Math.Comput.Sci. (2012) 6:427–456
DOI 10.1007/s11786-012-0135-4

Mathematics in Computer Science

Superposition Decides the First-Order Logic Fragment Over Ground Theories

Evgeny Kruglov · Christoph Weidenbach

Received: 15 December 2011 / Revised: 24 September 2012 / Accepted: 23 October 2012 / Published online: 19 December 2012

Abstract The hierarchic superposition calculus over a theory T, called SUP(T), enables sound reasoning on the hierarchic combination of a theory T with full first-order logic, FOL(T). If a FOL(T) clause set enjoys a sufficient completeness criterion, the calculus is even complete. Clause sets over the ground fragment of FOL(T) are not sufficiently complete, in general. In this paper we show that any clause set over the ground FOL(T) fragment can be transformed into a sufficiently complete one, and prove that SUP(T) terminates on the transformed clause set, hence constitutes a decision procedure provided the existential fragment of the theory T is decidable. Thanks to the hierarchic design of SUP(T), the decidability result can be extended beyond the ground case. We show SUP(T) is a decision procedure for the non-ground FOL fragment plus a theory T, if every non-constant function symbol from the underlying FOL signature ranges into the sort of the theory T, and every term of the theory sort is ground. Examples for T are in particular decidable fragments of arithmetic.

Keywords Theorem proving · Combination of theories · Decision procedure · Arithmetic

Mathematics Subject Classification (2010) Primary 68T15; Secondary 03B25

1 Introduction

Recent progress in automated reasoning has greatly encouraged numerous applications in soft- and hardware verification and the analysis of complex systems. The applications typically require to determine the validity/unsatisfiability of formulae over the combination of the free first-order logic with some background theories. Often such formulae include quantifiers and even quantifier alternations.

Satisfiability modulo theories (SMT) techniques are well-known to be efficient and scalable approaches towards reasoning over combinations of theories. SMT solvers [23,24] are restricted to ground formulae, or to formulae that can be effectively and efficiently reduced to a ground fragment. In comparison, superposition-based inference

E. Kruglov (✉) · C. Weidenbach
Universität des Saarlandes, Max-Planck-Institut für Informatik,
Campus E1 4, 66123 Saarbrücken, Germany
e-mail: ekruglov@mpi-inf.mpg.de

C. Weidenbach
e-mail: weidenbach@mpi-inf.mpg.de

systems (SUP) are complete for full first-order logic. After clause normal form translation they reason on equational clauses with universally quantified variables. The superposition calculus can be turned into a decision procedure for a number of decidable first-order fragments, e.g., [2,6,19,21], and is a good basis for actually proving decidability of fragments and obtaining efficient implementations. There are now several calculi available combining full first-order logic with a theory T: the hierarchic theorem proving approach SUP(T) for any theory T [7], an instance of this approach for the combination with linear arithmetic SUP(LA) [1] and non-linear arithmetic SUP(NLA) [12], superposition and chaining for totally ordered divisible abelian groups [27], superposition with linear arithmetic integrated [22], and model evolution extended with linear arithmetic [8].

The DPLL($\Gamma + \mathcal{T}$) calculus of [9] is a tight integration of DPLL, specialized SMT solvers for $\mathcal{T}$, and first-order reasoning Γ based on Superposition and Resolution. For example, DPLL($\Gamma + \mathcal{T}$) can be turned into a decision procedure for theories that axiomatize type systems relevant to program checking. The crucial feature of such theories is that they need to be essentially finite: they have one unary function symbol whose range is finite. The general form of initial clause sets DPLL($\Gamma + \mathcal{T}$) deals with, is $\mathcal{R} \uplus \mathcal{P}$, where $\mathcal{P}$ is ground formula containing $\mathcal{T}$ symbols, and $\mathcal{R}$ is a set of non-ground clauses without occurrences of $\mathcal{T}$ symbols axiomatizing an application specific theory.

The hierarchic theorem proving approach [7] offers an abstract completeness result for the combination via a sufficient completeness criterion that goes beyond ground problems or specific theories, as studied for example in local theory extensions [17,20]. Furthermore, it enables the handling of the theory T part in a modular way that can be eventually implemented via efficient off-the-shelf solvers (in contrast to [22,27]). In this paper we show that the SUP(T) calculus is a decision procedure for the ground FOL(T) fragment and can be extended for the FOL fragment including FOL variables plus a theory T, where every non-constant function symbol from the underlying FOL signature ranges into the sort of the theory T, and all terms of the theory sort are ground. This actually closes a gap for SUP(T) that was already shown a decision procedure for certain FOL(T) fragments with variables [15,16], whereas the ground case was not yet solved.

The paper is organized as follows: first we give an introduction to the most important notions, Sect. 2, and present the SUP(T) calculus, Sect. 3. In Sect. 4, we prove SUP(T) to be a decision procedure for the ground FOL(T) fragment. In Sect. 5 we show that SUP(T) can decide beyond the ground fragment. The paper ends with a discussion of the achieved results and a presentation of further directions of research, Sect. 6.

2 Preliminaries

For the presentation of the superposition calculus, we refer to the notation and notions of [28], where for the specific notions serving the hierarchical combination of a theory T with first-order logic, we refer to [1,7,12]. We will not introduce the full apparatus of the hierarchical theorem proving technology, but just the notions that enable us to state the inference rules, the sufficient completeness criterion, the completeness result and the redundancy notion.

Multisets A *multiset* over a set X is a mapping M from X to the non-negative integer (natural) numbers. Intuitively, $M(x)$ specifies the number of occurrences of $x \in X$ in M. We say that x is an element of M if $M(x) > 0$. The union, intersection, and difference of multisets are defined by the identities: $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$, $(M_1 \cap M_2)(x) = \min\{M_1(x),\ M_2(x)\}$, and $(M_1 \backslash M_2)(x) = \min\{0,\ M_1(x) - M_2(x)\}$. A multiset M is finite if the set $\{x \mid M(x) > 0\}$ is finite. If M is a multiset and N a set, we write $M \subseteq N$ to indicate that every element of M is an element of N, and use $M \backslash N$ to denote the multiset M' for which $M'(x) = 0$ for any x in N, and $M'(x) = M(X)$, otherwise. We often use sequence- or set-like notation to denote multisets and write, for instance, M_1, M_2 instead of $M_1 \cup M_2$, or M, L instead of $M \cup \{L\}$. For example, $\{F, G, G\}$ denotes the multiset M over formulae for which $M(F) = 1$, $M(G) = 2$, and $M(H) = 0$, for any other formula H.

Signatures and Terms A *many-sorted signature* is a pair $\Sigma = (\mathcal{S}, \Omega)$ where $\mathcal{S}$ is a finite set of sort symbols and Ω a set of operator symbols over $\mathcal{S}$. In addition to the sets $\mathcal{S}$ and Ω, we assume a set of sorted variables $\mathcal{X}$, such that for every single sort $S \in \mathcal{S}$ there is a countably infinite set of different variables of the sort S in $\mathcal{X}$. The set of well-sorted terms over a variable set $\mathcal{X}$ is denoted by $T_\Omega(\mathcal{X})$; the set of all terms of a specific sort S is denoted by

$T_\Omega(S, \mathcal{X})$. Given a term $t \in T_\Omega(\mathcal{X})$, we write $sort(t)$ to denote the sort of t. The function $vars$ maps a term to the set of all variables occurring in it. A term t is called ground if $vars(t) = \emptyset$. The set of all ground terms built over Ω is denoted by $T_\Omega(\emptyset)$, or shortly T_Ω; the set of all ground terms of a specific sort S is denoted by $T_\Omega(S)$.

A *position* is a word over the natural numbers. The set $\rho(f(t_1, \ldots, t_n))$ of positions of a given term $f(t_1, \ldots, t_n)$ is defined as follows: (i) the empty word ε is a position in any term t and $t/\varepsilon = t$, (ii) if $t/p' = f(t_1, \ldots, t_n)$, then $p = p'.i$ is a position in t for all $i = 1, \ldots, n$, and $t/p = t_i$. The *length* $|p|$ of a position p is its length as a sequence: $|\varepsilon| = 0, |p'.i| = |p'| + 1$. The *depth* of a term is the maximal length of a position in the term: $depth(t) = \max(|p| \mid p \in \rho(t))$. We write $top(t)$ to denote the top symbol of t, and $t[s]_p$ to denote that $s = t/p$ is a subterm of t at some position $p \in \rho(t)$; we may simply write $t[s]$ if the position of s in t is not important in the context of discussion. Ambiguously, we write $t[s']$ (or $t[s']_p$) to denote the term obtained from t by replacing the occurrence of s (at the position p) with the term s'. The subterm s is called *strict* if the position p of s in t has non-zero length, and it is called *immediate* if $|p| = 1$, i.e. if s occurs immediately below the top function symbol.

Substitutions and Unifiers A *substitution* σ is a mapping from variables $\mathcal{X}$ to terms $T_\Omega(\mathcal{X})$ such that every variable $x \in \mathcal{X}$ is mapped to a term $\sigma(x) = t$ of the same sort $S = sort(x) = sort(t)$, and there only finitely many variables x for which $\sigma(x) \neq x$. The application $\sigma(x)$ of a substitution σ to a variable x is usually written in postfix notation as $x\sigma$; the composition of two substitutions σ and τ is written as a juxtaposition $\sigma\tau$, i.e. $t\sigma\tau = (t\sigma)\tau$. For a substitution σ, we write $dom(\sigma)$ to denote the set of all variables which are not mapped by σ to themselves: $dom(\sigma) = \{x \in \mathcal{X} \mid x\sigma \neq x\}$; $cdom(\sigma)$ is the set of all variables occurring in the image of $dom(\sigma)$ under the substitution: $cdom(\sigma) = \{vars(t) \mid t = x\sigma, x \in dom(\sigma)\}$. The result of an application of a substitution σ to a term t yields a term $s = t\sigma$ obtained from t by replacing each variable occurrence x with $x\sigma$, for all $x \in vars(t)$. The term s is called an *instance* of t; if the s is ground, then it is called a *ground* instance of t. The set of all ground instances of a term t is denoted by $gi(t)$. We write $t[s/x]$ to denote $t\sigma$ for $\sigma = \{x \mapsto s\}$.

Two terms s and t are said to be *unifiable* if there exists a substitution σ such that $s\sigma = t\sigma$; the substitution σ is called then a *unifier* of s and t. It is called a *most general unifier*, written $\sigma = mgu(s, t)$, if any other unifier τ of s and t can be represented as $\tau = \sigma\tau'$, for some substitution τ'. Notably, most general unifiers are unique up to variable renaming. A substitution σ is called a *matcher* from s to t if $s\sigma = t$.

Clauses As usual in the superposition context, all considered *atoms* are equations $t \approx s$ where s and t are of the same sort. Every predicate symbol is encoded as a function into a set with one distinguished element; without loss of generality, we assume that there is a distinct sort for every predicate symbol. Non-equational atoms are thus turned into equations $P(t_1, \ldots, t_n) \approx true_P$ which are simply abbreviated by $P(t_1, \ldots, t_n)$. A *literal* L is either an atom or a negated atom.

A clause C is a pair of multisets of atoms, written $C = (\Gamma \to \Delta)$, where all variables are assumed to be universally quantified, if not stated otherwise; the multiset Γ is called the *antecedent*, and the multiset Δ the *succedent*. Logically, the atoms in Γ denote negative literals while the atoms in Δ denote the positive literals in the clause. Semantically, a clause $C = \{A_1, \ldots, A_m\} \to \{B_1, \ldots, B_n\}$ represents an implication $A_1 \wedge \ldots \wedge A_m \to B_1 \vee \ldots \vee B_n$, or equivalently a disjunction $\neg A_1 \vee \ldots \vee \neg A_m \vee B_1 \vee \ldots \vee B_n$. Sometimes, we interpret clauses as (multi)sets of their literals, thus given a clause $C = \{A_1, \ldots, A_m\} \to \{B_1, \ldots, B_n\}$ we may write $C = \{L_1, \ldots, L_{m+n}\}$, where $\{L_1, \ldots, L_{m+n}\} = \{\neg A_1, \ldots, \neg A_m, B_1, \ldots, B_n\}$. Different clauses are assumed to be variable disjoint. The clause $C = \Gamma \to \Delta$ is called *negative* (*positive*) if it contains only negative (positive) literals; *Horn* if its succedent contains at most one atom, i.e. $|\Delta| \leq 1$; *unit* if it consists of a single literal, i.e. $|\Gamma| + |\Delta| = 1$; *empty* if it does not contain a literal, i.e. $\Gamma = \Delta = \emptyset$. We write $\square$ to denote an empty clause, a contradiction.

A clause C_1 is said to *subsume* a clause C_2 if there exists a matcher σ from C_1 to C_2 such that $C_1\sigma \subseteq C_2\sigma$, where C_1 and C_2 considered as multisets of literals. The clause C_1 is said to *strictly subsume* the clause C_2 if C_2 does not subsume C_1. It is worth mentioning that the subsumed clause C_2 is a logical consequence (see a presentation of semantics below) of the subsuming clause C_1. Two clauses C and D are *duplicates* of each other, if they are equal up to variable renaming. Obviously, the clauses C and D are equivalent and subsume each other.

Semantics and Specifications A Σ-*algebra* $\mathcal{A}$ consists of a Σ-sorted family of non-empty carrier sets $S_\mathcal{A}$ for each sort $S \in \mathcal{S}$ and of a function $f_\mathcal{A} \colon (S_1)_\mathcal{A} \times \ldots \times (S_n)_\mathcal{A} \to S_\mathcal{A}$ for every $f : S_1 \times \ldots \times S_n \to S \in \Omega$. The *interpretation* $t_\mathcal{A}$ of a ground term $t = f(t_1, \ldots, t_n)$ is defined recursively by $(f(t_1, \ldots, t_n))_\mathcal{A} = f_\mathcal{A}((t_1)_\mathcal{A}, \ldots, (t_n)_\mathcal{A})$. An algebra $\mathcal{A}$ is called *term-generated*, if every element of an $S_\mathcal{A}$ is the interpretation of some ground term of sort S. The *universe* of an algebra $\mathcal{A}$ is defined by $U_\mathcal{A} = \bigcup_{S \in \mathcal{S}} S_\mathcal{A}$.

An algebra $\mathcal{A}$ is said to *satisfy* an equation $t \approx s$, if $t_\mathcal{A} = s_\mathcal{A}$. A ground clause $C = \Gamma \to \Delta$ is *satisfied* by $\mathcal{A}$ if $\mathcal{A}$ satisfies an atom in Δ or at least one atom in Γ is not satisfied by $\mathcal{A}$. An algebra $\mathcal{A}$ *satisfies* a non-ground clause C if it satisfies all ground instances $gi(C)$ of the clause. If $\mathcal{A}$ satisfies every clause in a clause set N, then $\mathcal{A}$ is called a *model* of N. A clause set N is called *satisfiable* if it has a model, it is called *unsatisfiable* otherwise. We write $\mathcal{A} \models C$ to denote that $\mathcal{A}$ satisfies a clause C (analogously for atoms and sets of clauses). Also, we say that C is *true* in $\mathcal{A}$ if $\mathcal{A} \models C$, and *false* otherwise (analogously for atoms and sets of clauses). A clause is called *valid*, or a *tautology*, if it is true in all Σ-algebras, which is denoted by $\models C$. If N is unsatisfiable we denote this by $N \models \bot$. Given two clause sets N and M, we say N *entails* M, or M is a *consequence* of N, written $N \models M$, if each model $\mathcal{A}$ of N is a model of M. The sets N and M are called *equivalent*, written $N \models\!\!\mid M$, if $N \models M$ and $M \models N$. We call two clause sets N and M *equisatisfiable*, if N is satisfiable iff M is satisfiable (not necessarily in the same models).

A *specification* is a tuple $\mathsf{Sp} = (\Sigma, \mathscr{C})$ where $\mathscr{C}$ is a class of term-generated Σ-algebras, called *models* of the specification. We assume $\mathscr{C}$ is closed under isomorphisms. If Sp is the class of all Σ-models of a certain set of Σ-formulas Ax, then we write (Σ, Ax) instead of $(\Sigma, \mathscr{C})$.

