Superposition Modulo Linear Arithmetic SUP(LA)

Ernst Althaus, Evgeny Kruglov, and Christoph Weidenbach

Max Planc Institute for Informatics, Campus E1 4
D-66123 Saarbrücken
{althaus,ekruglov,weidenbach}@mpi-inf.mpg.de

Abstract. The hierarchical superposition based theorem proving calculus of Bachmair, Ganzinger, and Waldmann enables the hierarchic combination of a theory with full first-order logic. If a clause set of the combination enjoys a sufficient completeness criterion, the calculus is even complete. We instantiate the calculus for the theory of linear arithmetic. In particular, we develop new effective versions for the standard superposition redundancy criteria taking the linear arithmetic theory into account. The resulting calculus is implemented in SPASS(LA) and extends the state of the art in proving properties of first-order formulas over linear arithmetic.

1 Introduction

The superposition calculus can be turned into a decision procedure for a number of decidable first-order fragments, e.g., [HSG04, ABRS09], and is therefore a good basis for actually proving decidability of fragments and obtaining terminating implementations. There are now four calculi available combining full first-order logic with linear arithmetic (LA). The hierarchic theorem proving approach SUP(T) for any theory T [BGW94], superposition and chaining for totally ordered divisible abelian groups [Wal01], superposition with linear arithmetic integrated [KV07], and DPPL(T) extended with saturation [dMB08]. In this paper we instantiate the hierarchic theorem proving approach [BGW94] to linear arithmetic SUP(LA), because it offers an abstract completeness result for the combination via a sufficient completeness criterion that goes beyond ground problems (needed for DPLL(T) completeness) and it enables the handling of the LA theory part in a modular way that can be eventually implemented via efficient off the shelf solvers (in contrast to [Wal01, KV07]).

Our contribution is the definition of effective redundancy criteria for SUP(LA) out of the abstract non-effective hierarchic redundancy criteria, and its implementation in SPASS(LA) together with some experiments. In particular, due to completeness, we show by examples that the approach can even decide *satisfiability* of first-order theories with universal quantification modulo LA extending the state of the art. Our presentation assumes an LA theory over the rationals.

The paper is organized as follows. After an introduction to the most important notions, Section 2, we present the SUP(LA) calculus with specific definitions for

S. Ghilardi and R. Sebastiani (Eds.): FroCoS 2009, LNAI 5749, pp. 84–99, 2009.

tautology deletion and subsumption that are then turned into effective procedures via a mapping to LP-solving technology, Section 4. Key aspects of the overall implementation of SPASS(LA) are provided in Section 5. SPASS(LA) is applied to reachability problems of transition systems in Section 6. Here we also discuss the sufficient completeness requirement with respect to container data structure axiomatizations using the example of lists. The paper ends with a discussion of the achieved results and the presentation of further directions of research (Section 7).

2 Definitions and Notations

For the presentation of the superposition calculus, we refer to the notation and notions of [Wei01], where for the specific notions serving the hierarchical combination we refer to [BGW94]. We won't 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.

A *hierarchic specification* is the extension of a base specification over a set of function symbols Ω with free function symbols Ω', $\Omega \subseteq \Omega'$. We assume a many-sorted setting with an explicit *base sort*, sorting Ω. The semantics of the base specification is given by a *base theory*, either through a set of axioms or a collection of models. The semantics of a hierarchic specification is then the extension of the base theory by the standard first-order semantics for the function symbols in $\Omega' \setminus \Omega$. Intuitively, if a hierarchic specification is sufficiently complete, Definition 10, then the restriction of a model of a hierarchic specification to the function symbols Ω of the base specification yields the base theory, in our particular case the standard LA model.

The terms build over Ω and base sort variables are called *base terms*. The base sort may contain further terms build over function symbols from Ω' and in particular $\Omega' \setminus \Omega$ ranging into the base sort. For variables of the base sort we write u, v, w possibly indexed or primed and for non-base sort variables we write x, y, z, possibly indexed and primed. We say that a term is *pure*, if it does not contain both a base operator and a non-base operator.

As usual in the superposition context, all considered atoms are equations where predicates are encoded by a mapping of a function to a distinguished element *true*. Non-equational atoms are thus turned into equations $P(t_1, \ldots, t_n) \approx \mathit{true}$ which are abbreviated $P(t_1, \ldots, t_n)$.

A substitution is called *simple*, if it maps every variable of a base sort 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 simple ground instances of a clause C is denoted by $sgi(C)$, analogously $sgi(N)$ is the set of all simple ground instances of a clause set N.

For linear arithmetic over the rationals $\mathbb{Q}$ considered here we use the signature $\{+, -, \leq, <, \approx, >, \geq\} \subset \Omega$ plus additional symbols to represent the fractions which we denote in this paper simply by decimal numbers.

Clauses are of the form $\Lambda \parallel \Gamma \to \Delta$, where Λ is a sequence of base specification literals, only containing signature symbols from Ω plus variables, called the *clause constraint*. The sequences Γ, Δ of first-order atoms only contain signature symbols from the free first-order theory, i.e., equations over terms with variables over $\Omega' \setminus \Omega$. All parts share universally quantified variables. $\square$ denotes the empty clause. Semantically a clause is interpreted as the universal closure of the implication

$$\bigwedge \Lambda \wedge \bigwedge \Gamma \to \bigvee \Delta.$$

As usual Λ, Γ and Δ may be empty and are then interpreted as *true*, *true*, *false*, respectively. Different clauses are assumed to be variable disjoint. Upper Greek letters denote sequences of atoms $(\Lambda, \Gamma, \Delta)$ or theory literals, lower Greek letter substitutions (σ, τ), lower Latin characters non-variable terms (l, s, r, t) and variable terms (x, y, z, u, v, w) and upper Latin characters atoms (A, B, E). The function *vars* maps objects to their respective set of free variables, $dom(\sigma) = \{x \mid x\sigma \neq x\}$ and $cdom(\sigma) = \{t \mid x\sigma = t, x \in dom(\sigma)\}$ for any substitution σ.

