EasyChair Preprint

№ 9606

ALASCA: Reasoning in Quantified Linear Arithmetic (Extended Version)

Konstantin Korovin, Laura Kovacs, Giles Reger,
Johannes Schoisswohl and Andrei Voronkov

January 21, 2023

ALASCA: Reasoning in Quantified Linear Arithmetic (Extended Version)

Konstantin Korovin[3], Laura Kovács[1], Giles Reger[3,4], Johannes Schoisswohl[1], and Andrei Voronkov[2,3]

[1] TU Wien
[2] EasyChair
[3] University of Manchester
[4] Amazon AWS

Abstract. Automated reasoning is routinely used in the rigorous construction and analysis of complex systems. Among different theories, arithmetic stands out as one of the most frequently used and at the same time one of the most challenging in the presence of quantifiers and uninterpreted function symbols. First-order theorem provers perform very well on quantified problems due to the efficient superposition calculus, but support for arithmetic reasoning is limited to heuristic axioms. In this paper, we introduce the Alasca calculus that lifts superposition reasoning to the linear arithmetic domain. We show that Alasca is both sound and complete with respect to an axiomatisation of linear arithmetic. We implemented and evaluated Alasca using the Vampire theorem prover, solving many more challenging problems compared to state-of-the-art reasoners.

This paper is an extended version of the paper "ALASCA: Reasoning in Quantified Linear Arithmetic" published at TACAS 2023, written by the same set of the authors. In this extended version we provide the formal proofs of the results from our TACAS 2023 paper.

Keywords: Automated Reasoning · Linear Arithmetic · SMT · Quantified First-Order Logic · Theorem Proving

1 Introduction

Automated reasoning is undergoing a rapid development thanks to its successful use, for example, in mathematical theory formalisation [15], formal verification [16] and web security [13]. The use of automated reasoning in these areas is mostly driven by the application of SMT solving for quantifier-free formulas [6, 12, 28]. However, there exist many use case scenarios, such as expressing arithmetic operations over memory allocation and financial transactions [1, 18, 20, 31], which require complex first-order quantification. SMT solvers handle quantifiers using heuristic instantiation in domain-specific model construction [10, 27, 29, 35]. While being incomplete in most cases, instantiation requires instances to be produced to perform reasoning, which can lead to an explosion in work required for

quantifier-heavy problems. What is rather needed to address the above use cases is a reasoning approach able to handle both theories and complex applications of quantifiers. Our work tackles this challenge and designs a *practical, low-cost methodology* for proving first-order quantified linear arithmetic properties.

The problem of combining quantifiers with theories, and especially with arithmetic, is recognised as a major challenge in both SMT and first-order proving communities. In this paper *we focus on first-order, i.e. quantified, reasoning with linear arithmetic and uninterpreted functions.* In [25], it is shown that the validity problem for first-order reasoning with linear arithmetic and uninterpreted functions is Π^1_1-complete even when quantifiers are restricted to non-theory sorts. Therefore, there is no sound and complete calculus for this logic.

Quantified Reasoning in Linear Arithmetic – Related Works. In practice, there are two classes of methods of reasoning in first-order theory reasoning, and in particular with linear real arithmetic. SMT solvers use *instance-based methods*, where they repeatedly generate ground, that is quantifier-free, instances of quantified formulas and use decision procedures to check satisfiability of the resulting set of ground formulas [10, 27, 35]. Superposition-based first-order theorem provers use *saturation algorithms* [14, 26, 36]. In essense, they start with an initial set of clauses obtained by preprocessing the input formulas (initial search space) and repeatedly apply inference rules (such as superposition) to clauses in the search space, adding their (generally, non-ground) consequences to the search space. These two classes of methods are very different in nature and complement each other.

The superposition calculus [4, 30] is a refutationally complete calculus for first-order logic with equality that is used by modern first-order provers, for example, Vampire [26], E [36], iProver [17] and Zipperposition [14]. There have been a number of practical extensions to this calculus for reasoning in first-order theories, in particular for linear arithmetic [9, 11, 23]. Superposition theorem provers have become efficient and powerful on theory reasoning after the introduction of the AVATAR architecture [32, 37], which allows generated ground clauses to be passed to SMT solvers. Yet, superposition theorem provers have a major source of inefficiency. To work with theories, one has to add *theory axioms*, for example the transitivity of inequality $\forall x \forall y \forall z (x \leq y \wedge y \leq z \rightarrow x \leq z)$. In clausal form, this formula becomes $\neg x \leq y \vee \neg y \leq z \vee x \leq z$ where $\neg x \leq y$ can be resolved against *every* clause in which an inequality literal $s \leq t$ is selected. This, with other prolific theory axioms, results in a very significant growth of the search space. Note that SMT solvers do not use and do not need such theory axioms.

A natural solution is to try to eliminate some theory axioms, but this is notoriously difficult both in theory and in practice. In [25], the Lasca calculus was proposed, which replaced several theory axioms of linear arithmetic, including transitivity of inequality, by a new inference rule inspired by Fourier-Motzkin elimination and some additional rules. Lasca was shown to be complete for the ground case. But, after 15 years, Lasca is still not implemented, due to its complexity and lack of clear treatment for the non-ground case. As we argue

in Section 5, lifting Lasca to the non-ground setting is nearly impossible as a non-ground extension of the underlining ordering is missing in [25].

Lifting Lasca to Alasca– Our contributions. In this paper we introduce a new non-ground version of Lasca, which we call Abstracting Lasca (Alasca). Our Alasca calculus comes with new abstraction mechanisms (Section 4), inference rules and orderings (Section 5), which all together are proved to yield a sound and complete approach with respect to some partial axiomatisation of linear arithmetic (Theorem 5). In a nutshell, we make Alasca both work and scale by introducing (i) a novel variable elimination rule within saturation-based proof search (Figure 6); (ii) an analogue of *unification with abstraction* [33] needed for non-ground reasoning (Section 4); and (iii) a new non-ground ordering and powerful background theory for unification, which is not restricted to arithmetic but can be used with arbitrary theories (Section 5). As a result, Alasca improves [25] by ground modifications and lifting of Lasca in a finitary way, and complements [3, 39] with variable elimination rules that are competible with standard saturation algorithms. We also *demonstrate the practicality and efficiency* of Alasca (Section 6). To this end, we implemented Alasca in Vampire and show that this solves overall more problems than existing theorem provers.

2 Motivating Example

Consider the following mathematical property:

$$\forall x, y.\big(f(2x, y) > 2x + y \vee f(x + 1, y) > x + 2y\big) \rightarrow \forall x.\exists y.f(2, y) > x \quad (1)$$

where f is an uninterpreted function. While property (1) holds, deriving its validity is hard for state-of-the-art reasoners: only veriT [2] can solve it. Despite its seeming simplicity, this problem requires non-trivial handling of quantifiers and arithmetic. Namely, one would need to unify (modulo theory) the terms $2x$ and $x + 1$ (which can be done by instantiating x with 1) and then derive $f(2, y) > 2 + y \vee f(2, y) > 1 + 2y$. Further, one also needs to prove that $f(2, y)$ is always greater than the minimum of $2 + y$ and $1 + 2y$, for arbitrary y.

Vampire with Alasca finds a remarkably short proof as shown in Fig. 1. To prove (1) its negation is shown unsatisfiable by first negating and translating into clausal form (by using skolemization and normalisation, which shifts arithmetic terms to be compared to 0), as listed in lines 1–4. Next a lower bound for $f(2x, y)$ is established: In line 5, using our new inequality factoring (IF) rule with unification with abstraction (see Fig. 5), the constraint $2x \not\approx x + 1$ is introduced, and establishing thereby that if $2x \approx 1 + x$ and $y + 2x \leq 2y + x$, then $f(2x, y) > 2x + y$. After further normalisation, the inequalities $sk \geq f(2, y)$ and $f(2x, y) > 2x + y$ are used to derive $sk > 2x + y$ in line 7, using the Fourier-Motzkin Elimination rule (FM), while still keeping track of the constraint $2x \not\approx x + 1$. By applying the Variable Elimination rule (VE) twice, the empty clause $\square$ is derived in line 10, showing the unsatisfiability of the negation of (1).

1.	$f(2x, y) > 2x + y \vee f(x + 1, y) > x + 2y$	Hypothesis
2.	$\neg f(2, y) > sk$	Skolemized, Negated Conjecture
3.	$f(2y, x) - 2y - x > 0 \vee f(1 + y, x) - y - 2x > 0$	Normalisation 1
4.	$-f(2, y) + sk \geq 0$	Normalisation 2
5.	$f(2x, y) - 2x - y > 0 \vee y + 2x - 2y - x > 0 \vee 2x \not\approx 1 + x$	(IF) 3
6.	$f(2x, y) - 2x - y > 0 \vee x - y > 0 \vee 0 \not\approx x - 1$	Normalisation 5
7.	$-2x - y + sk > 0 \vee x - y > 0 \vee 0 \not\approx x - 1 \vee 2x \not\approx 2$	(FM) 6,4
8.	$-2x - y + sk > 0 \vee x - y > 0 \vee 0 \not\approx x - 1$	Normalisation 7
9.	$0 \not\approx x - 1$	(VE) 8
10.	$\square$	(VE) 9

Fig. 1. A refutational proof using the calculus introduced in this paper. Variables x, y are implicitly universally quantified, and sk is an uninterpreted constant.

The key steps in the proof (and the reason why it was found in a short time) are: (1) the use of the theory rules (FM), and (IF); (2) the use of the new variable elimination rule (VE), and finally, a consistent use of unification with abstraction. These rules give a significant reduction compared to the number of steps required using theory axioms. In particular, not using (FM) would require the use of transitivity and generation of several intermediate clauses. As well as shortening the proof, we eliminate the fatal impact on proof search from generating a large number of irrellevant formulas from theory axioms.

Indeed, such short proofs are also found quickly. Similar to the previous example, $\forall x, y.\big(f(g(x)+g(a), y) > 2x+y \vee f(2g(x)) > x+2y\big) \rightarrow \exists k.\forall x \exists z.f(2g(k), z) > x$ has a short proof of 7 steps, excluding CNF transformation and normalisation steps, found by Vampire with Alasca. This proof was found in almost no time (only 37 clauses were generated) but cannot be solved by any other solver. This shows the power of the calculus.

3 Background and Notation

Multi-Sorted First-Order Logic. We assume familiarity with standard first-order logic with equality, with all standard boolean connectives and quantifiers in the language. We consider a multi-sorted first-order language, with sorts $\tau_{\mathbb{Q}}, \tau_1, \ldots, \tau_n$. The sort $\tau_{\mathbb{Q}}$ is the *sort of rationals*, whereas $\tau_1, \ldots, \tau_n$ are *uninterpreted sorts*. We write $\approx_\tau$ for the equality predicate of τ. We denote the set of all terms as $\mathbf{T}$, variables as $\mathbf{V}$, and literals as $\mathbf{L}$. Throughout this paper, we denote terms by s, t, u, variables by x, y, z, function symbols by f, g, h, all possibly with indices. Given a term t such that t is $f(\ldots)$, we write $\mathsf{sym}(t)$ for f, referring that f is the top level symbol of t. We write $t : \tau$ to denote that t is a term of sort τ. A term, or literal is called *ground*, when it does not contain any variables. We refer to the sets of all ground terms, and literals as $\mathbf{T}^\theta$, and $\mathbf{L}^\theta$ respectively.

We denote predicates by P, Q, literals by L, clauses by C, D, formulas by F, G, and sets of formulas (axioms) by $\mathcal{E}$, possibly with indices. We write $F \models G$ to denote that whenever F holds in a model, then G does as well. We call a

function (similarly, for predicates) f uninterpreted wrt some set of equations $\mathcal{E}$ if whenever $\mathcal{E} \models f(s_1 \dots s_n) \approx f(t_1 \dots t_n)$, then $\mathcal{E} \models s_1 \approx t_1 \wedge \dots \wedge s_n \approx t_n$. A function f is interpreted wrt $\mathcal{E}$ if it is not uninterpreted.

Rational Sort. We assume the signature contains a countable set of unary functions $k : \tau_{\mathbb{Q}} \mapsto \tau_{\mathbb{Q}}$ for every $k \in \mathbb{Q}$ and refer to k as *numeral multiplications*. In addition, the signature is assumed to also contain a constant $1 : \tau_{\mathbb{Q}}$, a function $+ : \tau_{\mathbb{Q}} \times \tau_{\mathbb{Q}} \mapsto \tau_{\mathbb{Q}}$, and predicate symbols $>, \geq$: $\mathbf{P}(\tau_{\mathbb{Q}} \times \tau_{\mathbb{Q}})$, as well as an arbitrary number of other function symbols. For every numeral multiplication $k \in \mathbb{Q} \setminus \{1\}$, we simply write k to denote the term $k(1)$ obtained by the numeral multiplication k applied to 1; in these cases, we refer to k as *numerals*. Throughout this paper, we use j, k, l to denote numerals, or numeral multiplications, possibly with indices.

We write $-t$ to denote the term $-1(t)$. If j, k are two numeral multiplications, by (jk) and $(j + k)$ we denote the numeral multiplication that corresponds to the result of multiplying and adding the rationals/numerals j and k, respectively. For applications of numeral multiplications $j(t)$ we may omit the parenthesis and write jt instead. If we write $+k$, or $-k$ for some numeral k, we assume k itself is positive. We write $\pm$ (and $\mp$) to denote either of the symbols $+$ or $-$ (and respectively $-$ or $+$). For $q \in \mathbb{Q}$ we define $\mathbf{sign}(q)$ to be 1 if $q > 0$, -1 if $q < 0$, and 0 otherwise. We call $+, \geq, >, 1$, and the numeral multiplications the $\mathbb{Q}$ *symbols*. Finally, an *atomic term* is either a logical variable, or the term 1, or a term whose top level function symbol is not a $\mathbb{Q}$ symbol.

A $\mathbb{Q}$*-model* interprets the sort $\tau_{\mathbb{Q}}$ as $\mathbb{Q}$, and all $\mathbb{Q}$ symbols as their corresponding functions/predicates on $\mathbb{Q}$. We write $\mathbb{Q} \models C$ iff for every $\mathbb{Q}$-model M, $M \models C$ holds. If $\mathcal{E}$ is a set of formulas, we call a model M a $\mathcal{E}$*-model* if $M \models \mathcal{E}$.

Term Orderings. Let $\equiv$ be some equivalence relation. Then we say that s is a subterm of t wrt $\equiv$ ($s \trianglelefteq_{\equiv} t$) if $s \equiv t$, or $t \equiv f(t_1 \dots t_n)$ and $s \trianglelefteq t_i$ for some i. We say that s is a strict subterm of t wrt $\equiv$ ($s \triangleleft_{\equiv} t$) if $s \trianglelefteq_{\equiv} t$ and $s \not\equiv t$. We say s is a subterm of t ($s \trianglelefteq t$) if $s \trianglelefteq_{=} t$, and s is a strict subterm of t ($s \triangleleft t$) if $s \triangleleft_{=} t$. We write $u[s]$ to denote that s is a subterm of u. That is, $s \trianglelefteq u$; similar notation will also be used for literals $L[s]$ and clauses $C[s]$. We denote by $u[s \mapsto t]$ the term resulting from replacing all subterms s of u by t.

Multisets (of term, literals) are denoted with $\dot{\{} \dots \dot{\}}$. For a multiset S and natural number $n \in \mathbb{N}$, we define $0 * S = \emptyset$, and $n * S = (n - 1 * S) \cup S$ for $n > 0$.

Let $\prec$ be a relation and $\equiv$ be an equivalence relation. By $\prec^{\mathsf{mul}}_{\equiv}$ we denote the *multiset extension* of $\prec$, defined as the smallest relation satisfying $M \cup \dot{\{}s_1, \dots, s_n\dot{\}} \prec^{\mathsf{mul}}_{\equiv} N \cup \dot{\{}t\dot{\}}$, where $M \equiv N$, $n \geq 0$, and $s_i \prec t$ for $1 \leq i \leq n$. For $n, m \in \mathbb{N}$, by $\prec^{\mathsf{wmul}}_{\equiv}$ we denote the *weighted multiset extension*, defined by $\langle \frac{1}{n}, S\rangle \prec^{\mathsf{wmul}}_{\equiv} \langle \frac{1}{m}, T\rangle$ iff $m * S \prec^{\mathsf{mul}}_{\equiv} n * T$. We omit the equivalence relation $\equiv$ if it is clear in the context.

Let $\prec$ be a relation and $\equiv$ be an equivalence relation. By $\prec^{\mathsf{lex}}_{\equiv}$ we denote the *lexicographic extension* of $\prec$, written as $\langle s_1 \dots s_n\rangle \prec^{\mathsf{lex}}_{\equiv} \langle t_1 \dots t_n\rangle$, iff there is an i such that $s_i \prec t_i$ and $s_1 \equiv t_1 \dots s_{i-1} \equiv t_{i-1}$.

Let s, t, t_i be terms, θ, θ' be ground substitutions and $\mathcal{E}$ be a set of axioms. We write $s \equiv_{\mathcal{E}} t$ for $\mathcal{E} \models s \approx t$ and $\theta \equiv_{\mathcal{E}} \theta'$ iff for all variables x we have $x\theta \equiv_{\mathcal{E}} x\theta'$.

We say that s is a $\mathcal{E}$-subterm of t ($s \trianglelefteq_{\mathcal{E}} t$) if $s \trianglelefteq_{\equiv_{\mathcal{E}}} t$, and s is a strict $\mathcal{E}$-subterm of t ($s \triangleleft_{\mathcal{E}} t$) if $s \triangleleft_{\equiv_{\mathcal{E}}} t$.

4 Theoretical Foundation for Unification with Abstraction

Our motivating example from Section 2 showcases that first-order arithmetic reasoning requires (i) establishing syntactic difference among terms (e.g. $2x$ and $x + 1$), while (ii) deriving they are semantically the same in models of a background theory $\mathcal{E}$ (e.g. the theory $\mathbb{Q}$).

A naive approach addressing (i)-(ii) would be to use an axiomatisation of the background theory $\mathcal{E}$, and use this axiomatisation for proof search in uninterpreted first-order logic. Such an approach can however be very costly. For example, even a relatively simple background theory **AC** axiomatizing commutativity and associativity of $\approx$, that is $\mathbf{AC} = \{x + y \approx y + x, x + (y + z) \approx (x + y) + z\}$, would make a superposition-based theorem prover derive a vast amount of useless/redundant formulas as equational tautologies. An approach to circumvent such inefficient handling of equality reasoning is to use *unification modulo* **AC**, or in general *unification modulo* $\mathcal{E}$, as already advocated in [22, 33, 39]. In this section we describe the adjustments we made towards unification modulo $\mathcal{E}$, allowing us to introduce *unification with abstraction* (Sect. 4.1). We also show under which condition our method can be used to turn a complete superposition calculus using unification modulo $\mathcal{E}$ into a complete superposition calculus using unification with abstraction. Concretely, we show how this can be used for the specific theory of arithmetic $\mathcal{A}_{\mathsf{eq}}$ in the calculus Alasca (Sect. 4.2).

4.1 Unification with Abstraction – UWA

In a nutshell, unification modulo $\mathcal{E}$ finds substitiutions σ that make two terms s, t equal in the background theory, i.e. $\mathcal{E} \models s\sigma \approx t\sigma$. While unification modulo $\mathcal{E}$ removes the need for axiomatisation of $\mathcal{E}$ during superposition reasoning, it comes with some inefficiencies. Most importantly, in contrast to syntactic unification, there is no unique most general unifier $\mathsf{mgu}(s, t)$ when unifying modulo $\mathcal{E}$ but only minimal complete sets of unifiers $\mathsf{mcu}_{\mathcal{E}}(s, t)$, which can be very large; for example, unification modulo **AC** is doubly exponential in general [22].

Bypassing the need for unification modulo $\mathcal{E}$, *fully abstracted clauses* are used in [39], without the need for axiomatisation of the theory $\mathcal{E}$ and without compromising completeness of the underlining superposition-based calculus. Our work extends ideas from [39] and adjusts *unification with abstraction* (uwa) from [33], allowing us to prove completeness of a calculus using uwa (Theorem 3).