The specification Sp is called *compact* if every set of formulae over Σ is satisfiable in $\mathscr{C}$ whenever each of its finite subsets is satisfiable in $\mathscr{C}$, or equivalently if every set of formulae over Σ is unsatisfiable in $\mathscr{C}$ whenever some its finite subset is unsatisfiable in $\mathscr{C}$.

Hierarchic Specification Assume $\mathsf{Sp} = (\Sigma, \mathscr{C})$ is a specification consisting of a signature $\Sigma = (\mathcal{S}, \Omega)$ and a class $\mathscr{C}$ of term-generated algebras closed under isomorphism. Let $\mathsf{Sp}' = (\Sigma', Ax')$ be a specification, where $\Sigma' = (\mathcal{S}', \Omega')$ is a signature augmenting Σ, i.e. $\mathcal{S} \subseteq \mathcal{S}'$ and $\Omega \subseteq \Omega'$, and Ax' is an axiom (formula) set over Σ'. A pair $\mathsf{HSp} = (\mathsf{Sp}, \mathsf{Sp}')$ is called a *hierarchic specification*, where Sp is called the *base specification*, and Sp' is called the *body* of the hierarchic specification. The models in $\mathscr{C}$ of the base specification are also called *base algebras*. The sorts $\mathcal{S}'' = \mathcal{S}' \backslash \mathcal{S}$ and operators $\Omega'' = \Omega' \backslash \Omega$ are called *free*.

If $\mathcal{A}'$ is a Σ'-algebra, the *restriction of* $\mathcal{A}'$ *to* Σ, written $\mathcal{A}'|_\Sigma$, is the Σ-algebra that is obtained from $\mathcal{A}'$ by removing every carrier set $S''_{\mathcal{A}'}$ for all free sorts $S'' \in \mathcal{S}' \backslash \mathcal{S}$ and every function $f_{\mathcal{A}'}$ for all free operator symbols $f \in \Omega' \backslash \Omega$. A Σ'-algebra $\mathcal{A}'$ is called *hierarchic* if its restriction to Σ is a base algebra, i.e. if $\mathcal{A}'|_\Sigma \in \mathscr{C}$. Informally speaking, a Σ'-algebra $\mathcal{A}'$ is hierarchic, if $\mathcal{A}'$ extends some base model $\mathcal{A} \in \mathscr{C}$ (meaning that $\mathcal{A}'$ interprets all base operator symbols in Ω the same as $\mathcal{A}$ does) and neither collapses any its sort nor adds new elements to it, i.e. if $\mathcal{A}'$ is a conservative extension of $\mathcal{A}$. A hierarchic algebra $\mathcal{A}'$ is called a *model of a hierarchic specification* $\mathsf{HSp} = (\mathsf{Sp}, \mathsf{Sp}')$, where $\mathsf{Sp}' = (\Sigma', Ax')$, if it is a model of the axiom set Ax'.

Let N and M be two sets of Σ'-clauses. A Σ'-algebra $\mathcal{A}'$ is called a *hierarchic model* of N, denoted by $\mathcal{A}' \models_\mathscr{C} N$, if $\mathcal{A}' \models N$ and $\mathcal{A}'|_\Sigma \in \mathscr{C}$, i.e. if $\mathcal{A}'$ entails N and is a hierarchic algebra. We say N *implies* M *relative to* $\mathscr{C}$, written $N \models_\mathscr{C} M$, if every hierarchic model of N is also a model of M. We call N *consistent/satisfiable relative to* $\mathscr{C}$ (or equivalently $\mathscr{C}$*-consistent/satisfiable*, or *consistent/satisfiable with respect to the theory* T), if there exists a hierarchic model of N, and, otherwise, we call it *inconsistent/unsatisfiable relative to* $\mathscr{C}$ (or equivalently $\mathscr{C}$*-inconsistent/unsatisfiable*, or *inconsistent/unsatisfiable with respect to the theory* T), written $N \models_\mathscr{C} \bot$. Alternatively, we can say $N \models_\mathscr{C} M$ if given a base specification Sp and a body signature Σ', the hierarchic specification $\mathsf{HSp} = (\mathsf{Sp}, (\Sigma', N \to M))$ has a model. If N is a set of base clauses, it is $\mathscr{C}$-consistent if and only if some base algebra $\mathcal{A}$ in $\mathscr{C}$ is a model of N. The set N is $\mathscr{C}$*-valid* if it is true in every model $\mathcal{A} \in \mathscr{C}$. Note that for any clause sets N and M, $N \models_\mathscr{C} M$ follows from $N \models M$, but not other way around, in general.

Refutational theorem proving for hierarchic theories is aimed at answering one the following questions: given a clause set N over the body signature Σ',

- is N false in all models of the hierarchic specification HSp?
- does a hierarchic specification (Sp, $(\Sigma', Ax' \cup N)$) have no model? or
- is $Ax' \cup N$ inconsistent relative to $\mathscr{C}$?

Each of the questions is a reformulation of the others, and, hence, they are all equivalent.

For the sake of simplicity, for the rest we assume a single base sort S and a single free sort S'', i.e. $\mathcal{S} = \{S\}$ and $\mathcal{S}' = \{S, S''\}$. The base sort S may contain further terms built over function symbols from Ω' and in particular Ω'' ranging into the base sort. In addition to operator symbols we assume two distinct countably infinite sets of variables $\mathcal{X}$ and $\mathcal{X}''$, consisting of all variables ranging into the base and free sorts, respectively; the union of sets is denoted by $\mathcal{X}' = \mathcal{X} \cup \mathcal{X}''$. Variables in $\mathcal{X}$ are called *base*, and ones in $\mathcal{X}''$ *non-base*. We write $T_{\Omega'}(\mathcal{X}')$ to denote the set of all terms over operators Ω' and variables $\mathcal{X}'$; the set of all ground terms over Ω' is denoted by $T_{\Omega'}$. A term is called *pure* if it does not contain both a base operator and a free operator. Pure terms built over base variables and base operator symbols are called *base*, pure terms built over variables (base and non-base) and free operator symbols and that are not single base variables are called *free* (that is, a free term may contain base variables, but cannot be a base variable-term). We write $T_{\Omega}(\mathcal{X})$ for the set of all base terms, and $T_{\Omega''}(\mathcal{X}') \setminus \mathcal{X}$ for the set of all free terms; the respective sets of ground terms are denoted by T_{Ω} and $T_{\Omega''}$. A substitution is called *simple* if it maps every base variable to a base term. If σ is a simple substitution, $t\sigma$ is called a *simple instance* of t (analogously for equations and clauses). The set of all simple ground instances of a clause C is denoted by $sgi(C)$ (analogously for terms, atoms/literals, clause sets).

In order to enable a modular combination of free FOL and a background theory T, we deal with so-called *abstracted clauses* that consist only of pure literals. An abstracted clause C is usually written in the form $C = \Lambda \parallel \Gamma \to \Delta$, where Λ is a multiset of base literals built over base operator symbols from Ω and base variables from $\mathcal{X}$, and Γ and Δ multisets of free atoms built over free operator symbols from Ω'' and variables (base and non-base) from $\mathcal{X}'$. The parts Λ, Γ, and Δ of the abstracted clause C are called the *constraint*, the *antecedent* and the *succedent* of C, respectively. The part $\Gamma \to \Delta$ of the clause C to the right from the double bar $\parallel$ separator is called the *free part* of the clause. In Sect. 3 we show that any clause over Σ' can be transformed into an equivalent abstracted clause.

Semantically, an abstracted clause $C = \Lambda \parallel \Gamma \to \Delta$ is an implication $\bigwedge \Lambda \wedge \bigwedge \Gamma \to \bigvee \Delta$ between the conjunction of literals in Λ and atoms in Γ on one hand side, and disjunction of atoms in Δ, on the other. Thus, the double bar $\parallel$ symbol is simply used to separate the base part of a clause from the free one and does not carry itself any logical meaning. The constraint Λ and the free part $\Gamma \to \Delta$ of an abstracted clause C may share base variables. As usual, Λ, Γ and Δ may be empty and are then interpreted as *true*, *true*, *false*, respectively.

An abstracted clause $C = \Lambda \parallel \Gamma \to \Delta$ is called *negative* (*positive*) if its free part contains only negative (positive) literals; *Horn* if its succedent contains at most one atom, i.e. $|\Delta| \leq 1$; *unit* if its free part consists of a single literal, i.e. $|\Gamma| + |\Delta| = 1$; *empty* if its constraint and free part are empty. As in the plain case, we write $\square$ to denote an empty clause, a contradiction.

Calculus The technical part, "mechanics", of automated theorem proving is realized by means of special rules, called *deduction rules*, that manipulate a given clause set N by adding to or deleting clauses from N. The rules are described in a fraction-like manner, where the clauses above the fraction bar are called the *premises* of the rule, and the clauses below the bar the *conclusions* of the rule. There are three basic types of rules:

- *inference rules*

$$\mathcal{I}\frac{C_1 \ldots C_n}{D}$$

that add the conclusion D to the current clause set N yielding a clause set $N' = N \cup \{D\}$, provided $C_1, \ldots, C_n \in N$;
- *reduction rules*

$$\mathcal{R}\frac{C_1 \ldots C_n}{D_1 \ldots D_m}$$

that replace in N the premises $C_1 \dots C_n$ by the conclusions $D_1 \dots D_m$ yielding a clause set $N' = N \backslash \{C_1, \dots, C_n\} \cup \{D_1, \dots, D_m\}$;

- and *splitting rules*

$$\mathcal{S}\frac{C}{D \mid D'}$$

that replace the whole set N by disjunction of two sets $N_1 = N \cup \{D\}$ and $N_2 = N \cup \{D'\}$, provided $C \in N$.

Deduction rules are usually constrained by a list of conditions that have to be satisfied in order to apply the rules and help to reduce the search space. An application of an inference, reduction, or splitting rule is called *inference*, *reduction*, or *splitting*, respectively.

Given a clause set N, inference rules are aimed to derive new clauses (the conclusion D) from already existing ones (the premises $C_1 \dots C_n$) in N, whereas reduction rules are devoted to "simplify" N by replacing clauses in it (the premises $C_1 \dots C_n$) by some "simpler" ones (the conclusions $D_1 \dots D_m$). As a special case, if the number of conclusions of a reduction rule is smaller than the number of its premises, the rule can be actually used to delete clauses from N, thus reducing the size of N. Splitting rules have a dual nature: on one hand, they replace the current clause set N by two sets and extend them by new clauses D and D', respectively, serving this way like inference rules; on the other hand, the new clauses D and D' are "simpler" then the premise C and usually subsume C, so that C can be deleted just after the splitting rule application, reducing thus the original problem to "simpler" subproblems, and serving this way like reduction rules. The aim of splitting is to split a clause set into two simpler *independent* clause sets, such that they can be considered *separately from each other*.

A set of deduction rules is a called a *calculus*. An inference/reduction rule is *sound* (with respect to an entailment relation $\models$) if it yields a clause set N' entailed (with respect to $\models$) by the original set N. A splitting rule is sound if the disjunction of the resulting clause sets N_1 and N_2 is entailed by the original set N. A calculus is sound if every deduction rule thereof is so. Traditionally, refutational theorem proving has been formally described as a closure process that systematically adds conclusions of inferences to a given set of clauses and deletes redundant clauses until a "closed", or "saturated", set N^* is reached, where the conclusion of every inference from N^* is already contained in N^*, or a premise or the conclusion of an inference is redundant with respect to N^*. A calculus is called *refutationally complete* (with respect to an entailment relation $\models$), if a saturated set of clauses is unsatisfiable (with respect to $\models$) if and only if it contains the empty clause. □

Orderings and Selection Functions A binary relation $\to$ over terms is called *compatible with contexts* if for all terms s_1, s_2, and t it holds that $s_1 \to s_2$ implies $t[s_1] \to t[s_2]$, provided $t[s_1]$ and $t[s_2]$ are well-formed/sorted. We call a binary relation $\to$ *stable under substitutions* if for any substitution σ and any two terms t and s, if $t \to s$ then $t\sigma \to s\sigma$. A binary relation $\to$ is called a *rewrite relation* if it is compatible with contexts and stable under substitutions. A binary relation $\to$ is called *well-founded* (or *terminating*) if there is no infinite descending chain $t_1 \to t_2 \to \dots$.

A (*partial*) *ordering* is a transitive, reflexive, and antisymmetric binary relation. A *strict ordering* is a transitive and irreflexive relation. Every partial ordering $\succeq$ induces a strict ordering $\succ$ by $t \succ s$ iff $t \succeq s$ and $t \neq s$. A strict ordering is said to be *total* if for any two distinct elements s and t either $s \succ t$ or $t \succ s$. An ordering $\succ$ is called a *reduction ordering* if it is a well-founded and transitive rewrite relation (that is, a well-founded, transitive relation that is compatible with contexts, stable under substitutions, and possessing the subterm property).

The *lexicographic extension* $\succ_{\text{lex}}$ on tuples of some strict ordering is defined by $(t_1, \dots, t_n) \succ_{\text{lex}} (s_1, \dots, s_n)$ if $t_i \succ s_i$ for some $i \in \{1, \dots, n\}$, and $t_j = s_j$ for every $1 \leq j \leq i$. The *multiset extension* $\succ_{\text{mul}}$ of a partial ordering $\succ$ is defined as follows: $M \succ_{\text{mul}} N$ if $M \neq N$ and for every s such that $N(s) > M(s)$, there exists some t such that $M(t) > N(s)$ and $t \succ s$. Given a multiset, any smaller multiset can be obtained by (repeatedly) replacing an element by zero or more occurrences of smaller elements. If an ordering $\succ$ is total (respectively, well-founded), so is its multiset extension [11].

Definition 1 (*Lexicographic path ordering [4]*) Let $\succ$ be a strict ordering on function symbols, called *precedence*. The *lexicographic path ordering* $\succ_{\text{lpo}}$ is defined by: $t \succ_{\text{lpo}} s$ iff:

1. $s \in vars(t)$ and $s \neq t$, or
2. $t = f(t_1, \ldots, t_m)$, $s = g(s_1, \ldots, s_n)$, and
   (a) $t_i \succeq_{\text{lpo}} s$ for some $i \in \{1, \ldots, m\}$, or
   (b) $f \succ g$, and $t \succeq_{\text{lpo}} s_j$ for $\forall\, j \in \{1, \ldots, n\}$, or
   (c) $f = g$, $t \succeq_{\text{lpo}} s_j$ for $\forall\, j \in \{1, \ldots, n\}$, and $(t_1, \ldots, t_m)\ (\succ_{\text{lpo}})_{\text{lex}}\ (s_1, \ldots, s_m)$,

where $(\succ_{\text{lpo}})_{\text{lex}}$ is the lexicographic extension of $\succ_{\text{lpo}}$.

Theorem 2 (LPO [4]) *The LPO is a reduction ordering. If the precedence $\succ$ is total on the set of operator symbols, then LPO $\succ_{\text{lpo}}$ augmenting $\succ$ is total on the set of ground terms.*

Any ordering on terms can be extended to (abstracted) clauses in the following way. We consider clauses as multisets of occurrences of (dis)equations (recall that the antecedent and the succedent of a abstracted clause consist of equations, whereas the constraint may contain also disequations). An occurrence of an equation $s \approx t$ in the antecedent is identified with the multiset $\{s, s, t, t\}$, the occurrence of an equation $s \approx t$ in the succedent is identified with the multiset $\{s, t\}$; analogously, an occurrence of an equation $s \not\approx t$ in the constraint is identified with $\{s, s, t, t\}$, and an occurrence of a disequation $s \approx t$ in the constraint is identified with $\{s, t\}$. Now we overload $\succ$ on (dis)equation occurrences to be the multiset extension of $\succ$ on terms, and $\succ$ on clauses to be the multiset extension of $\succ$ on (dis)equation occurrences. Observe that an occurrence of an equation $s \approx t$ in the antecedent is strictly greater than an occurrence of $s \approx t$ in the succedent.