The reduction ordering $\prec$ underlying the superposition calculus is as usual lifted to equational atoms, literals, and clauses [Wei01]. We distinguish inference from reduction rules, where the clause(s) below the bar, the conclusions, of an inference rule are added to the current clause set, while the clause(s) below the bar of a reduction rule replace the clause(s) above the bar, the premises. For example,

$$\mathcal{I}\frac{\Gamma_1 \to \Delta_1 \qquad \Gamma_2 \to \Delta_2}{\Gamma_3 \to \Delta_3} \qquad\qquad \mathcal{R}\frac{\Gamma_1' \to \Delta_1' \qquad \Gamma_2' \to \Delta_2'}{\Gamma_3' \to \Delta_3'}$$

an application of the above inference adds the clause $\Gamma_3 \to \Delta_3$ to the current clause set, while the above reduction replaces the clauses $\Gamma_1' \to \Delta_1'$, $\Gamma_2' \to \Delta_2'$ with the clause $\Gamma_3' \to \Delta_3'$. Note that reductions can actually be used to delete clauses, if there are no conclusions.

3 Superposition Modulo Linear Arithmetic

We will first define the calculus for the general case of a hierarchic specification [BGW94]. Later on, we will then instantiate the general rules for the case of LA.

Any given disjunction of literals can be transformed into a clause of the form $\Lambda \parallel \Gamma \to \Delta$, where Λ only contains base terms and all base terms in Γ, Δ are variables by the following transformation [BGW94]. Whenever a subterm t, whose top symbol is a base theory symbol from Ω, occurs immediately below a non-base operator symbol, it is replaced by a new base sort variable u ("abstracted out") and the equation $u \approx t$ is added to Λ. Analogously, if a subterm t, whose top symbol is not a base theory symbol from Ω, occurs immediately below a base operator symbol, it is replaced by a 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 non-base

literals to Γ, Δ, respectively. Recall that Γ, Δ are sequences of atoms whereas Λ holds theory literals. Moreover, 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. For example, the disjunction

$$S(u + 40, x) \vee v > 60 \vee \neg T(f(u - 3.5), y)$$

is abstracted to the clause

$$v' \approx u + 40, u' \approx u - 3.5, v \leq 60 \parallel T(f(u'), y) \rightarrow S(v', x).$$

During a derivation new base sort atoms $u \approx v$ may be created by inferences or reductions in the free part that are then moved to the constraint part.

Definition 1 (Superposition Left). *The hierarchic superposition left rule is*

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

where σ is simple and a most general unifier of l and l', $l\sigma \not\preccurlyeq r\sigma$, $s\sigma \not\preccurlyeq t\sigma$, l' is not a variable, $(l \approx r)\sigma$ is strictly maximal in $(\Gamma_1 \rightarrow \Delta_1, l \approx r)\sigma$, and $(s[l'] \approx t)\sigma$ is maximal in $(s[l'] \approx t, \Gamma_2 \rightarrow \Delta_2)\sigma$.

For simplicity, we don't consider selection nor sort constraints. Note that we also do not consider the terms in the theory part of the clauses for our maximality criteria. As clauses are abstracted, this is justified by considering a suitable path ordering, like LPO, where the function symbols from Ω are smaller in the precedence than all symbols in $\Omega' \setminus \Omega$. The hierarchic superposition calculus suggested by Bachmair et al [BGW94] introduces a further refined maximality criterion with respect to simple grounding substitutions. The criterion is not decidable, in general. Therefore, we currently don't use it and define the inference rules such that they include the more restricted original versions but provide a decidable maximality criterion. The rules and redundancy criteria are implemented exactly in the way described below in SPASS(LA).[1]

Definition 2 (Superposition Right). *The hierarchic superposition right rule is*

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

where σ is simple and a most general unifier of l and l',$l\sigma \not\preccurlyeq r\sigma$, $s\sigma \not\preccurlyeq t\sigma$, l' is not a variable, $(l \approx r)\sigma$ is strictly maximal in $(\Gamma_1 \rightarrow \Delta_1, l \approx r)\sigma$, and $(s[l'] \approx t)\sigma$ is strictly maximal in $(s[l'] \approx t, \Gamma_2 \rightarrow \Delta_2)\sigma$.

Definition 3 (Equality Factoring). *The hierarchic equality factoring rule is*

$$\mathcal{I}\frac{\Lambda \parallel \Gamma \rightarrow \Delta, l \approx r, l' \approx r'}{(\Lambda \parallel \Gamma, r \approx r' \rightarrow \Delta, l' \approx r')\sigma}$$

[1] The current implementation does not support equality but predicative reasoning (resolution). Equality reasoning will follow soon.

where σ is simple and a most general unifier of l and l', $l\sigma \not\leq r\sigma$, $l'\sigma \not\leq r'\sigma$, and $(l \approx r)\sigma$ is maximal in $(\Gamma \to \Delta, l \approx r, l' \approx r')\sigma$.

Definition 4 (Ordered Factoring). *The hierarchic ordered factoring rule is*

$$\mathcal{I}\frac{\Lambda \parallel \Gamma \to \Delta, E_1, E_2}{(\Lambda \parallel \Gamma \to \Delta, E_1)\sigma}$$

where σ is simple and a most general unifier of E_1 and E_2 and $E_1\sigma$ is maximal in $(\Gamma \to \Delta, E_1, E_2)\sigma$.

Definition 5 (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 σ is simple and a most general unifier of s and t and $(s \approx t)\sigma$ is maximal in $(\Gamma, s \approx t \to \Delta)\sigma$.

Definition 6 (Constraint Refutation). *The constraint refutation rule is*

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

where $\Lambda_1 \parallel\to \wedge \ldots \wedge \Lambda_n \parallel\to$ is inconsistent in the base theory.

The completeness theorem will require compactness of the base theory to justify the constraint refutation rule. For the LA theory considered here, where we, e.g., do not consider shared parameters between clauses, the case $n = 1$ is sufficient.

We will now define new tautology deletion and subsumption deletion rules that will eventually be turned into effective algorithms for LA in Section 4.

