Hierarchic Superposition With Weak Abstraction

Peter Baumgartner[1] and Uwe Waldmann[2]

[1] NICTA⋆ and Australian National University, Canberra, Australia
Peter.Baumgartner@nicta.com.au
[2] MPI für Informatik, Saarbrücken, Germany
uwe@mpi-inf.mpg.de

Abstract. Many applications of automated deduction require reasoning in first-order logic modulo background theories, in particular some form of integer arithmetic. A major unsolved research challenge is to design theorem provers that are "reasonably complete" even in the presence of free function symbols ranging into a background theory sort. The hierarchic superposition calculus of Bachmair, Ganzinger, and Waldmann already supports such symbols, but, as we demonstrate, not optimally. This paper aims to rectify the situation by introducing a novel form of clause abstraction, a core component in the hierarchic superposition calculus for transforming clauses into a form needed for internal operation. We argue for the benefits of the resulting calculus and provide a new completeness result for the fragment where all background-sorted terms are ground.

1 Introduction

Many applications of automated deduction require reasoning modulo background theories, in particular some form of integer arithmetic. Developing corresponding automated reasoning systems that are also able to deal with quantified formulas has recently been an active area of research. One major line of research is concerned with extending (SMT-based) solvers [15] for the quantifier-free case by instantiation heuristics for quantifiers [10,11, e. g.]. Another line of research is concerned with adding black-box reasoners for specific background theories to first-order automated reasoning methods (resolution [4,12,1], sequent calculi [17], instantiation methods [9,5,6], etc). In both cases, a major unsolved research challenge is to provide reasoning support that is "reasonably complete" in practice, so that the systems can be used more reliably for both proving theorems and finding counterexamples.

In [4], Bachmair, Ganzinger, and Waldmann introduced the hierarchical superposition calculus as a generalization of the superposition calculus for black-box style theory reasoning. Their calculus works in a framework of hierarchic specifications. It tries to prove the unsatisfiability of a set of clauses with respect to interpretations that extend a background model such as the integers with linear arithmetic conservatively, that is, without identifying distinct elements of old sorts ("confusion") and without adding new elements to old sorts ("junk"). While confusion can be detected by first-order theorem

⋆ NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

proving techniques, junk can not – in fact, the set of logical consequences of a hierarchic specifications is usually not recursively enumerable. Refutational completeness can therefore only be guaranteed if one restricts oneself to sets of formulas where junk can be excluded a priori. The property introduced by Bachmair, Ganzinger, and Waldmann for this purpose is called "sufficient completeness with respect to simple instances". Given this property, their calculus is refutationally complete for clause sets that are fully abstracted (i. e., where no literal contains both foreground and background symbols). Unfortunately their full abstraction rule may destroy sufficient completeness with respect to simple instances. We show that this problem can be avoided by using a new form of clause abstraction and a suitably modified hierarchical superposition calculus. Since the new hierarchical superposition calculus is still refutationally complete and the new abstraction rule is guaranteed to preserve sufficient completeness with respect to simple instances, the new combination is strictly more powerful than the old one.

In practice, sufficient completeness is a rather restrictive property. While there are application areas where one knows in advance that every input is sufficiently complete, in most cases this does not hold. As a user of an automated theorem prover, one would like to see a best effort behaviour: The prover might for instance try to *make* the input sufficiently complete by adding further theory axioms. In the calculus by Bachmair, Ganzinger, and Waldmann, however, this does not help at all: The restriction to a particular kind of instantiations ("simple instances") renders theory axioms essentially unusable in refutations. We show that this can be prevented by introducing two kinds of variables of the background theory sorts instead of one, that can be instantiated in different ways, making our calculus significantly "more complete" in practice. We also include a definition rule in the calculus that can be used to establish sufficient completeness by linking foreground terms to background parameters, thus allowing the background prover to reason about these terms.

The following trivial example demonstrates the problem. Consider the clause set $N = \{C\}$ where $C = \mathsf{f}(1) < \mathsf{f}(1)$. Assume that the background theory is integer arithmetic and that f is an integer-sorted operator from the foreground (free) signature. Intuitively, one would expect N to be unsatisfiable. However, N is not sufficiently complete, and it admits models in which $\mathsf{f}(1)$ is interpreted as some junk element $¢$, an element of the domain of the integer sort that is not a numeric constant. So both the calculus in [4] and ours are excused to not find a refutation. To fix that, one could add an instance $C' = \neg(\mathsf{f}(1) < \mathsf{f}(1))$ of the irreflexivity axiom $\neg(x < x)$. The resulting set $N' = \{C, C'\}$ is (trivially) sufficiently complete as it has no models at all. However, the calculus in [4] is not helped by adding C', since the abstracted version of N' is again not sufficiently complete and admits a model that interprets $\mathsf{f}(1)$ as $¢$. Our abstraction mechanism always preserves sufficient completeness and our calculus will find a refutation.

With this example one could think that replacing the abstraction mechanism in [4] with ours gives all the advantages of our calculus. This is not the case, however. Let $N'' = \{C, \neg(x < x)\}$ be obtained by adding the more realistic axiom $\neg(x < x)$. The set N'' is still sufficiently complete with our approach thanks to having two kinds of variables at disposal, but it is not sufficiently complete in the sense of [4]. Indeed, in that calculus adding background theory axioms *never* helps to gain sufficient completeness, as variables there have only one kind.

Another alternative to make N sufficiently complete is by adding a clause that forces $\mathsf{f}(1)$ to be equal to some background domain element. For instance, one can add a “definition” for $\mathsf{f}(1)$, that is, a clause $\mathsf{f}(1) \approx \alpha$, where α is a fresh symbolic constant belonging to the background signature (a “parameter”). The set $N''' = \{C, \mathsf{f}(1) \approx \alpha\}$ is sufficiently complete and it admits refutations with both calculi. The definition rule in our calculus mentioned above will generate this definition automatically. Moreover, the set N belongs to a syntactic fragment for which we can guarantee not only sufficient completeness (by means of the definition rule) but also refutational completeness.

We present the new calculus in detail and provide a general completeness result, modulo compactness of the background theory, and a specific completeness result for clause sets over ground background-sorted terms that does not require compactness. We also report on experiments with a prototypical implementation on the TPTP problem library. Complete proofs, which are omitted here for lack of space, can be found in [7].

Related Work. The relation with the predecessor calculus in [4] is discussed above and also further below. What we say there also applies to other developments rooted in that calculus, [1, e. g.]. The specialised version of hierarchic superposition in [13] will be discussed in Sect. 7 below. The resolution calculus in [12] has built-in inference rules for linear (rational) arithmetic, but is complete only under restrictions that effectively prevent quantification over rationals. Earlier work on integrating theory reasoning into model evolution [5,6] lacks the treatment of background-sorted foreground function symbols. The same applies to the sequent calculus in [17], which treats linear arithmetic with built-in rules for quantifier elimination. The instantiation method in [9] requires an answer-complete solver for the background theory to enumerate concrete solutions of background constraints, not just a decision procedure. All these approaches have in common that they integrate specialized reasoning for background theories into a general first-order reasoning method. A conceptually different approach consists in using first-order theorem provers as (semi-)decision procedures for specific theories in DPLL(T)(-like) architectures [14,2,8]. Notice that in this context the theorem provers do not need to reason modulo background theories themselves, and indeed they don’t. The calculus and system in [14], for instance, integrates superposition and DPLL(T). From DPLL(T) it inherits splitting of ground non-unit clauses into their unit components, which determines a (backtrackable) model candidate M. The superposition inference rules are applied to elements from M and a current clause set F. The superposition component guarantees refutational completeness for pure first-order clause logic. Beyond that, for clauses containing background-sorted variables, (heuristic) instantiation is needed. Instantiation is done with ground terms that are provably equal w. r. t. the equations in M to some ground term in M in order to advance the derivation. The limits of that method can be illustrated with an (artificial but simple) example. Consider the unsatisfiable clause set $\{i \leq j \vee \mathsf{P}(i+1, x) \vee \mathsf{P}(j+2, x),\ i \leq j \vee \neg\mathsf{P}(i+3, x) \vee \neg\mathsf{P}(j+4, x)\}$ where i and j are integer-sorted variables and x is a foreground-sorted variable. Neither splitting into unit clauses, superposition calculus rules, nor instantiation applies, and so the derivation gets stuck with an inconclusive result. By contrast, the clause set belongs to a fragment that entails sufficient completeness (“no background-sorted foreground function symbols”) and hence is refutable by our calculus. On the other hand, heuristic instantiation does have a place in our calculus, but we leave that for future work.