An antecedent or succedent occurrence of an equation $s \approx t$ is *maximal* (*minimal*) in a clause $C = \Lambda \parallel \Gamma \rightarrow \Delta$ if there is no occurrence of an equation in $\Gamma \rightarrow \Delta$ that is strictly greater (smaller) then the occurrence $s \approx t$ with respect to $\succ$. An antecedent or succedent occurrence of an equation $s \approx t$ is *strictly maximal* (*strictly minimal*) in the clause C if there is no occurrence of an equation in $\Gamma \rightarrow \Delta$ that is greater (smaller) then or equal to the occurrence $s \approx t$ with respect to $\succ$. We do not consider occurrences of (dis)equations in the constraint of a clause for our maximality criteria. As clauses are abstracted, this is justified by considering a suitable path ordering, like LPO, where the operator symbols from Ω are smaller than all symbols in $\Omega'' = \Omega' \backslash \Omega$.

Selection functions, like ordering restrictions, are aimed to reduce the search space of a calculus. A *selection function Sel* assigns to each abstracted clause $C = \Lambda \parallel \Gamma \rightarrow \Delta$ a multiset of atoms in the antecedent Γ of the clause. The atoms returned by $Sel(C)$ are called *selected* in the clause C. A selection function *Sel* may select atoms *arbitrarily* irregardless of ordering restrictions. The strategy, when all atoms are selected in the antecedent of a clause, is called *eager selection*. Eager selection is an instance of *positive superposition strategy*, which is called this way because in this strategy only positive clauses can be superposed into other clauses. In this work, we shall exploit eager selection as it allows to keep the length of Horn abstracted clauses bounded.

For the rest of the paper we assume a hierarchic specification $\mathsf{HSp} = (\mathsf{Sp}, \mathsf{Sp}')$, where $\mathsf{Sp} = (\Sigma, \mathscr{C})$ is the base specification, $\mathsf{Sp}' = (\Sigma', Ax')$ the body of HSp, with respective signatures $\Sigma = (\{S\}, \Omega)$ and $\Sigma' = (\{S, S''\}, \Omega')$, and countably infinite sets of base $\mathcal{X}$ and free $\mathcal{X}''$ variables, constituting the variable set $\mathcal{X}' = \mathcal{X} \cup \mathcal{X}''$. Sometimes, in the examples illustrating our approach, we shall instantiate Σ and Σ' with concrete signatures. For the sake of simplicity, we set the axiom set Ax' to be empty, which in the light of the observation from Sect. 2, page 430, regarding the main concerns of refutational theorem proving for hierarchic theories, does *not* diminish the generality of the overall approach.

Upper Greek letters Λ, Γ, Δ denote sequences of atoms or theory literals, lower Greek letters σ, τ substitutions, lower Latin characters l, r, s, t terms, a, b, c constants of the free sort, $\mathsf{a}, \mathsf{b}, \mathsf{c}$ constants of the base sort, u, v, w variables of the free sort, x, y, z variables of the base sort, and upper Latin characters A, B, E atoms.

3 Superposition Modulo a Theory T

One of the main aims of hierarchic superposition calculus is processing problems over a combination of a theory T and the FOL in a modular fashion, where internal reasoning in each part remains hidden from another. For this

reason we stick to abstracted clauses (defined in Sect. 2) which consist of two parts: one contains only base literals, and another only free FOL literals, such that base parts of abstracted clauses could be directly loaded into a theory solver, and free parts could be treated as ordinary plain clauses by a superposition-based procedure (for instance, an automated theorem prover) slightly adjusted to support *simple* unification and extension of reduction rules with extra checks for redundancy conditioned by the theory T (see below for details). Any given clause $C = \Gamma \to \Delta$ can be transformed into an abstracted clause $C' = \Lambda' \parallel \Gamma' \to \Delta'$, where Λ' contains only base literals and all base terms in Γ', Δ' are variables, by the following transformation [7]. Whenever a subterm t whose top symbol is a base symbol from Ω occurs immediately below a free operator symbol, it is replaced by a new base sort variable x ("abstracted out") and the equation $x \approx t$ is added to Γ. Analogously, if a subterm t whose top symbol is not a base symbol from Ω occurs immediately below a base operator symbol, it is replaced by a new general variable x and the equation $x \approx t$ is added to Γ. This transformation is repeated until all terms in the clause are pure; then all base literals are moved to Λ' and all free atoms to Γ', Δ', respectively. Recall that Γ', Δ' are multisets of atoms whereas Λ' holds theory *literals* (including negative literals). We need to "purify" clauses only once—just before saturating the clauses, since if the premises of an inference are abstracted clauses, then the conclusion is also abstracted (a formal proof is given in Lemma 8 in [7]). For an example of abstraction, see Sect. 4.2. The abstraction procedure presented is essentially the *Nelson-Oppen* method's preprocessing step, usually referred to as *purification*.

The overall hierarchic superposition calculus is based on a reduction ordering $\succ$ that is total on ground terms [10]. Moreover, following [7] we impose the requirement that in the ordering $\succ$ all ground base terms are smaller than all ground non-base terms.

Definition 3 (*Hierarchic Splitting)* The hierarchic splitting rule is

$$\mathcal{S}\,\frac{\Lambda_1, \Lambda_2 \parallel \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2}{\Lambda_1 \parallel \Gamma_1 \to \Delta_1 \;\mid\; \Lambda_2 \parallel \Gamma_2 \to \Delta_2}$$

where

(i) $vars(\Lambda_1 \parallel \Gamma_1 \to \Delta_1) \cap vars(\Lambda_2 \parallel \Gamma_2 \to \Delta_2) = \emptyset$,
(ii) $\Delta_1 \neq \emptyset$ and $\Delta_2 \neq \emptyset$.

The hierarchic splitting rule splits a given clause into two *variable disjoint* parts which are "semantically" independent in the sense that the premise $C = \Lambda_1, \Lambda_2 \parallel \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2$ is equivalent to the disjunction of the conclusions $C_1 = \Lambda_1 \parallel \Gamma_1 \to \Delta_1$ and $C_2 = \Lambda_2 \parallel \Gamma_2 \to \Delta_2$, therefore, N is satisfiable iff one of the sets $N_1 = N \backslash C \cup C_1$ or $N_2 = N \backslash C \cup C_2$ is satisfiable, or equivalently, N is unsatisfiable iff each N_i is unsatisfiable, for $i \in \{1, 2\}$. Note, that we may delete the clause C in N_1 and N_2 because each C_i subsumes C. After the splitting, the original clause set N is replaced by two clause sets N_1 and N_2, that are considered then independently from each other. The main reason behind using splitting is to reduce a given problem to the Horn fragment, which is essential for achieving termination. For this reason, we only apply splitting if the two succedent parts Δ_1, Δ_2 are non-empty (for a more detailed motivation see Sect. 4.3.1).

Definition 4 (*Hierarchic Equality Resolution)* The hierarchic equality resolution rule is

$$\mathcal{I}\,\frac{\Lambda \parallel \Gamma, s \approx t \to \Delta}{(\Lambda \parallel \Gamma \to \Delta)\sigma}$$

where

(i) $\sigma = mgu(s, t)$ and σ is simple,
(ii) $(s \approx t)\sigma$ is maximal in $(\Gamma, s \approx t \to \Delta)\sigma$ and no literal in Γ is selected
or
$s \approx t$ is selected.

Definition 5 (*Hierarchic Superposition Left)* The hierarchic superposition left rule is

$$\mathcal{I}\frac{\Lambda_1 \parallel \Gamma_1 \to \Delta_1, l \approx r \qquad \Lambda_2 \parallel s[l'] \approx t, \Gamma_2 \to \Delta_2}{(\Lambda_1, \Lambda_2 \parallel s[r] \approx t, \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma}$$

where

(i) $\sigma = mgu(l, l')$ and σ is simple,
(ii) l' is not a variable,
(iii) $l\sigma \not\prec r\sigma$,
(iv) $s\sigma \not\prec t\sigma$,
(v) $(l \approx r)\sigma$ is strictly maximal in $(\Gamma_1 \to \Delta_1, l \approx r)\sigma$,
(vi) $(s \approx t)\sigma$ is maximal in $(s \approx t, \Gamma_2 \to \Delta_2)\sigma$ and no literal in Γ_2 is selected
or
$s \approx t$ is selected,
(vii) no literal in Γ_1 is selected.

Definition 6 (*Hierarchic Superposition Right)* The hierarchic superposition right rule is

$$\mathcal{I}\frac{\Lambda_1 \parallel \Gamma_1 \to \Delta_1, l \approx r \qquad \Lambda_2 \parallel \Gamma_2 \to \Delta_2, s[l'] \approx t}{(\Lambda_1, \Lambda_2 \parallel \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2, s[r] \approx t)\sigma}$$

where

(i) $\sigma = mgu(l, l')$ and σ is simple,
(ii) l' is not a variable,
(iii) $l\sigma \not\prec r\sigma$,
(iv) $s\sigma \not\prec t\sigma$,
(v) $(l \approx r)\sigma$ is strictly maximal in $(\Gamma_1 \to \Delta_1, l \approx r)\sigma$,
(vi) $(s[l'] \approx t)\sigma$ is strictly maximal in $(s[l'] \approx t, \Gamma_2 \to \Delta_2)\sigma$,
(vii) no literal in Γ_1, Γ_2 is selected.

As we mentioned above, an application of Hierarchic Superposition Left/Right rule may produce an assignment of base variables $(s[r] \approx t)\sigma$ in the case if r and t are base variables, l and s are non-variable terms, and $l' = s$. The resulting assignment of base variables is moved to the constraint of the inference resolvent (where it gets negated if produced by the Hierarchic Superposition Right rule as it is moved from succedent of the clause to its constraint). We do not explicitly mention this rule as it is obvious. It has not been needed in the previous formulation of the calculus [7] because there no strict separation between subsets of base and free literals of an abstracted clause has been made. However, it is very useful to separate the base literals from the free ones in order to explore the reasoning mechanisms for the base specification. This also emphassis the fact that (ground) base terms are strictly smaller than (ground) non-base terms.

It is worthwhile to notice that the unifier σ underlying the inference rules Hierarchic Equality Resolution and Hierarchic Superposition Left/Right can unify a base variable only with another base variable, because σ is required to be simple (meaning that σ can substitute base variables only with base terms) and premises are abstracted (meaning that the only base terms occurring in the free part of premises are just base variables).

Definition 7 (*Constraint Refutation)* The constraint refutation rule is

$$\mathcal{I}\frac{\Lambda_1 \parallel \to \quad \ldots \quad \Lambda_n \parallel \to}{\square}$$

where $(\Lambda_1 \parallel \to), \ldots, (\Lambda_n \parallel \to) \models_{\mathscr{C}} \bot$.

The rule Constraint Refutation is applied if the set of base clauses $(\Lambda_1 \parallel \rightarrow), \ldots, (\Lambda_n \parallel \rightarrow)$ is unsatisfiable in the base theory T.

We do not give a definition of the Hierarchic Factoring inference rule, because clauses of the FOL(T) fragments considered here can be always split into Horn clauses, herewith Factoring gets obsolete (for more details regarding splitting into Horn clauses see Sect. 4.3.1), and completeness is preserved though.

Discarding separation of base and free parts of clauses and simple unifiers, the rules Hierarchic Splitting, Hierarchic Superposition Left/Right, and Hierarchic Equality Resolution are obvious instances of the respective standard superposition inference rules [5,28], therefore the hierarchic rules are sound with respect to the general entailment relation $\models$, hence sound with respect to the hierarchic entailment relation $\models_{\mathscr{C}}$ (recall that $N \models M$ implies $N \models_{\mathscr{C}} M$). The rule Constraint Refutation is not sound with respect to the general entailment relation $\models$ (as for a set of base clauses inconsistent relative to the theory T, there might exist a non-standard model of T,—a model out of the base model class $\mathscr{C}$,—that actually satisfies the given set of base clauses), but the rule is evidently sound with respect to $\models_{\mathscr{C}}$. Thus, the overall inference system consisting of the rules Hierarchic Splitting, Hierarchic Superposition Left/Right, Hierarchic Equality Resolution, and Constraint Refutation is sound with respect to $\models_{\mathscr{C}}$ [7].

We do not introduce the hierarchic redundancy criterion proposed in [7], as the only concrete redundancy criterion we need here is deletion of duplicate clauses modulo renaming. This is equivalent to omitting inferences which produce such clauses. For this reason, we also do not define the reduction rules Hierarchic Tautology Deletion and Hierarchic Subsumption Deletion given in [7]. Nevertheless, in practice the overall approach would considerably benefit from using these rules, as clauses, that could be eliminated by the rules, naturally arise in SUP(T) derivations from clause sets over the FOL(T) fragments considered here.

Definition 8 (*Sufficient Completeness [7]*) A set N of clauses is called *sufficiently complete with respect to simple instances*, if for every model $\mathcal{A}'$ of $sgi(N)$ and every ground term $t \in T_{\Omega'}(S)$ of the base sort S there exists a ground base term $t' \in T_{\Omega}$ such that $t \approx t'$ is true in $\mathcal{A}'$.

Theorem 9 (Completeness [7]) *If the base specification is compact, then the hierarchic superposition calculus is refutationally complete for all sets of clauses that are sufficiently complete with respect to simple instances.*

The compactness requirement is obsolete in cases where all possible hierarchic superposition derivations for some clause set are finite. We will show this to be the case for the fragments we consider here. The given definition of the sufficient completeness criterion is very strong, but still the calculus may be complete for some clause sets, that do not enjoy the criterion. In practice, we only need to sufficiently define only those non-base terms of the base sort that actually occur in a given clause set N. For a finite clause set N sufficient completeness can be obtained by transforming N into a clause set N', in which every non-base term t of the base sort occurs in a positive unit of the form $t \approx t'$, where t' is a base term. If, by a proper instantiation of the SUP(T) calculus the existence of such equations remains an invariant for any SUP(T) derivation $N_0 \vdash N_1 \vdash \ldots$, where N_0 is the clause set N' abstracted, sufficient completeness is preserved and the overall approach is complete. Completeness of the calculus for clause sets over the fragments we consider in the current work is achieved exactly in this way—by sufficiently defining all non-base terms of the base sort that do occur in a given clause set (the approach is further described in the following sections).

Example Here is an example, for which the hierarchical calculus is not complete, but where completeness can be recovered by sufficiently defining present terms. Let the background theory T be rational linear arithmetic; $<, \geq, 0 \in \Omega$, and $f, a \in \Omega''$; the function symbol f ranges into the base sort S, and a into the free sort S''. Consider the two clauses

$$(1) \qquad \rightarrow f(a) \geq 0$$
$$(2) \qquad \rightarrow f(a) < 0$$

which are abstracted to

$$(1') \quad x \not\geq 0 \parallel f(a) \approx x \rightarrow$$
$$(2') \quad x \not< 0 \parallel f(a) \approx x \rightarrow$$

Recall that variables in clauses are universally quantified, hence all clauses are variable disjoint. Clearly, the set of these two clauses is unsatisfiable relative to $\mathscr{C}$, however no SUP(T) inference is possible. Note that hierarchic equality resolution is not applicable as $\sigma = \{x \mapsto f(a)\}$ is not a simple substitution. The two clauses are not sufficiently complete, because the clause set does not imply simple instances of the term $f(a)$ to be equal to some base term.

Note that the set is satisfiable in a Σ'-model $\mathcal{A}'$, that extends the base sort with an extra "junk" element, which we denote by "$\diamond$", exclusively generated by the term $f(a)$, i.e. $t_{\mathcal{A}'} = \diamond$ iff $t = f(a)$; and interprets the predicates "$<$" and "$\geq$" such that:

- $\diamond \geq_{\mathcal{A}'} 0_{\mathcal{A}'}$ and $\diamond <_{\mathcal{A}'} 0_{\mathcal{A}'}$ are true; and
- $\mathcal{A}' \models s < s'$ iff $s < s'$ holds in the theory of linear arithmetic (analogously for $\geq$), for any ground base terms $s, s' \in T_\Omega$.