Definition 7 (Tautology Deletion). *The hierarchic tautology deletion rule is*

$$\mathcal{R}\frac{\Lambda \parallel \Gamma \to \Delta}{}$$

if $\Gamma \to \Delta$ is a tautology or the existential closure of $\bigwedge \Lambda$ is unsatisfiable in the base theory.

Definition 8 (Subsumption Deletion). *The hierarchic subsumption deletion rule is*

$$\mathcal{R}\frac{\Lambda_1 \parallel \Gamma_1 \to \Delta_1 \qquad \Lambda_2 \parallel \Gamma_2 \to \Delta_2}{\Lambda_1 \parallel \Gamma_1 \to \Delta_1}$$

for a simple matcher σ with $\Gamma_1\sigma \subseteq \Gamma_2$, $\Delta_1\sigma \subseteq \Delta_2$, $vars(\Lambda_1\sigma) \subseteq vars(\Lambda_2)$, and the universal closure of $\bigwedge \Lambda_2 \Rightarrow \bigwedge \Lambda_1\sigma$ holds in the base theory.

In general, a matcher δ with $\Gamma_1\delta \subseteq \Gamma_2$ and $\Delta_1\delta \subseteq \Delta_2$ does not guarantee $vars(\Lambda_1\delta) \subseteq vars(\Lambda_2)$. The substitution $\sigma = \delta\tau$ can be obtained by first computing the standard subsumption matcher δ between and then establishing a simple theory matcher τ where $dom(\tau) = vars(\Lambda_1\delta) \setminus vars(\Lambda_2)$ and $vars(cdom(\tau)) \subseteq vars(\Lambda_2)$.

Note that $u \in vars(\Lambda_1\delta) \setminus vars(\Lambda_2)$ implies $u \notin (vars(\Gamma_1) \cup vars(\Delta_1))$. If the base theory enables quantifier elimination, then u could in fact be eliminated in $\Lambda_1 \parallel \Gamma_1 \rightarrow \Delta_1$ and eventually the need to find an additional theory matcher τ becomes obsolete. However, for LA there is a worst case exponential price to pay for eliminating all variables u as defined above. This is not tractable for subsumption deletion that typically needs to be checked several hundred thousand times for an "average" derivation. Therefore, in Section 4 we propose a polynomial transformation to linear programming for finding the matcher τ that eventually solves the clause constraint implication problem in polynomial time in the size of the two theory parts Λ_1, Λ_2. The basic idea is to map the above variables u to linear terms over variables from Λ_2.

Actually, efficient algorithms for establishing the theory matcher τ is a crucial part in getting the hierarchical superposition calculus to practically work for some base theory. In general, a decision procedure for an unsatisfiability check of clause constraints plus an implication test, potentially modulo a theory substitution are needed for the base theory.

Definition 9 (Hierarchic Redundancy [BGW94]). *A clause $C \in N$ is called redundant if for all $C' \in sgi(C)$ there are clauses $C'_1, \ldots, C'_n \in sgi(N)$ such that $C'_1 \wedge \ldots \wedge C'_n \models C'$ and $C'_i \prec C'$ for all i.*

Our definition of tautology deletion is an obvious instance of the hierarchic redundancy notion. This holds for subsumption deletion as well, because we only consider simple matchers, the notion on the free part is identical to the standard notion and the condition on the clause constraints is a reformulation of the above entailment requirement.

Definition 10 (Sufficient Completeness [BGW94]). *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 non-base term t of the base sort there exists a ground base term t such that $t' \approx t$ is true in $\mathcal{A}'$.*

Note that if the base sort solely contains base terms, then any clause set N over the base and free sort is sufficiently complete.

Theorem 1 (Completeness [BGW94]). *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.*

Here is an example of a clause set where the hierarchical calculus is not complete. Consider LA over the rationals and the two clauses

$$u < 0 \parallel f(v) \approx u \rightarrow$$
$$u' \geq 0 \parallel f(v') \approx u' \rightarrow$$

Then clearly these two clauses are unsatisfiable, however no superposition inference is possible. Note that equality resolution is not applicable as $\{u \mapsto f(v)\}$ is not a simple substitution. Even worse, the theory is no longer compact. Any finite set of ground instances of the two clauses is satisfiable. The two clauses are not sufficiently complete, because the clause set does not imply simple instances of the non-base term $f(v)$ of the base sort to be equal to a base term. Note that in this case a model $\mathcal{A}'$ of the clauses is a non-standard model of linear arithmetic. The equality to a base term can be forced by the additional clause

$$\| \rightarrow f(w) \approx w$$

and now the three clauses are sufficiently complete. For the three clauses there is a refutation where already the superposition left inference between the first and third clause yields

$$u < 0, w \approx u \ \| \rightarrow$$

and a constraint refutation application of the resulting clause yields the contradiction. □

4 Linear Arithmetic Constraint Solving

As discussed in Section 3, we have to provide procedures for satisfiability and for an implication test for linear arithmetic, potentially modulo a theory matcher getting rid of extra constraint variables.

In the following, we use the standard notation from linear programming theory and assume that the reader is familiar with this topic. We write vectors in bold print. Vectors denoted with $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ or $\boldsymbol{d}$ refer to vectors of rational numbers, vectors $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$ or $\boldsymbol{p}$ denote vectors of variables and using alternations $\bar{}$, $\tilde{}$ and $\hat{}$ of those vectors like $\bar{\boldsymbol{x}}$ or $\tilde{\boldsymbol{x}}$ we denote assignments of values to the variables. $\boldsymbol{a}^T$ denotes the transposed vector of $\boldsymbol{a}$. A linear (in)equality is of the form $\boldsymbol{a}^T\boldsymbol{x} \circ c$ with $\boldsymbol{a} \in \mathbb{Q}^n, c \in \mathbb{Q}$, and $\circ \in \{\leq, <, \approx, >, \geq\}$. We can rewrite $\boldsymbol{a}^T\boldsymbol{x} = c$ as $\boldsymbol{a}^T\boldsymbol{x} \leq c \wedge \boldsymbol{a}^T\boldsymbol{x} \geq c$ and $\boldsymbol{a}^T\boldsymbol{x} \geq c$ as $-\boldsymbol{a}^T\boldsymbol{x} \leq -c$.