Example 1. Let us first consider the example of factoring the clause $p(2x) \vee p(x+1)$, a simplified version of the unification step performed in line 5 in Fig. 1. That is, unifying the literals $p(2x)$ and $p(x + 1)$, in order to remove duplicate literals. Within the setting of [39], these literals would only exist in their fully abstracted form, which can be obtained by replacing every subterm $t : \tau_{\mathbb{Q}}$ that is not a

```
1  fn uwa(s,t)
2  |  eqs ← {s ≈ t}; σ ← ∅; C ← ∅;
3  |  while eqs ≠ ∅
   |  |  ; // (I1)  E ⊨ ⋀ eqs → (s ≈ t)σ ∨ ⋁ C
   |  |  ; // (I2)  (s' ≈ t' ∈ eqs ‖ s' ≉ t' ∈ C) ⟹ s' ⊴ sσ & t' ⊴ tσ
   |  |  ; // (I3)  (s' ≈ t' ∈ eqs ‖ s' ≉ t' ∈ C) ⟹ E ⊨ s' ≉ t' → sσ ≉ tσ
   |  |  ; // (I4)  ∀θ.(E ⊨ sθ ≈ tθ ⟹ ∃ρ.θ ≡_E σρ)
   |  |  ; // (I5)  xσ ≠ x ⟹ x ∉ eqs
4  |  |  ṡ ≈ ṫ ← eqs.pop();
5  |  |  if ṡ ≈ ṫ ∈ {x ≈ u, u ≈ x} for some x ∈ V, x ∉ u
6  |  |  |  σ ← σ{x ↦ u};
7  |  |  |  eqs ← eqs{x ↦ u};
8  |  |  |  C ← C{x ↦ u};
9  |  |  else if canAbstract(ṡ, ṫ)
10 |  |  |  C.push(ṡ ≉ ṫ);
11 |  |  else if ṡ = f(s1 ... sn), ṫ = f(t1 ... tn)
12 |  |  |  eqs.push({s1 ≈ t1 ... sn ≈ tn})
13 |  |  else
14 |  |  |  return ⊥;
15 |  return ⟨σ, C⟩;
```

Algorithm 1: Computing an abstracting unifier uwa.

variable by a fresh variable x, and adding the constraint $x \not\approx t$ to the corresponding clause. Hence, the clause $p(2x) \vee p(x+1)$ is transformed to $p(y) \vee p(z) \vee y \not\approx 2x \vee z \not\approx x+1$ in [39]. Unification then becomes trivial: we would derive the clause $p(y) \vee y \not\approx 2x \vee y \not\approx x+1$ by factoring, from which $p(2x) \vee 2x \not\approx x+1$ is inferred using equality factoring and resolution.

Within unification with abstraction, we aim at cutting out intermediate steps of applying abstractions, equality resolution and factoring. As a result, we skip unnecessary consequences of intermediate clauses, and derive the conclusion of Fig. 2 directly from its first, top-most premise. To this end, we introduce constraints only for those $s, t : \tau_{\mathbb{Q}}$ on which unification fails. We thus gain the advantage that clauses are not present in the search space in their abstracted forms, increasing efficiency in proof search. This is so because abstracted forms of clauses come with a vast number of fresh variables, which makes literals incomparable wrt to simplification orderings within superposition. Further, our unification with abstraction approach is parametrized by a predicate canAbstract to control the application of abstraction, as listed in Algorithm 1. This is yet another significant difference compared to fully abstracted clauses, as in the latter, abstraction is performed for every subterm $t : \tau_{\mathbb{Q}}$ without considering the terms with which t might be later unified.

Full Abstraction. As a first step we will have a look at how completeness using full abstraction instead of unification modulo $\mathcal{E}$ is achieved in [39].

$$\dfrac{\dfrac{\dfrac{p(2x) \vee p(x+1)}{p(y) \vee p(z) \vee y \not\approx 2x \vee z \not\approx x+1}}{p(y) \vee y \not\approx 2x \vee y \not\approx x+1}}{p(2x) \vee 2x \not\approx x+1}$$

Fig. 2. Example of using unification with full abstraction. Unification with abstraction skips the intermediate steps and derives the last clause directly from the first one.

Definition 1 (Full Abstraction). *We say a clause C is* fully abstracted *iff for every term $f(t_1 \dots t_n) \lhd_{\mathcal{E}} C$, then either f is interpreted, or every t_i of sort $\tau_{\mathbb{Q}}$ is a variable.*

Note that every clause can be transformed into an equivalent fully abstracted clause by successively rewriting $C[t] \Rightarrow C[x] \vee x \not\approx t$ for every t that violates full abstraction. We call this transformation *abstracting* a clause.

The calculus presented in [39], has two key properties: Firstly every rule that is applied to fully abstracted clauses, will yield fully abstracted clauses. Secondly every unification preformed in the calculus, does only involve terms without interpreted symbols when if the premises are fully abstracted clauses. Therefore unification modulo $\mathcal{E}$ and syntactic unification are equivalent for these clauses, hence there is no need for a specialized unification algorithm.

Practically speaking this means usual highly efficient term indexing structures can be used, and that there is a single unique most general unifier, instead of a potentially very big set of them. Fully abstracted clauses have a downside too however. That is, they contain a large number of variables, which tends to make literals incomparable, and makes a big number of rule applications possible. Further abstracted clauses tend to have more literals than the non-abstracted ones, which again degrades performance, as more rule applications are possible.

Unification with abstraction. In order to overcome these shortcomings of full abstraction in [33] *Unification with Abstraction* was introduced. The idea of unification with abstraction is to perform abstraction only on demand.

We illustrate this with Fig. 2. The example shows how the initial clause is first being fully abstracted (1), and then processed using the equality factoring rule (2), and afterwards the two literals introduced by abstraction being factored into one (3). Unification with abstraction compresses this sort of derivation introducing the *constraint* $a + b \not\approx b + a$, during unification of $f(a + b)$, and $f(b + a)$, when applying equality factoring to the corresponding literal.

In order to do this, in contrast usual unification modulo $\mathcal{E}$, unification with abstraction, does not compute complete sets of unifiers, but a so-called abstracting unifier; a pair $\langle \sigma, \mathcal{C} \rangle$ of a substitution σ, and a set of constraint literals $\mathcal{C}$. In contrast to the work [33], our definition of an abstracting unifier will independent of the algorithm that computes it, which makes it possible to formalize properties of abstracting unifiers and allow for other, more general versions of

unification with abstraction in the future. Nevertheless we provide an algorithm (Algorithm 1) that is a slight modification of the one in [33], which we use for our experiments (Sect. 6). This concrete implementation of an abstracting unifier will help us to illustrate properties we want from abstracting unifiers. The algorithm performs usual syntactic unification on two terms s, t, but whenever the predicate $\mathsf{canAbstract}$ holds for two subterms s', t', a constraint $s' \not\approx t'$ is introduced instead of continuing unification on these terms. The intuition behind the $\mathsf{canAbstract}$ predicate is that it returns true whenever there is a substitution that can make the two terms equal in the background theory.

Definition 2 (Abstracting Unifier). *Let σ be a substitution and $\mathcal{C}$ a set of literals. A partial function* uwa *that maps two terms s, t either to $\bot$ or to a pair $\langle \sigma, \mathcal{C} \rangle = \mathsf{uwa}(s,t)$ is called an* abstracting unifier.

To ensure that unification with abstraction can replace unification modulo $\mathcal{E}$, we impose the following additional properties over the abstract unifier $\mathsf{uwa}(s,t)$.

Definition 3 (uwa Properties). *Let σ be a substitution and $\mathcal{C}$ a set of literals. Consider $s, t \in \mathbf{T}$ be such that $\mathsf{uwa}(s,t) = \langle \sigma, \mathcal{C} \rangle$ and let θ be an arbitrary ground substituion. We say* uwa *is*

- $\mathcal{E}$-sound *iff* $\mathcal{E} \models (s \approx t)\sigma \vee \mathcal{C}$;
- $\mathcal{E}$-general *iff* $\forall \mu \in \mathsf{mcu}_{\mathcal{E}}(s,t). \exists \rho. \sigma\rho \equiv_{\mathcal{E}} \mu$;
- $\mathcal{E}$-minimal *iff* $\mathcal{E} \models (s \approx t)\sigma\theta \Longrightarrow \mathcal{E} \vDash (\neg \mathcal{C})\theta$;
- subterm-founded *with respect to the clause ordering $\prec$, iff for every uninterpreted function or predicate f, every literal $L[\circ]$, it holds that $\mathcal{E} \models (s \approx t)\theta \Longrightarrow \mathcal{C}\theta \prec L[f(s)]\theta$ or $\mathcal{C}\theta \prec L[f(t)]\theta$.*

Further, uwa *is* $\mathcal{E}$-complete *if, for all $s, t \in \mathbf{T}$ with $\mathsf{uwa}(s,t) = \bot$, we have $\mathsf{mcu}_{\mathcal{E}}(s,t) = \emptyset$.*

Definition 3 is necessary to lift inferences using unification with abstraction. We thereby want to assure that, whenever C does not hold, then s and t are equal; hence abstracting unifiers $\mathsf{uwa}(x,y) = \langle \emptyset, x + y \not\approx y + x \rangle$ would be unsound. The $\mathcal{E}$-generality property enforces that substitutions introduced by uwa are general enough in order to still be turned into a complete set of unifiers. As such, $\mathcal{E}$-generality is needed to rule out cases like $\mathsf{uwa}(x + y, 2) = \langle \{x \mapsto 0, y \mapsto 2\}, \emptyset \rangle$, which would not be able to capture, for example, the substitution $\{x \mapsto 1, y \mapsto 1\}$. We note that we use uwa to extend counterexample-reducing inference systems (see Definition 5), allowing inductive completeness proofs. As these inference systems need to derive conclusions that are smaller than the premises, we need the subterm-foundedness property to make sure to only introduce constraints that are smaller than the premises as well. If we have a look at the previous properties, we see that all of them are fulfilled if $\mathsf{uwa}(s,t) = \bot$. Therefore we need to make sure that uwa only returns $\bot$ when s and t are not unifiable modulo $\mathcal{E}$; this is captured by $\mathcal{E}$-completeness.

In addition to properties of abstract unifiers $\mathsf{uwa}(s,t)$, we also impose conditions over the $\mathsf{canAbstract}$ relation that parametrizes $\mathsf{uwa}(s,t)$. As Algorithm 1

only introduces equality constraints for subterm pairs that should be unified, a resulting abstracting unifier $\mathsf{uwa}(s,t)$ is sound. Further, under the assumption that clause ordering happens as in standard superposition (e.g. using multiset extensions of simplification ordering that fulfill the subterm property), the abstracting unifier $\mathsf{uwa}(s,t)$ is also subterm founded. However, to ensure that $\mathsf{uwa}(s,t)$ is also minimal, interpreted functions should not be treated as uninterpreted ones; hence the canAbstract relation needs to always trigger abstraction on interpreted functions. Finally, imposing that canAbstract does not skip terms potentially equal modulo $\mathcal{E}$ is needed to guarantee that completeness is achieved. In summary, the following two properties are necessary over canAbstract.

Definition 4 (canAbstract **Properties).** *Let* $s,t \in \mathbf{T}$. *The* canAbstract *relation*

- captures $\mathcal{E}$, *iff for all* s, t, *where* $\mathsf{sym}(s)$, *or* $\mathsf{sym}(t)$ *is an interpreted function, it holds that* $\exists\rho.\mathcal{E} \models (s \approx t)\rho \Longrightarrow \mathsf{canAbstract}(s,t)$;
- guards interpreted functions, *iff for all* s, t, *where* $\mathsf{sym}(s) = \mathsf{sym}(t)$ *is an interpreted function,* $\mathsf{canAbstract}(s,t)$ *holds.*

We will now show that the abstracting unifier computed by Algorithm 1 fulfils all the desired properties, if the canAbstract captures $\mathcal{E}$, and guards uninterpreted functions. In order to show this we will need some invariants of the loop in the algorithm. The structure of the full proof is visualized in Figure 3.

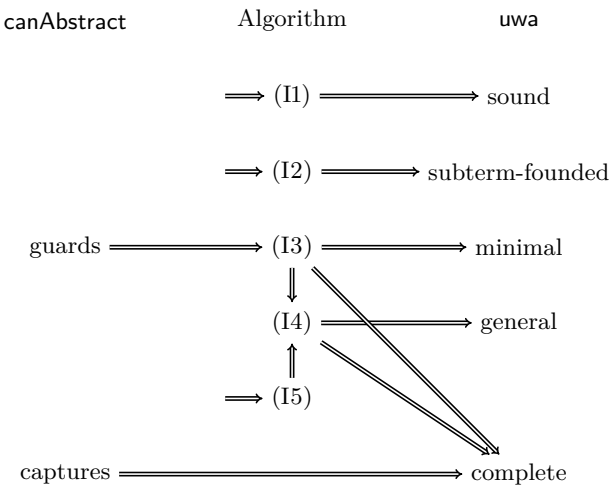

Fig. 3. Implication structure between properties of canAbstract, and the abstracting unifier computed by Algorithm 1.

Lemma 1. *Consider the following invariants.*

(I1) $\mathcal{E} \models \bigwedge eqs \to (s \approx t)\sigma \vee \bigvee \mathcal{C}$

(I2) $(s' \approx t' \in eqs \parallel s' \not\approx t' \in \mathcal{C}) \Longrightarrow s' \trianglelefteq s\sigma \;\&\; t' \trianglelefteq t\sigma$
(I3) $(s' \approx t' \in eqs \parallel s' \not\approx t' \in \mathcal{C}) \Longrightarrow \mathcal{E} \models s' \not\approx t' \rightarrow s\sigma \not\approx t\sigma$
(I4) $\forall\theta.(\mathcal{E} \models s\theta \approx t\theta \Longrightarrow \exists\rho.\theta \equiv_{\mathcal{E}} \sigma\rho)$
(I5) $x\sigma \neq x \Longrightarrow x \notin eqs$

The loop in Algorithm 1 fulfils the invariants (I5)-(I2), and if canAbstract *guards interpreted functions it also fulfils (I3), (I4).*

Proof. First of all, obviously all of the invariants hold when we enter the loop. We'll show that they hold after each loop iteration considering invariant by invariant.

(I1) Does not hold anymore after line 4. After the first branch, it holds again, since

$$\begin{aligned} &\mathcal{E} \models \bigwedge eqs \wedge x \approx u \rightarrow (s \approx t)\sigma \vee \bigvee \mathcal{C} \\ \Longrightarrow\; &\mathcal{E} \models (\bigwedge eqs \wedge x \approx u \rightarrow (s \approx t)\sigma \vee \bigvee \mathcal{C})\{x \mapsto u\} \\ \Longleftrightarrow\; &\mathcal{E} \models \bigwedge eqs \rightarrow (s \approx t)\sigma\{x \mapsto u\} \vee \bigvee \mathcal{C}\{x \mapsto u\} \end{aligned}$$

which is exactly (I1) after that branch. After the second branch (I1) obviously holds again since obviously

$$\begin{aligned} &\mathcal{E} \models \bigwedge eqs \wedge \dot{s} \approx \dot{t} \rightarrow (s \approx t)\sigma \vee \bigvee \mathcal{C} \\ \Longrightarrow\; &\mathcal{E} \models \bigwedge eqs \rightarrow (s \approx t)\sigma \vee \bigvee \mathcal{C} \vee \dot{s} \not\approx \dot{t} \end{aligned}$$

After the third branch it holds as $\models s_1 \approx t_1 \ldots s_n \approx t_n \rightarrow f(s_1 \ldots s_n) \approx f(t_1 \ldots t_n)$.

(I2) Still holds after line 4. In the first branch it is broken by the modification of σ, but fixed again by modifying eqs, and $\mathcal{C}$. After the second branch it holds because $\dot{s} \approx \dot{t}$, was in eqs before. After the third branch it holds because $s_i \triangleleft \dot{s} \trianglelefteq s\sigma$, and $t_i \triangleleft \dot{t} \trianglelefteq t\sigma$.

(I3) Holds after line 4. Again it is broken by modifying σ, but fixed by modifying eqs, and $\mathcal{C}$ in the first branch. After the second branch it holds because $\dot{s} \approx \dot{t}$, was in eqs before. For third branch, if we assume that canAbstract guards interpreted functions, then we know that f must be uninterpreted, hence for every i we have that $\mathcal{E} \models s_i \not\approx t_i \rightarrow \dot{s} \not\approx \dot{t}$, which means that $\mathcal{E} \vDash s_i \not\approx t_i \rightarrow s\sigma \not\approx t\sigma$.

(I4) Holds after line 4. In the first branch the property still holds after modifying σ, since we can reason as follows. Let θ be a substitution.

$$\begin{aligned}
\mathcal{E} \vDash (s \approx t)\theta \Longrightarrow & \mathcal{E} \vDash (s \approx t)\sigma\rho \;\&\; \sigma\rho = \theta && \text{due to (I4)} \\
\Longrightarrow & \mathcal{E} \vDash (\dot{s} \approx \dot{t})\rho && \text{due to (I3)} \\
\Longleftrightarrow & \mathcal{E} \vDash (x \approx u)\rho && \text{due to the if condition}
\end{aligned}$$

$$\rho' := \{y \mapsto y\rho \mid y \in \mathbf{V}, y \neq x\}$$

$$\begin{aligned}
\Longrightarrow & \rho \equiv_{\mathcal{E}} \{x \mapsto u\}\rho' && \text{due to (I5)} \\
\Longrightarrow & \sigma\{x \mapsto u\}\rho' \equiv_{\mathcal{E}} \theta
\end{aligned}$$

In all other branches the property is trivially preserved.

(I5) Still holds after the line 4. In the branch in line 5, it is broken by changing σ, but restored by re-assigning *eqs*. In all other branches it holds trivially.

□

Theorem 1. *The abstracting unifier* uwa *computed by Algorithm 1 is subterm-founded and sound. If* canAbstract *guards interpreted functions, then* uwa *is $\mathcal{E}$-general and $\mathcal{E}$-minimal. If* canAbstract *guards interpreted functions and captures $\mathcal{E}$, then* uwa *is $\mathcal{E}$-complete.*

Proof. Note that we only return a value different from $\bot$ when $eqs = \emptyset$. This means soundness follows from invariant (I1), subterm-foundedness from (I2), minimality from (I3), and generality from (I4).

Let's show $\mathcal{E}$-completeness contrapositively. $\bot$ is only returned in line 14. We know that $\mathsf{canAbstract}(\dot{s}, \dot{t})$ does not hold, and as canAbstract captures $\mathcal{E}$. this means

$$\begin{aligned}
& \nexists\theta.\mathcal{E} \models (\dot{s} \approx \dot{t})\theta \\
\Longrightarrow & \forall\theta.\mathcal{E} \not\models (\dot{s} \approx \dot{t})\theta \\
\Longrightarrow & \forall\theta.\mathcal{E} \not\models (s \approx t)\sigma\theta && \text{due to (I3)} \\
\Longrightarrow & \forall\theta.\mathcal{E} \not\models (s \approx t)\theta && \text{due to (I4)} \\
\Longrightarrow & \nexists\theta.\mathcal{E} \models (s \approx t)\theta
\end{aligned}$$

□

4.2 UWA Completeness

We now show how unification with abstraction (uwa) can be used to replace unification modulo $\mathcal{E}$ in saturation-based theorem proving [3]. We recall from [3] that in order to show refutational completeness of an inference-system Γ, one constructs a *model functor* I that maps sets of ground clauses N to candidate models I_N. In order to show that Γ is refutationally complete, one needs to show that if N is saturated with respect to Γ, then $I_N \vDash N$. For this, the notion of a counterexample-reducing inference system is introduced.

Definition 5 (Counterexample-Reducing Inference System). *We say an inference system Γ is* counterexample reducing*, with respect to a model functor I and a well-founded ordering on ground clauses $\prec$, if for every ground set of clauses N and every minimal $C \in N$ such that $I_N \not\models C$, there is an inference*

$$\frac{C_1 \quad \dots \quad C_n \quad C}{D}$$

where $\forall i. I_N \models C_i$, $\forall i. C_i \prec C$, $D \prec C$, and $I_N \not\models D$.

We then have the following key result.

Theorem 2 (Bachmair&Ganzinger [3]). *Let $\prec$ be a well-founded ordering on ground clauses and I be a model functor. Then, every inference system that is counterexample-reducing wrt $\prec$ and I is refutationally complete.*

This result also holds for an inference system being refutationally complete wrt $\mathcal{E}$ if for every N it holds that $I_N \models \mathcal{E}$.