2 Signatures, Clauses, and Interpretations

We work in the context of standard many-sorted logic with first-order signatures comprised of sorts and operator (or function) symbols of given arities over these sorts. A *signature* is a pair $\Sigma = (\Xi, \Omega)$, where Ξ is a set of *sorts* and Ω is a set of *operator symbols* over Ξ. If $\mathcal{X}$ is a set of sorted variables with sorts in Ξ, then the set of well-sorted terms over $\Sigma = (\Xi, \Omega)$ and $\mathcal{X}$ is denoted by $\mathrm{T}_\Sigma(\mathcal{X})$; T_Σ is short for $\mathrm{T}_\Sigma(\emptyset)$. We require that Σ is a *sensible* signature, i. e., that T_Σ has no empty sorts. As usual, we write $t[u]$ to indicate that the term u is a (not necessarily proper) subterm of the term t. The position of u in t is left implicit.

A *Σ-equation* is an unordered pair (s, t), usually written $s \approx t$, where s and t are terms from $\mathrm{T}_\Sigma(\mathcal{X})$ of the same sort. For simplicity, we use equality as the only predicate in our language. Other predicates can always be encoded as a function into a set with one distinguished element, so that a non-equational atom is turned into an equation $P(t_1, \dots, t_n) \approx true_P$; this is usually abbreviated by $P(t_1, \dots, t_n)$.[3] A *literal* is an equation $s \approx t$ or a negated equation $\neg(s \approx t)$, also written as $s \not\approx t$. A *clause* is a multiset of literals, usually written as a disjunction; the empty clause, denoted by $\square$ is a contradiction. If F is a term, equation, literal or clause, we denote by *vars*(F) the set of variables that occur in F. We say F is *ground* if $\mathrm{vars}(F) = \emptyset$

A *substitution* σ is a mapping from variables to terms that is sort respecting, that is, maps each variable $x \in \mathcal{X}$ to a term of the same sort. Substitutions are homomorphically extended to terms as usual. We write substitution application in postfix form. A term s is an *instance* of a term t if there is a substitution σ such that $t\sigma = s$. All these notions carry over to equations, literals and clauses in the obvious way.

The *domain* of a substitution σ is the set $\mathrm{dom}(\sigma) = \{x \mid x \neq x\sigma\}$. We work with substitutions with finite domains only, written as $\sigma = [x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$ where $\mathrm{dom}(\sigma) = \{x_1, \dots, x_n\}$. A *ground substitution* is a substitution that maps every variable in its domain to a ground term. A *ground instance* of F is obtained by applying some ground substitution with domain (at least) $\mathrm{vars}(F)$ to it.

A *Σ-interpretation* I consists of a Ξ-sorted family of carrier sets $\{I_\xi\}_{\xi \in \Xi}$ and of a function $I_f : I_{\xi_1} \times \cdots \times I_{\xi_n} \to I_{\xi_0}$ for every $f : \xi_1 \dots \xi_n \to \xi_0$ in Ω. The *interpretation* t^I of a ground term t is defined recursively by $f(t_1, \dots, t_n)^I = I_f(t_1^I, \dots, t_n^I)$ for $n \geq 0$. An interpretation I is called *term-generated*, if every element of an I_ξ is the interpretation of some ground term of sort ξ. An interpretation I is said to *satisfy* a ground equation $s \approx t$, if s and t have the same interpretation in I; it is said to *satisfy* a negated ground equation $s \not\approx t$, if s and t do not have the same interpretation in I. The interpretation I *satisfies* a ground clause C if at least one of the literals of C is satisfied by I. We also say that a ground clause C is *true in I*, if I satisfies C; and that C is *false in I*, otherwise. A term-generated interpretation I is said to *satisfy* a non-ground clause C if it satisfies all ground instances $C\sigma$; it is called a *model* of a set N of clauses, if it satisfies all clauses of N.[4] We abbreviate the fact that I is a model of N by $I \models N$; $I \models C$ is short for

[3] Without loss of generality we assume that there exists a distinct sort for every predicate.

[4] This restriction to term-generated interpretations as models is possible since we are only concerned with refutational theorem proving, i. e., with the derivation of a contradiction.

$I \models \{C\}$. We say that N *entails* N', and write $N \models N'$, if every model of N is a model of N'; $N \models C$ is short for $N \models \{C\}$.

If $\mathcal{J}$ is a class of Σ-interpretations, a Σ-clause or clause set is called *$\mathcal{J}$-satisfiable* if at least one $I \in \mathcal{J}$ satisfies the clause or clause set; otherwise it is called *$\mathcal{J}$-unsatisfiable*.

A *specification* is a pair $SP = (\Sigma, \mathcal{J})$, where Σ is a signature and $\mathcal{J}$ is a class of term-generated Σ-interpretations called *models* of the specification SP. We assume that $\mathcal{J}$ is closed under isomorphisms.

We say that a class of Σ-interpretations $\mathcal{J}$ or a specification $(\Sigma, \mathcal{J})$ is *compact*, if every infinite set of Σ-clauses that is $\mathcal{J}$-unsatisfiable has a finite subset that is also $\mathcal{J}$-unsatisfiable.

3 Hierarchic Theorem Proving

In hierarchic theorem proving, we consider a scenario in which a general-purpose foreground theorem prover and a specialized background prover cooperate to derive a contradiction from a set of clauses. In the sequel, we will usually abbreviate "foreground" and "background" by "FG" and "BG".

The BG prover accepts as input sets of clauses over a *BG signature* $\Sigma_{\mathrm{B}} = (\Xi_{\mathrm{B}}, \Omega_{\mathrm{B}})$. Elements of Ξ_{B} and Ω_{B} are called *BG sorts* and *BG operators*, respectively. We fix an infinite set X_{B} of *BG variables* of sorts in Ξ_{B}. Every BG variable has (is labeled with) a *kind*, which is either *"abstraction"* or *"ordinary"*. Terms over Σ_{B} and X_{B} are called *BG terms*. A BG term is called *pure*, if it does not contain ordinary variables; otherwise it is *impure*. These notions apply analogously to equations, literals, clauses, and clause sets.

The BG prover decides the satisfiability of Σ_{B}-clause sets with respect to a *BG specification* $(\Sigma_{\mathrm{B}}, \mathcal{B})$, where $\mathcal{B}$ is a class of term-generated Σ_{B}-interpretations called *BG models*. We assume that $\mathcal{B}$ is closed under isomorphisms.

In most applications of hierarchic theorem proving, the set of BG operators Ω_{B} contains a set of distinguished constant symbols $\Omega_{\mathrm{B}}^{D} \subseteq \Omega_{\mathrm{B}}$ that has the property that $d_1^I \neq d_2^I$ for any two distinct $d_1, d_2 \in \Omega_{\mathrm{B}}^{D}$ and every BG model $I \in \mathcal{B}$. We refer to these constant symbols as *(BG) domain elements*.

While we permit arbitrary classes of BG models, in practice the following three cases are most relevant:

(1) $\mathcal{B}$ consists of exactly one Σ_{B}-interpretation (up to isomorphism), say, the integer numbers over a signature containing all integer constants as domain elements and $\leq, <, +, -$ with the expected arities. In this case, $\mathcal{B}$ is trivially compact; in fact, a set N of Σ_{B}-clauses is $\mathcal{B}$-unsatisfiable if and only if some clause of N is $\mathcal{B}$-unsatisfiable.
(2) Σ_{B} is extended by an infinite number of *parameters*, that is, additional constant symbols. While all interpretations in $\mathcal{B}$ share the same carrier sets $\{I_\xi\}_{\xi \in \Xi_{\mathrm{B}}}$ and interpretations of non-parameter symbols, parameters may be interpreted freely by arbitrary elements of the appropriate I_ξ. The class $\mathcal{B}$ obtained in this way is in general not compact; for instance the infinite set of clauses $\{\, n \leq \beta \mid n \in \mathbb{N} \,\}$, where β is a parameter, is unsatisfiable in the integers, but every finite subset is satisfiable.
(3) Σ_{B} is again extended by parameters, however, $\mathcal{B}$ is now the class of all interpretations that satisfy some first-order theory, say, the first-order theory of linear integer

arithmetic.[5] Since $\mathcal{B}$ corresponds to a first-order theory, compactness is recovered. It should be noted, however, that $\mathcal{B}$ contains non-standard models, so that for instance the clause set $\{n \leq \beta \mid n \in \mathbb{N}\}$ is now satisfiable (e. g., $\mathbb{Q} \times \mathbb{Z}$ with a lexicographic ordering is a model).