Matrices are denoted with capital letters like A, B, S, T, P and X, where A and B denote matrices of rational values and P, S, T, and X matrices of variables. A system of linear (in)equalities is the conjunction of a set of linear (in)equalities, its feasible region is typically written as $\{\boldsymbol{x} \in \mathbb{Q}^n \mid \boldsymbol{a}^{iT}\boldsymbol{x} \circ_i \boldsymbol{c}_i \text{ for all } 1 \leq i \leq m\}$ for $\boldsymbol{a}^i \in \mathbb{Q}^n$, $\boldsymbol{c}_i \in \mathbb{Q}$ and $\circ_i \in \{\leq, <, \approx, >, \geq\}$. Let A be the matrix with rows $\boldsymbol{a}^{iT}$, $\boldsymbol{c}$ be the vector with entries $\boldsymbol{c}_i$ and $\circ$ be the vector with entries $\circ_i$. We can rewrite the feasible region of the system of linear (in)equalities above as $\{\boldsymbol{x} \in \mathbb{Q}^n \mid A\boldsymbol{x} \circ \boldsymbol{c}\}$. In the following, we often resign to give the dimensions of the defining matrices and vectors.

The description of the feasible region of any system of linear (in)equalities can be transformed into the standard form $\{\boldsymbol{x} \in \mathbb{Q}^n \mid A'\boldsymbol{x} \leq \boldsymbol{c}', A''\boldsymbol{x} < \boldsymbol{c}''\}$. For a matrix $A \in \mathbb{Q}^{m \times n}$, let A_i be the ith row of A.

It is well known that the feasibility of a linear program can be tested in weakly polynomial time using the Ellipsoid method. For further details we refer to [Sch89].

Assume the set of variables of the base theory part of a clause Λ is $\{x_1, \ldots, x_n\}$ and let $\boldsymbol{x}$ be the vector with entries $x_1, \ldots, x_n$. We can identify the base theory part of the clause as $\Lambda = \bigwedge_{i=1}^{m} \boldsymbol{a}^{i^T} \boldsymbol{x} \circ_i \boldsymbol{c}_i$ for $\boldsymbol{a}^i \in \mathbb{Q}^n$, $\boldsymbol{c}_i \in \mathbb{Q}$ and $\circ_i \in \{\leq, <, \approx, >, \geq\}$ with the subset of $\mathbb{Q}^n$ of satisfying assignments, i.e. the set $\{\boldsymbol{x} \in \mathbb{Q}^n \mid \boldsymbol{a}^{i^T} \boldsymbol{x} \circ_i \boldsymbol{c}_i \text{ for all } 1 \leq i \leq m\}$. The base theory part of the clause is satisfiable, if the corresponding subset of $\mathbb{Q}^n$ is non-empty. Hence the satisfiability of the theory part of a clause corresponds to testing the feasibility of a linear program.

Assume the base theory part of two clauses Λ_1, Λ_2 have the same set of variables. Λ_2 implies Λ_1 if the corresponding subset of $\mathbb{Q}^n$ of Λ_2 is contained in the subset corresponding to Λ_1 (all satisfying assignments for Λ_2 satisfy Λ_1). In Section 4.1, we show this containment problem can be reduced to testing the feasibility of a linear program.

For the subsumption test, we have, in addition, to compute a matcher τ such that Λ_2 implies $\Lambda_1\delta\tau$ for given Λ_1, Λ_2, and δ. In Section 4.2, we show how to reduce this problem to testing the feasibility of a linear program when restricted to matchers that are affine transformations (see Figure 1).

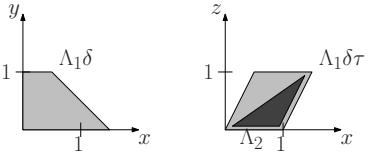

Fig. 1. In the left figure, we show the set $\Lambda_1\delta = \{(x, y) \in \mathbb{Q}^2 \mid x \geq 0, y \geq 0, y \leq 1, 2x + 2y \leq 3\}$, choosing $\tau : \mathbb{Q} \mapsto \mathbb{Q}^2$ with $\tau(y) = -x + z/2 + 1$ gives the set $\Lambda_1\delta\tau = \{(x, z) \in \mathbb{Q}^2 \mid x \geq 0, -x + z/2 \geq -1, -x + z/2 \leq 0, z \leq 1\}$ shown in light gray the right figure. $\Lambda_1\delta\tau$ is implied by the set $\Lambda_2 = \{(x, z) \in \mathbb{Q}^2 \mid z \geq 0, z - x \leq 0, -x + z/2 \geq -1\}$ shown in dark gray as this set is contained in $\Lambda_1\delta\tau$.

4.1 Implication Test

In this section we discuss our approach to solve the implication test, i.e. given $\Lambda_1 = \{\boldsymbol{x} \in \mathbb{Q}^n \mid A'\boldsymbol{x} \leq \boldsymbol{c}', A''\boldsymbol{x} < \boldsymbol{c}''\}$ and $\Lambda_1 = \{\boldsymbol{x} \in \mathbb{Q}^n \mid B'\boldsymbol{x} \leq \boldsymbol{d}', B''\boldsymbol{x} < \boldsymbol{d}''\}$, to tell if $\Lambda_2 \subseteq \Lambda_1$.

We first shortly discuss the case if all inequalities are non-strict, which is based on the well known Farkas' Lemma. Then we extend Farkas' Lemma to the case of mixed strict and non-strict inequalities, which can be used to solve the general case.