Proof. Proof by contradiction. We assume that there is a counterexample in N. By well-foundedness of $\prec$ on ground clauses, we know that there must be a least counterexample. As the inference system is counterexample-reducing there must be an even smaller counterexample, which contradicts minimality of the former. This means that there cannot be any counterexample in the first place, which means that $I_N \models N$, which means the inference system is complete wrt. $\mathcal{E}$ as $I_N \models \mathcal{E}$. □

When constructing a refutationally complete calculus, one usually first defines a ground counterexample-reducing inference system and then lifts this calculus to a non-ground inference system. Lifting is done such that, if the ground inference system is counterexample reducing, then its lifted non-ground version is also counterexample reducing.

We next show how to transform a lifting of a counterexample-reducing inference system that uses unification modulo $\mathcal{E}$ into a lifting using unification with abstraction. That is, given a counterexample-reducing inference-system using unification modulo $\mathcal{E}$ to define its rules, we construct another counterexample-reducing inference system that uses uwa instead. As we only transform rules that use unification, we introduce the notion of a unifying rule.

Definition 6 (Unifying Rule). *An inference rule γ is a* unifying rule *if it is of the form*

$$\frac{C_1 \quad \dots \quad C_n \quad C}{D\sigma}, \quad \text{where } \sigma \in \mathsf{mcu}_{\mathcal{E}}(s,t).$$

We also define the mapping $\circ_{\mathsf{uwa}}$ that maps unifying inferences γ to γ_{uwa} as

$$\gamma = \quad \Big(\frac{C_1 \quad \dots \quad C_n \quad C}{D\sigma} \text{ where } \sigma \in \mathsf{mcu}_{\mathcal{E}}(s,t) \Big)$$

$$\Downarrow$$

$$\gamma_{\mathsf{uwa}} = \Big(\frac{C_1 \quad \dots \quad C_n \quad C}{D\sigma \vee \mathcal{C}} \text{ where } \langle \sigma, \mathcal{C} \rangle = \mathsf{uwa}(s,t) \Big)$$

Soundness of the unifying rule γ alone however does not suffice to show soundness of γ_{uwa}. Therefore we introduce a stronger notion of soundness that holds for all the rules we will consider to lift.

Definition 7 (Strong soundness). *Let γ be a unifying rule. We say γ is* strongly sound *iff $\mathcal{E}, C_1 \dots C_n, C \models s \approx t \to D$.*

Note that strong soundness of γ implies soundness, as $\sigma \in \mathsf{mcu}_{\mathcal{E}}(s,t)$ is a unifier of s and t.

Lemma 2. *Assume that γ is strongly sound and* uwa *is sound. Then, γ_{uwa} is sound.*

Proof. Let $M \vDash C_1 \dots C_n, C$. Let $\langle \sigma, \mathcal{C} \rangle = \mathsf{uwa}(s,t)$. Then due to strong soundness of *gamma* we have $M \models s \approx t \to D$, hence $M \models (s \approx t)\sigma \to D\sigma$. Let ξ be some variable interpretation. If $M, \xi \models \mathcal{C}$, then $M, \xi \vDash \mathcal{C} \vee D\sigma$. If $M, \xi \not\models \mathcal{C}$, then by soundness of uwa we know that $M, \xi \models (s \approx t)\sigma$, which in turn means that $M, \xi \models D\sigma$, which in turn means that $M, \xi \vDash \mathcal{C} \vee D\sigma$ in this case as well.

Therefore we know that the rule is sound. □

We note that not every inference can be transformed using $\circ_{\mathsf{uwa}}$, without compromising completeness. One example of a rule that cannot be lifted using uwa is the equality resolution rule in the classic superposition calculus. Consider for example the clause $D \vee a + b \not\approx b + a$, as well as $\mathsf{uwa}(a+b, b+a) = \langle \emptyset, \mathcal{C} \rangle$ and $\mathcal{C} = a + b \not\approx b + a$. Using equality resolution, we would then derive $D \vee \mathcal{C}$ which is the same as the premise. To circumvent this problem, we consider the notion of compatibility with respect to transformations.

Definition 8 (Unifies Strict Subterms). *Let γ be a unifying inference. Then, γ* unifies strict subterms *iff for every grounding θ, $u \in \{s,t\}$ there is an uninterpreted function or predicate f, a literal $L[f(u)]$, and clause $C' \in \{C_1 \dots C_n, C\}$, such that $L[f(u)]\theta \preceq C'\theta$.*

Note that in the above definition we usually have that $L[f(s)]$ or $L[f(t)]$ is some literal of one of the premises.

Definition 9 (uwa-Compatibility). *We say an inference γ is* uwa compatible *if it is a unifying inference, strongly sound, and unifies strict subterms.*

Theorem 3. *Let* uwa *be a general, compatible, subterm-founded, complete, and minimal abstracting unifier. If Γ is the lifting of a counterexample-reducing inference system Γ^ϑ with respect to I and $\prec$, then $\Gamma_{\mathsf{uwa}} = \{\gamma_{\mathsf{uwa}} \mid \gamma \in \Gamma,\ \gamma$ is* uwa*-compatible$\} \cup \{\gamma \in \Gamma \mid\ \gamma$ is not* uwa*-compatible$\}$ is the lifting of an inference system $\Gamma^\vartheta_{\mathsf{uwa}}$ that is counterexample-reducing with respect to I and $\prec$.*

Proof. For the sake of simplicity we assume that all the rules are uwa compatible. The proof can be easily extended to the case where other rules are contained as well, as we do not modify these rules.

Let us define

$$\begin{aligned}\Gamma^\vartheta_{\mathsf{uwa}} = \{\gamma_{\mathsf{uwa}}\theta \mid &\gamma_{\mathsf{uwa}} \in \Gamma_{\mathsf{uwa}},\\ &\theta \text{ is a ground substitution },\\ &\mathcal{E} \models (s \approx t)\theta\}\end{aligned}$$

Obviously $\Gamma^\vartheta_{\mathsf{uwa}}$ is a lifting of Γ_{uwa}, and due to strong soundness of the original inferences the grounded inferences are sound as well. Let's show that $\Gamma^\vartheta_{\mathsf{uwa}}$ is counterexample-reducing.

Let N be a set of clauses, and N^ϑ be the grounding of N. As Γ^ϑ is a counterexample-reducing inference system, we know that for every minimal counterexample $C^\vartheta \in N^\vartheta$ there is an inference $\gamma^\vartheta \in \Gamma^\vartheta$ such that $D^\vartheta \prec C^\vartheta$, and $I^{N^\vartheta} \not\models D^\vartheta$.

$$\gamma^\vartheta = \frac{C_1^\vartheta \quad \dots \quad C_n^\vartheta \quad C^\vartheta}{D^\vartheta}$$

As Γ is a lifting of Γ^ϑ, we know that there must be an inference $\gamma \in \Gamma$, and a ground substitution θ, such that $\gamma\theta = \gamma^\vartheta$. As γ is uwa-compatible, it must be of the following form:

$$\gamma = \frac{C_1 \quad \dots \quad C_n \quad C}{D\sigma} \text{ where } \sigma \in \mathsf{mcu}_{\mathcal{E}}(s,t)$$

Which means that we have

$$\gamma^\vartheta = \gamma\theta = \frac{C_1\theta \quad \dots \quad C_n\theta \quad C\theta}{D\sigma\theta}$$

with $D\sigma\theta \prec C\theta$, $C_i\theta \prec C\theta$, and $I^{N^\vartheta} \not\models D\sigma\theta$.

As uwa is general (Definition 3), and complete (Definition 3), and due to the definition of $\Gamma^\vartheta_{\mathsf{uwa}}$ we know that there is an inference $\gamma^\vartheta_{\mathsf{uwa}} \in \Gamma^\vartheta_{\mathsf{uwa}}$ with $\sigma'\rho \equiv_{\mathcal{E}} \sigma$.

$$\gamma^\vartheta_{\mathsf{uwa}} = \frac{C_1\theta \quad \dots \quad C_n\theta \quad C\theta}{D\sigma\theta \vee \mathcal{C}\theta}$$

$$
\begin{aligned}
&\mathcal{A}_{\mathbb{Q}} = \mathcal{A}_{\mathsf{eq}} \cup \mathcal{A}_{\mathsf{ineq}} && \mathcal{A}_{\mathsf{ineq}} = \{x > y \wedge y > z \to x > y\}\\
&\mathcal{A}_{\mathsf{eq}} = \mathbf{AC} && \quad \cup \{x > y \to x + z > x + z\}\\
&\quad \cup \{jx + kx \approx (j + k)x \mid j, k \in \mathbb{Q}\} && \quad \cup \{x > y \vee x \approx y \vee y > x\}\\
&\quad \cup \{j(k(x)) \approx (jk)x \mid j, k \in \mathbb{Q}\} && \quad \cup \{\neg(x > x)\}\\
&\quad \cup \{1(x) \approx x\} && \quad \cup \{x \geq y \leftrightarrow (x > y \vee x \approx y)\}\\
&\quad \cup \{k(x + y) \approx kx + ky \mid k \in \mathbb{Q}\} && \quad \cup \{x > y \to +kx > +ky \mid +k \in \mathbb{Q}\}\\
&\quad \cup \{x + 0 \approx x, 0x \approx 0\} && \quad \cup \{x > y \to -ky > -kx \mid -k \in \mathbb{Q}\}
\end{aligned}
$$

Fig. 4. Axioms handled by the Alasca calculus. All are implicity universally quantified.

As $\sigma \in \mathsf{mcu}_{\mathcal{E}}(s, t)$, we know that $s\sigma \equiv_{\mathcal{E}} t\sigma$, which means because $\sigma'\rho \equiv_{\mathcal{E}} \sigma$ that $s\sigma'\rho \equiv_{\mathcal{E}} s\sigma'\rho$. This means since uwa is minimal (Definition 3), that $\mathcal{E} \models (\neg\mathcal{C})\rho\theta$, hence $I^{N^\vartheta} \not\models \mathcal{C}\rho\theta$, which means that the conclusion of the rule is a counterexample as well.

What is left to show is that the conclusion is smaller than the main premise. As γ unifies strict subterm (Definition 8), and uwa is subterm-founded (Definition 3), and $C\theta \succ C_i\theta$ we know that $C\theta \succeq L[f(s)]\theta \succ \mathcal{C}\theta$, or $C\theta \succeq L[f(t)]\theta \succ \mathcal{C}\theta$. As $C\theta \succ D\sigma\theta$, this means that $C\theta \succ D\sigma\theta \vee \mathcal{C}\theta$. □

Theorem 1 and Theorem 3 together imply that, given a compatible inference system, we need to only specify the right $\mathsf{canAbstract}$ predicate in order to perform a lifting using uwa. In Section 5 we introduce the calculus Alasca, a concrete inference system with the desired properties, for which a suitable predicate $\mathsf{canAbstract}$ can easily be found.

5 ALASCA Reasoning

We use the lifting results of Section 4 to introduce our Alasca calculus for reasoning in quantified linear arithmetic, by combining superposition reasoning with Fourier-Motzkin type inference rules. While an instance of such a combination has been studied in the Lasca calculus of [25], Lasca is restricted to ground, i.e. quantifier-free, clauses. Our Alasca extends Lasca with uwa and provides an altered ground version Alasca$^\theta$ (Sect. 5.1) which efficiently can be lifted to the quantified domain (Sect. 5.2). As quantified reasoning with linear real arithmetic and uninterpreted functions is inherently incomplete, we provide formal guarantess about what Alasca can prove. Instead of focusing on completeness with respect to $\mathbb{Q}$-models as in [25], we show that Alasca is complete with respect to a partial axiomatisation $\mathcal{A}_{\mathbb{Q}}$ of $\mathbb{Q}$-models (Sect. 5.2).

5.1 The ALASCA Calculus – Ground Version

The Alasca calculus uses a partial axiomatisation $\mathcal{A}_{\mathbb{Q}}$ of $\mathbb{Q}$-models, and handles some $\mathbb{Q}$-axioms via inferences and some via uwa. We therefore split the axiom

set $\mathcal{A}_{\mathbb{Q}}$ into $\mathcal{A}_{\mathsf{eq}}$ and $\mathcal{A}_{\mathsf{ineq}}$, as listed in Figure 4. In order to show our completeness result we will therefore introduce an alternative model functor $\mathcal{I}_{\infty}$, that maps to $\mathcal{A}_{\mathbb{Q}}$-models. We will then show that the ground calculus ALASCA$^{\theta}$, is counterexample-reducing, hence complete for this class of models, and then show how ALASCA$^{\theta}$ can be lifted to a non-ground clauses, while still staying complete for this class of models.

Our ALASCA calculus modifies the LASCA framework [25] to enable an efficient lifting for quantified reasoning. For simplicity, we first present the ground version of ALASCA, which we refer to ALASCA$^{\theta}$, whose one key benefit is illustrated next.

Example 2. One central rule of ALASCA is the Fourier-Motzkin variable elimination rule (FM). We use (FM) in line 7 of Figure 1, when proving the motivating example of Sect. 2, given in formula (1). Namely, using (FM), we derive $-2x - y + sk > 0$ from $f(2x, y) - 2x - y > 0$ and $-f(2, y) + sk \geq 0$, under the assumption that $2x \approx 2$. The (FM) rule can be seen as a version of the inequality chaining rules of [3] for normalized arithmetic terms, chaining the inequalities $sk \geq f(2, y)$ and $f(2x, y) > 2x + y$. Morover, the (FM) rule can also be considered a version of binary resolution, as it resolves the positive summand $f(2x, y)$ with the negative summand $-f(2, y)$, mimicing thus resolution over subterms, instead of literals. The main benefit of (FM) comes with its restricted application to maximal atomic terms in a sum (instead of its naive application whenever possible).

Variable Elimination and (FM). As discussed in Example 2, the (FM) rule of ALASCA$^{\theta}$ is similar to binary resolution, as it can be seen as "resolving" atomic subterms instead of literals. To formalize such handling of terms in (FM), we distinguish so-called $\mathsf{atoms}(t)$, atoms of some term t. Doing so, given a term $t = \frac{1}{k}(\pm_1 k_1 t_1 + \ldots \pm_n k_n t_n)$, we define $\mathsf{atoms}^{\pm}(t) = \langle k, k_1 * \dot{\{} \pm_1 t_1 \dot{\}} \cup \ldots \cup k_n * \dot{\{} \pm_n t_n \dot{\}} \rangle$ and $\mathsf{atoms}(t) = \langle k, k_1 * \dot{\{} t_1 \dot{\}} \cup \ldots \cup k_n * \dot{\{} t_n \dot{\}} \rangle$. We extend both of these functions $f \in \{\mathsf{atoms}, \mathsf{atoms}^{\pm}\}$ to literals as follows: $f(t \diamond 0) = f(t)$, assuming that the term t has been normalised to $\frac{1}{k} = 1$ before. For (dis)equalities $s \approx t$ ($s \not\approx t$) of uninterpreted sorts, we define atoms to be $\langle 1, \dot{\{} s, t \dot{\}} \rangle$. Further we define $\mathsf{maxAtoms}(t)$, to be the set of maximal terms in $\mathsf{atoms}(t)$ with respect $\prec$, and $\mathsf{maxAtom}(t) = t_0$ if $\mathsf{maxAtoms}(t) = \{t_0\}$.

ALASCA$^{\theta}$ Normalization and Orderings. Compared to LASCA [25], the major difference of ALASCA$^{\theta}$ comes with focusing on which terms are being considered equal within inferences; this in turn requires careful adjustements in the underlying orderings and normalization steps of ALASCA$^{\theta}$, and later also in unification within ALASCA. In LASCA terms are rewritten in their so-called $\mathbb{Q}$-normalized form, while equality inference rules exploit equivalence modulo **AC**. Lifting such inference rules is however tricky. Consider for example the application of the rewrite rule $j(ks) \to (jk)s$ (triggered by $j(ks) \approx (jk)s$) over the clause $C[jx, x]$. In order to lift all instances of this rewrite rule, we would need to derive $C[(jk)x, kx]$ for every $k \in \mathbb{Q}$, which would yield an infinite number of conclusions. In order to

resolve this matter, Alasca$^\theta$ takes a different approach to term normalization and handling equivalence. That is, unlike Lasca, we formulate all inference rules using equivalence modulo $\mathcal{A}_{\mathsf{eq}}$, and do not consider the normalization of terms as simplification rules.

As Alasca$^\theta$ rules use equivalence modulo $\mathcal{A}_{\mathsf{eq}}$, we also need to impose that the simplification ordering used by Alasca$^\theta$ is $\mathcal{A}_{\mathsf{eq}}$-compatible[5]. Intuitively, $\mathcal{A}_{\mathsf{eq}}$-compatibility means that terms that are equivalent modulo $\mathcal{A}_{\mathsf{eq}}$ are in one equivalence class wrt the ordering. This allows us to replace terms by an arbitrary normal form wrt these equivalence classes before and after applying any inference rules, allowing it to use a normalization similar to $\mathbb{Q}$-normalization that does not need to be lifted. Hence, we introduce $\mathcal{A}_{\mathsf{eq}}$-*normalized terms* as being terms whose sort is not $\tau_{\mathbb{Q}}$ or of the form $\frac{1}{k}(k_1 t_1 + \cdots + k_n t_n)$, such that $\forall i.k_i \in \mathbb{Z} \setminus 0$, $\forall i \neq j.t_i \not\equiv t_j$, $\forall i.t_i$ is atomic, k is positive, and $\gcd(\{k, k_1 \dots k_n\}) = 1$. Obviously every term can be turned into a $\mathcal{A}_{\mathsf{eq}}$-normalized term. For the rest of this section we assume terms are $\mathcal{A}_{\mathsf{eq}}$-normalized, and write $\equiv$ for $\equiv_{\mathcal{A}_{\mathsf{eq}}}$. We also assume that literals with interpreted predicates $\diamond$ are being normalized (during preprocessing) and to be of the form $t \diamond 0$. We write $s \mathrel{\hat{\approx}} t$ for equalities, with sorts different from $\tau_{\mathbb{Q}}$, and for equalities of sort $\tau_{\mathbb{Q}}$ that can be rewritten to $s \approx t$ such that s is an atomic term. Finally, Alasca$^\theta$ also extends Lasca by not only handling the predicates $>$ and $\approx$, but also $\geq$, and $\not\approx$, which has the advantage that inequalities are not being introduced in purely equational problems in Alasca$^\theta$.

Further, in order to present the calculus more compactly, and to account for our normalization of equalities ($s \approx t \Rightarrow s - t \approx 0$) we merge the rules Gaussian Elimination, and Ordered Paramodulation, that are present in Lasca, into one rule. In order to do this we introduce the following equivalence relation on equality literals: Let $\equiv_{\approx}$ be the least relation such that $L \equiv L' \Longrightarrow L \equiv_{\approx} L'$, $s \approx t \equiv_{\approx} t \approx s$, and $ks + t \approx 0 \equiv_{\approx} s \approx -\frac{1}{k}t$. We write $s \mathrel{\hat{\approx}} t$ to denote some literal L such that $L \equiv_{\approx} s \approx t$.

Alasca$^\theta$ Inferences. The inference rules of Alasca$^\theta$ are summarized in Figure 5. All rules are parametrized by a $\mathcal{A}_{\mathsf{eq}}$-compatible ordering relation $\prec$ on ground terms, literals and clauses. Underlining a literal in a clause or an atomic term in a sum means that the underlined expression is non-strictly maximal wrt to the other literals in the clause, or atomic terms in the sum. We use double-underlining to denote that the expression is strictly maximal.

In the classic superposition calculus rewriting factoring is performed only for positive literals. The reason for this is that only these literals can be productive (a notion we will define later). As we, for example, normalize $\neg s > 0$ to $-s \geq 0$ there are no negative inequalities therefore we need to generalize this notion of potentially productive literals. Therefore we define the set $\mathbf{L}^\theta_+$, called the set of

[5] A formal definition of $\mathcal{A}_{\mathsf{eq}}$-compatibility is given later (See Property 3).