The FG theorem prover accepts as inputs clauses over a signature $\Sigma = (\Xi, \Omega)$, where $\Xi_{\mathrm{B}} \subseteq \Xi$ and $\Omega_{\mathrm{B}} \subseteq \Omega$. The sorts in $\Xi_{\mathrm{F}} = \Xi \setminus \Xi_{\mathrm{B}}$ and the operator symbols in $\Omega_{\mathrm{F}} = \Omega \setminus \Omega_{\mathrm{B}}$ are called *FG sorts* and *FG operators*. Again we fix an infinite set $\mathcal{X}_{\mathrm{F}}$ of *FG variables* of sorts in Ξ_{F}. All FG variables have the kind "ordinary". We define $\mathcal{X} = \mathcal{X}_{\mathrm{B}} \cup \mathcal{X}_{\mathrm{F}}$.

In examples we use $\{0, 1, 2, \dots\}$ to denote BG domain elements, $\{+, -, <, \leq\}$ to denote (non-parameter) BG operators, and the possibly subscripted letters $\{x, y\}$, $\{X, Y\}$, $\{\alpha, \beta\}$, and $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}, \mathsf{g}\}$ to denote ordinary variables, abstraction variables, parameters, and FG operators, respectively. The letter ζ denotes an ordinary variable or an abstraction variable.

We call a term in $\mathrm{T}_{\Sigma}(\mathcal{X})$ a *FG term*, if it is not a BG term, that is, if it contains at least one FG operator or FG variable (and analogously for equations, literals, or clauses). We emphasize that for a FG operator $\mathsf{f} : \xi_1 \dots \xi_n \to \xi_0$ in Ω_{F} any of the ξ_i may be a BG sort, and that consequently FG terms may have BG sorts.

If I is a Σ-interpretation, the *restriction* of I to Σ_{B}, written $I|_{\Sigma_{\mathrm{B}}}$, is the Σ_{B}-interpretation that is obtained from I by removing all carrier sets I_{ξ} for $\xi \in \Xi_{\mathrm{F}}$ and all functions I_f for $f \in \Omega_{\mathrm{F}}$. Note that $I|_{\Sigma_{\mathrm{B}}}$ is not necessarily term-generated even if I is term-generated. In hierarchic theorem proving, we are only interested in Σ-interpretations that extend some model in $\mathcal{B}$ and neither collapse any of its sorts nor add new elements to them, that is, in Σ-interpretations I for which $I|_{\Sigma_{\mathrm{B}}} \in \mathcal{B}$. We call such a Σ-interpretation a *$\mathcal{B}$-interpretation*.

Let N and N' be two sets of Σ-clauses. We say that N *entails N' relative to $\mathcal{B}$* (and write $N \models_{\mathcal{B}} N'$), if every model of N whose restriction to Σ_{B} is in $\mathcal{B}$ is a model of N'. Note that $N \models_{\mathcal{B}} N'$ follows from $N \models N'$. If $N \models_{\mathcal{B}} \Box$, we call N *$\mathcal{B}$-unsatisfiable*; otherwise, we call it *$\mathcal{B}$-satisfiable*.[6]

Our goal in refutational hierarchic theorem proving is to check whether a given set of Σ-clauses N is false in all $\mathcal{B}$-interpretations, or equivalently, whether N is $\mathcal{B}$-unsatisfiable.

We say that a substitution is *simple* if it maps every abstraction variable in its domain to a *pure* BG term. For example, $[x \mapsto 1 + Y + \alpha]$, $[X \mapsto 1 + Y + \alpha]$ and $[x \mapsto \mathsf{f}(1)]$ all are simple, whereas $[X \mapsto 1 + y + \alpha]$ and $[X \mapsto \mathsf{f}(1)]$ are not. Let F be a clause or (possibly infinite) clause set. By $sgi(F)$ we denote the set of simple ground instances of F, that is, the set of all ground instances of (all clauses in) F obtained by simple ground substitutions. Standard unification algorithms can be modified in a straightforward way for computing simple mgus. Note that a simple mgu can map an ordinary variable to an abstraction variable but not vice versa, as ordinary variables are not pure BG terms.

[5] To satisfy the technical requirement that all interpretations in $\mathcal{B}$ are term-generated, we assume that in this case Σ_{B} is suitably extended by an infinite set of constants (or by one constant and one unary function symbol) that are not used in any input formula or theory axiom.

[6] If $\Sigma = \Sigma_{\mathrm{B}}$, this definition coincides with the definition of satisfiability w. r. t. a class of interpretations that was given in Sect. 2. A set N of BG clauses is $\mathcal{B}$-satisfiable if and only if some interpretation of $\mathcal{B}$ is a model of N.

For a BG specification $(\Sigma_{\mathrm{B}}, \mathcal{B})$, we define $\mathrm{GndTh}(\mathcal{B})$ as the set of all ground Σ_{B}-formulas that are satisfied by every $I \in \mathcal{B}$.

Definition 3.1 (Sufficient completeness). *A Σ-clause set N is* sufficiently complete w. r. t. simple instances *iff for every Σ-model J of $sgi(N) \cup \mathrm{GndTh}(\mathcal{B})$*[7] *and every ground BG-sorted FG term s there is a ground BG term t such that $J \models s \approx t$.*[8]

For brevity, we will from now on omit the phrase "w. r. t. simple instances" and speak only of "sufficient completeness". It should be noted, though, that our definition differs from the classical definition of sufficient completeness in the literature on algebraic specifications.

4 Orderings

A *hierarchic reduction ordering* is a strict, well-founded ordering on terms that is compatible with contexts, i. e., $s \succ t$ implies $u[s] \succ u[t]$, and stable under simple substitutions, i. e., $s \succ t$ implies $s\sigma \succ t\sigma$ for every simple σ. In the rest of this paper we assume such a hierarchic reduction ordering $\succ$ that satisfies all of the following: (i) $\succ$ is total on ground terms, (ii) $s \succ d$ for every domain element d and every ground term s that is not a domain element, and (iii) $s \succ t$ for every ground FG term s and every ground BG term t. These conditions are easily satisfied by an LPO with an operator precedence in which FG operators are larger than BG operators and domain elements are minimal with, for example, $\cdots \succ -2 \succ 2 \succ -1 \succ 1 \succ 0$ to achieve well-foundedness.

Condition (iii) and stability under *simple* substitutions together justify to always order $s \succ X$ where s is a non-variable FG term and X is an abstraction variable. By contrast, $s \succ x$ can only hold if $x \in \mathrm{vars}(s)$. Intuitively, the combination of hierarchic reduction orderings and abstraction variables affords ordering more terms.

The ordering $\succ$ is extended to literals over terms by identifying a positive literal $s \approx t$ with the multiset $\{s, t\}$, a negative literal $s \not\approx t$ with $\{s, s, t, t\}$, and using the multiset extension of $\succ$. Clauses are compared by the multiset extension of $\succ$, also denoted by $\succ$.

The non-strict orderings $\succeq$ are defined as $s \succeq t$ iff $s \succ t$ or $s = t$ (the latter is multiset equality in case of literals and clauses). We say that a literal L is *maximal* (*strictly maximal*) in a clause $L \vee C$ iff there is no $K \in C$ with $K \succ L$ ($K \succeq L$).

5 Weak Abstraction

To refute an input set of Σ-clauses, hierarchic superposition calculi derive BG clauses from them and pass the latter to a BG prover. In order to do this, some separation of the FG and BG vocabulary in a clause is necessary. The technique used for this separation is known as *abstraction*: One (repeatedly) replaces some term t in a clause by a new variable and adds a disequations to the clause, so that $C[t]$ is converted into the equivalent clause $\zeta \not\approx t \vee C[\zeta]$, where ζ is a new (abstraction or ordinary) variable.

[7] In contrast to [4], we include $\mathrm{GndTh}(\mathcal{B})$ in the definition of sufficient completeness. (This is independent of the abstraction method used; it would also have been useful in [4].)