Recall the following variant of Farkas' Lemma (see e.g. [Sch89])

Lemma 1 (Farkas' Lemma (affine variant)). *Let $\Lambda = \{\boldsymbol{x} \in \mathbb{Q}^n \mid B\boldsymbol{x} \leq \boldsymbol{d}\}$ be non-empty. All points $\boldsymbol{x} \in \Lambda$ satisfy the inequality $\boldsymbol{a}^T\boldsymbol{x} \leq c$, iff there is $\boldsymbol{p} \geq 0$ such that $\boldsymbol{p}^T B = \boldsymbol{a}^T$ and $\boldsymbol{p}^T\boldsymbol{d} \leq c$.*

Informally speaking, if $\bar{\boldsymbol{p}}$ is a solution, we multiply every inequality $B_i\boldsymbol{x} \leq \boldsymbol{d}_i$ with $\bar{\boldsymbol{p}}_i$ and sum up the obtained inequalities. This gives us for each feasible

$\bar{\boldsymbol{x}}$ that $\boldsymbol{a}^T \bar{\boldsymbol{x}} = \sum_i \bar{\boldsymbol{p}}_i^T B_i \bar{\boldsymbol{x}} \leq \sum_i \bar{\boldsymbol{p}}_i \boldsymbol{d}_i \leq c$. We call the value $\bar{\boldsymbol{p}}_i$ the multiplier of the inequality $B_i \boldsymbol{x} \leq \boldsymbol{d}_i$. Hence, if such a $\bar{\boldsymbol{p}}$ exists, the inequality $\boldsymbol{a}^T \boldsymbol{x} \leq c$ clearly holds for every $\boldsymbol{x} \in \Lambda$.

For the other direction, let $\boldsymbol{a}^T \boldsymbol{x} \leq c$ be an inequality that holds for all $\boldsymbol{x} \in \Lambda$. Consider the linear program $\max\{\boldsymbol{a}^T \boldsymbol{x} \mid \boldsymbol{x} \in \Lambda\}$ and its dual $\min\{\boldsymbol{p}^T \boldsymbol{d} \mid \boldsymbol{p}^T B = \boldsymbol{a}^T, \boldsymbol{p} \geq 0\}$. As the first is feasible and bounded from above by c, we know from the Strong Duality Theorem that there is a solution $\boldsymbol{p}$ of the dual with value at most c.

From Lemma 1 we conclude:

Corollary 1. *A set $\{\boldsymbol{x} \in \mathbb{Q}^n \mid A\boldsymbol{x} \leq \boldsymbol{c}\}$ contains a non-empty set $\{\boldsymbol{x} \in \mathbb{Q}^n \mid B\boldsymbol{x} \leq \boldsymbol{d}\}$, iff each inequality $A_i \boldsymbol{x} \leq \boldsymbol{c}_i$ can be obtained by a non-negative linear combination of the inequalities of $B\boldsymbol{x} \leq \boldsymbol{d}$, i.e. if there are $\boldsymbol{p}^i \geq 0$ with ${\boldsymbol{p}^i}^T B = A_i$ and ${\boldsymbol{p}^i}^T \boldsymbol{d} \leq \boldsymbol{c}_i$.*

Rewriting the rows ${\boldsymbol{p}^i}^T$ as matrix P yields the existence of a matrix P with $PB = A$, $P \geq 0$ and $P\boldsymbol{d} \leq \boldsymbol{c}$. Such a matrix P can be determined by testing the feasibility of a linear program.

Notice that so far, we could solve m independent small systems of linear constraints, where m is the number of rows of A. This is no longer possible in our subsumption test (see 4.2), as we have to compute a single substitution for which all (in)equalities of the contained system of linear (in)equalities can be derived from the (in)equalities of the containing system.

This result can be extended to strict inequalities as follows:

Lemma 2 (Farkas' Lemma for strict inequalities). *Let $\Lambda = \{\boldsymbol{x} \in \mathbb{Q}^n \mid B'\boldsymbol{x} \leq \boldsymbol{d}', B''\boldsymbol{x} < \boldsymbol{d}''\}$. We assume that $\Lambda \neq \emptyset$ and that $B''\boldsymbol{x} < \boldsymbol{d}''$ includes the trivial inequality $0 < 1$.*

- *All points $\boldsymbol{x} \in \Lambda$ satisfy the inequality $\boldsymbol{a}^T \boldsymbol{x} \leq c$, iff there are $\boldsymbol{p}', \boldsymbol{p}'' \geq 0$ such that ${\boldsymbol{p}'}^T B' + {\boldsymbol{p}''}^T B'' = \boldsymbol{a}^T$ and ${\boldsymbol{p}'}^T \boldsymbol{d}' + {\boldsymbol{p}''}^T \boldsymbol{d}'' \leq c$.*
- *All points $\boldsymbol{x} \in \Lambda$ satisfy the inequality $\boldsymbol{a}^T \boldsymbol{x} < c$, iff there are $\boldsymbol{p}', \boldsymbol{p}'' \geq 0$ such that ${\boldsymbol{p}'}^T B' + {\boldsymbol{p}''}^T B'' = \boldsymbol{a}^T$, ${\boldsymbol{p}'}^T \boldsymbol{d}' + {\boldsymbol{p}''}^T \boldsymbol{d}'' \leq c$ and $\boldsymbol{p}'' \neq 0$.*

For the proof please consider our technical report.

Corollary 2. *A set $\{\boldsymbol{x} \in \mathbb{Q}^n \mid A'\boldsymbol{x} \leq \boldsymbol{c}', A''\boldsymbol{x} < \boldsymbol{c}''\}$ contains a non-empty set $\{\boldsymbol{x} \in \mathbb{Q}^n \mid B'\boldsymbol{x} \leq \boldsymbol{d}', B''\boldsymbol{x} < \boldsymbol{d}''\}$, iff each inequality of the first system can be obtained by a non-negative linear combination of the inequalities of $B'\boldsymbol{x} \leq \boldsymbol{d}'$, $B'' < \boldsymbol{d}''$, or $0 < 1$, where at least one multiplier of a strict inequality has to be different from zero if the inequality of the first system is strict.*

Assuming that the system $B''\boldsymbol{x} < \boldsymbol{d}''$ contains the inequality $0 < 1$, we have to test whether the following system of linear (in)equalities is feasible

$$\{(P_1, P_2, P_3, P_4) \in \mathbb{Q}^n \mid P_1, P_2, P_3, P_4 \geq 0,$$
$$P_1 B' + P_2 B'' = A',$$
$$P_3 B' + P_4 B'' = A'',$$
$$P_4 \mathbb{1} > 0,$$
$$P_1 \boldsymbol{d}' + P_2 \boldsymbol{d}'' \leq \boldsymbol{c}',$$
$$P_3 \boldsymbol{d}' + P_4 \boldsymbol{d}'' \leq \boldsymbol{c}''\},$$

where n is the total number of variables in the matrices P_1, P_2, P_3, and P_4 and $\mathbb{1}$ is the all ones vector with appropriate dimension.