Fourier-Motzkin Elimination

$$\frac{C_1 \vee +j\underline{s} + t_1 \gtrsim_1 0 \qquad C_2 \vee -k\underline{\underline{s'}} + t_2 \gtrsim_2 0}{C_1 \vee C_2 \vee kt_1 + jt_2 > 0}\ (\mathsf{FM})$$

where
- $js + t_1 > 0 \succ C_1$
- $-ks' + t_2 > 0 \succeq C_2$
- $s \equiv s'$
- $\{>\} \subseteq \{\gtrsim_1, \gtrsim_2\} \subseteq \{>, \geq\}$

Tight Fourier-Motzkin Elimination

$$\frac{C_1 \vee +j\underline{s} + t_1 \geq 0 \qquad C_2 \vee -k\underline{\underline{s'}} + t_2 \geq 0}{C_1 \vee C_2 \vee kt_1 + jt_2 > 0 \vee -ks' + t_2 \approx 0}\ (\mathsf{FM}^{\geq})$$

where
- $js + t_1 > 0 \succ C_1$
- $-ks' + t_2 > 0 \succeq C_2$
- $s \equiv s'$

Term Factoring

$$\frac{C \vee j\underline{s} + k\underline{s'} + t \diamond 0}{C \vee (j+k)s' + t \diamond 0}\ (\mathsf{TF})$$

where
- $s \equiv s'$
- $\diamond \in \{>, \geq, \approx, \not\approx\}$
- $s, s' \in \mathsf{maxAtoms}(C \vee js + ks' + t \diamond 0)$
- there is no uninterperted literal C

Inequality Factoring

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

where
- $s \equiv s'$
- $\forall L \in (C \vee js\ + t_1 \gtrsim_1 0).ks' + t_2 \gtrsim_2 0 \succeq L$ or $\forall L \in (C \vee ks' + t_2 \gtrsim_2 0).js\ + t_1 \gtrsim_1 0 \succeq L$
- $\gtrsim_i \in \{>, \geq\}$
- $\gtrsim_3 = \begin{cases} \geq & \text{if } \gtrsim_1 = \geq, \text{ and } \gtrsim_2 = > \\ > & \text{else} \end{cases}$

Contradiction

$$\frac{\cancel{C \vee \pm k \diamond 0}}{C}\ (\mathsf{Triv})$$

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

Superposition

$$\frac{C_1 \vee \underline{s} \,\tilde{\approx}\, t \qquad C_2 \vee L[s']}{C_1 \vee C_2 \vee L[s' \to t]}\ (\mathsf{Sup})$$

where
- $s \equiv s'$
- $s \,\tilde{\approx}\, t \succ C_1$
- $L[s'] \in \mathbf{L}^{\theta}_{+}$ & $L[s'] \succ C_2$ or $L[s'] \notin \mathbf{L}^{\theta}_{+}$ & $L[s'] \succeq C_2$
- $s' \trianglelefteq x \in \mathsf{maxAtoms}(L[s'])$
- $s \approx t \vee C_1 \prec C_2 \vee L[s']$

Equality Resolution

$$\frac{C \vee s \not\approx s'}{C}\ (\mathsf{ER})$$

where
- $s \equiv s'$
- $s \not\approx s' \succeq C$

Equality Factoring

$$\frac{C \vee \underline{s} \,\tilde{\approx}\, t_1 \vee \underline{\underline{s'}} \,\tilde{\approx}\, t_2}{C \vee t_1 \not\approx t_2 \vee s \approx t_1}\ (\mathsf{EF})$$

where
- $s \equiv s'$
- $s' \approx t_2 \succeq C \vee s \approx t_1$

Binary Resolution

$$\frac{C \vee \underline{\underline{P(t_1 \dots t_n)}} \qquad D \vee \underline{\neg P(t'_1 \dots t'_n)}}{C \vee D}\ (\mathsf{BR})$$

where
- $P(t_1 \dots t_n) \equiv P(t'_1 \dots t'_n)$

Factoring

$$\frac{C \vee \underline{P(t_1 \dots t_n)} \vee \underline{P(t'_1 \dots t'_n)}}{C \vee P(t_1 \dots t_n)}\ (\mathsf{F})$$

where
- $P(t_1 \dots t_n) \equiv P(t'_1 \dots t'_n)$

Fig. 5. Rules of the ground calculus ALASCA$^{\theta}$.

Variable Elimination

$$\frac{C \vee \bigvee_{i \in I} x + b_i \gtrsim_i 0 \vee \bigvee_{j \in J} -x + b_j \gtrsim_j 0 \vee \bigvee_{k \in K} x + b_k \approx 0 \vee \bigvee_{l \in L} x + b_l \not\approx 0}{\bigwedge_{K^+ \subseteq K} \left(\begin{array}{lllll} C \vee \bigvee_{i \in I, j \in J} b_i + b_j \gtrsim_{i,j} 0 \vee & \bigvee_{i \in I, k \in K^-} b_i - b_k \geq 0 & \vee & \bigvee_{i \in I, l \in L} b_i - b_l \gtrsim_i 0 \\ \vee & \bigvee_{j \in J, k \in K^+} b_j + b_k \geq 0 & \vee & \bigvee_{j \in J, l \in L} b_j + b_l \gtrsim_j 0 \\ \vee & \bigvee_{k_1 \in K^+, k_2 \in K^-} b_{k_1} - b_{k_2} \geq 0 \vee & & \bigvee_{k \in K^+, l \in L} b_k - b_l \geq 0 \\ & & \vee & \bigvee_{k \in K^-, l \in L} b_l - b_k \geq 0 \\ & & \vee & \bigvee_{\langle l_1, l_2 \rangle \in \mathsf{pairs}(L)} b_{l_1} - b_{l_2} \not\approx 0 \end{array} \right)} \text{(VE)}$$

where

- x is an unshielded variable
- $K^- = K \setminus K^+$
- C does not contain x
- $\gtrsim_i, \gtrsim_j \in \{\geq, >\}$
- $(\gtrsim_{i,j}) = \begin{cases} (\geq) & \text{if } \geq \in \{\gtrsim_i, \gtrsim_j\} \\ (>) & \text{otherwise} \end{cases}$

Fig. 6. Variable elimination rule used for lifting Alasca$^\theta$.

potentially productive literals, as follows.

$$\begin{aligned} \mathbf{L}^\theta_+ =& \{P(t_1 \dots t_n) \mid t_i \in \mathbf{T}^\theta\} \\ & \cup \{s \approx t \mid s, t \in \mathbf{T}^\theta\} \\ & \cup \{+k\underline{s} + t \gtrsim 0 \mid s, t \in \mathbf{T}^\theta, \gtrsim \in \{>, \geq\}\} \end{aligned}$$

Further, the classic superposition calculus is only performs rewrites within maximal argumens of an equality literal. This is done, as only these literals need to be processed by other rules. In our calculus we can restrict this even further for arithmetic terms, only rewriting with maximal atomic terms in their respective sums. Further, for uninterpreted predicates we need to rewrite all of their arguments. Therefore we introduce the helper function active, returning the set of terms that need to be rewritten in a given literal.

$$\mathsf{active}(L) = \begin{cases} \{t_1 \dots t_n\} & \text{if } L \in \{P(t_1 \dots t_n), \neg P(t_1 \dots t_n)\} \\ \mathsf{maxAtoms}(L) & \text{else} \end{cases}$$

Finding a right ordering relation is non-trivial, as many different requirements, like compatibility, subterm property, well-foundedness, and stability under substitutions, need to be met [24, 25, 38, 40]. For Alasca, we use a modified version of the Qkbo ordering of [25], with the following two modifications.

(i) Firstly, the Alasca ordering is defined for non-ground terms. This means that the abstraction function abs (See in the definition of the ordering below)

might be undefined for terms in cases where there is no maximal element in a sum. An example for this are terms like $x + y$. These terms are incomparable to any other term with respect to our ordering. In addition, our ordering needs to be stable under substitutions in order to work with non-ground terms. Note however that our atom functions atoms and $\mathsf{atoms}^{\pm}$ are not stable under substitutions, as the term $f(x) - f(y)$ and the substitution $\{x \mapsto y\}$ demonstrates. Therefore, we parametrize our ALASCA ordering by the relation $\mathsf{subsSafe}$. The $\mathsf{subsSafe}$ relation fulfils the property that if $\mathsf{subsSafe}(\frac{1}{k}(\pm_1 k_1 t_1 + \cdots \pm_n k_n t_n))$, then there is no substitution θ such that $\pm_i k_i t_i \theta \equiv \mp_j k_j t_j \theta$, for any i, j. In general, checking the existence of such a θ is as hard as unifying modulo $\mathcal{A}_{\mathsf{eq}}$. Nevertheless, we can overapproximate the $\mathsf{subsSafe}$ relation using the $\mathsf{canAbstract}$ predicate (see later). For each of the functions atoms, and $\mathsf{atoms}^{\pm}$ we define a respective function $\mathsf{atoms}_{\mathsf{safe}}$, and $\mathsf{atoms}^{\pm}_{\mathsf{safe}}$ that is equal to the respective function on all arguments where $\mathsf{subsSafe}$ holds, and undefined otherwise. We only use this guarded version of the functions in the definition of the ordering in order to ensure that it is stable under substitutions.

(ii) Secondly, we adjusted the ALASCA ordering to be $\mathcal{A}_{\mathsf{eq}}$-compatible, instead of **AC**-compatible. This is done as follows: The case corresponding to 2b in the original QKBO makes sure that two terms that are equivalent modulo **AC** are ordered in the same way, that is they are in one equivalence class wrt to the ordering. As we require $\mathcal{A}_{\mathsf{eq}}$-compatibility (See Property 3), we modified this case such that terms that are equivalent modulo $\mathcal{A}_{\mathsf{eq}}$ are being ordered in the same way.

For defining orderings we introduce the following additional notation. Consider relations $\prec$ and $\equiv$ are relations on the same set, and relation $\prec'$ on a potentially different set. We define $s \prec t \otimes_{\equiv} s' \prec' t'$ iff either $s \prec t$, or $s \equiv t$ and $s' \prec' t'$. We consider $\otimes_{\equiv}$ to be right-associative, and write $\otimes$ for $\otimes_{\equiv_{\mathcal{A}_{\mathsf{eq}}}}$.

Definition 10 (Qkbo). *Let KBO be a Knuth-Bendix Ordering with a weight function w and a precedence relation $\ll$, such that for all terms $t : \tau_{\mathbb{Q}}$ that do not contain a numeral multiplication, or addition, we have $1 \preceq_{KBO} t$.*

We define $s \prec_{QKBO} t$ iff

1. $\mathsf{abs}(s) \prec_{KBO} \mathsf{abs}(t)$, *or*
2. $\mathsf{abs}(s) = \mathsf{abs}(t)$, *and either*
   (a) $s = f(s_1 \dots s_n)$, $t = f(t_1 \dots t_n)$, *and* $\langle s_1 \dots s_n \rangle \prec_{QKBO}^{\mathsf{lex}} \langle t_1 \dots t_n \rangle$ *for some uninterpreted f*
   (b) $\mathsf{atoms}^{\pm}_{\mathsf{safe}}(s) \dot{\prec}^{\mathsf{wmul}} \mathsf{atoms}^{\pm}_{\mathsf{safe}}(t)$

where
- $\mathsf{abs}(x) = x$
- $\mathsf{abs}(f(t_1 \dots t_n)) = f(\mathsf{abs}(t_1) \dots \mathsf{abs}(t_n))$
- $\mathsf{abs}(k_1 t_1 + \dots + k_n t_n) = \max_{\prec_{KBO}}\{\mathsf{abs}(t_1) \dots \mathsf{abs}(t_n)\}$
- $k_1 s \dot{\prec} k_2 t \iff s \prec_{QKBO} t \otimes k_1 \ddot{\prec} k_2$
- $+ \ddot{\prec} -$

The ordering in defined in such a way that if fulfills the following properties.

In the following lemmas let a, a_i, b be arbitrary ground atomic terms, and s, t, s', t', t_i, u be arbitrary ground terms.

Property 1 (Atomic Subterm Property). $a \lhd_{\mathcal{A}_{\mathsf{eq}}} t \Longrightarrow a \prec t$

Property 2 (Totality modulo $\mathcal{A}_{\mathsf{eq}}$). $s \prec t \parallel s \succ t \parallel s \equiv t$

Property 3 ($\mathcal{A}_{\mathsf{eq}}$-compaility). $s' \equiv s \succ t \equiv t' \Longrightarrow s' \succ t'$

Property 4 (Well-founded). There are no infinite descending chains $t_1 \succ t_2 \succ \ldots$.

Property 5 (Context Stability). $a \succ t \Longrightarrow u[a] \succ u[a \mapsto t]$

Property 6 (Product Property). $a \succ b \Longrightarrow ja \succ kb$

Property 7 (Sum Property). $ka \succ k_i a_i \Longrightarrow ka \succ k_1 a_1 + \cdots + k_n a_n$

Property 8 (One Minimality). If $s \neq 0$ and s is of sort $\tau_{\mathbb{Q}}$, then $s \succeq 1$.

Property 9 (Irreflexivity modulo $\mathcal{A}_{\mathsf{eq}}$). $s \equiv s' \Longrightarrow s \not\prec s'$

Property 10 (Transitivity). $s \prec t \prec u \Longrightarrow s \prec u$

Property 11 (Antisymmetry). $s \prec t \Longrightarrow t \not\prec s$

Property 12 (Negative Maximality). $-ja \succ +ka$

Property 13 (Sub-Sum Property). If there is no i such that $a_i \equiv a$, and $\mathbf{sign}(k_i) \neq \mathbf{sign}(k)$, then $ka \prec ka + \sum_{i \in I} k_i a_i$.

Definition 11 (Literal Ordering). *We extend the term ordering $\prec$ to literals as follows. $L_1 \prec L_2$ iff*

1. $\mathsf{sym}(L_1)$ *is interpreted, and* $\mathsf{sym}(L_2)$ *is uninterpreted*
2. $\mathsf{sym}(L_1)$ *and* $\mathsf{sym}(L_2)$ *are uninterpreted, and*

$$\mathsf{sym}(L_1) \ll \mathsf{sym}(L_2) \otimes_{=} \mathsf{args}(L_1) \prec^{\mathsf{lex}} \mathsf{args}(L_2) \otimes \mathsf{pol}(L_1) \prec \mathsf{pol}(L_2)$$

3. $\mathsf{sym}(L_1)$, *and* $\mathsf{sym}(L_2)$ *are interpreted, and*

$$\mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L_1) \dot{\prec}^{\mathsf{wmul}} \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L_2) \otimes L_1 \ddot{\prec} L_2$$

where
- $\mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L) = \langle k, \dot{\{}\langle t, \mathsf{lvl}(L)\rangle \mid t \in T\dot{\}}\rangle$ *where* $\langle k, T\rangle = \mathsf{atoms}_{\mathsf{safe}}(L)$
- $\mathsf{lvl}(s \approx t) = 0$
- $\mathsf{lvl}(s \diamond t) = 1$, *for* $\diamond \in \{>, \geq, \not\approx\}$
- $\langle s, j\rangle \dot{\prec} \langle t, k\rangle$ *iff* $s \prec t \otimes j < k$
- $s \gtrsim_1 0 \ddot{\prec} t \gtrsim_2 0 \Longleftrightarrow s \prec t \otimes (\gtrsim_1) \ll (\gtrsim_2)$ *for* $\gtrsim_i \in \{>, \geq\}$
- $s \approx 0 \ddot{\prec} t \approx 0 \Longleftrightarrow s \not\approx 0 \ddot{\prec} t \not\approx 0 \Longleftrightarrow \mathsf{nf}_{\not\approx}(s) \prec^{\mathsf{mul}} \mathsf{nf}_{\not\approx}(t) \otimes s \prec t$
- $\mathsf{nf}_{\not\approx}(t) = \dot{\{} \sum_{i:k_i \geq 0} k_i t_i, \sum_{i:k_i \leq 0} k_i t_i \dot{\}}$, *where* $\frac{1}{k} \sum k_i t_i = \mathsf{nf}_{\mathsf{eq}}(t)$
- $\mathsf{pol}(\neg P(\ldots)) = -$, $\mathsf{pol}(P(\ldots)) = +$, *and* $+ \prec -$
- $\mathsf{args}(L) = \langle t_1 \ldots t_n\rangle$, *for* $L \in \{P(t_1 \ldots t_n), \neg P(t_1 \ldots t_n)\}$
- $\mathsf{sym}(L) = P$, *for* $L \in \{P(t_1 \ldots t_n), \neg P(t_1 \ldots t_n)\}$

Definition 12 (Clause Ordering). *We extend the literal ordering $\prec$ to clauses C, D as follows.*

$$C \prec D \text{ iff } C \prec^{\mathsf{mul}} D \otimes C \dot{\prec}^{\mathsf{mul}} D$$

where $s \diamond 0 \dot{\prec} t \diamond 0 \iff \mathsf{atoms}_{\mathsf{safe}}(s) \equiv^{\mathsf{wmul}} \mathsf{atoms}_{\mathsf{safe}}(t)$ *for* $\diamond \in \{\approx, \not\approx, >, \geq\}$

In order to establish completeness for our calculus, we will need the following lemmas for our literal ordering. In the following lemmas, we consider L, L', K, K', to be ground literals, a, a', a'' to be a ground atomic term, s, t, t_i, t'_i, u to be a ground term, C to be a ground clause, $\gtrsim_i \in \{>, \geq\}$, and $\diamond \in \{>, \geq, \approx, \not\approx\}$.

Lemma 3 (Totality on Ground Literals). $L \prec L' \parallel L \succ L' \parallel L \equiv L'$

Proof. Assume assuming that $L \not\succ L'$ and $L' \not\succ L$. Then either of the following cases must hold. Both L and L' are uninterpreted, and their symbols and polarites are the same, and their arguments are equivalent, or both are positive or negative equalities $s_1 \diamond s_2$, $t_1 \diamond t_2$, and $\dot{\{}s_1, s_2\dot{\}} \equiv \dot{\{}t_1, t_2\dot{\}}$, which obviously means that $L \equiv L'$, or both are inequalities $s \gtrsim 0$, $t \gtrsim 0$, and $s \equiv t$, which again means that $L \equiv L'$. □

Lemma 4 (Context stability). $a \succ t \Longrightarrow L[a] \succ L[a \mapsto t]$

Proof. If $L[a] = P(\ldots)$, for some uninterpreted predicate P it follows from context stability of terms (Property 5).

Further if $a \succ t$, then obviously $\mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L[a]) \succ^{\mathsf{wmul}} \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L[a \mapsto t])$, which means that the lemma holds for interpreted $L[a]$ as well. □

Lemma 5. *If $t_i \equiv t'_i$ for all i, then $\neg P(t_1 \ldots t_n) \succ P(t'_1 \ldots t'_n)$*

Proof. Follows straight from the definition. □

Lemma 6. *If $\diamond \in \{>, \geq, \approx, \not\approx\}$, and $a \equiv a' \equiv a''$, then $C \vee ja' + ka'' + t \diamond 0 \succ C \vee (j + k)a + t \diamond 0$*

Proof. This follows straight from the definition of the clause ordering. □

Lemma 7. $s \prec t \Longrightarrow s \gtrsim_1 0 \prec t \gtrsim_2 0$

Proof. If $s \prec t$ then it obviously must be the case that $\mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(s \gtrsim_1 0) = \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(s \gtrsim_2 0) \preceq \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(t \gtrsim_2 0)$. Therefore it must be the case that $s \gtrsim_1 0 \prec t \gtrsim_2 0$. □

Lemma 8. $+k\underline{a} + t \gtrsim_1 0 \prec -k\underline{a} - t \gtrsim_2 0$

Proof. Obviously $\mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(+k\underline{a} + t \gtrsim_1 0) = \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(-k\underline{a} - t \gtrsim_2 0)$.

$$\begin{aligned} & +ka \prec -ka && \text{Property 12} \\ \Longrightarrow\ & +ka - t \prec -ka && \text{sum property Property 7} \\ \Longrightarrow\ & +ka - t \prec -ka - t && \text{Property 13} \end{aligned}$$

Hence by the definition of the literal ordering this means that the lemma holds. □

Lemma 9 (Weak $\equiv_\approx$-Compability). $K' \equiv_\approx K \not\equiv_\approx L \equiv_\approx L' \;\&\; K \prec L \Longrightarrow K' \prec L'$

Proof. For uniterpreted L, K this obviously holds.