[8] Note that J need *not* be a $\mathcal{B}$-interpretation.

The calculus by Bachmair, Ganzinger, and Waldmann [4] works on "fully abstracted" clauses: Background terms occuring below a FG operator or in an equation between a BG and a FG term or vice versa are abstracted out until one arrives at a clause in which no literal contains both FG and BG operator symbols.

A problematic aspect of any kind of abstraction is that it tends to increase the number of incomparable terms in a clause, which leads to an undesirable growth of the search space of a theorem prover. For instance, if we abstract out the subterms t and t' in a ground clause $\mathsf{f}(t) \approx \mathsf{g}(t')$, we get $x \not\approx t \vee y \not\approx t' \vee \mathsf{f}(x) \approx \mathsf{g}(y)$, and the two new terms $\mathsf{f}(x)$ and $\mathsf{g}(y)$ are incomparable in any reduction ordering. The approach used in [4] to reduce this problem is to consider only instances where BG-sorted variables are mapped to BG terms: In the terminology of the current paper, all BG-sorted variables in [4] have the kind "abstraction". This means that, in the example above, we obtain the two terms $\mathsf{f}(X)$ and $\mathsf{g}(Y)$. If we use an LPO with a precedence in which f is larger than g and g is larger than every BG operator, then for every simple ground substitution τ, $\mathsf{f}(X)\tau$ is strictly larger that $\mathsf{g}(Y)\tau$, so we can still consider $\mathsf{f}(X)$ as the only maximal term in the literal.

The advantage of full abstraction is that this clause structure is preserved by all inference rules. There is a serious drawback, however: Consider the clause set $N = \{\, 1{+}\mathsf{c} \not\approx 1{+}\mathsf{c}\,\}$. Since N is ground, we have $\mathrm{sgi}(N) = N$, and since $\mathrm{sgi}(N)$ is unsatisfiable, N is trivially sufficiently complete. Full abstraction turns N into $N' = \{\, X \not\approx \mathsf{c} \vee 1 + X \not\approx 1 + X\,\}$. In the simple ground instances of N', X is mapped to all pure BG terms. However, there are Σ-interpretations of $\mathrm{sgi}(N')$ in which c is interpreted differently from any pure BG term, so $\mathrm{sgi}(N') \cup \mathrm{GndTh}(\mathcal{B})$ does have a Σ-model and N' is no longer sufficiently complete. In other words, the calculus of [4] is refutationally complete for clause sets that are fully abstracted and sufficiently complete, but full abstraction may destroy sufficient completeness. (In fact, the calculus is not able to refute N'.)

The problem that we have seen is caused by the fact that full abstraction replaces FG terms by abstraction variables, which may not be mapped to FG terms later on. The obvious fix would be to use ordinary variables instead of abstraction variables whenever the term to be abstracted out is not a pure BG term, but as we have seen above, this would increase the number of incomparable terms and it would therefore be detrimental to the performance of the prover.

Full abstraction is a property that is stronger than actually necessary for the completeness proof of [4]. In fact, it was claimed in a footnote in [4] that the calculus could be optimized by abstracting out only non-variable BG terms that occur below a FG operator. This is incorrect, however: Using this abstraction rule, neither our calculus nor the calculus of [4] would not be able to refute $\{\, 1 + 1 \approx 2,\ (1 + 1) + \mathsf{c} \not\approx 2 + \mathsf{c}\,\}$, even though this set is unsatisfiable and trivially sufficiently complete. We need a slightly different abstraction rule to avoid this problem:

Definition 5.1. *A BG term t occurring in a clause C is called* target term *if t is neither a domain element nor a variable*[9] *and if C has the form $C[f(s_1, \dots, t, \dots, s_n)]$, where f is a FG operator or at least one of the s_i is a FG or impure BG term.*

[9] The reason why it is permissible to treat domain elements in a special way will become clear in Sect. 6.

A clause is called weakly abstracted *if it does not have any target terms.*

The weakly abstracted version of a clause *is the clause that is obtained by exhaustively replacing $C[t]$ by*

- *$C[X] \vee X \not\approx t$, where X is a new abstraction variable, if t is a pure target term in C,*
- *$C[y] \vee y \not\approx t$, where y is a new ordinary variable, if t is an impure target term in C.*

The weakly abstracted version of C is denoted by $\mathrm{abstr}(C)$.

For example, weak abstraction of the clause $\mathsf{g}(1, \alpha, \mathsf{f}(1)+(\alpha+1), z) \approx \beta$ yields $\mathsf{g}(1, X, \mathsf{f}(1)+Y, z) \approx \beta \vee X \not\approx \alpha \vee Y \not\approx \alpha + 1$. Note that the terms 1, $\mathsf{f}(1) + (\alpha + 1)$, z, and β are not abstracted out: 1 is a domain element; $\mathsf{f}(1)+(\alpha+1)$ has a BG sort, but it is not a BG term; z is a variable; and β is not a subterm of a FG term. The clause $\mathsf{write}(\mathsf{a}, 2, \mathsf{read}(\mathsf{a}, 1)+1) \approx \mathsf{b}$ is already weakly abstracted. Every BG clause is trivially weakly abstracted.

Proposition 5.2. *If N is a set of clauses and N' is obtained from N by replacing one or more clauses by their weakly abstracted versions, then $sgi(N)$ and $sgi(N')$ are equivalent and N' is sufficiently complete whenever N is.*

In contrast to full abstraction, the weak abstraction rule does not require abstraction of FG terms (which can destroy sufficient completeness, if done using abstraction variables, and which is detrimental to the performance of a prover if done using ordinary variables). BG terms are usually abstracted out using abstraction variables. The exception are BG terms that are impure, i. e., that contain ordinary variables themselves. In this case, we cannot avoid to use ordinary variables for abstraction, otherwise, we might again destroy sufficient completeness. For example, the clause set $\{\, \mathsf{P}(1 + y),\ \neg\mathsf{P}(1 + \mathsf{c}) \,\}$ is sufficiently complete. If we used an abstraction variable instead of an ordinary variable to abstract out the impure subterm $1+y$, we would get $\{\, \mathsf{P}(X) \vee X \not\approx 1+y,\ \neg\mathsf{P}(1+\mathsf{c}) \,\}$, which is no longer sufficiently complete.

In input clauses (that is, before abstraction), BG-sorted variables may be declared as "ordinary" or "abstraction". As we have seen above, using abstraction variables can reduce the search space; on the other hand, abstraction variables may be detrimental to sufficient completeness. Consider the following example: The set of clauses $N = \{\, \neg\mathsf{f}(x) > \mathsf{g}(x) \vee \mathsf{h}(x) \approx 1, \neg\mathsf{f}(x) \leq \mathsf{g}(x) \vee \mathsf{h}(x) \approx 2, \neg\mathsf{h}(x) > 0 \,\}$ is unsatisfiable w. r. t. linear integer arithmetic, but since it is not sufficiently complete, the hierarchic superposition calculus does not detect the unsatisfiability. Adding the clause $X > Y \vee X \leq Y$ to N does not help: Since the abstraction variables X and Y may not be mapped to the FG terms $\mathsf{f}(x)$ and $\mathsf{g}(x)$ in a simple ground instance, the resulting set is still not sufficiently complete. However, if we add the clause $x > y \vee x \leq y$, the set of clauses becomes (vacuously) sufficiently complete and its unsatisfiability is detected.

One might wonder whether it is also possible to gain anything if the abstraction process is performed using ordinary variables instead of abstraction variables. The following proposition shows that this is not the case:

Proposition 5.3. *Let N be a set of clauses, let N' be the result of weak abstraction of N as defined above, and let N'' be the result of weak abstraction of N where all newly introduced variables are ordinary variables. Then $sgi(N')$ and $sgi(N'')$ are equivalent and $sgi(N')$ is sufficiently complete if and only if $sgi(N'')$ is.*

6 Base Inference System

An *inference system* $\mathcal{I}$ is a set of inference rules. By an $\mathcal{I}$ *inference* we mean an instance of an inference rule from $\mathcal{I}$ such that all conditions are satisfied.