Notice again, that we can solve m smaller systems of linear equations, where m is the total number of linear rows in A' and A''.

4.2 Subsumption Test

We discuss our approach for the subsumption test when all inequalities are non-strict. The extension to strict inequalities is straight-forward with the discussion above.

Assume $\Lambda_1 = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Q}^{n_x+n_y} \mid A'\boldsymbol{x} + A''\boldsymbol{y} \leq \boldsymbol{c}\}$ and $\Lambda_2 = \{(\boldsymbol{x}, \boldsymbol{z}) \in \mathbb{Q}^{n_x+n_z} \mid B'\boldsymbol{x} + B''\boldsymbol{z} \leq \boldsymbol{d}\}$, where n_x, n_y, n_z equal the number of entries in the respective vectors. Furthermore let m_A be the number of rows in A' and A'' and m_B be the number of rows in B' and B''. Consider arbitrary affine transformations $y_i := \boldsymbol{s^i}\boldsymbol{x} + \boldsymbol{t^i}\boldsymbol{z} + \beta^i$ with $\boldsymbol{s^i} \in \mathbb{Q}^{n_x}$, $\boldsymbol{t^i} \in \mathbb{Q}^{n_z}$, $\beta^i \in \mathbb{Q}$ for $1 \leq i \leq n_y$. Let S be the matrix with rows $\boldsymbol{s}_1^T, \ldots \boldsymbol{s}_{n_y}^T$, T be the matrix with rows $\boldsymbol{t}_1^T, \ldots \boldsymbol{t}_{n_y}^T$ and $\boldsymbol{\beta}$ be the vector with entries β^i. Let $\tau(S, T, \boldsymbol{\beta})$ be the substitution corresponding to these affine transformations. Hence $\boldsymbol{y}\tau(S, T, \boldsymbol{\beta}) = S\boldsymbol{x} + T\boldsymbol{z} + \boldsymbol{\beta}$ and $\Lambda_1\delta\tau(S, T, \boldsymbol{\beta}) = \{(\boldsymbol{x}, \boldsymbol{z}) \in \mathbb{Q}^{n_x+n_z} \mid A'\boldsymbol{x} + A''S\boldsymbol{x} + A''T\boldsymbol{z} + A''\boldsymbol{\beta} \leq \boldsymbol{c}\}$.

We have to check whether there are S, T and $\boldsymbol{\beta}$ such that $\Lambda_1\delta\tau(S, T, \boldsymbol{\beta})$ contains Λ_2.

With the approach described above these are $S, T, \boldsymbol{\beta}$ such that there are $\boldsymbol{p}^1, \ldots, \boldsymbol{p}^{m_A} \in \mathbb{Q}^{m_B}$ with $\boldsymbol{p}^1, \ldots, \boldsymbol{p}^{m_A} \geq 0$, $\boldsymbol{p^i}^T B' = (A' + A''S)_i$, $\boldsymbol{p^i}^T B'' = (A''T)_i$, and $\boldsymbol{p^i}^T \boldsymbol{d} + (A''\boldsymbol{\beta})_i \leq \boldsymbol{c}_i$ for $1 \leq i \leq m_A$. Let P be the matrix with rows $p^1, \ldots, p^{m_A}$. We have to test the feasibility of the following linear program

$$\{(P, S, T, \beta) \in \mathbb{Q}^n \mid PB' = A' + A''S, PB'' = A''T, P \geq 0 \text{ and } Pd + A''\boldsymbol{\beta} \leq \boldsymbol{c}\},$$

where $n = m_A \times m_B + n_x \times n_y + n_z \times n_y + n_y$

If the system contains strict inequalities, we have to find a solution of a system of linear (in)equalities that contains strict inequalities.

4.3 Summary

We summarize the discussion above by giving the complexity of the three tests. Let Λ_i consist of n_i variables and m_i linear (in)equalities for $i = 1, 2$.

The satisfiability of Λ_1 can be verified by testing the feasibility of a linear program with n_1 variables and m_1 linear (in)equalities.

The test whether Λ_1 implies Λ_2 can be verified by testing the feasibility of m_1 linear programs, each with m_2 variables and $n_1+1 = n_2+1$ linear (in)equalities, where we do not count the non-negativity conditions.

The test whether there is an affine matcher τ such that $\Lambda_1\tau$ contains Λ_2 can be verified by testing the feasibility of one linear program with $m_1m_2+(n_2+1)n'$ variables and $(n_2+1)m_1$ linear constraints, where n' is the number of variables that appear in Λ_1 but not in Λ_2.

All linear systems can contain strict and non-strict inequalities.

5 Implementation

The free part of the inference rules of the hierarchic superposition calculus described in Section 3 is identical to the standard calculus, except that only simple substitutions are considered. For the theory part of clauses an implementation needs to provide instantiation and union of two clause constraints.

The operations resulting from subsumption or tautology deletion are much more involved because here the clause constraints need to be mapped to linear programming problems. The existence of solutions to the linear programs eventually decides on the applicability of the reduction rules. As reductions are more often checked than inferences computed, it is essential for an efficient implementation to support the operations needed for reductions. Therefore, we decided to actually store the clause constraint not in a symbolic tree like representation, as it is done for first-order terms, but directly in the input format data structure of LP solvers, where for the published SPASS(LA) binary `http://spass-prover.org/prototypes` we rely on QSopt `http://www2.isye.gatech.edu/~wcook/qsopt/`.