We observe that $L \equiv L'$ implies that $\mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L) \equiv \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L')$, $\mathsf{nf}_{\mathsf{eq}}(L) \equiv \mathsf{nf}_{\mathsf{eq}}(L')$, and $sym(L) = sym(L')$, and the same for K, K' respectively. From these observations the lemma follows straight away. □

Lemma 10. *Let* $a \in \mathbf{Atomic}^\theta, t_1, t_2 \in \mathbf{T}^\theta$.

$$a \succ t_1 \;\&\; a \succ t_2 \Longrightarrow a \approx t_2 \succ t_1 \not\approx t_2$$

Proof. We observe that due to the subterm property (Property 1), we have $a \succ u \Longrightarrow a \succ u_i$ for $u_i \in \mathsf{atoms}_{\mathsf{safe}}(u)$. Further as $\langle a, 0\rangle \in \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(a \approx t_2)$, and $\mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(t_1 \not\approx t_2) = \dot{\{}\langle u, 1\rangle \mid u \in \mathsf{atoms}_{\mathsf{safe}}(t_1 - t_2)\dot{\}}$, this means that $a \approx u \succ t_1 \not\approx t_2$. □

Lemma 11. $s \succeq t \Longrightarrow \exists s' \in \mathsf{atoms}_{\mathsf{safe}}(s) \forall t' \in \mathsf{atoms}_{\mathsf{safe}}(t).s' \succeq t$

Proof. Proof by contraposition. Assume $\forall s' \in \mathsf{atoms}_{\mathsf{safe}}(s) \exists t' \in \mathsf{atoms}_{\mathsf{safe}}(t).s' \not\succeq t'$. This means by totality (Property 2) that the maximal of all these t' is strictly greater than all s', which means by the sum and the product property (Property 7, Property 6), that $t \succ s$. □

Lemma 12. *If* $L[u]$ *is not a positive equality, then* $u \succeq s \Longrightarrow L[u] \succ s \approx s$

Proof. Suppose $u \succeq s$. If $L[u]$ is uninterpreted the lemma obviously holds. So let's consider the case where $L[u]$ is interpreted.

If u is a subterm of some atomic term u' such that $\dot{\{}u', u'\dot{\}} \in \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L[u])$, then by the subterm property $u' \succeq s$, and furhter $u' \succeq s'$ for any $s' \in \mathsf{atoms}_{\mathsf{safe}}(s)$.

If u is not the subterm of some atomic term then u must be an interpreted term. This means that $\dot{\{}\langle u', 1\rangle \mid u' \in \mathsf{atoms}_{\mathsf{safe}}(u)\dot{\}} \subseteq \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L[u])$. By Lemma 11, we know that there is some u' such that $u' \succeq s'$ for $s' \in \mathsf{atoms}_{\mathsf{safe}}(s)$.

This means in both cases we have $\langle u', 1\rangle \in \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L[u])$, which is obviously bigger than any element of $\mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(s \approx s) = \dot{\{}\langle s', 0\rangle \mid s' \in \mathsf{atoms}_{\mathsf{safe}}(s - s)\dot{\}}$.

Hence $L[u] \succ s \approx s$. □

Lemma 13. $t \geq 0 \succ t \approx 0$

Proof. Follows straight from the defintion. □

Lemma 14. *If* L_1, L_2 *are interpreted, then* $\mathsf{maxAtom}(L_1) \succ \mathsf{maxAtom}(L_2) \Longrightarrow L_1 \succ L_2$

Proof. Follows straight from the defintion. □

Lemma 15. $\mathsf{maxAtoms}(L) = \mathsf{maxAtoms}(\neg L)$

Proof. Follows straight from the defintion. □

Lemma 16. *If L_1, L_2 are interpreted, but not positive equalities, then* $\mathsf{maxAtoms}(L_1) \succ \mathsf{maxAtoms}(L_2) \Longrightarrow L_1 \succ L_2$

Proof. Follows straight from the defintion. □

Lemma 17. $\underline{a} \not\approx t_1 \succ L \succ \underline{a} \approx t_1 \Longrightarrow \exists t_2 \in \mathbf{T}^\theta \Big(L \equiv_\approx \underline{a} \approx t_2 \Big)$

Proof. Obviously L cannot be uninterpreted.

Due to Lemma 14, Lemma 15, and totality on ground terms (Property 2) we know that $\mathsf{maxAtom}(L) = \mathsf{maxAtom}(L') = a' \equiv a$.

If L would be an inequality, then from $L \prec a \approx t_1 \prec \neg L$ it follows that $\mathsf{atoms}_{\mathsf{safe}}(L) \prec \mathsf{atoms}_{\mathsf{safe}}(a \approx t_1) \prec \mathsf{atoms}_{\mathsf{safe}}(\neg L)$. But this is not possible as $\mathsf{atoms}_{\mathsf{safe}}(L) = \mathsf{atoms}_{\mathsf{safe}}(\neg L)$ for inequalities.

If L would be a negative equality then $\neg L \prec L$, which is also not possible.

Therefore we know that L must be a positive equality with $\mathsf{maxAtom}(L) = a'$, hence $L \equiv_\approx a \approx t_2$ with $a \succ t_2$.

□

Lemma 18. $-k\underline{a} - t_1 \gtrsim_1 0 \succ L \succ +k\underline{a} + t_1 \gtrsim_2 0 \Longrightarrow \exists t_2 \in \mathbf{T}^\theta \Big(L \equiv j\underline{a} + t_2 \gtrsim_3 0 \Big)$

Proof. Obviously L cannot be uninterpreted. Further $\mathsf{atoms}_{\mathsf{safe}}(-ka - t_1 \gtrsim_1 0) = \mathsf{atoms}_{\mathsf{safe}}(+ka + t_1 \gtrsim_2 0)$, hence it must be the case $\mathsf{atoms}_{\mathsf{safe}}(+ka + t_1 \gtrsim_2 0) \equiv \mathsf{atoms}_{\mathsf{safe}}(L)$.

This further means that L must be an inequality $L = j_1 a_1 + \ldots + j_n a_n \gtrsim_3 0$. It cannot be the case that $\forall i. a_i \prec a$, because otherwise by the sum property (Property 7), the product property (Property 6), and stability under contexts (Property 5), we would know that $-j_1 a_1 - \ldots - j_n a_n \prec a \prec ka + t_1$, which means by Lemma 7 mean that $\neg L \prec +ka + t_1 \gtrsim_2 0$.

Further it cannot be the case that $\exists i. a_i \succ a$, as otherwise due to the same assumptions we would have that $j_1 a_1 + \ldots + j_n a_n \succ a_i \succ -a - t_1$, which would mean that $L \succ -ka - t_1 \gtrsim_1 0$.

Let $j = \sum_{i:a_i \equiv a} j_i$. It's easy to see then that $L \equiv js + t_2 \gtrsim_3 0$, for some $t_2 \in \mathbf{T}^\theta$. □

Lemma 19. *If $L[a] \not\equiv_\approx a \approx t_1$, then* $\underline{a} \approx t_1 \succ L[a] \Longrightarrow \exists t_2 \in \mathbf{T}^\theta. \Big(L[a] \equiv_\approx \underline{a} \approx t_2 \Big)$

Proof. Obviously $L[a]$ cannot be interpreted.

Suppose $L[a]$ is not a positive equality. Then $\exists u[a]. \langle u[a], 1 \rangle \in \mathsf{atoms}^{\mathsf{lvl}}_{\mathsf{safe}}(L[a])$. This means that $\mathsf{atoms}_{\mathsf{safe}}(a \approx t_2) \prec^{\mathsf{mul}} \mathsf{atoms}_{\mathsf{safe}}(L[a])$, since the maximal element in this multiset is $\langle a, 0 \rangle \prec \langle u[a], 1 \rangle$ due to the subterm property (Property 1), which cannot be the case.

Therefore we know that $L[a]$ is a positive equality. Suppose a occurs as a strict subterm of some uninterpreted term. That would mean that $\exists u[a]. \langle u[a], 0 \rangle \in \mathsf{atoms}_{\mathsf{safe}}(L[a])$ & $a \lhd_{\mathcal{A}_{\mathsf{eq}}} u[a]$ which would again by the subterm property (Property 1)mean that $\mathsf{atoms}_{\mathsf{safe}}(a \approx t_2) \prec \mathsf{atoms}_{\mathsf{safe}}(L[a])$, which is not possible.

Therefore we know that $L[a] \equiv_\approx a \approx t_2$ for some $t_2 \in \mathbf{T}^\theta$ with $a \succ t_2$. □

Model construction. We will now define our model functor $\mathcal{I}^{\circ}_{\infty}$, mapping sets of clauses to what we call the canonical model of the respective set. In this model, as in the classic Bachmair-Ganzinger model construction for regular superposition, the domain will be equivalence classes over the Herbrand universe of the signature. This means that $\tau_{\mathbb{Q}}$ will not be interpreted as $\mathbb{Q}$. Its domain can be rather thought of as some sort of ordered vector spaces over field of rationals. It is to note that $\mathbb{Q}$ is a vector space of this very kind, hence everything we prove with our calculus does indeed hold for $\mathbb{Q}$.

The idea of the model construction is – as in the model construction a lá Bachmair & Ganzinger – to first build a rewrite system from a set of ground clauses, and then to build a model, which has the normal forms wrt to this rewrite system as its domain. We will be reasoning modulo an equational background theory $\mathcal{A}_{\mathsf{eq}}$, so our rewrite system as well as the domain of our model will use equivalence classes modulo $\mathcal{A}_{\mathsf{eq}}$ instead of plain terms t. On the semantics side we will just write t for the equivalence class of t modulo $\mathcal{A}_{\mathsf{eq}}$ in order to keep our definitions more readable. As we do not only want to interpret equality, but also, inequalities and uninterpreted predicates, we need to collect the data to interpret the other predicates in parallel with the data to interpret equality (i.e. the rewrite system).

The model construction is structured as follows. First we will build a sequence of sets of so-called *atomic assertions*, based on the clause ordering $\prec$. Then we will define the actual model as the limit of this sequence.

Definition 13. *Let s, t, $t_1, \dots t_n$ be equivalence classes of ground terms terms modulo $\mathcal{A}_{\mathsf{eq}}$, and P be a predicate symbol, and s be atomic, and $s \succ t$. We define an* atomic assertion *is one of the following:*

- *a rewrite rule $s \to t$*
- *an atom $P(t_1 \dots t_n)$*
- *a lower bound $s > t$*

Definition 14. *Let $\mathcal{I}$ be a set of atomic assertions. We define*

$$\mathcal{I}^{\approx} = \{s \to t \in \mathcal{I}\}$$
$$\mathcal{I}^{P} = \{P(t_1 \dots t_n) \in \mathcal{I} \mid P \text{ is an uninterpreted predicate symbol.}\}$$
$$\mathcal{I}^{>} = \{s > t \in \mathcal{I}\}$$

Each set of atomic assertions is meant to represent an $\mathcal{A}_{\mathbb{Q}}$-model. Therefore we need to define how to turn a $\mathcal{A}_{\mathbb{Q}}$-model into a first-order structure. We will call this first-order structure an $\mathcal{A}_{\mathbb{Q}}$-structure. Its domain are equivalence classes of terms; more specifically tall the normal forms of $\mathcal{I}^{\approx}$ modulo $\mathcal{A}_{\mathsf{eq}}$. In that way we make sure that our $\mathcal{A}_{\mathbb{Q}}$-model actually models $\mathcal{A}_{\mathsf{eq}}$. We will denote the normal form of a term t wrt the rewritesystem R by $t \downarrow_R$.

Definition 15. *Let $\mathcal{I}$ be a set of atomic assertions. We call $\mathcal{I}$ an* $\mathcal{A}_{\mathbb{Q}}$-structure *if for all for all $s \to t \in \mathcal{I}$ we have that s is irreducible in $\mathcal{I}^{\approx} \setminus \{s \to t\}$,for all $P(t_1 \dots t_n) \in \mathcal{I}, t_1 \dots t_n$ are irreducible in $\mathcal{I}^{\approx}$,and for all $s > t \in \mathcal{I}^{\approx}$, s is irreducible in $\mathcal{I}^{\approx}$.*

Definition 16. *Let $\mathcal{I}$ be an $\mathcal{A}_{\mathbb{Q}}$-structure.*

Then we identify $\mathcal{I}$ with the first-order structure interpreting symbols as follows:

$$\mathcal{I}(\tau) = \{t \downarrow_{\mathcal{I}^{\approx}} \mid t \in \mathbf{T}, t : \tau\}$$
$$\mathcal{I}(t) = t \downarrow_{\mathcal{I}^{\approx}}$$

$$\mathcal{I}(P)(t_1 \dots t_n) \iff P(t_1 \dots t_n) \in \mathcal{I}^P$$
$$\mathcal{I}(\geq 0)(t) \iff (\mathcal{I}(> 0)(t) \parallel \mathcal{I}(t) = 0)$$
$$\mathcal{I}(> 0)(t) \iff \begin{cases} \mathbb{Q} \vDash t > 0 & \text{if } t \in \mathbb{Q} \\ \exists s > l \in \mathcal{I}^{>}. \mathcal{I} \models kl + t' \geq 0 & \text{if } t = +k\underline{s} + t' \\ \mathcal{I} \not\models +k\underline{s} - t' \geq 0 & \text{if } t = -k\underline{s} + t' \end{cases}$$

Lemma 20. *If $\mathcal{I}$ is an $\mathcal{A}_{\mathbb{Q}}$-structure, then $\mathcal{I} \models \mathcal{A}_{\mathbb{Q}}$.*

Proof. As a first step we need to show that our term-interpretation is well-defined. That is, we need to show that the rewrite system $\mathcal{I}^{\approx}$ is confluent, and terminating. The rewrite system is obviously terminating as for every rewrite rule $s \to t \in \mathcal{I}^{\approx}$, we have that $s \succ t$, and by Property 4, s is well-founded.

Let us now show local confluence.

Let $s_1 \to t_1, s_2 \to t_2 \in \mathcal{I}$, and u be a term that can be rewritten by both of the rules. By definition of an $\mathcal{A}_{\mathbb{Q}}$-structure, we know that it cannot be the case that $s_1 \trianglelefteq_{\mathcal{A}_{\mathsf{eq}}} s_2$, or $s_2 \trianglelefteq_{\mathcal{A}_{\mathsf{eq}}} s_1$. From this it is easy to see that $u[s_1 \to t_1][s_2 \to t_2] \equiv u[s_2 \to t_2][s_1 \to t_1]$.

All axioms in $\mathcal{A}_{\mathsf{eq}}$ obviously hold as our domain is terms modulo $\mathcal{A}_{\mathsf{eq}}$.

So let us have a look at the other axioms.

We will first show transitivity holds. First we normalize the axiom to the clause $\neg y - x > 0 \vee \neg z - y > 0 \vee z - x > 0$. The clause holds, iff it holds for every $\mathcal{I}(x), \mathcal{I}(y), \mathcal{I}(z)$. We proof transitivity holds by induction on the term ordering $\prec$.

Let $I(x) = j_x \underline{\underline{s_x}} + \hat{s}_x$, $I(y) = j_y \underline{\underline{s_y}} + \hat{s}_y$, and $I(z) = j_z \underline{\underline{s_z}} + \hat{s}_z$.

Suppose transitivity does not hold for these. That would mean $\mathcal{I} \models I(y) - I(x) > 0 \wedge I(z) - I(y) > 0 \wedge I(x) - I(z) \geq 0$. If $I(x) - I(z) \approx 0$, then we would have $I(y) - I(x) > 0 \wedge I(x) - I(y) > 0$, which is a contradiction. This means that we have $\mathcal{I} \models I(y) - I(x) > 0 \wedge I(z) - I(y) > 0 \wedge I(x) - I(z) > 0$.

Let us now show the following lemma.

$$\mathcal{I} \models (j_1 \underline{\underline{s_1}} + \hat{s}_1) - (j_2 \underline{\underline{s_2}} + \hat{s}_2) > 0 \qquad (A)$$
$$\&\mathcal{I} \models (j_2 \underline{\underline{s_2}} + \hat{s}_2) - (j_3 \underline{\underline{s_3}} + \hat{s}_3) > 0 \qquad (B)$$
$$\&\mathcal{I} \models (j_3 \underline{\underline{s_3}} + \hat{s}_3) - (j_1 \underline{\underline{s_1}} + \hat{s}_1) > 0 \qquad (C)$$
$$\Longrightarrow (s_3 \succ s_1 \Longrightarrow s_2 \succ s_3)$$

This can be shown as follows: Assume $s_3 \succ s_1$, and $s_3 \succ s_2$

case $j_3 > 0$**:**

$$\begin{aligned}
(C) &\iff \exists s_3 > l \in \mathcal{I}.\mathcal{I} \models j_3 l + \hat{s}_3 - (j_1 s_1 + \hat{s}_1) \geq 0 \\
&\implies \exists s_3 > l \in \mathcal{I}.\mathcal{I} \models j_3 l + \hat{s}_3 - (j_2 s_2 + \hat{s}_2) \geq 0 \quad \text{by I.H. and (A)} \\
&\iff \mathcal{I} \models j_3 s_3 + \hat{s}_3 - (j_2 s_2 + \hat{s}_2) > 0
\end{aligned}$$

Which contradicts (B). ⊣

case $j_3 < 0$**:**

$$\begin{aligned}
(C) &\iff \mathcal{I} \not\models -j_3 \underline{\underline{s_3}} - \hat{s}_3 + j_1 s_1 + \hat{s}_1 \geq 0 \\
&\implies \mathcal{I} \not\models -j_3 \underline{\underline{s_3}} - \hat{s}_3 + j_1 s_1 + \hat{s}_1 > 0 \\
&\iff \nexists s_3 > l \in \mathcal{I}.\mathcal{I} \models -j_3 l - \hat{s}_3 + j_1 s_1 + \hat{s}_1 > 0 \\
&\iff \forall s_3 > l \in \mathcal{I}.\mathcal{I} \models j_3 l + \hat{s}_3 - j_1 s_1 - \hat{s}_1 \geq 0 \\
&\implies \forall s_3 > l \in \mathcal{I}.\mathcal{I} \models j_3 l + \hat{s}_3 - j_2 s_2 - \hat{s}_2 \geq 0 \quad \text{by I.H. and (A)} \\
&\iff \mathcal{I} \not\models -j_3 s_3 - \hat{s}_3 + j_2 s_2 + \hat{s}_2 > 0
\end{aligned}$$

Which contradicts (B). ⊣

As we have

$$\mathcal{I} \models (j_x \underline{\underline{s_x}} + \hat{s}_x) - (j_y \underline{\underline{s_y}} + \hat{s}_y) > 0$$
$$\mathcal{I} \models (j_y \underline{\underline{s_y}} + \hat{s}_y) - (j_z \underline{\underline{s_z}} + \hat{s}_z) > 0$$
$$\mathcal{I} \models (j_z \underline{\underline{s_z}} + \hat{s}_z) - (j_x \underline{\underline{s_x}} + \hat{s}_x) > 0$$

which means by the lemma, that we get

$$s_x \succ s_y \Longrightarrow s_z \succ s_x \Longrightarrow s_y \succ s_z$$
$$s_y \succ s_x \Longrightarrow s_z \succ s_y \Longrightarrow s_x \succ s_z$$

Both of these implication chains are contradictory, hence transitivity must hold.

For all other axioms one can easily see that they follow from the definition of a $\mathcal{A}_{\mathbb{Q}}$ model by simple definition unfolding.

Definition 17 (Canonical Model). *Let N be a ground set of clauses, and $C \in N$, then we call $\mathcal{I}^N_{\prec C}$ the* $\mathcal{A}_{\mathbb{Q}}$-model up to C wrt. N, *$\mathcal{I}^N_C$ the* $\mathcal{A}_{\mathbb{Q}}$-model for C wrt. N, *and $\mathcal{I}^N_\infty$ the* canonical model wrt. N.

$$\begin{aligned}\mathcal{I}^N_\infty &= \bigcup_{C \in N} \mathcal{I}^N_C \\ \mathcal{I}^N_C &= \mathcal{I}^N_{\prec C} \cup \mathsf{produce}^N(C) \\ \mathcal{I}^N_{\prec C} &= \bigcup_{D \in N, D \prec C} \mathcal{I}^N_D\end{aligned}$$

$$\mathsf{produce}^N(C' \vee \underline{\underline{L}}) = \begin{cases} \emptyset & \text{if } \mathcal{I}^N_{\prec C} \models C' \vee L \\ \emptyset & \text{else if } t \in \mathsf{active}(L) \text{ that is reducible in } \mathcal{I}^N_{\prec C} \\ \{s > -\frac{1}{k}t\} & \text{else if } L = +k\underline{\underline{s}} + t \gtrsim 0, \text{ where } \gtrsim \in \{>, \geq\} \\ & \quad \& \ \forall k_1 t_1 + \ldots + k_n t_n \gtrsim' 0 \in C'. \mathbb{Q} \models \Sigma_{i:t_i \equiv s} k_i \leq 0 \\ \{s \to t\} & \text{else if } L \equiv_{\approx} \underline{\underline{s}} \approx t \\ & \quad \& \ \forall s \hat{\approx} u \in C'. \mathcal{I}^N_{\prec C} \models t \not\approx u \\ \{P(t_1, \ldots, t_n)\} & \text{else if } L = P(t_1 \ldots t_n) \\ \emptyset & \text{else} \end{cases}$$

We say a clause $C = C' \vee \underline{\underline{L}}$ is productive for L iff $\mathsf{produce}^N(C) \neq \emptyset$.