The model $\mathcal{A}'$ is not hierarchic, as its restriction $\mathcal{A}'|_\Sigma$ to the base signature is a non-standard model of linear arithmetic (because of the junk "$\diamond$").

The equality of $f(a)$ to a base term can, for example, be forced by the additional clause

$$(3) \qquad \rightarrow f(a) \approx 0$$

which is abstracted to

$$(3') \quad y \approx 0 \parallel \rightarrow f(a) \approx y$$

Now the three clauses are sufficiently complete. For the clauses there is a refutation where Hierarchic Superposition Left Inferences between the clauses (1) and (3), and (2) and (3) yield respectively (note the move of base variable assignments $y \approx x$ from the free parts of the conclusions to the constraints):

$$(4) \quad x < 0, y \approx 0, y \approx x \parallel \rightarrow$$
$$(5) \quad x \geq 0, y \approx 0, y \approx x \parallel \rightarrow$$

which are equivalent to

$$(4') \qquad 0 < 0 \parallel \rightarrow$$
$$(5') \qquad 0 \geq 0 \parallel \rightarrow$$

respectively. A Constraint Refutation application to the clause (4) (or equivalently to (4′)) results in the empty clause. □

4 Deciding Ground FOL(T)

In this section we prove that the SUP(T) calculus is a decision procedure for the ground FOL(T) fragment.

4.1 Basification

A key idea of the approach to decide the ground FOL(T) fragment we describe here is to *sufficiently define* ground base sort terms, occurring in an input clause set and whose top symbol is a free function symbol, by extending the clause set by extra positive unit clauses consisting of an equation between every such a term and a fresh (i.e. not occurring in the clause set) base constant (parameter) uniquely introduced for the term, thus obtaining a sufficiently complete clause set which is satisfiable iff the original one is so. Here we assume that the background theory T actually provides such free (Skolem) constants. Semantically, they play the role of existentially quantified variables, i.e., for our decision result to be effective the base theory must provide decidability with respect to quantifier alternations. This is, for example, the case for the theory of linear arithmetic, considered as the running example. Furthermore, extending the background theory with new constant symbols may cause losing compactness of the overall approach. We show that following a certain strategy, the SUP(T) calculus may produce only *finitely many* irredundant clauses in the fragment considered, so compactness is guaranteed that way.

Given a clause set N, a ground term t, whose top symbol is a free function $f \in \Omega''$ ranging into the base sort S, is called *basified*, if the set N contains a positive unit clause $C \in N$ consisting of an equation $t \approx \mathsf{a}$, where a is a base constant (theory parameter). The clause C is called *basifying* (*for the term* t). A ground clause set N is called *basified* if every ground term t, occurring in N and whose top symbol is a free function $f \in \Omega''$ ranging into the base sort S, is basified by a clause in N.

Every ground clause set N over FOL(T) can be transformed into an equisatisfiable basified clause set in the following way: traverse every literal in every clause *from innermost* (i.e. discover the literal's term tree bottom up) and search for occurrences of subterms whose top symbol is a free function symbol ranging into the base sort; whenever such a term t is being observed, there are two possible ways of proceeding:

- if this is the *first occurrence* of t considered, then: (i) introduce a fresh base constant a, (ii) replace the occurrence of t with a, and (iii) extend the input clause set N with a basifying clause $C = (\rightarrow t \approx \mathsf{a})$;
- for otherwise, replace the current occurrence of t with the parameter a, that has been introduced previously up to this point, when the first occurrence of t has been discovered.

As the transformation of terms is performed from innermost to outwards, in every term t basified, there is no subterm whose top symbol is a free function symbol ranging into the base sort, thus no further basification in an added basifying clause is needed. Besides, all subsequent occurrences of the same term t appearing in the input clause set N are replaced with the same base parameter a exclusively introduced for t. Any basified clause set N enjoys the sufficient completeness criterion, because for every term t occurring in N, whose top symbol is a free function symbol ranging into the base sort, there is a basifying clause $(\rightarrow t \approx \mathsf{a})$ in N, where a is a base constant; consequently, for every such t there exists a base term $t' = \mathsf{a}$ such that $t \approx t'$ is true in every model of N. From a semantics perspective, the base parameters can be thought of as base-sorted variables existentially quantified on the top of the overall problem.

Example Let N be a given clause set to be basified, containing just one clause:

$$N = \{\ P(f(h(1 + g(a)))) \ \rightarrow\ f(h(1)) + g(a) \geq 0\ \}$$

where the background theory T is rational linear arithmetic; $+, \geq, 0, 1 \in \Omega$, and $P, f, g, a, h \in \Omega''$; the function symbols f, g range into the base sort $\mathcal{S}$, and a, h into the free sort $\mathcal{S}''$.

Assume the literal $\neg P(f(h(1 + g(a))))$ is discovered first. Its (only) innermost subterm, whose top symbol is a free function symbol ranging into the base sort, is the term $g(a)$. First, a fresh base parameter a is introduced; second, the current occurrence of $g(a)$ is replaced with a; and then a basifying clause $(\rightarrow g(a) \approx \mathsf{a})$ is added, yielding the following clause set:

$$\begin{aligned}\{\ P(f(h(1 + \mathsf{a}))) \ &\rightarrow\ f(h(1)) + g(a) \geq 0,\\ &\rightarrow\ g(a) \approx \mathsf{a}\ \}\end{aligned}$$

Next, the basification procedure processes the subterm $f(h(1+\mathsf{a}))$ by, likewise, introducing a fresh parameter, replacing the subterm with the new parameter, and adding the respective basifying clause, resulting in the following clause set:

$$\begin{aligned}\{\ P(\mathsf{b}) &\rightarrow\ f(h(1)) + g(a) \geq 0,\\ &\rightarrow\ g(a) \approx \mathsf{a},\\ &\rightarrow\ f(h(1+\mathsf{a})) \approx \mathsf{b}\ \}\end{aligned}$$

At this point, every subterm of the first literal becomes basified, so the second literal $f(h(1)) + g(a) \geq 0$ is being processed. Assume the subterm $f(h(1))$ is considered first. After basifying the subterm in the above manner, we obtain:

$$\begin{aligned}\{\ P(\mathsf{b}) &\rightarrow\ \mathsf{c} + g(a) \geq 0,\\ &\rightarrow\ g(a) \approx \mathsf{a},\\ &\rightarrow\ f(h(1+\mathsf{a})) \approx \mathsf{b},\\ &\rightarrow\ f(h(1)) \approx \mathsf{c}\ \}\end{aligned}$$

In the next step, the subterm $g(a)$ is to be basified. Since another occurrence of the term has been already processed, and for which a unique base parameter has been introduced, the parameter a, no new parameter is produced now and no basifying clause is added, and the current occurrence of $g(a)$ is replaced with the a, yielding the following clause set N':

$$\begin{aligned}N' = \{\ P(\mathsf{b}) &\rightarrow\ \mathsf{c} + \mathsf{a} \geq 0,\\ &\rightarrow\ g(a) \approx \mathsf{a},\\ &\rightarrow\ f(h(1+\mathsf{a})) \approx \mathsf{b},\\ &\rightarrow\ f(h(1)) \approx \mathsf{c}\ \}\end{aligned}$$

The obtained clause set N' can be split into two parts: N_I and N_B. The set N_I is the set of the basified clauses from the initially given clause set N, the set N_B is the set of clauses basifying those in N_I. So, for the example above, we have:

$$\begin{aligned}N_I = \{\ P(\mathsf{b}) &\rightarrow\ \mathsf{c} + \mathsf{a} \geq 0\ \},\\ N_B = \{\ &\rightarrow\ g(a) \approx \mathsf{a},\\ &\rightarrow\ f(h(1+\mathsf{a})) \approx \mathsf{b},\\ &\rightarrow\ f(h(1)) \approx \mathsf{c}\ \}\end{aligned}$$

Let N be a ground clause set, $N' = N_I \cup N_B$ a clause set obtained from N by basification, where N_I is the set of the basified clauses from N, and N_B is the set of clauses basifying those in N_I. Let $L = t_1 \mathrel{\dot{\approx}} t_2$ be an arbitrary literal from a clause $C \in N$, where $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$. We want to determine the structure of the literal $L' = t_1' \mathrel{\dot{\approx}} t_2'$, which the literal L becomes after basification in the corresponding clause $C' \in N_I$. Regarding the structure of t_1 – analogously for t_2 – there are two possible cases:

1. any subterm s of t_1 with a free top symbol $top(s) = f \in \Omega''$ is of the free sort $sort(s) = S''$, or more formally: $\forall\, p \in \rho(t_1) : top(t_1/p) \in \Omega'' \Rightarrow sort(t_1/p) = S''$, implying that any base sort subterm of t_1 is actually a base term; in this case there is no subterm in t_1 to be basified, therefore t_1 remains unchanged: $t_1' = t_1$.

2. t_1 contains a subterm s of the base sort $sort(s) = S$ with a free top symbol $top(s) = f \in \Omega''$; then every such a subterm s is replaced with a unique base parameter, yielding a term t_1' such that any subterm of t_1' with a free top symbol is of the free sort S'', or more formally: $\forall\, p \in \rho(t_1') : top(t_1'/p) \in \Omega'' \Rightarrow sort(t_1'/p) = S''$, implying that any base sort subterm of t_1' is actually a base term.

Consider an arbitrary clause $(\parallel t \approx \mathsf{a} \rightarrow) \in N_B$. The clause is basifying for some term t whose top symbol is a free function symbol $f \in \Omega''$ ranging into the base sort S. Since basification runs from innermost in a backward depth-first-search manner, at the iteration of basification, when the first occurrence of t is basified, all immediate subterms of t have been basified already, hence any *strict* subterm s of t has a top symbol $top(s) = g$, that is either a base function symbol including newly introduced base parameters, or a free function symbol ranging into the free sort, i.e. $\forall\, p \in \rho(t) : p > \varepsilon, top(t/p) \in \Omega'' \Rightarrow sort(t/p) = S''$, implying that any *immediate* base sort subterm of t is actually a base term.

In the following proposition and subsequent corollary we summarize the observations regarding the structure of terms occurring in N' obtained by basification from a ground clause set N:

Proposition 10 *Let $N' = N_I \cup N_B$ be a clause set obtained from a ground clause set N by basification. For any literal $L = t_1 \mathrel{\dot{\approx}} t_2$ in any clause from N_I, where $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$, it holds that*

$$\forall\, p \in \rho(t_i) : top(t_i/p) \in \Omega'' \Rightarrow sort(t_i/p) = S'',$$

for every $i \in \{1, 2\}$. The only literal $L = t \approx \mathsf{a}$ in any clause from N_B is an equation between a base parameter a and a term t, such that $top(t) \in \Omega''$, $sort(t) = S$, and

$$\forall\, p \in \rho(t) : p > \varepsilon, top(t/p) \in \Omega'' \Rightarrow sort(t/p) = S''.$$

Corollary 11 *If a clause set N' is obtained from a ground clause set N by basification, then for any literal $L = t_1 \mathrel{\dot{\approx}} t_2$ in any clause from N', where $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$, it holds that*

$$\forall\, p \in \rho(t_i) : top(t_i/p) \in \Omega \Rightarrow t_i/p \in T_\Omega,$$

for every $i \in \{1, 2\}$.

For a given ground clause set N, basification of N produces an equisatisfiable ground clause set N'. From Proposition 10 it follows that all non-base terms t of the base sort S occurring in N' are sufficiently defined and occur only in positive literals of the form $t \approx t'$, where t' is ground base term, which makes the set N' sufficiently complete.

Lemma 12 *Let N be a ground clause set. If N' is obtained from N by basification, then N' is sufficiently complete, and N' and N are equisatisfiable.*

The clause set N' obtained from a ground clause set N by basification contains one extra clause for each different ground base sort terms with free top symbol that appears in N, and its size asymptotically equals the size of N in the number of symbols in N. Basification takes loglinear (i.e. $n \log n$) time in the number of symbols in N.

4.2 Abstraction

Abstraction introduced in Sect. 3 is used to split a clause into base and free parts such that each part consists of base and free literals, respectively, and the only symbols shared by the parts of an abstracted clause are base variables. After a given clause set N has been basified to a clause set $N' = N_I \cup N_B$, the set N' is to be abstracted.

Example Consider the clause set $N' = N_I \cup N_B$ from the previous example:

$$\begin{aligned} N_I &= \{\ P(\mathsf{b}) \ \rightarrow\ \mathsf{c} + \mathsf{a} \geq 0\ \}, \\ N_B &= \{\ \ \rightarrow\ g(a) \approx \mathsf{a}, \\ &\quad\ \ \rightarrow\ f(h(1 + \mathsf{a})) \approx \mathsf{b}, \\ &\quad\ \ \rightarrow\ f(h(1)) \approx \mathsf{c}\ \} \end{aligned}$$

Abstraction yields the following clause set (recall that all variables are universally quantified):

$$\begin{aligned} N'_I &= \{\ x \approx \mathsf{b}, \mathsf{c} + \mathsf{a} < 0 \parallel P(x) \ \rightarrow\ \}, \\ N'_B &= \{\ \ x \approx \mathsf{a} \parallel \ \rightarrow\ g(a) \approx x, \\ &\quad\ \ x \approx \mathsf{b}, y \approx 1 + \mathsf{a} \parallel \ \rightarrow\ f(h(y)) \approx x, \\ &\quad\ \ x \approx \mathsf{c}, y \approx 1 \parallel \ \rightarrow\ f(h(y)) \approx x\ \} \end{aligned}$$

According to Corollary 11, any subterm t, occurring in a clause $C = \Gamma \rightarrow \Delta$ from the clause set N' and whose top symbol is a base operator symbol, is a base term (meaning that t contains no free symbol), therefore only base terms are abstracted out. These implies that every variable x in the abstracted clause $C' = \Lambda' \parallel \Gamma' \rightarrow \Delta'$ is *base* and assigned to some *ground* base term t via the equation $x \approx t$ in Λ'. Thus, abstraction populates the constraint Λ' of the clause C' by literals of two types:

- assignments $x \approx t$ of a variable x to a ground base term t, arising as the result of abstracting out base subterms occurring below a free operator symbol; and
- base literals $s \mathrel{\dot{\approx}} t$, where s and t are ground base terms and $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$, which have been already present in the clause $C \in N'$ and then moved to the constraint of C' at the final step of abstraction[1]; the literal gets negated if it comes from the succedent of the clause.

Since no free term is abstracted out, the number of literals in each subset of the free part of the abstracted clause $C' = \Gamma' \rightarrow \Delta'$ may only decrease in comparison to that of the corresponding original clause $C = \Lambda \parallel \Gamma \rightarrow \Delta$, i.e. $|\Gamma'| \leq |\Gamma|$ and $|\Delta'| \leq |\Delta|$.

In the sequel of Sect. 4 we state different properties regarding variables occurring in clauses derived by the SUP(T) calculus from a clause set obtained by basification and abstraction of an input set of ground clauses. Although all variables occurring in such clauses are base, we formulate the properties for explicitly distinguished base variables. This enables us to directly extend the results obtained for the ground FOL(T) to non-ground fragments involving non-ground free terms containing variables of the free sort, Sect. 5.

Next we show that the above properties regarding base variables and the structure of the clauses' constraints are actually invariants for any SUP(T) derivation $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$, where N_0 is obtained from a ground clause set by basification and abstraction.

Proposition 13 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained from a ground clause set by basification and abstraction; let $C = \Lambda \parallel \Gamma \rightarrow \Delta$ be an arbitrary clause in N_i, $i \geq 0$. Then for every base variable $x \in vars(C)$ there exists a ground base term $t \in T_\Omega$ such that[2] the equation $x \approx t$ is in the constraint Λ of C.*

[1] Recall that predicates are encoded as functions; for instance, the literal $\mathsf{c} + \mathsf{a} < 0$ from the example above is an abbreviation for the equation $<(\mathsf{c} + \mathsf{a}, 0) \approx true_<$. Encoding of predicates as functions is needed to compactly define the calculus. According to the definition of the SUP(T) calculus, only free literals are superposed, therefore encoding base predicates as functions is not strictly necessary. Still, we do it as it helps to compress the theoretical discussion by saving one extra case (of considering predicative atoms), like in Proposition 14. Whenever the Constraint Refutation rule is performed and the background T-solver is invoked, every such base predicative (dis)equation $P(t_1, \ldots, t_n) \mathrel{\dot{\approx}} true_P$, where $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$, is passed to the solver as the corresponding predicate $P(t_1, \ldots, t_n)$.