QSopt uses the simplex-method to solve systems of linear constraints which is not polynomial time but very efficient in practice. Alternatively, we could use an interior-point method to become polynomial while staying efficient in practice.

Furthermore, QSopt relies on floating-point arithmetic and therefore is not guaranteed to find the correct answer. Using an exact solver like QSOpt_Ex `http://www.dii.uchile.cl/~daespino/ESolver_doc/main.html` would lead to a large increase in running-time. Currently we are integrating a floating-point-filter for linear programming [AD09], i.e. a method that certifies the feasibility of linear programs which are numerically not too difficult. This method will allow us to prove or disprove the feasibility of most of the linear programs without using exact arithmetic and hence gives a safe implementation without a large increase in running time. We verified that the linear programs that appear in the solution process of the examples in Section 6 are solved correctly.

A key aspect in implementing SPASS(LA) is the representation of the LA constraints. We decided to use a representation that is close to standard LP solver interfaces such that performing the satisfiability test can be done by calling the LP solver directly with our LA constraint representation. The LP format is a "column-oriented" sparse format, meaning that the problems are represented column by column (variable) rather than row by row (constraint) and only non-zero coefficients are stored. So given an LP by a system $A\boldsymbol{x} \circ \boldsymbol{c}$, the variable

numcols holds the number of columns (or different variables) in the constraint matrix A, *numrows* holds the number of rows in the constraint matrix, *rhs* is an array containing the right-hand sides, *sense* is an array containing the sense vector for the different rows and finally, the three arrays *matval*, *matind*, and *matcnt* hold the values of the matrix A. The non-zero coefficients of the constraint matrix A are grouped by column in the array *matval*. Each column $A_{*,j}$ is stored in a separate array *matval[j]*, and *matcnt[j]* stores the number of entries in *matval[j]*. For each i and j, *matind[j][i]* indicates the row number of the corresponding coefficient A_{ij}, stored in *matval[j][i]*. For example, the clause constraint

$$\Lambda = -u_1 + 2u_2 + 3u_3 + 4.4u_4 \leq 0,\ 5u_2 + 6u_3 \leq 1,\ 7u_3 + 8u_4 < -2.5$$

is represented by the data structure shown in Figure 2.

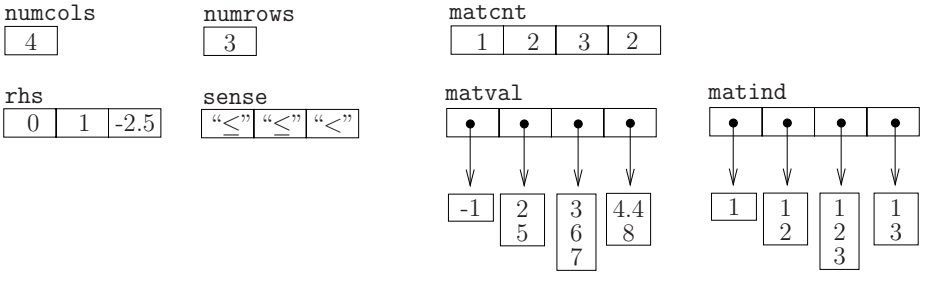

Fig. 2. Constraint Data Structures

The above representation implies that clause constraint satisfiability can directly be mapped from the clause constraint representation to a call of the LP solver. Furthermore, the SPASS variable normalization in clauses , done for sharing on the free theory part, fits perfectly to the above described LP format. All variables occurring in clauses are subsequently named starting with the “smallest” variable. In addition, the other operations on the free part eventually ranging into the constraints like the application of substitutions and the variable disjoint union of constraints can be efficiently mapped to the above suggested representation.

6 Example Applications

6.1 Transition Systems

In the following we present two examples showing that our notion on tautology and subsumption deletion is strong enough to decide formal safety properties of transition systems and our implementation SPASS(LA) is able to perform saturations in a reasonable amount of time. Furthermore, the proofs for the two presented properties of the examples are both satisfiability proofs and hence rely on the completeness of the calculus which is an important feature distinguishing our combination approach from others. Both examples are contained in the experimental SPASS(LA) version provided at `http://spass-prover.org/prototypes/`.

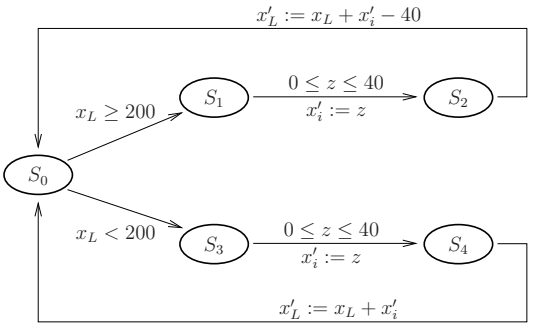

Fig. 3. Water Tank Controller

The transition system of the water tank controller depicted in Figure 3 is meant to keep the level of a water tank below 240 units, provided the initial level at state S_0 is less or equal 240 units. There is a non-controllable inflow from the outside of the system that adds at most 40 units per cycle to the water tank and a controllable outflow valve that can reduce the content of the tank by 40 units per cycle.

We model reachability of the transition system by introducing two place predicates in the variables of the water tank level x_L and the inflow x_i for each state and translate the transitions into implications between states. For example, the transition from S_1 to S_2 becomes

$$\forall u, v, w\,((S_1(u, v) \wedge w \le 40 \wedge w \ge 0) \Rightarrow S_2(u, w))$$

and after normal form translation and abstraction the clause

$$w \le 40, w \ge 0 \parallel S_1(u, v) \rightarrow S_2(u, w)$$

and the transition from S_2 to S_0 eventually results in the clause

$$u' \approx u + v - 40 \parallel S_2(u, v) \rightarrow S_0(u', v)$$

The conjecture is translated into the formula

$$[\forall u, v\,((u \le 240 \wedge S_0(u, v))] \Rightarrow [\exists u', v'(S_0(u', v') \wedge u' > 240))]$$

meaning that starting with an initial state S_0 with a level below 240 units we can reach a state S_0 with a level strictly above 240 units. SPASS(LA) finitely saturates this conjecture together with the theory of the transition system without finding the empty clause in less than one second on any reasonable PC hardware. Due to completeness this shows that the level of the water tank is always below 240 units. Obviously, the hierarchic superposition calculus is complete for this example, because there are no free function symbols at all. SPASS(LA) may also positively prove reachability conjectures. There is one such conjecture commented out in the example file of the distribution. For a number of classes of transition systems, it can actually be shown that the translation used here always results in a fragment where the hierarchic superposition calculus is complete[Non00, Dim09].