Completeness. In order to show that our Alasca$^\theta$ is refutationally complete, by using the framework mentioned in Section 4, we need to establish that our calculus reduces counterexamples. For that we will need to establish some lemmas first.

First of all we need to make sure that every of our potential models can indeed be used to interpret clauses.

Lemma 21. *For every set of clauses N, $\mathcal{I}^N_C$, $\mathcal{I}^N_{\prec C}$, and $\mathcal{I}^N_\infty$ are $\mathcal{A}_{\mathbb{Q}}$-structures.*

Proof. Follows straight from the definition. □

Next we observe that the interpretation of equality of some term stabilizes as soon as a literal greater than the term itself is processed in the model construction. Formally we establish this as the following lemma.

Lemma 22 (Equality Stabilisation).

$$C \succeq s \approx s \Longrightarrow \mathcal{I}^N_\infty(s) = \mathcal{I}^N_{\prec C}(s) = \mathcal{I}^N_C(s)$$

Proof. Firstly as $\mathcal{I}^N_{\prec C} \subseteq \mathcal{I}^N_C \subseteq \mathcal{I}^N_\infty$, any rule used to rewrite s in $\mathcal{I}^N_{\prec C}$ will also be in $\mathcal{I}^N_\infty$.

Let $u \to v \in \mathcal{I}^N_\infty$. If $u \to v$ is used in a rewriting of s, then there must be some atomic subterm $s' \trianglelefteq s$, such that $s' \succeq u$, which means by the subterm property that $s \succeq u$. Then there is some productive clause $D = D' \vee \underline{\underline{u \mathrel{\hat{\approx}} v}}$. Since $u \succ v$, we have $u \approx u \succ u \approx v$. Further by $\mathcal{A}_{\mathsf{eq}}$-compatibility (Property 3), and well-foundedness (Property 4) we know that $u \not\equiv_{\mathcal{A}_{\mathsf{eq}}} v$. Therefore by $\equiv_\approx$-compatibility (Lemma 9), this means that $D' \prec u \mathrel{\hat{\approx}} v \prec u \approx u \preceq s \approx s \preceq C$. Which further means that $D \prec C$, hence that $u \to v \in \mathcal{I}^N_{\prec C}$ as well. □

Lemma 23. *Let $L[u]$ be a ground literal that is not a positive equality*

$$C \succeq L[u] \;\&\; u \succeq s \Longrightarrow \mathcal{I}^N_\infty(s) = \mathcal{I}^N_{\prec C}(s) = \mathcal{I}^N_C(s)$$

Proof. This follows straight from Lemma 22, and Lemma 12. □

Lemma 24 (Monotonicity). *Let $L \in \mathbf{L}^\theta, C \in N$.*

$$L \preceq C \;\&\; \mathcal{I}^N_C \models L \Longrightarrow \mathcal{I}^N_\infty \models L$$

$$L \preceq C \;\&\; \mathcal{I}^N_{\prec C} \models L \Longrightarrow \mathcal{I}^N_\infty \models L$$

Proof. We proof by induction on L, and C along $\prec$.

Let's case split on L

case $s \approx t$: For $\mathcal{I} \in \{\mathcal{I}^N_{\prec C}, \mathcal{I}^N_C\}$ we have $\mathcal{I} \subset \mathcal{I}^N_\infty$, which means that if s, and t rewrite to the same term in $\mathcal{I}$, then they also do so in $\mathcal{I}^N_\infty$.

case $s \not\approx t$: By Lemma 23 we have that $\mathcal{I}^N_\infty(s) = \mathcal{I}^N_{\prec C}(s) = \mathcal{I}^N_C(s)$, and $\mathcal{I}^N_\infty(t) = \mathcal{I}^N_{\prec C}(t) = \mathcal{I}^N_C(t)$. This means both $\mathcal{I}^N_C \models s \not\approx t$, and $\mathcal{I}^N_{\prec C} \models s \not\approx t$ imply, that $\mathcal{I}^N_\infty \models s \not\approx t$.

case $t > 0$: By Lemma 23 we have that $\mathcal{I}^N_\infty(t) = \mathcal{I}^N_{\prec C}(t) = \mathcal{I}^N_C(t)$. We case split on $\mathcal{I}^N_\infty(t)$.

case k: This means by definition that if $\mathcal{I}^N_\infty \models t > 0 \Longleftrightarrow \mathbb{Q} \models k > 0 \Longleftrightarrow \mathcal{I}^N_C \models t > 0 \Longleftrightarrow \mathcal{I}^N_{\prec C} \models t > 0$.

case $+k\underline{\underline{s}} + t'$: Note that since $+ks + t' = \mathcal{I}^N_\infty(t) = \mathcal{I}^N_{\prec C}(t) = \mathcal{I}^N_C(t)$, it must be the case that $s = \mathcal{I}^N_\infty(s) = \mathcal{I}^N_{\prec C}(s) = \mathcal{I}^N_C(s)$. Further note that by construction for every $s > l \in \mathcal{I}^N_C \supseteq \mathcal{I}^N_{\prec C}$, $s \succ l$, and $s \in \mathbf{Atomic}^\theta$. This means that $\mathsf{maxAtom}(s) \succ \mathsf{maxAtom}(l)$, and $\mathsf{maxAtom}(s) \succ \mathsf{maxAtom}(t')$, which means that we have that $+ks + t' > 0 \succ +kl + t' \geq 0$. This means we can reason as follows.

$$\begin{aligned}
\mathcal{I} \models t > 0 \quad &\Longleftrightarrow \quad \exists s > l \in \mathcal{I}.\mathcal{I} \models kl + t' \geq 0 \\
&\Longrightarrow \quad \exists s > l \in \mathcal{I}^N_\infty.\mathcal{I} \models kl + t' \geq 0 \quad (\mathcal{I} \subseteq \mathcal{I}^N_\infty) \\
&\Longrightarrow \quad \exists s > l \in \mathcal{I}^N_\infty.\mathcal{I}^N_\infty \models kl + t' \geq 0 \quad \text{(by induction hypothesis)} \\
&\Longleftrightarrow \quad \mathcal{I}^N_\infty \models t > 0
\end{aligned}$$

case $-k\underline{s} + t'$**:** Suppose $\mathcal{I}^N_\infty \not\models t > 0$. This is the cases iff $\mathcal{I}^N_\infty \models +ks - t' \geq 0$.

if $\mathcal{I}^N_\infty \models +ks - t' > 0$**:** That means there is a $s' > l \in \mathcal{I}^N_\infty$ such that $\mathcal{I}^N_\infty \models kl - t' \geq 0$, and $s \equiv s'$. This means that there must be a productive clause $D = D' \vee j\underline{s'} - jl \gtrsim 0$.

$$
\begin{array}{rrll}
 & js \prec -ks & & \text{(Property 12)} \\
\implies & js - jl \prec -ks & & \text{(Property 7)} \\
\implies & js - jl \prec -ks + t' & & \text{(Property 13)} \\
\implies & js - jl \gtrsim 0 \prec -ks + t' > 0 & & \text{(Lemma 7)} \\
\implies & D' \vee js - jl \gtrsim 0 \prec -ks + t' > 0 & & \\
\implies & D' \vee js - jl \gtrsim 0 \prec t > 0 & & \text{(Lemma 7)} \\
\implies & D' \vee js - jl \gtrsim 0 \prec t > 0 \vee C' & &
\end{array}
$$

Hence by construction it must be the case that $s > l \in \mathcal{I}^N_{\prec C} \subset \mathcal{I}^N_C$. We know that $\mathcal{I}^N_\infty \models kl - t' \geq 0 \iff \mathcal{I}^N_\infty \not\models -kl + t' > 0$. As $s \succ l$, and $s \succ -t'$, by the sum property (Property 7), and Lemma 7 we know that $-kl + t' > 0 \prec L$, which means that by induction hypothesis for $\mathcal{I} \in \{\mathcal{I}^N_{\prec C}, \mathcal{I}^N_C\}$ we have $\mathcal{I} \not\models -kl + t' > 0$, hence $\mathcal{I} \models kl - t' \geq 0$. Combining that we get that $\mathcal{I} \models -ks + t' > 0$.

else: That means that $\mathcal{I}^N_\infty(t) = 0$, which means that $\mathcal{I}^N_C(t) = 0$ as well.

case $t \geq 0$**:** For $\mathcal{I} \in \{\mathcal{I}^N_{\prec C}, \mathcal{I}^N_C\}$, we can reason as follows. This is the case iff $\mathcal{I} \models t > 0$, or $\mathcal{I} \models t \approx 0$

If $\mathcal{I} \models t > 0$, then we can reason as in the case where $L = t > 0$ to establish that $\mathcal{I}^N_\infty \models t > 0$.

Otherwise if $\mathcal{I} \models t \approx 0$, by Lemma 23, we know that $\mathcal{I}(t) = \mathcal{I}^N_\infty(t)$, which means that $\mathcal{I}^N_\infty \models t \approx 0$ as well.

case $P(t_1 \dots t_n)$**:** For both $\mathcal{I} \in \{\mathcal{I}^N_{\prec C}, \mathcal{I}^N_C\}$, we can reason as follows.

$$
\begin{array}{rrll}
\mathcal{I} \models P(t_1 \dots t_n) & \iff & P(\mathcal{I}(t_1) \dots \mathcal{I}(t_n)) \in \mathcal{I} & \\
 & \implies & P(\mathcal{I}(t_1) \dots \mathcal{I}(t_n)) \in \mathcal{I}^N_\infty & (\text{as } \mathcal{I} \subseteq \mathcal{I}^N_\infty) \\
 & \implies & P(\mathcal{I}^N_\infty(t_1) \dots \mathcal{I}^N_\infty(t_n)) \in \mathcal{I}^N_\infty & (\text{due to Lemma 23}) \\
 & \iff & \mathcal{I}^N_\infty \models P(t_1 \dots t_n) &
\end{array}
$$

case $\neg P(t_1 \dots t_n)$**:** By Lemma 23 we know that $\mathcal{I}^N_\infty(t_i) = \mathcal{I}^N_C(t_i) = \mathcal{I}^N_{\prec C}(t_i)$. Suppose $P(\mathcal{I}^N_\infty(t_1) \dots \mathcal{I}^N_\infty(t_n)) \in \mathcal{I}^N_\infty$. Then there is some productive clause $D = D' \vee P(t'_1 \dots t'_n)$ with $\mathcal{I}^N_\infty(t'_i) = \mathcal{I}^N_\infty(t_i)$. As D is productive $t'_i = \mathcal{I}^N_D(t'_i)$, and by Lemma 23 $t'_i = \mathcal{I}^N_\infty(t'_i) = \mathcal{I}^N_\infty(t_i)$. This means as we only rewrite to smaller things that $P(t_1 \dots t_n) \succeq P(t'_1 \dots t_n)$. Together with Lemma 5 this means that $D \prec C$, hence $P(t'_1 \dots t'_n) \in \mathcal{I}^N_{\prec C} \subseteq \mathcal{I}^N_C$ as well.

Lemma 25. *If $C = C' \vee \underline{L}$ is a productive clause, then for $\mathcal{I} \in \{\mathcal{I}^N_\infty, \mathcal{I}^N_C\}$. $\mathcal{I} \models L$ & $\mathcal{I} \not\models C'$.*

Proof. Obviously due to Lemma 24, we know that if the lemma holds for $\mathcal{I} = \mathcal{I}_C^N$, then it will also hold for $\mathcal{I} = \mathcal{I}_\infty^N$. Hence we only need to have a look at the case where $\mathcal{I} = \mathcal{I}_C^N$.

We first show that $\mathcal{I}_C^N \models L$ by case splitting on $\mathsf{produce}^N(L)$

case $s \to t$**:** That means that $L = s \mathrel{\hat{\approx}} t$, which obviously holds in $\mathcal{I}_C^N$.
case $s > l$**:** That means that $L = js - jl \gtrsim 0$, which obviously holds in $\mathcal{I}_C^N$, since $jl - jl \geq 0$ holds in $\mathcal{I}_C^N$.
case $P(t_1 \dots t_n)$**:** That means that $L = P(t_1 \dots t_n)$, which obviously holds in $\mathcal{I}_C^N$.

Next we show that $\mathcal{I}_C^N \not\models C'$. Let L' be any literal of C'. As C is productive we know that $L' \prec L$, and that $\mathcal{I}_{\prec C}^N \not\models L'$. If $\neg L' \prec L$ then by Lemma 24 we know that $\mathcal{I}_\infty^N \models \neg L'$, which means $\mathcal{I}_\infty^N \not\models L'$, which means by the same lemma that $\mathcal{I}_C^N \not\models L'$, hence $\mathcal{I}_C^N \models \neg L'$.

Hence by Lemma 3 we can assume that $L' \prec L \preceq \neg L'$.

We case split on L'.

case $\underline{s} \not\approx t$**, or** $\neg P(t_0 \dots t_n)$**:** Not possible since then it would be the case that $\neg L' \prec L'$. ⊣
case $s \mathrel{\hat{\approx}} t_1$**:** Since C is productive we know that it cannot be the case that $L \equiv_\approx s \mathrel{\hat{\approx}} t_1$, due to the side condition of $\mathsf{produce}^N(C)$.
As $\neg L' \succ L \succ L'$, this means by weak compatibility (Lemma 9), that $s \not\approx t_1 \succ L \succ s \approx t_1$, which further means by Lemma 17 that $L \equiv_\approx \underline{\underline{s}} \mathrel{\hat{\approx}} t_2$.

As C is productive this means by definition that $\mathcal{I}_{\prec C}^N \models t_1 \not\approx t_2$, hence by Lemma 23 we know that $\mathcal{I}_C^N \models t_1 \not\approx t_2$. Futher as $\mathcal{I}_C^N \models L \equiv_\approx s \approx t_2$, this means that $\mathcal{I}_C^N \not\models s \approx t_1$.
case $t \gtrsim 0$**:** Obvisously $t \gtrsim 0 \equiv k\underline{\underline{s}} + t' \gtrsim 0$.

if $k < 0$**:** Then by Lemma 8, and weak compatibility (Lemma 9), it would be the case $\neg L' \prec L'$, which contradicts $L' \prec L \prec \neg L'$. ⊣
else $k > 0$**:** It cannot be the case that $L \equiv L'$ due to the side condition of $\mathsf{produce}^N C$.
Hence by $t \gtrsim 0 \prec L \prec \neg(t \gtrsim 0)$, and weak compatibility (Lemma 9), we know that $ks + t' \gtrsim 0 \prec L \prec -ks - t' \gtrsim' 0$. By Lemma 18 we therefore know that $L \equiv js + t_2 \gtrsim'' 0$. As C is productive this means by the same side condition of the definition of $\mathsf{produce}^N(C)$ that this cannot be the case. ⊣
case $P(t_1 \dots t_n)$**:** We know $\mathcal{I}_{\prec C}^N \not\models P(t_1 \dots t_n)$. Suppose $\mathcal{I}_C^N \models P(t_1 \dots t_n)$. Then it must be the case that C is productive with $L = P(t'_1 \dots t'_n)$, such that $t'_i \equiv t_i$. This would mean that L is not strictly maximal though, which would mean that C is not productive. ⊣

Theorem 4. *Alasca$^\theta$ is a counterexample-reducing inference system with respect to $\mathcal{I}_\infty$ and $\prec$.*

Proof. Let $C = C' \vee \underline{L}$ be a minimal counterexample that is not the empty clause.

if C is productive: Then by Lemma 25, we know that $\mathcal{I}^N_\infty \models C$, hence C cannot be a counterexample. $\dashv$.

else if $\mathcal{I}^N_{\prec C} \models C' \vee L$: Then by Lemma 24 $\mathcal{I}^N_\infty \models C' \vee L$, which means that C cannot be a counterexample either. $\dashv$

else if $C = C'' \vee \underline{L'} \vee \underline{L}$, and $L \in \mathbf{L}^\theta_+$: Then due totality (Lemma 3) we know that $L \equiv L'$. We case split on $\langle L, L' \rangle$

case $\langle s \approx_\tau t, s' \approx_\tau t' \rangle$ where $\tau \neq \tau_\mathbb{Q}$: If $s \equiv t$, or $s' \equiv t'$, then the C would not be a counterexample. So due to totality (Property 2) we can assume without loss of generality that $s \succ t, s' \succ t'$, which means by irreflexivity compatibility (Property 3 and Irreflexivity (Property 9), that $s \equiv s'$, and $t \equiv t'$

There is an instance of Rule (EF)

$$\frac{C' \vee s \approx t \vee s' \approx t'}{C' \vee t \not\approx t' \vee s' \approx t'}$$

Therefore since $\mathcal{I}^N_\infty \not\models C$ it must also be the case that $\mathcal{I}^N_\infty \not\models C' \vee t \not\approx t' \vee s' \approx t'$. Further the hypothesis is smaller than the conclusion, as from $s \succ t \equiv t'$, Property 3, and Lemma 14 it follows that $s \approx t \succ t \not\approx t'$.

case $\langle k \diamond 0, L' \rangle$ or $\langle L, k \diamond 0 \rangle$: Then either $\mathbb{Q} \models k \diamond 0$, hence C cannot be a counterexample, or there is an instance of (Triv) that reduces the counterexample.

case $\langle +j\underline{s} \pm k\underline{s'} + t \diamond 0, L'' \rangle$, or $\langle L'', +j\underline{s} \pm k\underline{s'} + t \diamond 0 \rangle$ where $s \equiv s'$: Then due to Lemma 6, there is an instance of rule (TF) that reduces the counterexample.

case $\langle +j\underline{\underline{s}} + t \diamond 0, +j\underline{\underline{s'}} + t' \diamond 0 \rangle$: Note that the coefficient of $+j$ must be posive as $L \in \mathbf{L}^\theta_+$, and the coefficients of s and s' must be the same as $L \equiv L'$, and $t \equiv t'$ for the same reason.

We case split on $\diamond$

case $\approx$: Then $L = s \mathrel{\hat{\approx}} -\frac{1}{j}t$, and $L' = s' \mathrel{\hat{\approx}} -\frac{1}{j}t'$. Hence there is an instance of Rule (EF) that reduces the counterexample.

case $\geq$, or $>$: Then there is an instance of the rule (IF) that reduces the counterexample.

case $\langle P(t_1 \dots t_n), P(t'_1 \dots t'_n) \rangle$ Then there is an instance of the Rule (F), that reduces the counterexample.

else if $\exists t \in \mathsf{active}(L)$ that is reducible in $\mathcal{I}^N_\infty$: In this case superposition will reduce the counterexample:

We case split on the rewrite rule that is used to rewrite t.

case $s \to t_1$ where $s \in \mathbf{Atomic}^\theta$: This means that there is some productive clause $D = D' \vee s \mathrel{\hat{\approx}} t_1$ with $s \lhd L$.

if $L \equiv s \approx t_1$: Then we know by Lemma 25 that $\mathcal{I}^N_\infty \vDash s \approx t_1$, hence $\mathcal{I}^N_\infty \vDash C$ which contradicts the assumption that C is a counterexample. $\dashv$

else if $s \mathrel{\hat{\approx}} t_1 \succ L$: Then we know by Lemma 19 that $L \equiv_\approx \underline{s} \mathrel{\hat{\approx}} t_2$.