[2] Recall that T_Ω stands for the set of all ground base terms. Notation for different types of term sets is introduced in Sect. 2.

Proof We give a proof by induction on the length of derivation of N_i.

Induction Base For N_0 the assertion follows from the definition of abstraction and Corollary 11.

Induction Hypothesis Assume the assertion holds for all clause sets N_i derived from N_0, where $i \leq n$, for some $n > 0$.

Induction Step Let $N_{n+1} = N_n \cup C$, where C is a conclusion of an inference rule application to some clauses in N_n. Consider the Hierarchic Equality Resolution rule, Definition 4,

$$\mathcal{I}\frac{\Lambda \parallel \Gamma, s \approx t \to \Delta}{(\Lambda \parallel \Gamma \to \Delta)\sigma}$$

the premise $C_1 \in N_n$ and conclusion C, respectively. As σ is simple and the premises are abstracted, the substitution σ can unify a base variable only with another base variable. The variables $vars(C_1) \setminus dom(\sigma)$ are not affected by σ, and $cdom(\sigma) \subseteq vars(s, t) \subseteq vars(C_1)$. Therefore, every variable $x \in vars(C)$ comes from the clause C_1, and $x = y\sigma$, where y is a base variable (not necessarily different from x) from $vars(C_1)$. By induction hypothesis, every base variable $y \in vars(C_1)$ is assigned to some ground base term t' via an equation $(y \approx t') \in \Lambda$ in the constraint of C_1, hence $(y \approx t')\sigma = (x \approx t'\sigma) \in \Lambda\sigma$, where $\Lambda\sigma$ is the constraint of C. Since t' is ground, we obtain $(x \approx t') \in \Lambda\sigma$.

Consider the inference rule Hierarchic Superposition Right, Definition 6,

$$\mathcal{I}\frac{\Lambda_1 \parallel \Gamma_1 \to \Delta_1, l \approx r \qquad \Lambda_2 \parallel \Gamma_2 \to \Delta_2, s[l'] \approx t}{(\Lambda_1, \Lambda_2 \parallel \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2, s[r] \approx t)\sigma}$$

with the premises $C_1, C_2 \in N_n$ and conclusion C, respectively. As in the case of Hierarchic Equality Resolution, we learn that every base variable $x \in vars(C)$ comes from a clause C_1 or C_2, and $x = y\sigma$, where y is a base variable (not necessarily different from x) from $vars(C_1, C_2)$. Likewise, we conclude that an equation $(x \approx t')$ is contained in the constraint $(\Lambda_1, \Lambda_2)\sigma$ of C', for some ground base term $t' \in T_\Omega$. Analysis of the Hierarchic Superposition Left rule is similar.

Consider the Hierarchic Splitting rule, Definition 3,

$$\mathcal{S}\frac{\Lambda_1, \Lambda_2 \parallel \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2}{\Lambda_1 \parallel \Gamma_1 \to \Delta_1 \mid \Lambda_2 \parallel \Gamma_2 \to \Delta_2}$$

with the premise $C \in N_n$ and conclusions C_1 and C_2, respectively. By induction hypothesis, we know that for every base variable $x \in vars(C)$ the equation $(x \approx t) \in (\Lambda_1, \Lambda_2)$, for some $t \in T_\Omega$. As, by Definition 3, $\Lambda_1, \Gamma_1, \Delta_1$ on one hand, and $\Lambda_2, \Gamma_2, \Delta_2$ on the other are variable-disjoint, it holds that for every base variable $x \in vars(C_i)$, there exists a term $t \in T_\Omega$, such that the atom $(x \approx t) \in \Lambda_i$, for each $i \in \{1, 2\}$.

An application of the Constraint Refutation rule is a trivial case as the rule's conclusion is an empty clause. □

An application of the Hierarchic Superposition Left/Right rule can produce an assignment of two variables, say $(x \approx y)$. If the variables are base, we replace $(x \approx y)$ with $(s \approx t)$, where s and t are the ground base terms, which the variables are respectively assigned to (Proposition 13), and move the equation $(s \approx t)$ to the constraint of the conclusion, where it gets negated in the case of the Hierarchic Superposition Right rule. We call this transformation the *variables assignments grounding*, and write VAG(C) to denote the result of applying it to a clause C. Obviously, VAG(C) is equivalent to C. The variable assignment grounding is implicitly applied to every clause derived.

Proposition 14 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained from a ground clause set by basification and abstraction; let $C = \Lambda \parallel \Gamma \to \Delta$ be an arbitrary clause in N_i, $i \geq 0$. If variable assignment grounding is applied to every clause derived, then every literal in Λ is either*

- *an assignment* $x \approx t$*, or*
- *a (dis)equation* $s_1 \mathrel{\dot{\approx}} s_2$*,*

where x *is a base variable,* t*,* s_1*, and* s_2 *ground base terms,* $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$.

Proof We give a proof by induction on the length of derivation of N_i.

Induction Base For N_0 the assertion follows from the definition of abstraction and Corollary 11.

Induction Hypothesis Assume the assertion holds for all clause sets N_i derived from N_0, where $i \leq n$, for some $n > 0$.

Induction Step Let $N_{n+1} = N_n \cup C$, where C is a conclusion of an inference rule application to some clauses in N_n. Consider a Hierarchic Superposition Left inference, Definition 5,

$$\mathcal{I}\frac{\Lambda_1 \parallel \Gamma_1 \to \Delta_1, l \approx r \qquad \Lambda_2 \parallel s[l'] \approx t, \Gamma_2 \to \Delta_2}{(\Lambda_1, \Lambda_2 \parallel s[r] \approx t, \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma}$$

with the premises $C_1, C_2 \in N_n$ and conclusion C, respectively. The constraint $\Lambda = (\Lambda_1, \Lambda_2)\sigma$ of C is the conjunction of the premises' constraints with the substitution σ applied onto them, therefore every literal $L' \in \Lambda$ equals $L\sigma$, for some $L \in \Lambda_1 \cup \Lambda_2$. As σ is simple and the premises are abstracted, the substitution σ can unify a base variable only with another base variable. By induction hypothesis, every literal $L \in \Lambda_1 \cup \Lambda_2$ is either:

- an assignment $x \approx t$, then $L\sigma = (x \approx t)\sigma = (y \approx t)$, where $y = x\sigma$; or
- a (dis)equation $s_1 \approx s_2$, then $L\sigma = (s_1 \approx s_2)\sigma = (s_1 \approx s_2)$.

If the rule produces an assignment $x \approx y$ of two base variables in the free part of the conclusion, it is moved to the constraint and grounded by the variable assignment grounding transformation, implicitly applied to every clause in the derivation.

Analysis of the other inference rules (except Constraint Refutation and Hierarchic Splitting) is similar. The Hierarchic Splitting rule produces two clauses, whose constraints are subsets of the premise's constraint, which enjoys the stated property by induction hypothesis. The induction step trivially holds for an application of the Constraint Refutation inference rule. □

4.3 Termination

Here we show that the SUP(T) calculus following a particular strategy terminates on all clause sets obtained from ground clause sets by basification and abstraction. We prove the termination by first showing the following three statements: (i) every non-Horn clause can be split into Horn clauses; (ii) every base variable occurring in a clause is assigned to some ground base term via the constraint of the clause, and the number of such base terms is limited by a certain finite number; since all variables are base, the number of different variables in a clause is bounded; and (iii) the maximum size of literals is also limited (with an appropriately chosen ordering). Then we show that these statements and the fact that the enrichment signature is finite imply that after finitely many new clauses have been derived any further clause inferred is a variant of some clause previously derived.

4.3.1 Limiting the Length of Derived Clauses by Exhaustive Splitting

A bounded length of clauses in any SUP(T) derivation can be gained by splitting all clauses to Horn clauses and using eager selection for SUP(T) inferences. In Proposition 15 we prove that all clauses derived from a set of Horn clauses by SUP(T) inferences with eager selection do not exceed a certain length.

First, we show that a clause set, basified and abstracted, can actually be split into Horn clauses. Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained from a ground clause set N by basification and abstraction. In Propositions 13 and 14, we have shown that for any clause set $N_i, i \geq 0$, in the derivation and any clause $C = \Lambda \parallel \Gamma \to \Delta$ in N_i the following hold:

- every base variable $x \in \mathit{vars}(C)$ is assigned to some ground base term t via an equation $(x \approx t) \in \Lambda$;
- the constraint Λ consists of literals of the two types:
  - assignments $x \approx t$ of a variable x to a ground base term t, and
  - (dis)equations $s_1 \mathrel{\dot{\approx}} s_2$, where s_1 and s_2 are ground base terms and $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$.

From the structure of Λ, we see that no two distinct variables $x, y \in \mathit{vars}(\Lambda)$ occur in the same literal, which means that the constraint Λ can always be split into n variable disjoint subsets, where $n = |\mathit{vars}(\Lambda)|$, the number of distinct variables occurring in Λ. Now, assume x is an arbitrary base variable occurring in the free part of C more than once, say $k > 1$ times; let us denote this by $C = \Lambda \parallel (\Gamma \to \Delta)[x : k]$. The variable x is assigned to some term $t \in T_\Omega$ via an equation $(x \approx t) \in \Lambda$. If we replace all subsequent occurrences of the variable x in Γ and Δ with fresh unique variables $x_2, \ldots, x_k$ and extend the constraint Λ with literals $x_2 \approx t, \ldots, x_k \approx t$, we obtain an equivalent clause

$$\Lambda', x_2 \approx t, \ldots, x_k \approx t \parallel (\Gamma' \to \Delta')[x : 1, x_2 : 1, \ldots, x_k : 1],$$

where x and each x_i occur in Γ' and Δ' only once. Applying this transformation to every variable in C that has multiple occurrences in the free part, gives us a clause $C' = \Lambda'' \parallel \Gamma'' \to \Delta''$ which is equivalent to C and where every base variable $x \in \mathit{vars}(C')$ has at most one occurrence in the free part $\Gamma'' \to \Delta''$ of the clause, implying that no two free literals share a base variable. We call the exhaustive application of the transformation to all variables in the clause *variable cloning* and write $\mathrm{VC}(C)$ to denote the result of applying it to a clause C. Note that the obtained clause $C' = \mathrm{VC}(C)$ enjoys the statements of Propositions 13 and 14. Since all variables in C' are base, literals in the free part of C' are all variable-disjoint. After applying variables cloning to a non-Horn clause, it can be split into Horn clauses by a sequence of the Hierarchic Splitting rule applications.

As any non-Horn clause in N_0 can be split into Horn clauses, we assume form now on that all clauses in N_0 are Horn, without loss of generality. Actually, it is sufficient to apply splitting only once exhaustively to clauses in N_0, and then the calculus generates only further Horn clauses.

Proposition 15 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation with eager selection. If N_0 is a Horn clause set, then every clause $C \in N_i, i \geq 1$, contains at most as many free literals as the longest premise of the inference, deriving C, does.*

Proof An application of the Hierarchic Equality Resolution rule to a clause $C \in N_0$ obviously yields a Horn conclusion with a decremented number of free literals. Consider the inference rules Hierarchic Superposition Left/Right. Since all free negative literals are selected in every clause in N_0, an inference application is possible only between either two positive unit clauses (Hierarchic Superposition Right), or between a positive unit clause and a Horn clause (Hierarchic Superposition Left), yielding in each case a Horn clause with the number of literals equal at most to the number of literals in the longest premise (actually, the number of free literals is smaller than in the longest premise, only if the inference produces an assignment of two base variables which is moved to the constraint of the conclusion). Applying this argument inductively, the assertion follows for every clause set N_i in the derivation $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$. □

Assume M is a set of abstracted clauses; we write $n_{\mathcal{L}}(M)$ to denote the maximum number of free literals in a clause from M:

$$n_{\mathcal{L}}(M) = \max_{C \in M}\{|\Gamma| + |\Delta| \mid C = \Lambda \parallel \Gamma \to \Delta\} \tag{4.1}$$

Corollary 16 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation with eager selection. If N_0 is a Horn clause set, then the number of free literals in any clause $C \in N_i$, for any $i \geq 0$, is at most $n_{\mathcal{L}}(N_0)$.*

Since basification, abstraction, and splitting, which produce N_0 out of a given ground clause set N, can only decrease the number of *free* literals in a clause $C' \in N_0$ in comparison to the number of *all* literals in the corresponding clause $C \in N$, we conclude that $n_{\mathcal{L}}(N_0) \leq \max_{C \in N}(|C|)$.

4.3.2 Limiting the Number of Variables in Derived Clauses

Given a set M of abstracted clauses, we write $n_{\mathcal{V}}(M)$ to denote the overall number of ground base terms, to each of which there is an assignment of a variable via an equation in the constraint of a clause in M; or more formally:

$$n_{\mathcal{V}}(M) = |\bigcup_{C \in M} \{t \in T_\Omega \,|\, (x \approx t) \in \Lambda, \text{ where } x \in \mathit{vars}(C) \quad \text{and} \quad C = \Lambda \parallel \Gamma \to \Delta\}| \tag{4.2}$$

Example For the clause set N'' from the previous example (page 441), we have $n_{\mathcal{V}}(N'') = 5$, and the terms, to which base variables are assigned, are (in the order of appearance): b, a, $1 + \mathsf{a}$, c, 1.

Proposition 17 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained from a ground clause set by basification and abstraction. Then $n_{\mathcal{V}}(N_i) \leq n_{\mathcal{V}}(N_0)$, for every $i \geq 0$.*

Proof For the case $i = 0$ the assertion trivially holds. Let $N_{i+1} = N_i \cup C$, where C is the conclusion of an inference with premises in N_i. Assume $n_{\mathcal{V}}(N_i) \leq n_{\mathcal{V}}(N_0)$, induction hypothesis. In the proofs of Propositions 13 and 14, we have shown that every ground base term s, appearing in the constraint of C, comes from one of the inference's premises. Therefore, every term t, for which there is an assignment $x \approx t$ in the constraint of C, is inherited from N_i, implying that $n_{\mathcal{V}}(N_{i+1}) \leq n_{\mathcal{V}}(N_i)$, hence, by induction hypothesis, $n_{\mathcal{V}}(N_{i+1}) \leq n_{\mathcal{V}}(N_0)$. □

Note that Proposition 17 requires neither eager selection for the derivation, nor N_0 being a Horn clause set, nor variables assignment grounding in clauses derived.

According to Proposition 14, for an arbitrary clause $C = \Lambda \parallel \Gamma \to \Delta$ in the derivation $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ with variable assignment grounding applied to all derived clauses, it holds that every base literal in Λ containing a variable x is of the form $x \approx t$, where $t \in T_\Omega$. Assume the variable x occurs in Λ more than once, say $k > 1$ times, then $L_1 = (x \approx t_1)$, …, $L_k = (x \approx t_k)$ are all literals in Λ with an entry of x, where $t_1, \ldots, t_k \in T_\Omega$ and k is the number of occurrences of x in Λ. By replacing all occurrences of x in each L_i except L_1 with the term t_1, we obtain literals $L'_2 = (t_1 \approx t_2)$, …, $L'_n = (t_1 \approx t_n)$; then the following clause

$$(\Lambda \backslash \{L_2, \ldots, L_n\} \cup \{L'_2, \ldots, L'_n\}) \parallel \Gamma \to \Delta,$$

whose constraint is obtained from Λ by replacing each L_i with L'_i, for all $i \in \{2, \ldots, k\}$, is equivalent to the original clause C. Applying this transformation to every variable that has multiple occurrences in Λ, we obtain an equivalent clause $C' = \Lambda' \parallel \Gamma \to \Delta$, in which every base variable $x \in \mathit{vars}(C')$ is assigned to a single ground base term $t \in T_\Omega$ via the equation $(x \approx t) \in \Lambda'$. We call the exhaustive application of the transformation to all variables in the clause *variable assignment propagation*, and write $\mathrm{VAP}(C)$ to denote the result of applying it to a clause C.