The *base inference system* $\mathrm{HSP}_{\mathrm{Base}}$ of the hierarchic superposition calculus consists of the inference rules Equality resolution, Negative superposition, Positive superposition, Equality factoring, and Close defined below.[10] All inference rules are applicable only to weakly abstracted premise clauses.

$$\textsf{Equality resolution}\quad \frac{s \not\approx t \vee C}{\mathrm{abstr}(C\sigma)}$$

if (i) neither s nor t is a pure BG term, (ii) σ is a simple mgu of s and t, and (iii) $(s \not\approx t)\sigma$ is maximal in $(s \not\approx t \vee C)\sigma$.[11]

For example, Equality resolution is applicable to $1 + \mathsf{c} \not\approx 1 + x$ with the simple mgu $[x \mapsto \mathsf{c}]$, but it is not applicable to $1 + \alpha \not\approx 1 + x$, since $1 + \alpha$ is a pure BG term.

$$\textsf{Negative superposition}\quad \frac{l \approx r \vee C \qquad s[u] \not\approx t \vee D}{\mathrm{abstr}((s[r] \not\approx t \vee C \vee D)\sigma)}$$

if (i) neither l nor u is a pure BG term, (ii) u is not a variable, (iii) σ is a simple mgu of l and u, (iv) $r\sigma \not\gtrsim l\sigma$, (v) $(l \approx r)\sigma$ is strictly maximal in $(l \approx r \vee C)\sigma$, (vi) $t\sigma \not\gtrsim s\sigma$, and (vii) $(s \not\approx t)\sigma$ is maximal in $(s \not\approx t \vee D)\sigma$.

$$\textsf{Positive superposition}\quad \frac{l \approx r \vee C \qquad s[u] \approx t \vee D}{\mathrm{abstr}((s[r] \approx t \vee C \vee D)\sigma)}$$

if (i) neither l nor u is a pure BG term, (ii) u is not a variable, (iii) σ is a simple mgu of l and u, (iv) $r\sigma \not\gtrsim l\sigma$, (v) $(l \approx r)\sigma$ is strictly maximal in $(l \approx r \vee C)\sigma$, (vi) $t\sigma \not\gtrsim s\sigma$, and (vii) $(s \not\approx t)\sigma$ is strictly maximal in $(s \approx t \vee D)\sigma$.

$$\textsf{Equality factoring}\quad \frac{l \approx r \vee s \approx t \vee C}{\mathrm{abstr}((l \approx t \vee r \not\approx t \vee C)\sigma)}$$

where (i) neither l nor s is a pure BG term, (ii) σ is a simple mgu of l and s, (iii) $(l \approx r)\sigma$ is maximal in $(l \approx r \vee s \approx t \vee C)\sigma$, (iv) $r\sigma \not\gtrsim l\sigma$, and (v) $t\sigma \not\gtrsim s\sigma$.

$$\textsf{Close}\quad \frac{C_1 \quad \cdots \quad C_n}{\square}$$

[10] With weak abstraction, it is *not* possible to replace Equality factoring by Factoring and Merging paramodulation. The inference system can be extended by selection functions, but only negative FG literals in clauses that do not contain ordinary BG variables may be selected.

[11] As in [4], it is possible to strengthen condition (iii) by requiring that there exists some simple ground substitution ψ such that $(s \not\approx t)\sigma\psi$ is maximal in $(s \not\approx t \vee C)\sigma\psi$ (and analogously for the other inference rules).

if $C_1, \dots, C_n$ are BG clauses and $\{C_1, \dots, C_n\}$ is $\mathcal{B}$-unsatisfiable, i. e., no interpretation in $\mathcal{B}$ is a Σ_B-model of $\{C_1, \dots, C_n\}$.

Notice that Close is not restricted to take *pure* BG clauses only. The reason is that also impure BG clauses admit simple ground instances that are pure.

In contrast to [4], the inference rules above include an explicit weak abstraction in their conclusion. Without it, conclusions would not be weakly abstracted in general. For example Positive superposition applied to the weakly abstracted clauses $\mathsf{f}(X) \approx 1 \lor X \not\approx \alpha$ and $\mathsf{P}(\mathsf{f}(1)+1)$ would then yield $\mathsf{P}(1+1) \lor 1 \not\approx \alpha$, whose P-literal is not weakly abstracted. Additionally, the side conditions of our rules differ somewhat from the corresponding rules of [4], this is due on the one hand to the presence of impure BG terms (which must sometimes be treated like FG terms), and on the other hand to the fact that, after weak abstraction, literals may still contain both FG and BG operators.

The inference rules are supplemented by a redundancy criterion, that is, a mapping $\mathcal{R}_{\mathrm{Cl}}$ from sets of formulae to sets of formulae and a mapping $\mathcal{R}_{\mathrm{Inf}}$ from sets of formulae to sets of inferences that are meant to specify formulae that may be removed from N and inferences that need not be computed. ($\mathcal{R}_{\mathrm{Cl}}(N)$ need not be a subset of N and $\mathcal{R}_{\mathrm{Inf}}(N)$ will usually also contain inferences whose premises are not in N.)

Definition 6.1. *A pair $\mathcal{R} = (\mathcal{R}_{\mathrm{Inf}}, \mathcal{R}_{\mathrm{Cl}})$ is called a* redundancy criterion *(*with respect to an inference system $\mathcal{I}$ and a consequence relation $\models$*), if the following conditions are satisfied for all sets of formulae N and N':*

(i) $N \setminus \mathcal{R}_{\mathrm{Cl}}(N) \models \mathcal{R}_{\mathrm{Cl}}(N)$.
(ii) If $N \subseteq N'$, then $\mathcal{R}_{\mathrm{Cl}}(N) \subseteq \mathcal{R}_{\mathrm{Cl}}(N')$.
(iii) If ι is an inference and its conclusion is in N, then $\iota \in \mathcal{R}_{\mathrm{Inf}}(N)$.
(iv) If $N' \subseteq \mathcal{R}_{\mathrm{Cl}}(N)$, then $\mathcal{R}_{\mathrm{Inf}}(N) \subseteq \mathcal{R}_{\mathrm{Inf}}(N \setminus N')$.

Inferences in $\mathcal{R}_{\mathrm{Inf}}(N)$ and formulae in $\mathcal{R}_{\mathrm{Cl}}(N)$ are said to be redundant *with respect to N.*

To define a redundancy criterion for $\mathrm{HSP}_{\mathrm{Base}}$ and to prove the refutational completeness of the calculus, we use the same approach as in [4] and relate $\mathrm{HSP}_{\mathrm{Base}}$ inferences to the corresponding ground inferences in the standard superposition calculus SSP [16].

Let N be a set of ground clauses. We define $\mathcal{R}^{\mathcal{S}}_{\mathrm{Cl}}(N)$ to be the set of all clauses C such that there exist clauses $C_1, \dots, C_n \in N$ that are smaller than C with respect to $\succ$ and $C_1, \dots, C_n \models C$. We define $\mathcal{R}^{\mathcal{S}}_{\mathrm{Inf}}(N)$ to be the set of all ground SSP inferences ι such that either a premise of ι is in $\mathcal{R}^{\mathcal{S}}_{\mathrm{Cl}}(N)$ or else C_0 is the conclusion of ι and there exist clauses $C_1, \dots, C_n \in N$ that are smaller with respect to $\succ^{\mathrm{c}}$ than the maximal premise of ι and $C_1, \dots, C_n \models C_0$. It is well known that ground SSP together with $(\mathcal{R}^{\mathcal{S}}_{\mathrm{Inf}}, \mathcal{R}^{\mathcal{S}}_{\mathrm{Cl}})$ is refutationally complete.

Let ι be an $\mathrm{HSP}_{\mathrm{Base}}$ inference with premises $C_1, \dots, C_n$ and conclusion $\mathrm{abstr}(C)$, where the clauses $C_1, \dots, C_n$ have no variables in common. Let ι' be a ground SSP inference with premises $C'_1, \dots, C'_n$ and conclusion C'. If σ is a simple substitution such that $C' = C\sigma$ and $C'_i = C_i\sigma$ for all i, and if none of the C'_i is a BG clause, then ι' is called a *simple ground instance* of ι. The set of all simple ground instances of an inference ι is denoted by $\mathrm{sgi}(\iota)$.