This means that there is an instance of Rule (Sup)

$$\frac{C' \vee s \mathrel{\hat{\approx}} t_2 \qquad D' \vee s \mathrel{\hat{\approx}} t_1}{C' \vee D' \vee t_2 \mathrel{\hat{\approx}} t_1}$$

As D is productive, and C is a counterexample we know by Lemma 25 that $\mathcal{I}^N_\infty \not\models D' \vee C' \vee s \approx t_2$, and $\mathcal{I}^N_\infty \models s \mathrel{\tilde{\approx}} t_1$. Therefore $\mathcal{I}^N_\infty \models s \approx t_1 \wedge s \not\approx t_2$, which implies $\mathcal{I}^N_\infty \models t_2 \not\approx t_1$, hence the conclusion is a smaller counterexample.

else: Note that either L is strictly maximal or it is weakly maximal and not in $\mathbf{L}^\theta_+$, which means that Rule (Sup) is applicable. Therefore there is an instance of Rule (Sup)

$$\frac{D' \vee s \mathrel{\tilde{\approx}} t_1 \qquad C' \vee L[s]}{C' \vee D' \vee L[t_1]}$$

As D is productive we know by Lemma 25, that $\mathcal{I}^N_\infty \not\models D'$. Further we already know that $\mathcal{I}^N_\infty \not\models C' \vee L[s]$, and as $\mathcal{I}^N_\infty \models s \approx t_1$ we know that also $\mathcal{I}^N_\infty \not\models L[t_1]$, hence the conclusion is a counterexample as well. Due to the stability under contexts (Lemma 4), we know that $L[t_1] \prec L[s]$. Further as $L[s] \prec s \mathrel{\tilde{\approx}} t_1$ we know that the conclusion is smaller than the premise, which means that superposition reduced the counterexample.

else if $+j\underline{s} \pm k\underline{s} + t' \in C'$ **and** s **is a max atom:** Then the Rule (TF) reduces the counterexample.

else: We know that

- $\forall t \in \mathsf{active}(L).t$ is irreducible in $\mathcal{I}^N_\infty$
- $\mathcal{I}^N_{\prec C} \not\models C'$
- C is not productive
- $L \in \mathbf{L}^\theta_+ \Rightarrow C = C' \vee \underline{\underline{L}}$
- $L \notin \mathbf{L}^\theta_+ \Rightarrow C = C' \vee \underline{L}$
- $\mathcal{I}^N_{\prec C} = \mathcal{I}^N_C$

Let's case split on L

case $+k\underline{\underline{s}} + t_2 \gtrsim_2 0$**:** As C is not productive it must be the case that $\exists + j\underline{\underline{s}} + t_1 \gtrsim_1 0 \in C'.s' \equiv s$.

if $\mathcal{I}^N_\infty \models kt_1 - jt_2 > 0$**:** This means that $\mathcal{I}^N_\infty \not\models jt_2 - kt_1 \geq 0$. Hence there is an instance of Rule (IF)

$$\frac{C'' \vee +js' + t_1 \gtrsim_1 0 \vee +ks + t_2 \gtrsim_2 0}{C'' \vee jt_2 - kt_1 \gtrsim_4 0 \vee +js' + t_1 \gtrsim_1 0}$$

which reduces the counterexample.

else if $\mathcal{I}^N_\infty \models jt_2 - kt_1 > 0$**:** This means that $\mathcal{I}^N_\infty \not\models kt_1 - jt_2 \geq 0$ Hence there is an instance of Rule (IF)

$$\frac{C'' \vee +js' + t_1 \gtrsim_1 0 \vee +ks + t_2 \gtrsim_2 0}{C'' \vee kt_1 - jt_2 \gtrsim_3 0 \vee +ks + t_2 \gtrsim_2 0}$$

which reduces the counterexample.

else $\mathcal{I}^N_\infty \models jt_2 - kt_1 \approx 0$**: if** $\gtrsim_1 = \gtrsim_2$**:** Then there is an instance of Rule (IF)

$$\frac{C'' \vee +js' + t_1 \gtrsim_1 0 \vee +ks + t_2 \gtrsim_1 0}{C'' \vee kt_1 - jt_2 > 0 \vee +ks + t_2 \gtrsim_2 0}$$

which reduces the counterexample.

else if $\gtrsim_1 = >$, and $\gtrsim_2 = \geq$: Then there is an instance of Rule (IF)

$$\frac{C'' \vee +js' + t_1 > 0 \vee +ks + t_2 \geq 0}{C'' \vee kt_1 - jt_2 > 0 \vee +ks + t_2 \geq 0}$$

which reduces the counterexample.

else if $\gtrsim_1 = \geq$, and $\gtrsim_2 = >$: Then there is an instance of Rule (IF)

$$\frac{C'' \vee +js' + t_1 \geq 0 \vee +ks + t_2 > 0}{C'' \vee jt_2 - kt_1 > 0 \vee +js' + t_1 \geq 0}$$

which reduces the counterexample.

case $\underline{s} \mathrel{\hat{\approx}} t$: As C is not productive we know that there is an $s' \mathrel{\hat{\approx}} u \in C'$ such that $s' \equiv s$, and $\mathcal{I}^N_{\prec C} \not\models t \not\approx u$. This means there is an instance of Equality Factoring (Rule (EF)) that reduces the counterexample.

case $s \approx s'$ where $s \equiv s'$: Then C cannot be a counterexample. ⊣

case $s \not\approx s'$ where $s \equiv s'$: Then there is an instance of equality resolution (ER) that reduces the counterexample.

case $s \not\approx t$ where $s \not\equiv t$: As s is not reducible, and $s \succ t$, and t can only be reduced to smaller terms, we know that $s \downarrow \neq t \downarrow$. Hence $\mathcal{I}^N_\infty \models s \not\approx t$, which means C cannot be a counterexample. ⊣

case $k \gtrsim 0$ where $\gtrsim \in \{>, \geq\}$: Here we can reason in the same way as in the case where $L = k \gtrsim 0$ was not strictly maximal.

case $-k\underline{s} + t > 0$: Note that $-k$ is negative as all positive coefficients have are being removed by termfactoring in the case before.

$$\begin{aligned}
\mathcal{I}^N_\infty \not\models -ks + t > 0 &\iff \mathcal{I}^N_\infty \not\models -k's + t > 0 \\
&\iff \mathcal{I}^N_\infty \models +ks - t \geq 0 \\
&\iff \mathcal{I}^N_\infty \models +ks - t > 0 \parallel \mathcal{I}^N_\infty \models +ks - t \approx 0
\end{aligned}$$

if $\mathcal{I}^N_\infty \models +ks - t \approx 0$: This is not possible as s is irreducible. ⊣

else: That means that $\mathcal{I}^N_\infty \models kl - t \geq 0$ for some $s' > l \in \mathcal{I}^N_\infty$.

By construction of $\mathcal{I}^N_\infty$ this means that there must be some productive clause $D = D' \vee j\underline{s'} - jl \gtrsim 0$. This further means that there is an application of (FM)

$$\frac{D' \vee +j\underline{s'} - jl \gtrsim 0 \qquad C' \vee -k\underline{s} + t > 0}{C' \vee D' \vee jt - jkl > 0}$$

As D is productive, by Lemma 25 we know that $\mathcal{I}^N_\infty \models \neg D' \wedge j\underline{s} - jl \gtrsim 0$. Hence we know that $\mathcal{I}^N_\infty \not\models C' \vee D'$. Further we know that $\mathcal{I}^N_\infty \models kl - t \geq 0$, which means that $\mathcal{I}^N_\infty \not\models t - kl > 0$, which means that the conclusion is a counterexample.

We know that if there is some summand $k''s''$ of t with $s \equiv s'$, then k'' must be negative, as otherwise term factoring would be applicable, which case we already handled before.

$$
\begin{aligned}
& & +js \prec -ks & \quad \text{(Property 12)} \\
\implies & & +js - jl \prec -ks & \quad \text{(Property 7)} \\
\implies & & +js - jl \prec -ks + t & \quad \text{(Property 13)} \\
\implies & D' \prec & +js - jl \prec -ks + t & \\
\implies & & D' \vee C' \prec -ks + t \vee C' &
\end{aligned}
$$

Further as $s' = \mathsf{maxAtom}(js' - jl \gtrsim 0)$, and due to totality (Property 2), we know that $\forall s'' \in \mathsf{maxAtoms}(-ks + t > 0).s'' \equiv s$. This implies that $\mathsf{maxAtoms}(-ks + t > 0) \succ \mathsf{maxAtoms}(jt - jkl > 0)$. This means by Lemma 16 that the conclusion is a smaller counterexample.

case $-k\underline{s} + t \geq 0$**:**

$$
\begin{aligned}
\mathcal{I}^N_\infty \not\models -ks + t \geq 0 &\iff \mathcal{I}^N_\infty \not\models -ks + t > 0 \;\&\; \mathcal{I}^N_\infty \not\models -ks + t \approx 0 \\
&\iff \mathcal{I}^N_\infty \models +ks - t \geq 0 \;\&\; \mathcal{I}^N_\infty \not\models -ks + t \approx 0 \\
&\iff \mathcal{I}^N_\infty \models +ks - t > 0 \;\&\; \mathcal{I}^N_\infty \not\models -ks + t \approx 0
\end{aligned}
$$

As previously means that $\mathcal{I}^N_\infty \models kl - t \geq 0$ for some $s' > l \in \mathcal{I}^N_\infty$, hence by construction of $\mathcal{I}^N_\infty$ this means that there must be some productive clause $D = D' \vee js' - jl \gtrsim 0$. If $\gtrsim$=$>$ then we can reason as in the case where $L = -ks + t > 0$.

If $\gtrsim$=$\geq$ then there is an instance of ($\mathsf{FM}^{\geq}$).

$$\frac{D' \vee +j\underline{s} - jl \geq 0 \qquad C' \vee -k\underline{s} + t \geq 0}{C' \vee D' \vee jt - jkl > 0 \vee -ks + t \approx 0}$$

As D is productive, by Lemma 25 we know that $\mathcal{I}^N_\infty \models \neg D' \wedge js - jl \geq 0$. Hence we know that $\mathcal{I}^N_\infty \not\models C' \vee D'$. Further we know that $\mathcal{I}^N_\infty \models kl - t \geq 0$, which means that $\mathcal{I}^N_\infty \not\models jt - jkl > 0$. Hence the conclusion is a counterexample.

Further we can reason as in the case where $L = -ks + t > 0$ to establish that $C' \vee D' \vee jt - jkl > 0 \prec -ks + t \geq 0$. Together with Lemma 13 we know that the conclusion is a smaller counterexample.

case $\neg P(t_1 \dots t_n)$**:** Then we know that $P(t_1 \dots t_n) \in \mathcal{I}^N_\infty$, hence there must be a productive clause $D = D' \vee P(t'_1 \dots t'_n)$, with $t_i \equiv t'_i$ for all i. This means that there is a resolution inference (BR), that reduces the counterexample.

$$\frac{D' \vee P(t_1 \dots t_n) \qquad C' \vee \neg P(t'_1 \dots t'_n)}{C' \vee D'}$$

As D is productive we know by Lemma 25 that $\mathcal{I}^N_\infty \not\models D'$, hence the conclusion is a smaller counterexample.

5.2 ALASCA Lifting and Completeness

Variable Elimination. Theorem 4 establishes completeness of ALASCA^θ for ground clauses wrt $\mathcal{A}_\mathbb{Q}$. We next lift this result (and calculus) to non-ground clauses.

We introduce the concept of an *unshielded variable*. We say a term $t : \tau_\mathbb{Q}$ is a top level term of a literal L if $t \in \mathsf{atoms}(L)$. We call a variable x *unshielded* in some clause C if x is a top level term of a literal in C, and there is no literal with an atomic top level term $t[x]$, and no uninterpreted literal $L[x]$. Observe that within the ALASCA^θ rules, only maximal atomic terms in sums are being used in rule applications. This means, lifting ALASCA^θ to ALASCA is straightforward for clauses where all maximal terms in sums are not variables. Further, due to the subterm property, if a variable is maximal in a sum then it must be unshielded. Hence, the only variables we have to deal within ALASCA rule applications are unshielded ones.

The work of [39] modifies a standard saturation algorithm by integrating it with a variable elimination rule that gets rid of unshielded variables, without compromising completeness of the calculus. Based on [39] and the variable elimination rule of [3], we extend ALASCA^θ with the Variable Elimination Rule (VE), as given in Figure 6. In what follows, we show that the handling of unshielded variables in Figure 6 can naturally be done within a standard saturation framework.

The (VE) rules replaces any clause with a set of clauses that is equivalent and does not contain unshielded variables. We assume that the clause is normalized, such that in every inequality x only occurs once with a factor 1 or -1, whereas for for equalities, x only occurs with factor 1. A simple example for the application of (VE) is the clause $a - x > 0 \vee x - b > 0 \vee a + b + x \geq 0$, where $x \in \mathbf{V}$, and a, b are constants. By reasoning about inequalities, it is easy to see that this is equivalent to $a > x \vee a + b \geq x \vee x > b$, thus further equivalent to $a > b \vee a + b \geq a$, which illustrates the benefit of variable elimination through (VE).

Lemma 26. *The conclusion of* (VE) *is equivalent to its premise.*

Proof. We will call a rule *equivalence preserving* iff its premises are equivalent to its conclusion.

We will proof the fact that the rule is equivalence preserving reducing it to the same fact about a simpler rule that does not contain equalities but only inequalities, using the fact that every positive and negative equality can be defined in terms of disequalities.

So let's first consider the simplified version of the rule, where all predicate symbols of the variable are either ($>$) or ($\geq$)

$$\frac{C \vee \bigvee_{i \in I} (x + b_i \gtrsim_i 0) \vee \bigvee_{j \in J} (-x + b_j \gtrsim_j 0)}{C \vee \bigvee_{i \in I, j \in J} (b_i + b_j \gtrsim_{i,j} 0)} \; (\mathsf{VE}^*)$$

where
- $(\gtrsim_i), (\gtrsim_j) \in \{(\geq), (>)\}$
- $(\gtrsim_{i,j}) = \begin{cases} (>) & \text{if } (\gtrsim_i) = (\gtrsim_j) = (>) \\ (\geq) & \text{otherwise} \end{cases}$

Lemma 27. *The conclusion of the rule* (VE^*) *is equivalent to its premise.*

Proof. "$\Longrightarrow$") The basic idea of this part of the proof is the following. Assuming the conclusion of the rule is false, it defines a set of bounds, that must contain a value m. This value m can then be used as an x to make the assumption false as well. We will now formalize this intuition:

Suppose $M \vDash \mathsf{Hyp}$. In case $M \vDash C$, we're done, so let's consider $M \nvDash C$. This means we need to show that $M \vDash \bigvee_{i\in I}(x + b_i \gtrsim_i 0) \vee \bigvee_{j\in J}(-x + b_j \gtrsim_j 0) \Longrightarrow M \vDash \bigvee_{i\in I, j\in J}(b_i + b_j \gtrsim_{i,j} 0)$

We will do that by contraposition

$$\begin{aligned}
& M \nvDash \bigvee_{i\in I, j\in J}(b_i + b_j \gtrsim_{i,j} 0) \\
\Leftrightarrow & M \vDash \bigwedge_{i\in I, j\in J}\neg(b_i + b_j \gtrsim_{i,j} 0) \\
\Leftrightarrow & M \vDash \bigwedge_{i\in I, j\in J}(b_j \not\gtrsim_{i,j} -b_i) \\
\Leftrightarrow & M(b_j) \not\gtrsim_{i,j} M(-b_i) \qquad (1)
\end{aligned}$$

Let us now find the least upper and greatest lower bounds for a value x that will make $\bigvee_{i\in I}(x + b_i \gtrsim_i 0) \vee \bigvee_{j\in J}(-x + b_j \gtrsim_j 0)$ false:

$$\begin{aligned}
j^- &= argmax_{j\in J}(M(b_j)) & i^+ &= argmin_{i\in I}(-M(b_i)) \\
b^- &= M(b_{j^-}) & b^+ &= -M(b_{i^+})
\end{aligned}$$

Let m be an arbitrary value such that

$$m \in \begin{cases} [b^-, b^+] & \text{if } \langle \gtrsim_{j^-}, \gtrsim_{i^+} \rangle \text{ is } \langle >, > \rangle \\ ]b^-, b^+] & \text{if } \langle \gtrsim_{j^-}, \gtrsim_{i^+} \rangle \text{ is } \langle \geq, > \rangle \\ [b^-, b^+[& \text{if } \langle \gtrsim_{j^-}, \gtrsim_{i^+} \rangle \text{ is } \langle >, \geq \rangle \\ ]b^-, b^+[& \text{if } \langle \gtrsim_{j^-}, \gtrsim_{i^+} \rangle \text{ is } \langle \geq, \geq \rangle \end{cases}$$

Note that due to the definition of $\gtrsim_{i,j}$, and (1), the value m exists: If on the one hand $(\geq) \in \{(\gtrsim_{i^+}), (\gtrsim_{j^-})\}$ then we have that $(\gtrsim_{i^+,j^-}) = (\geq)$, which means that by (1) we have that $b^- \not\geq -b^+$ which means that $b^- < -b^+$, hence we could choose the value $\frac{b^- - b^+}{2}$. On the other hand if $(>) = (\gtrsim_{i^+}) = (\gtrsim_{j^-})$, then we have $(\gtrsim_{i^+,j^-}) = (>)$, hence by (1) we know that $b^- \not> -b^+$, which means that $b^- \leq -b^+$, which means we can again choose the same value.

Let's show first that $M\{x \mapsto m\} \nvDash -x + b_j \gtrsim_j 0$ for any $j \in J$. If $m \neq M(b_j)$ then it is strictly greater, since it is greater or equal to b^- which is the maximum

of all $M(b_j)$. If $m = M(b_j)$, then (due to the definition of m), $\gtrsim_j$ must be $>$, which means that we are done because $\not\models -b_j + b_j > 0$.

Finally we need to show that $M\{x \mapsto m\} \not\models x + b_i \gtrsim_i 0$ for any $i \in I$. This can be done analogous to the case before: If $m \neq -M(b_i)$ then it is strictly less, since it is less or equal to b^+ which is the maximum of all $-M(b_i)$. If $m = -M(b_i)$, then (due to the definition of m), $\gtrsim_i$ must be $>$, which means that we are done because $\not\models -b_j + b_j > 0$.

"$\Longleftarrow$") The intuition here is that if conclusion of the rule is true, then must be some value m within the bounds of some $-b_i$, b_j. Which means that every x will either be less then this m, hence less than b_j, or it will be greater than this m, hence greater than $-b_i$, which means that the hypothesis is true.

Let us formalize this argument: As in the case before, if $M \models C$, we're done. So let's assume that $M \models b_i + b_j \gtrsim_{i,j} 0$ for some $\langle i, j\rangle \in I \times J$. We define $m = \frac{b_i + b_j}{2}$.

if $M(x) > m$**:** Then we have that $M(x) > m \geq -b_j$, which means $M \models x + b_j > 0$.
if $M(x) < m$**:** We have that $M(x) < m \leq b_i$, hence $M \models -x + b_i > 0$.
if $M(x) = m$**:** We make a case distinction base on $\gtrsim_i$ and $\gtrsim_j$:

- **if** $(\gtrsim_i) = (\gtrsim_j) = (>)$**:** Then $(\gtrsim_{i,j}) = (>)$. This means $-M(b_i) < m < M(b_j)$, which means that $M \models x + b_i > 0$.
- **if** $(\geq) = (\gtrsim_i)$**:** Then $(\gtrsim_{i,j}) = (>)$, hence $-M(b_i) \leq m \leq M(b_j)$, which means that $M \models x + b_i \geq 0$.
- **if** $(\geq) = (\gtrsim_j)$**:** Then $(\gtrsim_{i,j}) = (>)$, hence $-M(b_i) \leq m \leq M(b_j)$, which means that $M \models -x + b_j \geq 0$.

Now that we have established that (VE^*) is equivalence preserving we can proof it for the more general rule (VE) by reducing it to the simpler one. We first rewrite the premise of the general rule to be a disjunction of inequalities.

$$
\begin{aligned}
& C \vee \bigvee_{i\in I} x + b_i \gtrsim_i 0 \vee \bigvee_{j\in J} -x + b_j \gtrsim_j 0 \vee \bigvee_{k\in K} x + b_k \approx 0 \vee \bigvee_{l\in L} x + b_l \not\approx 0 \\
\Longleftrightarrow\; & C \vee \bigvee_{i\in I} x + b_i \gtrsim_i 0 \vee \bigvee_{j\in J} -x + b_j \gtrsim_j 0 \vee \bigvee_{k\in K} x + b_k \approx 0 \vee \bigvee_{l\in L} x + b_l > 0 \vee \bigvee_{l\in L} -x - b_l > 0 \\
\Longleftrightarrow\; & \left(\begin{array}{l} C \vee \bigvee_{i\in I} x + b_i \gtrsim_i 0 \vee \bigvee_{j\in J} -x + b_j \gtrsim_j 0 \vee \bigvee_{k\in K} \left(x + b_k \geq 0 \wedge\ -x - b_k \geq 0 \right) \\ \qquad\qquad \vee \bigvee_{l\in L} x + b_l > 0 \ \vee \bigvee_{l\in L} -x - b_l > 0 \end{array} \right) \\
\Longleftrightarrow\; & \bigwedge_{K^+ \subseteq K} \left(\begin{array}{l} C \vee \bigvee_{i\in I} x + b_i \gtrsim_i 0 \vee \bigvee_{j\in J} -x + b_j \gtrsim_j 0 \vee \bigvee_{k\in K^+} x + b_k \geq 0 \vee \bigvee_{k\in K^-} -x - b_k \geq 0 \\ \qquad\qquad \vee \bigvee_{l\in L} x + b_l > 0 \ \vee \bigvee_{l\in L} -x - b_l > 0 \end{array} \right)
\end{aligned}
$$

where $K^- = K \setminus K^+$.