Now, assume x, y are two arbitrary distinct variables in the obtained clause $C' = \mathrm{VAP}(C)$. The constraint Λ' contains two literals $x \approx s$, $y \approx t$, for some ground base $s, t \in T_\Omega$, which are uniquely defined. If $s = t$, then the clause $C'[x/y]$ obtained from C' by replacing every occurrence of y with x is equivalent to C'. Note that $\Lambda'[x/y]$ contains two identical literals $x \approx s$, one of which can be safely removed. After exhaustively merging all variables that are assigned to the same term, we obtain a clause $C'' = \Lambda'' \parallel \Gamma' \to \Delta'$, which is equivalent to C', hence, equivalent to C, and in which any two distinct base variables are assigned to distinct ground base terms via respective equations in the constraint Λ'' of C'':

$$x \neq y \;\Rightarrow\; \{x \approx s, y \approx t\} \subseteq \Lambda'' \quad \text{and} \quad s \neq t,$$

for any base variables $x, y \in vars(C'')$ and some ground base terms s and t. We call the exhaustive application of the transformation to all variables in the clause *variable merging*, and write $\mathrm{VM}(C)$ to denote the result of applying it to a clause C.

Note that variable assignment propagation and variable merging are compatible with Propositions 13 and 14. Although it is not mentioned in the definitions of the SUP(T) inference rules, we implicitly apply variable assignment grounding, variable assignment propagation, and variable merging to every clause in the derivation. We call the cascade of the three transformations—in the given order—the *grounding-propagation-merging of variables*, and write $\mathrm{MPG}(C)$ to denote the result of applying the combined transformation to a clause C: $\mathrm{MPG}(C) = \mathrm{VM}(\mathrm{VAP}(\mathrm{VAG}(C)))$. For the sake of conciseness, we use a term *MPG transformation* for the combined operation.

It is important to notice, that variable merging is antagonistic to variable cloning (described in Sect. 4.3.1), but the former is performed *after* application of an inference, whereas the latter is done *prior to* applying the Hierarchic Splitting rule, so that the two operations are not in conflict to each other, particularly because splitting is applied exhaustively before any other inference takes place.

Since every time a clause is derived all its base variables, which are assigned to the same term, are merged, the number of base variables in any clause derived is bounded by the number of different base terms, to which there are assignments in the constraint of the derived clause; and the number of all such terms, as we have shown in Proposition 17, is not increasing.

Corollary 18 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained from a ground clause set by basification and abstraction. If the MPG transformation is applied to every clause in the derivation, then*

$$|\{x \in vars(C) \mid sort(x) = S\}| \;\leq\; n_{\mathcal{V}}(N_0),$$

for any clause $C \in N_i$, $i \geq 0$.

Since all variables in $C \in N_i$ are base, we conclude that the number of *all* variables in C is bounded by $n_{\mathcal{V}}(N_0)$.

4.3.3 Limiting the Size of Literals in Derived Clauses

A key condition to ensure a finite saturation of a given clause set N_0 by the SUP(T) calculus is to keep the size of literals limited. Here we present a reduction ordering that limits the growth of the literals' size in any SUP(T) derivation with respect to this ordering.

Definition 19 (*Hierarchic lexicographic path ordering HLPO(γ)*) Let $\gamma : T_{\Omega'}(\mathcal{X}') \to \mathbb{N}$ be a total function. The *hierarchic lexicographic path ordering* $\succ_{\mathrm{H}(\gamma)}$ augmenting the function γ is defined by: $t \succ_{\mathrm{H}(\gamma)} s$ iff:

1. $\gamma(t) > \gamma(s)$, or
2. $\gamma(t) = \gamma(s)$ and $t \succ_{\mathrm{lpo}} s$,

where $\succ_{\mathrm{lpo}}$ is the standard LPO, see Definition 1 for LPO.

In general, the above HLPO ordering is not stable under substitutions. However in the context of simple substitutions and an appropriately defined function γ, it actually becomes a reduction ordering that is well-founded and total on ground terms.

We define a function $\omega : T_{\Omega'}(\mathcal{X}') \to \mathbb{N}$, which for a given term $t \in T_{\Omega'}(\mathcal{X}')$ returns the number of free operator symbol occurrences in t, as follows:

$$\omega(t) = |\{p \in \rho(t) \mid top(t/p) \in \Omega''\}|. \tag{4.3}$$

The function ω is extended to atoms and literals as

$$\omega(s \mathrel{\dot{\approx}} t) = \omega(s) + \omega(t),$$

where $s, t \in T_{\Omega'}(\mathcal{X}')$ and $\dot{\approx} \in \{\approx, \not\approx\}$.

An instance of the Hierarchic lexicographic path ordering HLPO(ω) augmenting the function ω compares two terms by, first, comparing the number of free operator symbol occurrences in the terms, and, second, if the number of free operator symbol occurrences is the same, by comparing the terms with respect to the standard LPO.

In the context of the SUP(T) calculus for the ground FOL(T) fragment, terms involved in inferences may contain only base variables from $\mathcal{X}$. Moreover, SUP(T) admits only simple substitutions, hence the variables may be substituted only with base terms. Next, we show that $\succ_{\mathrm{H}(\omega)}$ is a reduction ordering under these conditions. Propositions 20 and 21 followed hold only in the context of the ground FOL(T) fragment—for the non-ground case we give corresponding propositions in Sect. 5.

Proposition 20 *The hierarchic lexicographic path ordering $\succ_{\mathrm{H}(\omega)}$ augmenting the function ω is a reduction ordering for $T_{\Omega'}(\mathcal{X})$, with respect to simple substitutions.*

Proof We need to show that $\succ_{\mathrm{H}(\omega)}$ is irreflexive, transitive, and well-founded binary relation that is compatible with contexts, stable under simple substitutions, and has the subterm property. Let $t, s, r \in T_{\Omega'}(\mathcal{X})$ be arbitrary Ω'-terms.

Irreflexivity of $\succ_{\mathrm{H}(\omega)}$ follows from that of $>$ and $\succ_{\mathrm{lpo}}$.

Transitivity Assume $t \succ_{\mathrm{H}(\omega)} s \succ_{\mathrm{H}(\omega)} r$. We are to show that $t \succ_{\mathrm{H}(\omega)} r$. Note that $\omega(t) \geq \omega(s) \geq \omega(r)$. Consider two possible cases:

- $\omega(t) = \omega(s) = \omega(r)$, then $t \succ_{\mathrm{lpo}} s \succ_{\mathrm{lpo}} r$, and $t \succ_{\mathrm{lpo}} r$ follows by transitivity of $\succ_{\mathrm{lpo}}$, consequently $t \succ_{\mathrm{H}(\omega)} r$;
- $\omega(t) > \omega(s)$ or $\omega(s) > \omega(r)$, then $\omega(t) > \omega(r)$, consequently $t \succ_{\mathrm{H}(\omega)} r$.

Well-foundedness Let $t_1, t_2, \ldots$ be arbitrary terms from $T_{\Omega'}(\mathcal{X})$ such that $t_1 \succ_{\mathrm{H}(\omega)} t_2 \succ_{\mathrm{H}(\omega)} \ldots$. We are to show that any such a descending chain of terms is finite. We prove it by induction on $\omega(t_1)$.

Induction Base If $\omega(t_1) = 0$, then $\omega(t_1) = \omega(t_2) = \ldots = 0$. Therefore, $t_1 \succ_{\mathrm{lpo}} t_2 \succ_{\mathrm{lpo}} \ldots$, and the chain is finite by well-foundedness of $\succ_{\mathrm{lpo}}$.

Induction Hypothesis Assume that any descending chain $t_1 \succ_{\mathrm{H}(\omega)} t_2 \succ_{\mathrm{H}(\omega)} \ldots$, where $\omega(t_1) \leq n$ for some $n \geq 0$, is finite.

Induction Step Let $\omega(t_1) = n + 1$. There are two possible cases:

- $\omega(t_1) = \omega(t_2) = \ldots = n + 1$, then this case is analogous to the induction base;
- there exists an index $\ell > 1$ such that $\omega(t_1) > \omega(t_\ell)$. Let k be the smallest index such that $\omega(t_1) \geq \omega(t_k) + 1$. As $\omega(t_1) = \omega(t_2) = \ldots = \omega(t_{k-1})$, we know that $t_1 \succ_{\mathrm{lpo}} t_2 \succ_{\mathrm{lpo}} \ldots \succ_{\mathrm{lpo}} t_{k-1}$. By well-foundedness of $\succ_{\mathrm{lpo}}$, the left subchain $t_1 \succ_{\mathrm{H}(\omega)} t_2 \succ_{\mathrm{H}(\omega)} \ldots \succ_{\mathrm{H}(\omega)} t_{k-1}$ is finite (and so is k). By induction hypothesis, the right subchain $t_k \succ_{\mathrm{H}(\omega)} t_{k+1} \succ_{\mathrm{H}(\omega)} \ldots$ is finite as well.

Thus, any descending chain $t_1 \succ_{\mathrm{H}(\omega)} t_2 \succ_{\mathrm{H}(\omega)} \ldots$ is finite, and, hence, $\succ_{\mathrm{H}(\omega)}$ is well-founded.

Compatibility with Contexts Let $r = f(r_1, \ldots, r_n)$, for some $r_1, \ldots, r_n \in T_{\Omega'}(\mathcal{X})$ and $n \geq 1$. Assume $t \succ_{\mathrm{H}(\omega)} s$. Consider the terms $r' = r[t]_i$ and $r'' = r[s]_i$, for some $1 \leq i \leq n$, obtained from r by replacing its i-th immediate subterm r/i by t and s, respectively. Then

$$\omega(r') = \omega(r) - \omega(r/i) + \omega(t)$$
$$\omega(r'') = \omega(r) - \omega(r/i) + \omega(s)$$

If $\omega(t) > \omega(s)$, then $\omega(r') > \omega(r'')$, thus $r' \succ_{\mathrm{H}(\omega)} r''$ by condition 1 of Definition 19. If $\omega(t) = \omega(s)$, then $t \succ_{\mathrm{lpo}} s$ and $\omega(r') = \omega(r'')$. Let us compare r' and r'' with respect to $\succ_{\mathrm{lpo}}$. Since $r'/j = r''/j$, for every $1 \leq j \leq n$, $j \neq i$, and $r'/i = t \succ_{\mathrm{lpo}} s = r''/i$, we obtain, by condition 2a of Definition 1, that $r' \succ_{\mathrm{lpo}} r''/j$, for every $1 \leq j \leq n$. Moreover, $(r'/1, \ldots, r'/n)\ (\succ_{\mathrm{lpo}})_{\mathrm{lex}}\ (r''/1, \ldots, r''/n)$, hence, by condition 2c, we obtain $r' \succ_{\mathrm{lpo}} r''$, yielding, by condition 2 of Definition 19, that $r' \succ_{\mathrm{H}(\omega)} r''$. So, $\succ_{\mathrm{H}(\omega)}$ is compatible with contexts.

Stability Under Simple Substitutions Assume $t \succ_{\mathrm{H}(\omega)} s$. Let σ be an arbitrary simple substitution. Since all variables occurring in t and s are of the base sort, they can be mapped by σ only to base terms, hence we have

$\omega(t\sigma) = \omega(t)$ and $\omega(s\sigma) = \omega(s)$. If $\omega(t) > \omega(s)$, then $\omega(t\sigma) > \omega(s\sigma)$, hence $t\sigma \succ_{\mathrm{H}(\omega)} s\sigma$. If $\omega(t) = \omega(s)$, then it must hold that $t \succ_{\mathrm{lpo}} s$. Since $\succ_{\mathrm{lpo}}$ is stable under substitutions, we obtain $t\sigma \succ_{\mathrm{lpo}} s\sigma$, consequently, $t\sigma \succ_{\mathrm{H}(\omega)} s\sigma$, by condition 2 of Definition 19. So, $\succ_{\mathrm{H}(\omega)}$ is stable under simple substitutions.

Subterm Property Let $t = f(t_1, \ldots, t_n)$, for some $n \geq 1$, and $s = t_i$ be an immediate subterm of t, for some $1 \leq i \leq n$. If $f \in \Omega''$, then $\omega(t) \geq \omega(s) + 1$, consequently $\omega(t) > \omega(s)$, therefore $t \succ_{\mathrm{H}(\omega)} s$. For otherwise, $\omega(t) \geq \omega(s)$. If $\omega(t) > \omega(s)$, then $t \succ_{\mathrm{H}(\omega)} s$. For otherwise, $\omega(t) = \omega(s)$, but since $\succ_{\mathrm{lpo}}$ has the subterm property, we have $t \succ_{\mathrm{lpo}} s$, therefore $t \succ_{\mathrm{H}(\omega)} s$. So, $\succ_{\mathrm{H}(\omega)}$ has the subterm property. □

If M is a set of abstracted clauses, we write $n_{\mathcal{F}}(M)$ to denote the maximum number of free operator symbol occurrences in a free literal among all clauses in M:

$$n_{\mathcal{F}}(M) = \max_{C \in M} \{\omega(L) \mid L \in (\Gamma \to \Delta),\ C = \Lambda \parallel \Gamma \to \Delta\}. \tag{4.4}$$

Proposition 21 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained from a ground clause set by basification and abstraction. If $\succ_{\mathrm{H}(\omega)}$ is the underlying ordering, then for every free literal L in any clause $C \in N_i$, $i \geq 0$, the number of free operator symbol occurrences $\omega(L)$ is at most $n_{\mathcal{F}}(N_0)$.*

Proof We give a proof by induction on the length of derivation of N_i.

Induction Base For N_0 the assertion trivially holds.

Induction Hypothesis Assume the assertion holds for all clause sets N_i derived from N_0, where $i \leq n$, for some $n > 0$.

Induction Step Let $N_{n+1} = N_n \cup C$, where C is a conclusion of an inference rule application to some clauses in N_n. Let us first make the following observation: for any simple substitution σ and any term $t \in T_{\Omega'}(\mathcal{X})$ built over operators Ω' and base variables $\mathcal{X}$, the term $t\sigma$ contains as many free operator symbols as the term t does, i.e. $\omega(t\sigma) = \omega(t)$, because there is no non-base variable in t, and simple substitutions can map base variables only to base terms. Consider a Hierarchic Superposition Right inference, Definition 6,

$$\mathcal{I}\frac{\Lambda_1 \parallel \Gamma_1 \to \Delta_1, l \approx r \qquad \Lambda_2 \parallel \Gamma_2 \to \Delta_2, s[l'] \approx t}{(\Lambda_1, \Lambda_2 \parallel \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2, s[r] \approx t)\sigma}$$

with the premises $C_1, C_2 \in N_n$ and conclusion C, respectively. According to the definition of the rule,

$$\begin{array}{lll} & l\sigma = l'\sigma & \text{// by Cond. (i) of Def. 6} \\ \Rightarrow & \omega(l\sigma) = \omega(l'\sigma) & \\ \Rightarrow & \omega(l) = \omega(l') & \text{// as } l, l' \in T_{\Omega'}(\mathcal{X}) \text{ and } \sigma \text{ simple} \end{array}$$

Moreover,

$$\begin{array}{lll} & r\sigma \not\succ_{\mathrm{H}(\omega)} l\sigma & \text{// by Cond. (iii) of Def. 6} \\ \Rightarrow & \omega(r\sigma) \leq \omega(l\sigma) & \text{// acc. to Def. 19 of } \succ_{\mathrm{H}(\omega)} \\ \Rightarrow & \omega(r) \leq \omega(l) & \text{// as } l, r \in T_{\Omega'}(\mathcal{X}) \text{ and } \sigma \text{ simple} \end{array}$$

Therefore,

$$\begin{aligned} & \omega(s[r]\sigma) = \omega(s[r]) && \text{// as } s[r] \in T_{\Omega'}(\mathcal{X}) \text{ and } \sigma \text{ simple} \\ & \leq \omega(s[l]) && \text{// as } \omega(r) \leq \omega(l) \\ & = \omega(s[l']) && \text{// as } \omega(l) = \omega(l') \\ \Rightarrow \quad & \omega\big((s[r] \approx t)\sigma\big) \leq \omega(s[l'] \approx t) \\ & \leq n_{\mathcal{F}}(N_0) && \text{// by Induction Hypothesis} \end{aligned}$$

For any other literal $L\sigma$ in the inference's conclusion, where L is the corresponding literal from a premise, we have $\omega(L\sigma) = \omega(L) \leq n_{\mathcal{F}}(N_0)$, by Induction Hypothesis. The analysis of the other rules is similar. □

4.3.4 SUP(T) as a Decision Procedure for Ground FOL(T)

Lemma 22 *Let N_0 be obtained from a finite ground clause set N by basification and abstraction. If N_0 is a Horn clause set, then any SUP(T) derivation $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ with eager selection, MPG transformation, and reduction ordering $\succ_{\mathrm{H}(\omega)}$ is terminating.*

Proof According to Proposition 14, any literal in the constraint of a clause in the derivation is either a (dis)equation $s_1 \mathrel{\dot{\approx}} s_2$ between some ground base terms, where $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$, or an assignment $x \approx t$ of a base variable to a ground base term. In the proof of Proposition 14 we have shown that terms s_1, s_2, t are all inherited from N_0, meaning that no new base term is produced by an inference. In Proposition 18, we have shown that a clause in the derivation may contain at most $n_{\mathcal{V}}(N_0)$ different base variables. For finitely many different terms s_1, s_2, t and a limited number of variables, there can be only finitely many different literals of the above form, therefore there can be only finitely many different constraints Λ, up to variable renaming and removal of duplicate literals.