Definition 6.2. *Let N be a set of weakly abstracted clauses. We define $\mathcal{R}^{\mathcal{H}}_{\mathrm{Inf}}(N)$ to be the set of all inferences ι such that either ι is not a Close inference and $sgi(\iota) \subseteq \mathcal{R}^{\mathcal{S}}_{\mathrm{Inf}}(sgi(N) \cup$*

$\mathrm{GndTh}(\mathcal{B})$)*, or else* ι *is a Close inference and* $\Box \in N$*. We define* $\mathcal{R}^{\mathcal{H}}_{\mathrm{Cl}}(N)$ *to be the set of all weakly abstracted clauses* C *such that* $sgi(C) \subseteq \mathcal{R}^{\mathcal{S}}_{\mathrm{Cl}}(sgi(N) \cup \mathrm{GndTh}(\mathcal{B})) \cup \mathrm{GndTh}(\mathcal{B})$.[12]

To prove that $\mathrm{HSP}_{\mathrm{Base}}$ and $\mathcal{R}^{\mathcal{H}} = (\mathcal{R}^{\mathcal{H}}_{\mathrm{Inf}}, \mathcal{R}^{\mathcal{H}}_{\mathrm{Cl}})$ are refutationally complete for sets of weakly abstracted Σ-clauses and compact BG specifications $(\Sigma_{\mathrm{B}}, \mathcal{B})$, we use the same technique as in [4]:

First we show that $\mathcal{R}^{\mathcal{H}}$ is a redundancy criterion with respect to $\models_{\mathcal{B}}$, and that a set of clauses remains sufficiently complete if new clauses are added or if redundant clauses are deleted. The proofs for both properties are similar to the corresponding ones in [4]; the differences are due, on the one hand, to the fact that we include $\mathrm{GndTh}(\mathcal{B})$ in the redundancy criterion and in the definition of sufficient completeness, and, on the other hand, to the explicit abstraction steps in our inference rules.

We then encode arbitrary term-generated Σ_{B}-interpretation by sets of unit ground clauses in the following way: Let $I \in \mathcal{B}$ be a term-generated Σ_{B}-interpretation. For every Σ_{B}-ground term t let $m(t)$ be the smallest ground term of the congruence class of t in I. We define a rewrite system E'_I by $\mathrm{E}'_I = \{\, t \to m(t) \mid t \in \mathrm{T}_{\Sigma},\ t \neq m(t) \,\}$. Obviously, E'_I is terminating, right-reduced, and confluent. Now let E_I be the set of all rules $l \to r$ in E'_I such that l is not reducible by $\mathrm{E}'_I \setminus \{l \to r\}$. It is fairly easy to prove that E'_I and E_I define the same set of normal forms, and from this we can conclude that E_I and E'_I induce the same equality relation on ground Σ_{B}-terms. We identify E_I with the set of clauses $\{\, t \approx t' \mid t \to t' \in \mathrm{E}_I \,\}$. Let D_I be the set of all clauses $t \not\approx t'$, such that t and t' are distinct ground Σ_{B}-terms in normal form with respect to E_I.

Let N be a set of weakly abstracted clauses and $I \in \mathcal{B}$ be a term-generated Σ_{B}-interpretation, then N_I denotes the set $\mathrm{E}_I \cup \mathrm{D}_I \cup \{\, C\sigma \mid \sigma \text{ simple, reduced with respect to } \mathrm{E}_I,\ C \in N,\ C\sigma \text{ ground} \,\}$.

Theorem 6.3. *Let* $I \in \mathcal{B}$ *be a term-generated* Σ_{B}*-interpretation and let* N *be a set of weakly abstracted* Σ*-clauses. If* I *satisfies all BG clauses in* $sgi(N)$ *and* N *is saturated with respect to* $\mathrm{HSP}_{\mathrm{Base}}$ *and* $\mathcal{R}^{\mathcal{H}}$*, then* N_I *is saturated with respect to* SSP *and* $\mathcal{R}^{\mathcal{S}}$*.*

The crucial property of abstracted clauses that is needed in the proof of this theorem is that there are no superposition inferences between clauses in E_I and FG ground instances $C\sigma$, or in other words, that all FG terms occurring in ground instances $C\sigma$ are reduced w. r. t. E_I. Abstracting out FG terms as in [4] is not necessary to achieve this goal, and domain elements can also be excluded as target terms in Def. 5.1: Since two different domain elements must always be interpreted differently in I and since domain elements are smaller in the term ordering than any ground term that is not a domain element, every domain element is the smallest term in its congruence class. Domain elements occurring within FG terms are therefore trivially irreducible by E_I, so it is unnecessary to abstract them out.

If N is saturated with respect to $\mathrm{HSP}_{\mathrm{Base}}$ and $\mathcal{R}^{\mathcal{H}}$ and does not contain the empty clause, then Close cannot be applicable to N. If $(\Sigma_{\mathrm{B}}, \mathcal{B})$ is compact, this implies that there is some term-generated Σ_{B}-interpretation $I \in \mathcal{B}$ that satisfies all BG clauses in

[12] In contrast to [4], we include $\mathrm{GndTh}(\mathcal{B})$ in the redundancy criterion. (This is independent of the abstraction method used; it would also have been useful in [4].)

sgi(N). Hence, by Thm. 6.3, the set of *reduced simple* ground instances of N has a model that also satisfies $\mathrm{E}_I \cup \mathrm{D}_I$. Sufficient completeness allows us to show that this is in fact a model of *all* ground instances of clauses in N and that I is its restriction to Σ_B:

Theorem 6.4. *If the BG specification* $(\Sigma_\mathrm{B}, \mathcal{B})$ *is compact, then* $\mathrm{HSP}_\mathrm{Base}$ *and* $\mathcal{R}^\mathcal{H}$ *are refutationally complete for all sets of clauses that are sufficiently complete.*

We do not spell out in detail theorem proving *processes* here, because the well-known framework of standard resolution [3] can be readily instantiated with our calculus. In particular, it justifies the following version of a generic simplification rule for clause sets.

$$\mathsf{Simp}\ \frac{N \cup \{C\}}{N \cup \{D\}}$$

if (i) D is weakly abstracted, (ii) $\mathrm{GndTh}(\mathcal{B}) \cup N \cup \{C\} \models D$, and (iii) C is redundant w. r. t. $N \cup \{D\}$.

Condition (ii) is needed for soundness, and condition (iii) is needed for completeness. The Simp rule covers the usual simplification rules of the standard superposition calculus, such as demodulation by unit clauses and deletion of tautologies and (properly) subsumed clauses. It also covers simplification of arithmetic terms, e. g., replacing a subterm $(2 + 3) + \alpha$ by $5 + \alpha$ and deleting an unsatisfiable BG literal $5 + \alpha < 4 + \alpha$ from a clause. Any clause of the form $C \vee \zeta \not\approx d$ where d is domain element can be simplified to $C[\zeta \mapsto d]$.

7 Sufficient Completeness by Define

In this section we introduce an additional inference rule, Define. It augments the $\mathrm{HSP}_\mathrm{Base}$ inference system with complementary functionality: while the $\mathrm{HSP}_\mathrm{Base}$ inference system will derive a contradiction if the input clause set is inconsistent and sufficiently complete, the Define rule may turn input clause sets that are not sufficiently complete into sufficiently complete ones. Technically, the Define rule derives “definitions” of the form $t \approx \alpha$, where t is a ground BG-sorted FG term and α is a parameter. This way, sufficient completenessis is achieved “locally” for t, by forcing t to be equal to some element of the carrier set of the proper sort, denoted by the parameter α. For economy of reasoning, definitions are introduced only on a by-need basis, when t appears in a current clause, and $t \approx \alpha$ is used to simplify that clause immediately.

We need one more preliminary definition before introducing Define formally.

Definition 7.1 (Unabstracted clause). *A clause is* unabstracted *if it does not contain any disequation* $\zeta \not\approx t$ *between a variable* ζ *and a term* t *unless* $t \neq \zeta$ *and* $\zeta \in \mathit{vars}(t)$.