Then we can apply (VE^*)

$$\bigwedge_{K^+\subseteq K}\left(\begin{array}{llll} C \vee \bigvee\limits_{i\in I, j\in J} b_i + b_j \gtrsim_{i,j} 0 \vee & \bigvee\limits_{i\in I,k\in K^-} b_i - b_k \geq 0 & \vee & \bigvee\limits_{i\in I,l\in L} b_i - b_l \gtrsim_i 0 \\ \vee & \bigvee\limits_{j\in J,k\in K^+} b_j + b_k \geq 0 & \vee & \bigvee\limits_{j\in J,l\in L} b_j + b_l \gtrsim_j 0 \\ \vee & \bigvee\limits_{k_1\in K^+,k_2\in K^-} b_{k_1} - b_{k_2} \geq 0 & \vee & \bigvee\limits_{k\in K^+,l\in L} b_k - b_l \geq 0 \\ & & \vee & \bigvee\limits_{k\in K^-,l\in L} b_l - b_k \geq 0 \\ & & \vee & \bigvee\limits_{l_1\in L,l_2\in L} b_{l_1} - b_{l_2} > 0 \end{array}\right)$$

which we can simplify to

$$\bigwedge_{K^+\subseteq K}\left(\begin{array}{llll} C \vee \bigvee\limits_{i\in I, j\in J} b_i + b_j \gtrsim_{i,j} 0 \vee & \bigvee\limits_{i\in I,k\in K^-} b_i - b_k \geq 0 & \vee & \bigvee\limits_{i\in I,l\in L} b_i - b_l \gtrsim_i 0 \\ \vee & \bigvee\limits_{j\in J,k\in K^+} b_j + b_k \geq 0 & \vee & \bigvee\limits_{j\in J,l\in L} b_j + b_l \gtrsim_j 0 \\ \vee & \bigvee\limits_{k_1\in K^+,k_2\in K^-} b_{k_1} - b_{k_2} \geq 0 & \vee & \bigvee\limits_{k\in K^+,l\in L} b_k - b_l \geq 0 \\ & & \vee & \bigvee\limits_{k\in K^-,l\in L} b_l - b_k \geq 0 \\ & & \vee & \bigvee\limits_{\langle l_1,l_2\rangle\in \mathsf{pairs}(L)} b_{l_1} - b_{l_2} \not\gtrsim 0 \end{array}\right)$$

where $\mathsf{pairs}(S) = \{\langle x, y\rangle \in S \times S, x \prec y\}$ for some ordering $\prec$ on S. Which concludes our proof.

Alasca Calculus - Non-Ground Version with Unification with Abstraction. We now define our lifted calculus Alasca, as follows. Let Alasca$^-$ be the calculus Alasca$^\theta$ being lifted for clauses without unshielded variables. We define Alasca to be Alasca$^-$ chained with the variable elimination rule. That is, the result of every rule application is simplified using (VE) as long as applicable.

Formally define Alasca $= \{\gamma^{(\mathsf{VE})} \mid \gamma \in \text{Alasca}^-\}$, where $\circ^*$ and $\circ^{(\mathsf{VE})}$ are defined as follows:

$$\gamma^{(\mathsf{VE})} = \begin{cases} \gamma^* & \text{if the conclusion of } \gamma \text{ contains an unshielded variable} \\ \gamma & \text{else} \end{cases}$$

$$\left(\ \frac{C_1 \quad \dots \quad C_n}{D}\ \right)^* = \left(\ \frac{\dfrac{C_1 \quad \dots \quad C_n}{D}}{D'}\,(\mathsf{VE})\ \right)$$

Theorem 5. *Alasca is the lifting of a counterexample-reducing inference system for sets of clauses without unshielded variables.*

Proof. We observe that by definition ALASCA will never produce clauses with unshielded variables. Further by Lemma 26 it will derive sets of clauses equivalent to ALASCA^{-}. Thus the only thing to show is that in cases where ALASCA^{-} reduces a counterexample C_n to D, the clause D', the result of simplifying D using (VE), will be smaller than C_n as well. As ALASCA^{-} is a lifting of ALASCA$^{\theta}$, only ground instances of ALASCA^{-} inferences that actually are inferences of ALASCA$^{\theta}$ are relevant for reducing counterexamples. These inferences can only introduce unshielded variables x, if x has been shielded by some term s_i in a hypothesis s_i such that$s_i \in \mathsf{active}(C_i)$. Further for such a rule the conclusion D' only contains atoms that were contained in some hypothesis before, but does not contain the maximal atom s_i anymore. Therefore, as the literal ordering orders literals by their $\mathsf{atoms}_{\mathsf{safe}}$, and clauses by the multiset extension of that literal ordering, the result is also smaller than C_n. □

Theorem 5 imples that ALASCA is refutationally complete wrt $\mathcal{A}_{\mathbb{Q}}$ for sets of clauses without unshielded variables. As the (VE) rule can be applied as a step of preprocessing we can refute an arbitrary set of clauses using the calculus.

We conclude this section by specifying the lifting of ALASCA$^{\theta}$ to get ALASCA^{-}. To this end, we use our uwa results and properties for unification with abstraction (Sect. 4). We note that using unification modulo $\mathcal{A}_{\mathsf{eq}}$ would require us to develop an algorithmic approach that computes a complete set of unifiers modulo $\mathcal{A}_{\mathsf{eq}}$, which is a quite challenging task both in theory and in practice. Instead, using Theorem 1 and Theorem 3, we need to only specify a canAbstract predicate that guards interpreted functions and captures $\mathcal{A}_{\mathsf{eq}}$ within uwa. This is achieved by defining $\mathsf{canAbstract}(s,t)$ iff any function symbol $f \in \{\mathsf{sym}(s), \mathsf{sym}(t)\}$ is an interpreted function $f \in \mathbb{Q} \cup \{+\}$.This choice of the canAbstract predicate is a slight modification of the abstraction strategy `one_side_interpreted` of [33]. We note that this is not the only choice for the predicate to fulfil the canAbstract properties. Consider for example the terms $f(x) + a$, and $a + b$. There is no substitution that will make these two terms equal, but our abstraction predicate introduces a constraint upon trying to unify them. In order to address this, we introduce an alternative canAbstract predicate that compares the atoms of a term, instead of only looking at the outer most symbol:

We define $\mathsf{canAbstract}(s,t)$ iff

- $\mathsf{sym}(s) = \mathsf{sym}(t)$ is interpreted, or
- there is a variable in $\mathsf{atoms}_{\mathsf{safe}}(s) \cup \mathsf{atoms}_{\mathsf{safe}}(t)$, or
- $\langle n, \dot{\{} f \mid f(\ldots) \in A \dot{\}} \rangle =^{\mathsf{wmul}} \langle m, \dot{\{} f \mid f(\ldots) \in B \dot{\}} \rangle$,
  where $\langle n, A \rangle = \mathsf{atoms}_{\mathsf{safe}}(s), \langle m, B \rangle = \mathsf{atoms}_{\mathsf{safe}}(t)$.

6 Implementation and Experiments

We implemented ALASCA [6] in the extension of the VAMPIRE theorem prover [26].

[6] available at `https://github.com/vprover/vampire/commit/a766284fd9b85a4800572a6b63b8e22a68e187c9`

Benchmarks (#)	Alasca	Cvc5	Vampire	Yices	UltElim	SmtInt	veriT	solved
all (6374)	**5741**	5626	5585	5531	5218	829	465	5988
LRA (1722)	1574	1401	1399	**1722**	1469	623	89	1722
NRA (3814)	3795	3804	3803	**3809**	3669	0	0	3812
UFLRA (10)	**10**	**10**	**10**	0	0	**10**	**10**	10
Triangular (34)	**24**	10	14	0	0	0	6	25
Limit (280)	**100**	90	80	0	80	0	90	100
SH (514)	238	**311**	279	0	0	196	270	319

Table 1. Experimental results, showing the numbers of solved problems.

Benchmarks. We evaluated the practicality of Alasca using the following six sets of benchmarks, resulting all together in 6374 examples, as listed in Table 1 and detailed next. (i) We considered all sets of benchmarks from the SMT-LIB repository [7] set that involve real arithmetic and uninterpreted functions, but no other theories. These are the three benchmark sets corresponding to the LRA, NRA, and UFLRA logics in SMT-LIB. (ii) We further used new examples generated by [15], using the SMT-LIB syntax. From the examples of [15], we selected those benchmarks that involve real arithmetic but no other theories. We refer to this benchmark set as SH. (iii) Finally, we also created two new sets of benchmarks, Triangular, and Limit, exploiting various mathematical properties. The Triangular suite contains variations of our motivating example from Section 2, and thus comes with reasoning challenges about triangular inequalities and continuous functions. The Limit benchmark set is comprised of problems that combine various limit properties of real-valued functions.

Experimental Setup. We compared our implementation against the solvers from the `Arith` (arithmetic) division of the SMT-COMP competition 2022. These solvers, given in colums 3–8 of Table 1, are: Cvc5 [5], Vampire [34], Yices [19], UltElim [8], SmtInt [21], and veriT [2]. We point out that Vampire is run in its competition portfolio mode, which includes the work from [33]. Alasca uses the same portfolio, but implementing our modified version of unification with abstraction, disabling the use of theory axioms, and enabling the rules introduced in this paper instead. We ran our experiments using the SMT-COMP 2022 competition setup: based on the StarExec Iowa cluster, with a 20 minutes timeout, and using 4 cores.

Experimental Results. Table 1 summarizes our experimental findings and indicates the overall best performance of Alasca. For example, Alasca outperforms the two best arithmetic solvers of SMT-COMP 2022 by solving 115 more problems than Cvc5 and 156 more problems than Vampire. We believe that future combinations of Alasca with first-order simplification rules (such as subsumption resolution and demodulation) will further push the overall superiority of Alasca. An interesting fact to point out here is that Vampire outperforms Alasca on the problem set SH. Looking into the results in detail revealed that some problems can be solved by Vampire by applying only one rule, like demodulation, or unit resulting resolution, the applications of which are prevented in Alasca,

as literals are normalized so that the rules are not applicable anymore. For this reason believe that future adjustments of first-order simplification rules (such as subsumption resolution and demodulation) for Alasca will further push the overall superiority of Alasca.

7 Conclusions and Future Work

We introduced the Alasca calculus and drastically improved the performance of superposition theorem proving on linear arithmetic. Alasca eliminates the use of theory axioms by introducing theory-specific rules such as an analogue of Fourier-Motzkin elimination. We perform unification with abstraction with a general theoretical foundation, which, together with our variable elimination rules, serves as a replacement for unification modulo theory. Our experiments show that Alasca is competitive with state-of-the-art theorem provers, solving more problems than any prover that entered the arithmetic division in SMT-COMP 2022. Future work includes designing an integer version of Alasca, developing different versions for the canAbstract predicate, and improving literal/clause selections within Alasca.

Acknowledgements. This work was partially supported by the ERC Consolidator Grant ARTIST 101002685, the TU Wien Doctoral College SecInt, and the FWF SFB project SpyCoDe F8500.

References

1. Alt, L., Blicha, M., Hyvärinen, A.E.J., Sharygina, N.: SolCMC: Solidity Compiler's Model Checker. In: CAV, LNCS, vol. 13371, pp. 325–338, Springer (2022), https://doi.org/10.1007/978-3-031-13185-1_16
2. Andreotti, B., Barbosa, H., Fontaine, P., Schurr, H.J.: veriT at SMT-COMP 2022. https://smt-comp.github.io/2022/system-descriptions/veriT.pdf (2022)
3. Bachmair, L., Ganzinger, H.: Ordered Chaining Calculi for First-Order Theories of Transitive Relations. J. ACM **45**(6), 1007–1049 (1998), https://doi.org/10.1145/293347.293352, URL https://doi.org/10.1145/293347.293352
4. Bachmair, L., Ganzinger, H.: Resolution Theorem Proving. In: Handbook of Automated Reasoning, pp. 19–99, Elsevier and MIT Press (2001), https://doi.org/10.1016/b978-044450813-3/50004-7
5. Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., , Zohar, Y.: CVC5 at the SMT Competition 2022. https://smt-comp.github.io/2022/system-descriptions/cvc5.pdf (2022)
6. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A Versatile and Industrial-Strength SMT Solver. In: TACAS, LNCS, vol. 13243, pp. 415–442, Springer (2022), https://doi.org/10.1007/978-3-030-99524-9_24
7. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org (2016)

8. Barth, M., Dietsch, D., Heizmann, M., Podelski, A.: Ultimate Eliminator at SMT-COMP 2022. https://smt-comp.github.io/2022/system-descriptions/UltimateEliminator%2BMathSAT.pdf (2022)
9. Baumgartner, P., Bax, J., Waldmann, U.: Beagle - A Hierarchic Superposition Theorem Prover. In: CADE, LNCS, vol. 9195, pp. 367–377, Springer (2015), https://doi.org/10.1007/978-3-319-21401-6_25
10. Bonacina, M.P., Graham-Lengrand, S., Shankar, N.: Satisfiability Modulo Theories and Assignments. In: CADE, LNCS, vol. 10395, pp. 42–59, Springer (2017), https://doi.org/10.1007/978-3-319-63046-5_4
11. Bromberger, M., Fleury, M., Schwarz, S., Weidenbach, C.: SPASS-SATT - A CDCL(LA) solver. In: CADE, LNCS, vol. 11716, pp. 111–122, Springer (2019), https://doi.org/10.1007/978-3-030-29436-6_7
12. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The OpenSMT Solver. In: TACAS, LNCS, vol. 6015, pp. 150–153, Springer (2010), https://doi.org/10.1007/978-3-642-12002-2_12
13. Cook, B.: Formal Reasoning About the Security of Amazon Web Services. In: CAV, LNCS, vol. 10981, pp. 38–47, Springer (2018), https://doi.org/10.1007/978-3-319-96145-3_3
14. Cruanes, S.: Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. Ph.D. thesis, Ecole Polytechnique, Paris, France (2015)
15. Desharnais, M., Vukmirovic, P., Blanchette, J., Wenzel, M.: Seventeen Provers Under the Hammer. In: ITP, LIPIcs, vol. 237, pp. 8:1–8:18 (2022), https://doi.org/10.4230/LIPIcs.ITP.2022.8
16. Distefano, D., Fähndrich, M., Logozzo, F., O'Hearn, P.W.: Scaling Static Analyses at Facebook. Commun. ACM **62**(8), 62–70 (2019), https://doi.org/10.1145/3338112
17. Duarte, A., Korovin, K.: Implementing Superposition in iProver (System Description). In: IJCAR, LNCS, vol. 12167, pp. 388–397, Springer (2020), https://doi.org/10.1007/978-3-030-51054-1_24
18. Elad, N., Rain, S., Immerman, N., Kovács, L., Sagiv, M.: Summing up Smart Transitions. In: CAV, LNCS, vol. 12759, pp. 317–340, Springer (2021), https://doi.org/10.1007/978-3-030-81685-8_15
19. Graham-Lengrand, S.: Yices-QS 2022, an extension of Yices for quantified satisfiability. https://smt-comp.github.io/2022/system-descriptions/YicesQS.pdf (2022)
20. Gurfinkel, A.: Program Verification with Constrained Horn Clauses (Invited Paper). In: CAV, LNCS, vol. 13371, pp. 19–29, Springer (2022), https://doi.org/10.1007/978-3-031-13185-1_2
21. Hoenicke, J., Schindler, T.: SMTInterpol with Resolution Proofs. https://smt-comp.github.io/2022/system-descriptions/smtinterpol.pdf (2022)
22. Kapur, D., Narendran, P.: Double-exponential Complexity of Computing a Complete Set of AC-Unifiers. In: LICS, pp. 11–21, IEEE Computer Society (1992), https://doi.org/10.1109/LICS.1992.185515
23. Korovin, K., Tsiskaridze, N., Voronkov, A.: Conflict Resolution. In: CP, LNCS, vol. 5732, pp. 509–523, Springer (2009), https://doi.org/10.1007/978-3-642-04244-7_41
24. Korovin, K., Voronkov, A.: An AC-Compatible Knuth-Bendix Order. In: CADE, LNCS, vol. 2741, pp. 47–59, Springer (2003), https://doi.org/10.1007/978-3-540-45085-6_5
25. Korovin, K., Voronkov, A.: Integrating Linear Arithmetic into Superposition Calculus. In: CSLs, LNCS, vol. 4646, pp. 223–237, Springer (2007), https://doi.org/10.1007/978-3-540-74915-8_19
26. Kovács, L., Voronkov, A.: First-Order Theorem Proving and Vampire. In: CAV, LNCS, vol. 8044, pp. 1–35, Springer (2013), https://doi.org/10.1007/978-3-642-39799-8_1

27. de Moura, L.M., Bjørner, N.S.: Efficient E-Matching for SMT Solvers. In: CADE, LNCS, vol. 4603, pp. 183–198, Springer (2007), https://doi.org/10.1007/978-3-540-73595-3_13
28. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: TACAS, LNCS, vol. 4963, pp. 337–340, Springer (2008), https://doi.org/10.1007/978-3-540-78800-3_24
29. de Moura, L.M., Jovanovic, D.: A Model-Constructing Satisfiability Calculus. In: VMCAI, LNCS, vol. 7737, pp. 1–12, Springer (2013), https://doi.org/10.1007/978-3-642-35873-9_1
30. Nieuwenhuis, R., Rubio, A.: Paramodulation-Based Theorem Proving. In: Handbook of Automated Reasoning, pp. 371–443, Elsevier and MIT Press (2001), https://doi.org/10.1016/b978-044450813-3/50009-6
31. Passmore, G.O.: Some Lessons Learned in the Industrialization of Formal Methods for Financial Algorithms. In: FM, LNCS, vol. 13047, pp. 717–721, Springer (2021), https://doi.org/10.1007/978-3-030-90870-6_39
32. Reger, G., Bjørner, N.S., Suda, M., Voronkov, A.: AVATAR Modulo Theories. In: GCAI, EPiC Series in Computing, vol. 41, pp. 39–52, EasyChair (2016), https://doi.org/10.29007/k6tp
33. Reger, G., Suda, M., Voronkov, A.: Unification with Abstraction and Theory Instantiation in Saturation-Based Reasoning. In: TACAS, LNCS, vol. 10805, pp. 3–22, Springer (2018), https://doi.org/10.1007/978-3-319-89960-2_1
34. Reger, G., Suda, M., Voronkov, A., Kovács, L., Bhayat, A., Gleiss, B., Hajdu, M., Hozzova, P., Evgeny Kotelnikov, J.R., Rawson, M., Riener, M., Robillard, S., Schoisswohl, J.: Vampire 4.7-SMT System Description. `https://smt-comp.github.io/2022/system-descriptions/Vampire.pdf` (2022)
35. Reynolds, A., King, T., Kuncak, V.: Solving Quantified Linear Arithmetic by Counterexample-Guided Instantiation. FMSD **51**(3), 500–532 (2017), https://doi.org/10.1007/s10703-017-0290-y
36. Schulz, S., Cruanes, S., Vukmirovic, P.: Faster, Higher, Stronger: E 2.3. In: CADE, LNCS, vol. 11716, pp. 495–507, Springer (2019), https://doi.org/10.1007/978-3-030-29436-6_29
37. Voronkov, A.: AVATAR: The Architecture for First-Order Theorem Provers. In: CAV, LNCS, vol. 8559, pp. 696–710, Springer (2014), https://doi.org/10.1007/978-3-319-08867-9_46
38. Waldmann, U.: Extending Reduction Orderings to ACU-Compatible Reduction Orderings. Inf. Process. Lett. **67**(1), 43–49 (1998), https://doi.org/10.1016/S0020-0190(98)00084-2
39. Waldmann, U.: Superposition for Divisible Torsion-Free Abelian Groups. In: CADE, LNCS, vol. 1421, pp. 144–159, Springer (1998), https://doi.org/10.1007/BFb0054257
40. Yamada, A., Winkler, S., Hirokawa, N., Middeldorp, A.: AC-KBO Revisited. Theory Pract. Log. Program. **16**(2), 163–188 (2016), https://doi.org/10.1017/S1471068415000083