According to Corollary 16, any clause in the derivation may contain at most $n_{\mathcal{L}}(N_0)$ free literals, each of which, according to Proposition 21, may have at most $n_{\mathcal{F}}(N_0)$ function symbol occurrences, limiting thus the size of free literals. As all variables in the derivation are base, the number of different variables in the free part of a clause is also limited by $n_{\mathcal{V}}(N_0)$. For the finite set Ω'' of free function symbols occurring in N_0, a limited number and size of literals, and limited number of variables, there exist only finitely many different free parts $\Gamma \to \Delta$, up to variable renaming.

Combining the two observations above, we conclude that there can be only finitely many different clauses $\Lambda \parallel \Gamma \to \Delta$ in the derivation $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$, up to variable renaming. Thus, after sufficiently (and finitely) many clauses have been derived, any further inference is redundant. Since only irredundant inferences have to be performed, $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ is terminating. □

Theorem 23 *Let T be a base theory in which the $\exists^*$ fragment is decidable. Then the SUP(T) calculus with eager selection, MPG transformation, reduction ordering $\succ_{\mathrm{H}(\omega)}$, and variable cloning and splitting preferred to other inferences, is a decision procedure for the ground clause FOL(T) fragment.*

Proof Basification, presented in Sect. 4.1, transforms a finite set N of ground FOL(T) clauses into a finite equisatisfiable clause set N', which is sufficiently complete, Lemma 12. Let N'' be a clause set obtained from N' by abstraction, Sect. 4.2. In Sect. 4.3.1 we have shown that every clause in N'' can be split into Horn clauses, if variable cloning (Sect. 4.3.1, page 444) is first applied to it. Let $N_0^1, \ldots, N_0^n$ be the sets of Horn clauses obtained by exhaustively applying splitting to N''. The clause set N'' is satisfiable if and only if at least one of $N_0^1, \ldots, N_0^n$ is so. Since SUP(T) is refutationally complete for all sufficiently complete clause sets, every N_0^i, where $1 \leq i \leq n$, is satisfiable if and only if a SUP(T) derivation $N_0^i \vdash N_1^i \vdash N_2^i \vdash \cdots$ does not contain an empty clause $\square$. According to Lemma 22, any SUP(T) derivation $N_0^i \vdash N_1^i \vdash N_2^i \vdash \cdots$ with eager selection, MPG transformation (Sect. 4.3.2, page 446) and the reduction ordering $\succ_{\mathrm{H}(\omega)}$ is terminating, therefore satisfiability of every N_0^i can be determined,

hence satisfiability of N'' and N can be determined as well. Thus, the SUP(T) calculus with eager selection, reduction ordering $\succ_{\mathrm{H}(\omega)}$, and variable cloning and splitting preferred to other inferences, terminates for the ground clause FOL(T) fragment. If the base theory supports a decision procedure for the $\exists^*$ fragment, then Constraint Refutation can be decided. □

Note that the usage of the hierarchic LPO $\succ_{\mathrm{H}(\omega)}$ yields an upper bound on the maximum size of free literals and, hence, guarantees the existence of only finitely many different free literals over a finite set Ω'' of free operator symbols and a limited number of variables. Thus, even if the length of the free part of a clause not limited, there can be only finitely many different free parts $\Gamma \to \Delta$, up to variable renaming and duplication of literals. This observation supports an alternative strategy involving neither splitting nor eager selection: indeed, the two ingredients are not really needed for termination, but what is required in return is an additional Hierarchic Factoring rule [7], which is necessary for refutational completeness in the absence of the splitting rule.

According to the definition of SUP(T) inference rules, Sect. 3, an empty clause $\square$ can be produced only by an application of the Constraint Refutation rule, which is to apply to a set of base clauses (which are abstracted clauses with the free part empty). As every base variable x is assigned to a ground base term t, Proposition 13, any non-ground base clause can be grounded by propagation of all assignments $x \approx t$. The term t may contain base parameters introduced at the basification step. The parameters are essentially base variables existentially quantified on the top of the overall clause set. Thus, an application of the Constraint Refutation rule reduces to checking satisfiability of a base formula with all variables in it existentially quantified. For this reason, the only requirement we impose on the theory T is the decidability of its existential fragment.

5 Non-ground FOL(T)

In this section we consider an application of SUP(T) to a non-ground FOL(T) fragment, for which SUP(T) is a decision procedure as well. Due to the hierarchic design of SUP(T), the decidability result for the ground FOL(T) fragment can be easily extended to some non-ground fragments. Here we present an instance of such an extension.

Definition 24 (*BSHE(GBST) class)* We call the class consisting of all Horn clause sets, where

(i) every non-predicate operator symbol of non-zero arity ranges into the base sort, and
(ii) all base sort terms are ground,

the *Bernays-Schönfinkel Horn class with equality and ground base sort terms*, *BSHE(GBST)*.

Let N be a clause set from the BSHE(GBST) class, then any literal occurring in a clause from N may contain a base or free predicate symbol, equality symbol, variables of the free sort S'' which all occur immediately below a free predicate or equality symbol, base function symbols, free function symbols of non-zero arity all ranging into the base sort S, and constant symbols ranging either into the base S or the free sort S''. Thus, any non-predicative compound term occurring in N is ground and of the base sort S.

Example Let N be a given clause set containing just one clause:

$$N = \{f(a) + 2 > b,\ P(u, 3f(a), c) \ \to\ Q(u, b)\},$$

where the background theory T is the rational linear arithmetic; $+, >, 2, 3 \in \Omega$, and $P, Q, f, a, b, c \in \Omega''$; the function symbols f, b ranging into the base sort S, and a, c ranging into the free sort S''; $u \in \mathcal{X}''$ is a non-base variable. The set N agrees with conditions (i) and (ii) of Definition 24 of the BSHE(GBST) class.

As usual, the input clause set N is first basified resulting in a clause set $N' = N_I \cup N_B$:

$$\begin{aligned} N_I &= \{\ \mathsf{a} + 2 > \mathsf{b},\ P(u, 3\mathsf{a}, c) \ \to\ Q(u, \mathsf{b})\ \}, \\ N_B &= \{ \qquad\qquad\qquad\quad \to\ f(a) \approx \mathsf{a}, \\ &\qquad\qquad\qquad\quad \to\ b \approx \mathsf{b}\ \} \end{aligned}$$

and then abstracted into a clause set $N'' = N'_I \cup N'_B$:

$$\begin{aligned} N'_I &= \{\ \mathsf{a}+2 > \mathsf{b},\ x \approx 3\mathsf{a},\ y \approx \mathsf{b} \parallel P(u,x,c) \rightarrow Q(u,y)\ \}, \\ N'_B &= \{\ \qquad\qquad\quad x \approx \mathsf{a} \parallel \qquad\quad \rightarrow f(a) \approx x, \\ &\qquad\qquad\quad x \approx \mathsf{b} \parallel \qquad\quad \rightarrow b \approx x\ \}. \end{aligned}$$

Let N be an arbitrary clause set from the BSHE(GBST) class. From now on we write $N' = N_I \cup N_B$ to denote the clause set obtained from N by basification, where N_I is the set of the basified clauses from N, and N_B is the set of clauses basifying those in N_I; and N'' to denote the clause set N' abstracted. Let $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$ be a SUP(T) derivation with MPG transformation, where $N_0 = N''$. Most results for the ground fragment stated in propositions and corollaries of the previous section hold also for the BSHE(GBST) class, and their proofs are valid in the current context as well, therefore here we only reformulate the related properties to fit the BSHE(GBST) class without providing proofs.

The presence of non-base variables interferes neither basification nor abstraction because they may occur only immediately below equality or free predicate symbols. Therefore, terms occurring in basified BSHE(GBST) clause sets have the same structural properties as in the case of basified ground FOL(T) clause sets.

Proposition 25 *Let N be a clause set from the BSHE(GBST) class. Let $N' = N_I \cup N_B$ be a clause set obtained from N by basification. For any literal $L = (t_1 \mathrel{\dot{\approx}} t_2)$ in any clause from N_I, where $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$, it holds that*

$$\forall\, p \in \rho(t_i) : top(t_i/p) \in \Omega'' \Rightarrow sort(t_i/p) = S'',$$

for every $i \in \{1,2\}$. The only literal $L = (t \approx \mathsf{a})$ in any clause from N_B is an equation between a base parameter a and a term t, such that $top(t) \in \Omega''$, $sort(t) = S$, and

$$\forall\, p \in \rho(t) : p > \varepsilon, top(t/p) \in \Omega'' \Rightarrow sort(t/p) = S''.$$

Corollary 26 *Let N be a clause set from the BSHE(GBST) class. If a clause set N' is obtained from N by basification, then for any literal $L = (t_1 \mathrel{\dot{\approx}} t_2)$ in any clause from N', where $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$, it holds that*

$$\forall\, p \in \rho(t_i) : top(t_i/p) \in \Omega \Rightarrow t_i/p \in T_\Omega,$$

for every $i \in \{1,2\}$.

Likewise, basification of a BSHE(GBST) clause set N produces an equisatisfiable clause set N'. From Proposition 25 it follows that all non-base terms t of the base sort S occurring in N' are sufficiently defined and occur only in positive literals of the form $t \approx t'$, where t' is ground base term, which makes the set N' sufficiently complete.

Lemma 27 *Let N be a BSHE(GBST) clause set. If N' is obtained from N by basification, then N' is sufficiently complete, and N' and N are equisatisfiable.*

Since we consider only Horn clauses, the set N_0 is already Horn (as neither basification nor abstraction add atoms to the succedent of a clause), hence no splitting (and variable cloning) needs to be applied in any derivation from N_0. Thus, the properties regarding the maximal number of free literals stated in Proposition 15 and subsequent Corollary 16 are true for the BSHE(GBST) fragment as well.

The derivation invariants regarding base variables and structure of constraints hold in the context of the BSHE(GBST) class too.

Proposition 28 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained by basification and abstraction from a BSHE(GBST) clause set; let $C = \Lambda \parallel \Gamma \rightarrow \Delta$ be an arbitrary clause in $N_i, i \geq 0$. Then for*

every base variable $x \in vars(C)$ there exists a ground base term $t \in T_\Omega$ such that the equation $x \approx t$ is in the constraint Λ of C.

Proposition 29 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained by basification and abstraction from a BSHE(GBST) clause set; let $C = \Lambda \parallel \Gamma \to \Delta$ be an arbitrary clause in $N_i, i \geq 0$. If variable assignment grounding is applied to every clause derived, then every literal in Λ is either*

- *an assignment $x \approx t$, or*
- *a (dis)equation $s_1 \mathrel{\dot{\approx}} s_2$,*

where x is a base variable, t, s_1, and s_2 ground base terms, $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$.

Proposition 30 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained by basification and abstraction from a BSHE(GBST) clause set. Then[3] $n_\mathcal{V}(N_i) \leq n_\mathcal{V}(N_0)$, for every $i \geq 0$.*

Corollary 31 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained by basification and abstraction from a BSHE(GBST) clause set. If the MPG transformation is applied to every clause in the derivation, then*

$$|\{x \in vars(C) \mid sort(x) = S\}| \;\leq\; n_\mathcal{V}(N_0),$$

for any clause $C \in N_i, i \geq 0$.

However, we have to take a special care of *free* variables. For this reason, we introduce another instance of the hierarchic lexicographic path ordering, Definition 19, which limits the number of non-base variables in a derived clause and ensures a bounded literal growth. Since all free function symbols of non-zero arity range into the base sort, any term of the free sort occurring in the input clause set N is either a non-base variable, or a free constant. Clearly, this property is preserved by basification, abstraction, and any inference rule application, and holds thus for all N_i in $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$. Therefore, during the derivation a non-base variable can be unified either with another non-base variable, or with a free constant. If a non-base variable is unified with a free constant, the number of free operator symbol occurrences in the superposed term can actually grow. To cope with this, we need to count in addition the number of non-base variable occurrences. Here we use an instance of HLPO(γ), where for γ we use the function $\phi : T_{\Omega'}(\mathcal{X}') \to \mathbb{N}$ defined as:

$$\phi(t) = |\{p \in \rho(t) \mid top(t/p) \in \Omega'' \cup \mathcal{X}''\}|, \tag{5.1}$$

which in contrast to the function ω, which counts only free operator symbol occurrences (see Equation 4.3), counts in addition occurrences of variables ranging into the free sort. The resulting instance $\succ_{\mathrm{H}(\phi)}$ of HLPO is a reduction ordering for the BSHE(GBST) fragment, with respect to simple substitutions.

Proposition 32 *Let all non-constant function symbols in Ω'' range into the base sort. The hierarchic lexicographic path ordering $\succ_{\mathrm{H}(\phi)}$ augmenting the function ϕ is a reduction ordering for $T_{\Omega'}(\mathcal{X}')$ with respect to simple substitutions.*

Proof Let $t, s, r \in T_{\Omega'}(\mathcal{X}'), r = f(r_1, \ldots, r_n)$, where $n \geq 1$, all non-constant function symbols in Ω'' range into the base sort, and $\mathcal{X}' = \mathcal{X} \cup \mathcal{X}''$ is a set of base and non-base variables, respectively. Argumentation for the properties irreflexivity, transitivity, well-foundedness, context compatibility, and the subterm property is exactly the same as in Proposition 20 with the only difference being in using the function ϕ in place of ω (the presence of non-base variables does not harm the arguments of Proposition 20 regarding the listed properties).

[3] We write $n_\mathcal{V}(M)$ to denote the overall number of ground base terms, to each of which there is an assignment of a variable via an equation in the constraint of a clause in the set of abstracted clauses M (see Equation 4.2 on page 445).