Every clause can be unabstracted by repeatedly replacing $C \vee \zeta \not\approx t$ by $C[\zeta \mapsto t]$ whenever $t = \zeta$ or $\zeta \notin \mathrm{vars}(t)$. By unabstr($C$) we denote an unabstracted version of C that can be obtained this way.[13] If $t = t[\zeta_1, \ldots, \zeta_n]$ is a term in C and ζ_i is finally

[13] In general, unabstraction does not yield a unique result. All results are equivalent, however, and we can afford to select any one and disregard the others.

instantiated to t_i, we denote its unabstracted version $t[t_1, \ldots, t_n]$ by unabstr(t, C). For a clause set N let unabstr$(N) = \{\text{unabstr}(C) \mid C \in N\}$.

$$\textsf{Define}\ \frac{N \cup \{L[t[\zeta_1, \ldots, \zeta_n]] \vee D\}}{N \cup \{\text{abstr}(t[t_1, \ldots, t_n] \approx \alpha_{t[t_1,\ldots,t_n]}),\ \text{abstr}(L[\alpha_{t[t_1,\ldots,t_n]}] \vee D)\}}$$

if (i) $t[\zeta_1, \ldots, \zeta_n]$ is a minimal BG-sorted non-variable term with a toplevel FG operator, (ii) $t[t_1, \ldots, t_n] = \text{unabstr}(t[\zeta_1, \ldots, \zeta_n], L[t[\zeta_1, \ldots, \zeta_n]] \vee D)$, (iii) $t[t_1, \ldots, t_n]$ is ground, and (iv) $\alpha_{t[t_1,\ldots,t_n]}$ is a parameter, uniquely determined by the term $t[t_1, \ldots, t_n]$.

In condition (i), by minimality we mean that no proper subterm of $t[\zeta_1, \ldots, \zeta_n]$ is a BG-sorted non-variable term with a toplevel FG operator. In effect, the Define rule eliminates such terms inside-out. Conditions (iii) and (iv) are needed for soundness. Notice the Define-rule preserves $\mathcal{B}$-satisfiability, not $\mathcal{B}$-equivalence. In our main application, Thm. 7.5 below, every ζ_i will always be an abstraction variable.

Example 7.2. Consider the weakly abstracted clauses $\mathsf{P}(0)$, $\mathsf{f}(x) > 0 \vee \neg\mathsf{P}(x)$, $\mathsf{Q}(\mathsf{f}(x))$, $\neg\mathsf{Q}(x) \vee 0 > x$. Suppose $\neg\mathsf{P}(x)$ is maximal in the second clause. By superposition between the first two clauses we derive $\mathsf{f}(0) > 0$. With Define we obtain $\mathsf{f}(0) \approx \alpha_{\mathsf{f}(0)}$ and $\alpha_{\mathsf{f}(0)} > 0$, the latter replacing $\mathsf{f}(0) > 0$. From the third clause and $\mathsf{f}(0) \approx \alpha_{\mathsf{f}(0)}$ we obtain $\mathsf{Q}(\alpha_{\mathsf{f}(0)})$, and with the fourth clause $0 > \alpha_{\mathsf{f}(0)}$. Finally we apply Close to $\{\alpha_{\mathsf{f}(0)} > 0,\ 0 > \alpha_{\mathsf{f}(0)}\}$. □

In practice, it is interesting to identify conditions under which sufficient completeness can be established by means of Define *and* compactness poses no problems, so that a complete calculus results. The *ground BG-sorted term fragment (GBT fragment)* discussed below is one such case.

A clause set N belongs to the GBT fragment iff every clause $C \in N$ is a GBT clause, that is, all BG-sorted terms in C are ground. To get the desired completeness result we need to establish that the Define rule preserves the GBT property.

Lemma 7.3. *If* unabstr(N) *belongs to the GBT fragment and N' is obtained from N by a* Define *inference, then* unabstr(N') *also belongs to the GBT fragment.*

Below we will equip the HSP calculus with a specific strategy that first applies Define exhaustively before the derivation proper starts. In that, it may be beneficial to also apply Simp. But then, Simp needs to preserve the GBT property, too. Because this does not hold at the outset, and to make sure Split is well-behaved in the subsequent derivation, we have to make certain (mild) assumptions.

Definition 7.4. *Let $>_{fin}$ be any strict (partial) term ordering such that for every ground BG term s only finitely many ground BG terms t with $s >_{fin} t$ exist.*[14] *We say that a* Simp *inference with premise $N \cup \{C\}$ and conclusion $N \cup \{D\}$ is* suitable (for the GBT fragment) *iff (i) if* unabstr(C) *is a GBT clause then* unabstr(D) *is a GBT clause, (ii) for every BG term t occuring in* unabstr(D) *there is a BG term $s \in$* unabstr(C) *such that $s \geq_{fin} t$, and (iii) every term t in D contains a BG-sorted FG operator only at toplevel position, if at all. We say the* Simp *inference rule is* suitable *iff every* Simp *inference is.*

[14] A KBO with appropriate weights can be used for $>_{\text{fin}}$.

Expected simplification techniques like demodulation, subsumption deletion and evaluation of BG subterms are all covered by suitable Simp rules. The latter is possible because simplifications are not only decreasing w. r. t. $>$ but *additionally* also decreasing w. r. t. $\succeq_{\mathrm{fin}}$, as expressed in condition (ii). Without it, e. g., the clause $\mathsf{P}(1 + 1, 0)$ would admit infinitely many simplified versions $\mathsf{P}(2, 0)$, $\mathsf{P}(2, 0 + 0)$, $\mathsf{P}(2, 0 + (0 + 0))$, etc. Condition (i) makes sure that also Simp preserves GBT clauses. Condition (iii) is needed to make sure that no new BG terms are generated in derivations.

As said, we need to equip the HSP calculus with a specific strategy. Assume a suitable Simp rule and let N be a set of GBT clauses. By N^{pre} we denote any clause set obtained by a derivation of the form $(N_0 = \mathrm{abstr}(N)), N_1, \ldots, (N_k = N^{\mathrm{pre}})$ with the inference rules Define and Simp only, and such that every $C \in N^{\mathrm{pre}}$ either does not contain any BG-sorted FG operator or $\mathrm{unabstr}(C)$ is a ground positive unit clause of the form $\mathsf{f}(t_1, \ldots, t_n) \approx t$ where f is a BG-sorted FG operator and $t_1, \ldots, t_n, t$ do not contain BG-sorted FG operators.

For all GBT clause sets N, thanks to the effect of the Define rule and Lemma 7.3, all offending occurrences of BG-sorted FG terms in N can stepwisely be eliminated until a clause set N^{pre} results. Thanks to the additional assumptions about Simp we obtain the following main result on the GBT fragment.

Theorem 7.5. *The* HSP *calculus with a suitable* Simp *inference rule is refutationally complete for the ground BG-sorted term fragment. More precisely, if a set N of GBT clauses is $\mathcal{B}$-unsatisfiable then there is a refutation of N^{pre} without the* Define *rule.*

Because unabstraction can also be applied to fully abstracted clauses, it is possible to equip the hierarchic superposition calculus of [4] with a correspondingly modified Define rule and get Theorem 7.5 in that context as well.

In [13] it has been shown how to use hierarchic superposition as a decision procedure for ground clause sets (and for Horn clause sets with constants and variables as the only FG terms). Their method preprocesses the given clause set by "basification", a process that removes BG-sorted FG terms similarly as our Define rule. The resulting clause set then is fully abstracted and hierarchic superposition is applied. Certain modifications of the inference rules make sure derivations always terminate. Simplification is restricted to subsumption deletion. The effect of basification is achieved in our calculus by the Define rule. Moreover, for GBT clause sets, by Theorem 7.5, Define needs to be applied as preprocessing only. Applying Define beyond that for non-GBT clause sets can still be useful. Ex. 7.2, for instance, cannot be solved with basification during preprocessing. The fragment of [13] is a further restriction of the GBT fragment. We expect we can get decidability results for that fragment with similar techniques.

8 Implementation and Experiments

We have implemented the HSP calculus and carried out experiments with the TPTP Library [18]. Our implementation, "Beagle", is intended as a testbed for rapidly trying out theoretical ideas for their practical viability.[15] Beagle is in an early stage of

[15] `http://users.cecs.anu.edu.au/~baumgart/systems/beagle/`

development. Nevertheless it is a full implementation of HSP and accepts TPTP formulas over linear integer arithmetic ("TFF formulas", see [19]). The BG reasoner is a quantifier elimination procedure for linear integer arithmetic (LIA) based on Cooper's algorithm; it is called with all current BG clauses as inputs (with caching of simplified formulas) whenever a new BG clause has been derived. The HSP calculus itself is implemented in a straightforward way. Fairness is achieved through a combination of measuring clause lengths, depths and their derivation-age. Implemented simplification rules are evaluation of ground parameter-free BG terms and literals, expressing literals with the predicate symbols $\geq$ and $>$ in terms of $<$ and $\leq$, demodulation by unit clauses, proper subsumption deletion, and removing a positive literal L from a clause in presence of a unit clause that instantiates to the complement of L. Prover options allow the user to enable/disable the Define rule and to add certain LIA-valid clauses over ordinary variables. Unit clauses like $-(-x) \approx x$, $(x+(-y))+y \approx x$, $x+0 \approx x$, $x*0 \approx 0$, $\neg(x < x)$, etc, are always helpful as demodulators. Transitivity clauses for $<$ and $\leq$ are helpful sometimes. Optionally, a split rule can be enabled for branching out into complementary unit clauses if they simplify some current clause. Dependency-directed backtracking is used for search space pruning then. Beagle also implements the previous calculus [4], with abstraction variables only, full abstraction, and optionally a modified Define rule that uses full abstraction instead of weak abstraction. We refer to this setting by "HSPFA" below, and by "HSPWA" for the new calculus.

