2025.
- [6] Filip Bártek, Ahmed Bhayat, Robin Coutelier, Márton Hajdú, Matthias Hetzenberger, Petra Hozová, Laura Kovács, Jakob Rath, Michael Rawson, Giles Reger, Martin Suda, Johannes Schoisswohl, and Andrei Voronkov. The vampire diary. In *CAV*, volume 15933 of *LNCS*, pages 57–71. Springer, 2025.
- [7] Peter Baumgartner and Uwe Waldmann. Hierarchic superposition revisited. In Carsten Lutz, Uli Sattler, Cesare Tinelli, Anni-Yasmin Turhan, and Frank Wolter, editors, *Description Logic, Theory Combination, and All That - Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, *LNCS*, pages 15–56. Springer, 2019.
- [8] Ahmed Bhayat, Konstantin Korovin, Laura Kovács, and Johannes Schoisswohl. Refining unification with abstraction. In *LPAR*, volume 94 of *EPiC*, pages 36–47. EasyChair, 2023.
- [9] Nikolaj S. Bjørner and Mikoláš Janota. Playing with quantified satisfaction. In *LPAR*, volume 35 of *EPiC*, pages 15–27. EasyChair, 2015.
- [10] Nikolaj S. Bjørner and Lev Nachmanson. Arithmetic solving in Z3. In *CAV*, volume 14681 of *LNCS*, pages 26–41. Springer, 2024.
- [11] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition: a synopsis. *SIGSAM Bull.*, 10(1):10–12, 1976.
- [12] Simon Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. (Extensions de la Superposition pour l’Arithmétique Linéaire Entière, l’Induction Structurale, et bien plus encore)*. PhD thesis, École Polytechnique, Palaiseau, France, 2015.
- [13] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.
- [14] Martin Desharnais, Petar Vukmirovic, Jasmin Blanchette, and Makarius Wenzel. Seventeen provers under the hammer. In *ITP*, volume 237 of *LIPIcs*, pages 8:1–8:18, 2022.
- [15] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.
- [16] André Duarte and Konstantin Korovin. Implementing superposition in iprover (system description). In *IJCAR*, volume 12167 of *LNCS*, pages 388–397. Springer, 2020.
- [17] André Duarte and Konstantin Korovin. Ground joinability and connectedness in the superposition calculus. In *IJCAR*, volume 13385 of *LNCS*, pages 169–187. Springer, 2022.
- [18] Jean Baptiste Joseph Fourier. Solution d’une question particuliere du calcul des inégalités. *Nouveau Bulletin des Sciences par la Société philomatique de Paris*, 99:100, 1826.
- [19] Yeting Ge and Leonardo Mendonça de Moura. Complete instantiation for quantified formulas in satisfiability modulo theories. In *CAV*, volume 5643 of *LNCS*, pages 306–320. Springer, 2009.
- [20] Bernhard Gleiss, Laura Kovács, and Jakob Rath. Subsumption demodulation in first-order theorem

- proving. In *IJCAR*, volume 12166 of *LNCS*, pages 297–315. Springer, 2020.
- [21] Bernhard Gleiss and Martin Suda. Layered clause selection for theory reasoning. *CoRR*, abs/2001.09705, 2020.
 - [22] Krystof Hoder, Giles Reger, Martin Suda, and Andrei Voronkov. Selecting the selection. In *IJCAR*, volume 9706 of *LNCS*, pages 313–329. Springer, 2016.
 - [23] Mikolás Janota, Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. Fair and adventurous enumeration of quantifier instantiations. In *FMCAD*, pages 256–260. IEEE, 2021.
 - [24] Sam Kamin and Jean-Jacques Lévy. Two generalizations of the recursive path ordering. *Unpublished manuscript*, 1980.
 - [25] Donald E Knuth and Peter B Bendix. Simple word problems in universal algebras. *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 342–376, 1983.
 - [26] Konstantin Korovin, Laura Kovács, Giles Reger, Johannes Schoisswohl, and Andrei Voronkov. ALASCA: reasoning in quantified linear arithmetic. In *TACAS*, volume 13993 of *LNCS*, pages 647–665. Springer, 2023.
 - [27] Konstantin Korovin, Laura Kovács, and Johannes Schoisswohl. VIRAS: conflict-driven quantifier elimination for integer-real arithmetic. In *LPAR*, volume 100 of *EPiC*, pages 147–164. EasyChair, 2024.
 - [28] Konstantin Korovin, Laura Kovács, Johannes Schoisswohl, and Andrei Voronkov. Enhancing superposition reasoning in linear real arithmetic through selection and simplification (extended version). *EasyChair Preprint no. 16032*, 2026.
 - [29] Konstantin Korovin, Nestan Tsiskaridze, and Andrei Voronkov. Conflict resolution. In *CP*, volume 5732 of *LNCS*, pages 509–523. Springer, 2009.
 - [30] Konstantin Korovin and Andrei Voronkov. Integrating linear arithmetic into superposition calculus. In *CSL*, volume 4646 of *LNCS*, pages 223–237. Springer, 2007.
 - [31] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
 - [32] Amira T Mahmoud, Ahmad A Mohammed, Mahitap Ayman, Walaa Medhat, Sahar Selim, Hala Zayed, Ahmed H Yousef, and Nahla Elaraby. Formal verification of code conversion: A comprehensive survey. *Technologies*, 12(12):244, 2024.
 - [33] Theodore Samuel Motzkin. *Beiträge zur Theorie der linearen Ungleichungen*. Azriel, 1936.
 - [34] M. Saqib Nawaz, Moin Malik, Yi Li, Meng Sun, and Muhammad Ikram Ullah Lali. A survey on theorem provers in formal methods. *CoRR*, abs/1912.03028, 2019.
 - [35] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning*, volume 1, pages 371–443. Elsevier and MIT Press, 2001.
 - [36] Giles Reger, Nikolaj S. Bjørner, Martin Suda, and Andrei Voronkov. AVATAR modulo theories. In *GCAI*, volume 41 of *EPiC*, pages 39–52. EasyChair, 2016.
 - [37] Giles Reger, Johannes Schoisswohl, and Andrei Voronkov. Making theory reasoning simpler. In *TACAS*, volume 12652 of *LNCS*, pages 164–180. Springer, 2021.
 - [38] Giles Reger and Martin Suda. Set of support for theory reasoning. In *IWIL*, volume 1 of *Kalpa Publications in Computing*, pages 124–134. EasyChair, 2017.
 - [39] Giles Reger, Martin Suda, and Andrei Voronkov. Unification with abstraction and theory instantiation in saturation-based reasoning. In *TACAS*, volume 10805 of *LNCS*, pages 3–22. Springer, 2018.
 - [40] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In *TACAS*, volume 10806 of *LNCS*, pages 112–131. Springer, 2018.
 - [41] Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. Faster, higher, stronger: E 2.3. In *CADE*, volume 11716 of *LNCS*, pages 495–507. Springer, 2019.
 - [42] Thomas Sturm. Thirty years of virtual substitution: Foundations, techniques, applications. In *ISSAC*, pages 11–16. ACM, 2018.

- [43] Uwe Waldmann. Superposition and chaining for totally ordered divisible abelian groups. In IJCAR, volume 2083 of LNCS, pages 226–241. Springer, 2001.
- [44] Uwe Waldmann, Sophie Turret, Simon Robillard, and Jasmin Blanchette. A comprehensive framework for saturation theorem proving. J. Autom. Reason., 66(4):499–539, 2022.
- [45] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Renate A. Schmidt, editor, CADE, volume 5663 of LNCS, pages 140–145. Springer, 2009.