The proof of stability under simple substitutions is slightly different – due to the presence of non-base variables. Assume $t \succ_{\mathrm{H}(\phi)} s$. Let σ be a simple substitution. Any base variable $x \in vars(t)$ – analogously for s – can be mapped by σ only to a base term, and since base terms may contain only base symbols, we have $\phi(x\sigma) = 0 = \phi(x)$. As the only non-variable terms of the free sort are constants, any non-base variable $y \in vars(t)$ can be mapped by σ either to a non-base variable or a free constant, hence $\phi(y\sigma) = 1 = \phi(y)$. Thus, an application of σ to variables of t does not change the number of occurrences of free symbols (free operators and variables of the free sort), and, therefore, $\phi(t) = \phi(t\sigma)$; analogously $\phi(s) = \phi(s\sigma)$. If $\phi(t) > \phi(s)$, then $\phi(t\sigma) > \phi(s\sigma)$ as well, hence $t\sigma \succ_{\mathrm{H}(\phi)} s\sigma$. If $\phi(t) = \phi(s)$, then $\phi(t\sigma) = \phi(s\sigma)$, and it must also hold that $t \succ_{\mathrm{lpo}} s$. Since $\succ_{\mathrm{lpo}}$ is stable under substitutions, we know $t\sigma \succ_{\mathrm{lpo}} s\sigma$, consequently, $t\sigma \succ_{\mathrm{H}(\phi)} s\sigma$. So, $\succ_{\mathrm{H}(\phi)}$ augmenting ϕ is stable under simple substitutions. □

We redefine the function $n_{\mathcal{F}}$, Eq. 4.4, in the following way:

$$n_{\mathcal{F}}(M) = \max_{C \in M} \{\phi(L) \mid L \in \Gamma \cup \Delta,\ C = \Lambda \parallel \Gamma \to \Delta\}, \tag{5.2}$$

where M is a set of abstracted clauses.

Proposition 33 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation, where N_0 is obtained by basification and abstraction from a BSHE(GBST) clause set. If $\succ_{\mathrm{H}(\phi)}$ is the underlying ordering, then for every free literal L in any clause $C \in N_i$, $i \geq 0$, the number of free symbol occurrences $\phi(L)$ is at most $n_{\mathcal{F}}(N_0)$.*

Proof Analogous to Proposition 21. □

Lemma 34 *Let N_0 be obtained by basification and abstraction from a BSHE(GBST) clause set. Then any SUP(T) derivation $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ with eager selection, MPG transformation, and reduction ordering $\succ_{\mathrm{H}(\phi)}$ is terminating.*

Proof According to Proposition 29, any literal in the constraint of a clause in the derivation is either a (dis)equation $s_1 \mathrel{\dot{\approx}} s_2$ between some ground base terms, where $\mathrel{\dot{\approx}} \in \{\approx, \not\approx\}$, or an assignment $x \approx t$ of a base variable to a ground base term. The terms s_1, s_2, t are all inherited from N_0, meaning that no new base term is produced by an inference. According to Proposition 31, a clause in the derivation may contain at most $n_{\mathcal{V}}(N_0)$ different base variables. For finitely many different terms s_1, s_2, t and a limited number of variables, there can be only finitely many different literals of the above form, therefore there can be only finitely many different constraints Λ, up to variable renaming.

According to Corollary 16, any clause in the derivation may contain at most $n_{\mathcal{L}}(N_0)$ free literals, each of which, according to Proposition 33, may have at most $n_{\mathcal{F}}(N_0)$ function symbol occurrences, limiting thus the size of free literals. For a limited number and size of literals, there exist only finitely many different free parts $\Gamma \to \Delta$, up to variable renaming.

Combining the two observations above, we conclude that there can be only finitely many different clauses $\Lambda \parallel \Gamma \to \Delta$, up to variable renaming, in the derivation $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$. Thus, any inference from N_i, for some $i \geq 0$, with a conclusion C that can be matched to a clause $D \in N_i$ by variable renaming, is redundant. After sufficiently (and finitely) many clauses have been derived, any further inference is thus redundant. Since only irredundant inferences have to be performed, $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ is terminating. □

Theorem 35 *Let T be a base theory in which the existential fragment is decidable. Then SUP(T) with eager selection, MPG transformation, and reduction ordering $\succ_{\mathrm{H}(\phi)}$ augmenting the function ϕ is a decision procedure for the BSHE(GBST) class.*

Proof Basification, presented in Sect. 4.1, transforms a finite set N of BSHE(GBST) clauses into a finite equisatisfiable clause set N', which is sufficiently complete, Lemma 27.

Let N' be abstracted into a clause set N_0. As N is a Horn clause set, N_0 is so as well. Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a SUP(T) derivation with eager selection, MPG transformation, and underlying ordering $\succ_{\mathrm{H}(\phi)}$ augmenting the function ϕ. According to Lemma 34, the derivation is terminating.

If the original clause set N is unsatisfiable, so is N_0, and due to refutational completeness SUP(T) finitely refutes N_0 deriving an empty clause. If the original clause set N is satisfiable, so is N_0, and SUP(T) finitely saturates N_0 without deriving an empty clause. □

One application of the above decidability is reasoning (saturation and querying) in ontologies with arithmetical facts, such as time stamps, size/amount information, etc. Suda, Weidenbach, and Wischnewski have shown in [26] that the ontology YAGO[4] can be expressed in terms of the Bernays-Schönfinkel Horn class with equality, abbreviated BSHE, and presented a variant of superposition that decides the class. Typical examples of clauses in the representation of the ontology are (the examples are taken from [26]):

$$\begin{aligned} &\rightarrow bornIn(AlbertEinstein, Ulm) && \text{// for "Albert Einstein was born in Ulm"}\\ &\rightarrow human(AngelaMerkel) && \text{// for "Angela Merkel is a human"}\\ human(x) &\rightarrow mammal(x) && \text{// for "Every human is a mammal"} \end{aligned}$$

The ontology can be queried with questions like "who are the people who died in New York at the same place where their children were born?", which is formally encoded as the following conjecture

$$\exists x, y, z.\ diedIn(x, y) \wedge hasChild(x, z) \wedge bornIn(z, y) \wedge locatedIn(y, NewYork)$$

which after negating becomes the clause

$$diedIn(x, y), hasChild(x, z), bornIn(z, y), locatedIn(y, NewYork) \rightarrow$$

with all variables universally quantified.

If the ontology is further extended with arithmetical information, such as for instance:

$$\begin{aligned} &\rightarrow overInY(WWII, 1945) && \text{// for "The World War II ended in 1945"}\\ &\rightarrow diedInY(KarlTheGreat, 814) && \text{// for "Karl The Great died in 814"}\\ &\rightarrow gdp(Russia, 2011, 2.4 \cdot 10^{12}) && \text{// for "The GDP[5] of Russia was \$ 2.4 trillion in 2011"}\\ &\rightarrow population(USA, 2012, 313.8 \cdot 10^{6}) && \text{// for "The population of USA was 313.8 million in 2012"} \end{aligned}$$

then it can be addressed with queries like:

- "Was the GDP of Germany steadily growing every year by at least 3 % in 2008-2010?", which is encoded as the conjecture:
$$\forall x_1, x_2, y_1, y_2.\ gdp(Germany, y_1, x_1), gdp(Germany, y_2, x_2),$$
$$2008 \leq y_2, y_2 \leq 2010, y_2 = y_1 + 1 \rightarrow x_2 \geq 1.03 \cdot x_1$$
- "Was the GDP of Germany at least 3 % higher than that of any other country in Europe in the years 2008-2010?":
$$\forall u. \forall x_1, x_2, y.\ gdp(Germany, y, x_1), gdp(u, y, x_2), locatedIn(u, Europe),$$
$$u \not\approx Germany, 2008 \leq y, y \leq 2010 \rightarrow x_1 \geq 1.03 \cdot x_2$$

Now the query answering mechanisms presented in [29] can actually be extended according to the above decidability result to yield also a decision procedure for the extended language.

[4] YAGO (Yet Another Great Ontology) is the first automatically retrieved ontology out of Wikipedia and WordNet with accuracy of about 97% [25]; a well-known ontology in the information retrieval community.

[5] GDP (Gross domestic product)

6 Conclusion

We have shown that the SUP(T) calculus is a decision procedure for the ground FOL(T) fragment as well as a non-ground FOL(T) fragment where non-base variables can only be instantiated by constants. This answers the so far open question whether SUP(T) can actually decide the ground case. It was already known that SUP(T) is complete and for some cases even a decision procedure for certain non-ground fragments such as the combination of FOL and (non)linear arithmetic [1,12], and the analysis of fragments resulting from the translation of first-order probabilistic and (extended) timed automata [13–16].

There are several directions of future work. One is to extend the obtained results for SUP(T) to the combination of several theories. As long as these theories can be represented in FOL (e.g., lists, arrays, see [2,3]) such a combination is straightforward. However, if the theories are not FOL representable, then additional research is needed to understand such a combination in the hierarchic context. For example, a new variant of the sufficient completeness will probably be needed.

For the sake of simplicity, in this paper we have restricted our attention to a *Horn* clause fragment of FOL(T), where every non-constant function symbol from the underlying FOL signature ranges into the sort of the theory T and all base sort terms are ground. Actually, SUP(T) can decide the general clause fragment as well, because since in this case the only function symbols ranging into the free sort are constants, every non-Horn clause can be split into Horn clauses by instantiating every subsequent occurrence of the same variable by the free constants, reducing the original non-Horn problem to a Horn one.

Another topic for future investigation is the combination of FOL over explicit finite domain clause sets with a combination of background theories $\mathcal{T}$. A restrictive superposition calculus has been proven to be a decision procedure for FOL over finite domain fragment [18]. The sweet point of the combination of FOL over finite domain with $\mathcal{T}$, but with non-constant function symbols ranging into the free sort, is that any clause set (even non-ground) over the fragment is sufficiently complete by the transformations suggested in [18]. This is due to the fact that the cardinality clause

$$x \approx a_1 \vee \cdots \vee x \approx a_n$$

representing the finite domain $\{a_1, \ldots, a_n\}$ is reflected by the clauses

$$f(\vec{x}) \approx a_1 \vee \cdots \vee f(\vec{x}) \approx a_n \qquad \text{for any } f \in \Omega'' \setminus \{a_1, \ldots, a_n\}$$

which imply sufficient completeness for every term $t \in T_{\Omega'}$ of the base sort.

Acknowledgments The authors have been supported by the German Transregional Collaborative Research Center SFB/TR 14 AVACS. We also thank our anonymous reviewers for their detailed and valuable comments.

References

1. Althaus, E., Kruglov, E., Weidenbach, C.: Superposition modulo linear arithmetic SUP(LA). In: Ghilardi, S., Sebastiani, R. (eds.) Frontiers of Combining Systems: 7th International Symposium, FroCoS 2009, volume 5749 of Lecture Notes in Artificial Intelligence, pp. 84–99, Trento, Italy, September , Springer, Berlin (2009)
2. Armando, A., Bonacina, M.P., Ranise, S., Schulz, S.: New results on rewrite-based satisfiability procedures. ACM Trans. Comput. Log. **10**(1), 129–179 (2009)
3. Armando, A., Ranise, S., Rusinowitch, M.: A rewriting approach to satisfiability procedures. Inform. Comput. **183**(2), 140–164 (2003)
4. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
5. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Logic Comput. 4(3):217–247, 1994. Revised version of Max-Planck-Institut für Informatik technical report, MPI-I-91-208 (1991)

6. Bachmair, L., Ganzinger, H., Waldmann, U.: Superposition with simplification as a decision procedure for the monadic class with equality. In: Gottlob, G., Leitsch, A., Mundici, D. (eds.) Computational Logic and Proof Theory, Third Kurt Gödel Colloquium, volume 713 of LNCS, pp. 83–96. Springer, August (1993)
7. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. AAECC **5**(3/4), 193–212 (1994)
8. Baumgartner P., Fuchs A., Tinelli C.: ME(LIA)—model evolution with linear integer arithmetic constraints. In: LPAR 2008, volume 5330 of LNCS, pp. 258–273. Springer (2008)
9. Bonacina, M.P., Lynch, C., de Moura, L.M.: On deciding satisfiability by theorem proving with speculative inferences. J. Autom. Reason. **47**(2), 161–189 (2011)
10. Dershowitz, N.: Orderings for term-rewriting systems. Theor. Comput. Sci. **17**, 279–301 (1982)
11. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM. **22**(8), 465–476 (1979)
12. Eggers, A., Kruglov, E., Scheibler, K., Kupferschmid, S., Teige, T., Weidenbach, C.: SUP(NLA)—combining superposition and non-linear arithmetic. In: Sofronie-Stokkermans, V., Tinelli, C. (eds.) Frontiers of Combining Systems, 8th International Symposium, FroCos 2011, Lecture Notes in Computer Science. Springer, Berlin (2011)
13. Fietzke, A., Hermanns, H., Weidenbach C.: Superposition-based analysis of first-order probabilistic timed automata. In: Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'10, pp. 302–316, Berlin, Heidelberg, Springer Verlag (2010)
14. Fietzke, A., Kruglov, E.,Weidenbach C.: Automatic generation of inductive invariants by SUP(LA). Research Report MPI-I-2012-RG1-002, Max-Planck Institute for Informatics, Saarbrücken, Germany, March (2012)
15. Fietzke, A., Kruglov, E., Weidenbach C.: Automatic generation of invariants for circular derivations in SUP(LA). In: Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, volume 7180 of Lecture Notes in Computer Science, pp. 197–211. Springer, Berlin, March (2012)
16. Fietzke, A., Weidenbach, C.: Superposition as a decision procedure for timed automata. In: Ratschan, S. (ed.) Fourth International Conference on Mathematical Aspects of Computer and Information Sciences (MACIS 2011), Beijing, China, 2011. Internal Conference Proceedings (2011)
17. Ganzinger, H., Sofronie-Stokkermans, V., Waldmann, U.: Modular proof systems for partial functions with Evans equality. Inform. Comput. **204**(10), 1453–1492 (2006)
18. Hillenbrand, T., Weidenbach, C.: Superposition for finite domains. Research Report MPI-I-2007-RG1-002, Max-Planck Institute for Informatics, Saarbruecken, Germany, April (2007)
19. Hustadt, U., Schmidt, R.A., Georgieva, L.: A survey of decidable first-order fragments and description logics. J. Relat. Methods Comput. Sci. **1**, 251–276 (2004)
20. Ihlemann, C., Jacobs, S., Sofronie-Stokkermans V.: On local reasoning in verification. In: Ramakrishnan, C.R., Rehof, J. (eds.) Proceedings of TACAS 2008, volume **4963** of LNCS, pp. 265–281. Springer, Berlin (2008)
21. Jacquemard, F., Meyer, C., Weidenbach C.: Unification in extensions of shallow equational theories. In: Nipkow, T. (ed.) Rewriting Techniques and Applications, 9th International Conference, RTA-98, volume **1379** of LNCS, pp. 76–90. Springer, Berlin (1998)
22. Korovin, K., Voronkov A.: Integrating linear arithmetic into superposition calculus. In Duparc, J., Henzinger, T.A. (eds.) CSL 2007, volume 4646 of LNCS, pp. 223–237. Springer, Berlin (2007)
23. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving sat and sat modulo theories: from an abstract davis–putnam–logemann–loveland procedure to dpll(t). J. ACM. **53**, pp. 937–977, November (2006)
24. Sebastiani, R.: Lazy satisability modulo theories. JSAT **3**(3–4), 141–224 (2007)
25. Suchanek, F.M., Kasneci,G., Weikum G.: Yago: a core of semantic knowledge. In: Proceedings of the 16th international conference on World Wide Web, WWW '07, pp. 697–706, New York, NY, USA, ACM (2007)
26. Suda, M., Weidenbach, C., Wischnewski P.: On the saturation of yago. In: Proceedings of the 5th international conference on Automated Reasoning, IJCAR'10, pp. 441–456, Berlin, Heidelberg, Springer, Berlin (2010)
27. Waldmann, U.: Superposition and chaining for totally ordered divisible abelian groups. In Goré, R., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001, volume 2083 of LNAI, pp. 226–241. Springer, Berlin (2001)
28. Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol 2, chapter 27, pp. 1965–2012. Elsevier, Amsterdam (2001)
29. Weidenbach, C., Wischnewski, P.: Satisfiability checking and query answering for large ontologies. In: Fontaine, P., Schmidt, R., Schulz, S. (eds.) PAAR-2012: IJCAR'12 Workshop on Practical Aspects of Automated Reasoning, pp. 163–177 (2012)