Beagle is implemented in Scala. The choice of a slow programming language and, more severely, the absence of any form of term indexing limit Beagle's applicability to small problems only. Indeed, Beagle's performance on problems that require significant combinatorial search is poor. For example, the propositional pigeonhole problem with 8 pigeons takes more than two hours, SPASS solves it in under 4 seconds using settings to get a comparable calculus and proof procedure (including splitting). Nevertheless we tried Beagle on all first-order problems from the TPTP library (version 5.4.0) over linear integer arithmetic. The experiments were run on a MacBook Pro with a 2.4 GHz Intel Core 2 Duo processor. Here is our summary, by problem category.

ARI. Relevant are 223 problems. Many ARI problems are very simple, but roughly half of them are non-trivial by including integer-sorted non-constant FG function symbols and free predicates over the integers. The most difficult solved problems have a rating of 0.88. Beagle times out after 60 seconds on one problem (ARI184=1), terminates with an undecided result on one satisfiable problem (ARI603=1) and solves all other 221 problems correctly, 10 non-theorems and 211 theorems. All but three solved problems can be solved very quickly, the other three below 20 seconds. The Define rule is essential for HSPWA in 14 cases, for HSPFA in 17 cases. The problem ARI186=1 cannot be solved by HSPFA.

GEG. The five relevant problems are variations of each other. They deal with traversing weighted graphs and computing with the (sum of the) weights along paths. All five problems are solved within 60 seconds, the hardest problem has a rating of 0.67. For GEG025=1 the use of additional LIA-valid axioms, in particular transitivity of $\leq$ is essential. For two problems Define is essential, and one problem is unsolvable by HSPFA.

PUZ. The only relevant problem (PUZ133=2) is not solved.

NUM. The only non-easy problem that is solvable is NUM858=1 (rating 0.56), which is not solvable by HSPFA. All easy problems are solved easily, but for one problem Define is essential.

SEV/HWV. Six problems of SEV are relevant, about sets, stemming from a software verification context. Only SEV421=1 and SEV422=1 can be solved, in 3 and 100 seconds, respectively. The problem SEV422=1 cannot be solved by HSPFA. Solving the SEV-problems is dominated by pure foreground reasoning. The same applies to HWV, where no problem is solved.

SWV/SWW. Only one problem can be solved, SWV997=1 (rating 0.44), in 8 seconds. All other problems are too big to be converted into CNF in reasonable time. The same holds for all SWW problems.

SYO. Of the four problems, SYO521=1, SYO523=1 (rating 0.67) and SYO524=1 are solvable. The latter only with HSPWA, the Define rule and auxiliary lemmas.

The TSTP web page contains individual solutions to TPTP problems for various provers. About 12 provers are applicable to problems over linear integer arithmetic. Beagle solves 22 such problems with a rating of 0.60 or higher. Each of these problems can typically be solved by four or less systems, with CVC3, Princess, SPASS+T and Z3 the most reliable ones. There are five problems that only Princess and Beagle solve.

9 Conclusions

The main theoretical contribution of this paper is an improved variant of the hierarchic superposition calculus. The improvements over its predecessor [4] are grounded in a different form of “abstracted” clauses, the clauses the calculus works with internally. Because of that, a modified completeness proof is required. We have argued informally for the benefits over the old calculus in [4]. They concern making the calculus “more complete” in practice. It is hard to quantify that exactly in a general way, as completeness is impossible to achieve in presence of background-sorted foreground function symbols (e. g., “car” of integer-sorted lists). To compensate for that to some degree, we have reported on initial experiments with a prototypical implementation on the TPTP problem library. These experiments clearly indicate the benefits of our concepts, in particular the definition rule and the possibility of adding background theory axioms. They also confirm advantages of the new calculus over the old, the former solves strictly more more problems than the latter (and is never slower on the common set). Certainly more experimentation and an improved implementation is needed to also solve bigger-sized problems with a larger combinatorial search space.

We have also obtained a specific completeness result for clause sets over ground background-sorted terms and that does not require compactness. As far as we know this result is new. It is loosely related to the decidability results in [13], as discussed in Sect. 7. It is also loosely related to results in SMT-based theorem proving. For instance, the method in [11] deals with the case that variables appear only as arguments of, in our words, foreground operators. It works by ground-instantiating all variables in order to being able to use an SMT-solver for the quantifier-free fragment. Under certain conditions, finite ground instantiation is possible and the method is complete, otherwise it is complete only modulo compactness of the background theory (as expected). Treating different fragments, the theoretical results are mutually non-subsuming with ours. Yet, on the fragment they consider we could adopt their technique of finite ground instantiation before applying Thm. 7.5 (when it applies). However, according to Thm. 7.5 our calculus needs instantiation of *background-sorted variables only*, this way keeping reasoning with foreground-sorted terms on the first-order level, as usual with superposition.

References

1. E. Althaus, E. Kruglov, and C. Weidenbach. Superposition modulo linear arithmetic SUP(LA). In *FroCos*, 2009, *LNCS 5749*, pp. 84–99. Springer.
2. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 10(1), 2009.
3. L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In *Handbook of Automated Reasoning*. North Holland, 2001.
4. L. Bachmair, H. Ganzinger, and U. Waldmann. Refutational theorem proving for hierarchic first-order theories. *Appl. Algebra Eng. Commun. Comput*, 5:193–212, 1994.
5. P. Baumgartner, A. Fuchs, and C. Tinelli. ME(LIA) – Model Evolution With Linear Integer Arithmetic Constraints. In *LPAR*, 2008, *LNAI 5330*, pp. 258–273. Springer.
6. P. Baumgartner and C. Tinelli. Model evolution with equality modulo built-in theories. In *CADE-23*, 2011, *LNAI 6803*, pp. 85–100. Springer.
7. P. Baumgartner and U. Waldmann. Hierarchic superposition with weak abstraction. Technical Report MPI-I-2013-RG1-002, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2013. `http://www.mpi-inf.mpg.de/publications/index.html`.
8. M. P. Bonacina, C. Lynch, and L. M. de Moura. On deciding satisfiability by theorem proving with speculative inferences. *J. Autom. Reasoning*, 47(2):161–189, 2011.
9. H. Ganzinger and K. Korovin. Theory instantiation. In *LPAR*, 2006, *LNCS 4246*, pp. 497–511. Springer.
10. Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In F. Pfenning, ed., *CADE-21*, 2007, *LNCS 4603*, pp. 167–182. Springer.
11. Y. Ge and L. M. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *CAV*, 2009, *LNCS 5643*, pp. 306–320. Springer.
12. K. Korovin and A. Voronkov. Integrating linear arithmetic into superposition calculus. In *CSL*, 2007, *LNCS 4646*, pp. 223–237. Springer.
13. E. Kruglov and C. Weidenbach. Superposition decides the first-order logic fragment over ground theories. *Mathematics in Computer Science*, pp. 1–30, 2012.
14. L. M. de Moura and N. Bjørner. Engineering DPLL(T) + saturation. In *IJCAR*, 2008, *LNCS 5195*, pp. 475–490. Springer.
15. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.
16. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning*, pp. 371–443. Elsevier and MIT Press, 2001.
17. P. Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In *LPAR*, 2008, *LNAI 5330*, pp. 274–289. Springer.
18. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.
19. G. Sutcliffe, S. Schulz, K. Claessen, and P. Baumgartner. The TPTP typed first-order form with arithmetic. In *LPAR*, 2012, *LNAI 7180*, pp. 406–419. Springer.