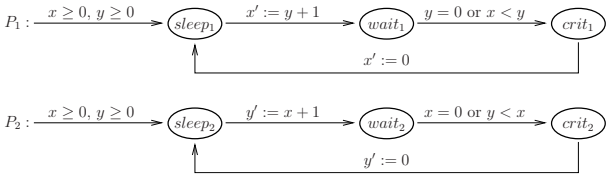

Fig. 4. Bakery Protocol

Note that the above clause set is out of reach for current SMT based approaches as their underlying calculus is typically not complete with respect to non-ground clauses and their instantiation based techniques are not able to show satisfiability. For example, Z3 "gives up" on the above (and the below) clause set. Both Z3 and any other Nelson-Oppen style solver are not complete for the two clause sets. The enhanced prover Z3(SP) [dMB08] is also not complete on both clause sets.

The bakery protocol is a protocol assuring mutual exclusion between two or more processes. We analyze the protocol for two processes by building the product transition system for the two processes shown in the Figure 4 and then modeling the transitions and states analogous to the water tank example.

The conjecture becomes

$$[\forall u, v\,((u \geq 0 \wedge v \geq 0 \wedge \mathit{Sl1Sl2}(u, v))] \Rightarrow [\exists u', v'\ \mathit{Cr1Cr2}(u', v'))]$$

stating that if u and v are both greater than 0 and the processes in their respective sleep states, then the critical section of both processes can be reached simultaneously. Again, SPASS(LA) saturates this conjecture together with the theory of the two processes without finding the empty clause in less than one second. This shows that the protocol is safe, i.e., the critical sections cannot be reached simultaneously by both processes.

6.2 Container Data Structures

Sufficient completeness is a strong prerequisite for the completion of SUP(LA) that can be difficult to obtain, in general. However, in addition to the before shown results for transition systems, in the following we show how to obtain sufficient completeness for axiomatizations of container data structures using the example of lists. A well-known axiom clause set for lists is the following [ABRS09]

$$\begin{aligned} &\rightarrow && \mathit{car}(\mathit{cons}(u, x)) \approx u \\ &\rightarrow && \mathit{cdr}(\mathit{cons}(u, x) \approx x \\ &\rightarrow && \mathit{cons}(\mathit{car}(x), \mathit{cdr}(x)) \approx x \end{aligned}$$

where we assume u to be of LA base sort and x to be of sort list with according sort definitions for the functions. Then the above axiom set over the above signature including *nil* for the empty list is "almost" sufficiently complete. The function *car* ranges into the LA sort and the first axiom can reduce any ground *car* term to a base term except $\mathit{car}(\mathit{nil})$. There are several ways to make the

axioms sufficiently complete. One is to move to an ordered-sorted setting and define a sort of non-empty lists as a subsort of all lists. Then *nil* is of sort list, *car* is defined on non-empty lists, so the ground term $car(nil)$ is not well-sorted anymore. Therefore, the axiom set using this extended sort structure is sufficiently complete.

The term $car(nil)$ is a term that should not occur anyway as it has no well-defined meaning with respect to lists. Therefore, another possibility is simply to map the term $car(nil)$ to some LA constant, e.g. 0. We add $\rightarrow car(nil) \approx 0$ to the above axioms making it sufficiently complete. Then in order to preserve the intended list semantics, we replace the third clause by $\rightarrow cons(car(x), cdr(x)) \approx x, x \approx nil$. The resulting set is sufficiently complete and preserves the list semantics. In general, additional atoms $t \approx nil$ have to be added to any clause containing a $car(t)$ subterm.

Now having a complete axiom set it is subject to the same superposition based techniques for decidability results as they have been suggested, e.g., by Armando Et Al [ABRS09] for the standard superposition calculus. Sufficient completeness can also be obtained using the above described encodings for other container data structures, e.g., arrays, making SUP(LA) a complete calculus for the hierarchic combination of LA over the rationals and arrays.

7 Conclusion

We have presented an instance of the hierarchic superposition calculus [BGW94] with LA with effective notions for subsumption and tautology deletion. Compared to other approaches combining LA with first-order logic, the hierarchic superposition calculus is complete, if the actual clause sets enjoys the sufficient completeness criterion. We showed that for the theories of transition systems over LA and container data structures the sufficient completeness criterion can be fulfilled. The calculus is implemented in a first prototype version called Spass(LA) for the resolution fragment of the hierarchic superposition calculus. By two examples we show that it can already be effectively used to decide satisfiability of clause sets that are out of scope for other approaches, in particular SMT based procedures. On the other hand the hierarchic approach cannot be extended to deal simultaneously with several different theories outside the free part in a straight forward way as it is the case for SMT based procedures using the Nelson-Oppen method, even if queries are ground. Here further research is needed.

For a bunch of examples we tested also Spass(LA) with an SMT solver in place of the LP solver. In contrast to the results presented in [FNORC08], the LP solver turns out to be faster for our satisfiability and implication tests. Although we did not do enough experiments to arrive at a final conclusion, the usage of solvers in Spass(LA) differs from the SMT scenario, because we do not incrementally/decrementally change the investigated LA theory as done by SMT solvers but ask for solving different "small" LA problems containing typically not more that 10–30 (dis)equations.

Finally, our definition of the calculus and the subsumption deletion and tautology deletion rules is not only applicable to a combination with LA, but potentially to any other theory providing an effective constraint satisfiability and implication test.

Acknowledgements. We thank our reviewers for their detailed and valuable comments. The authors are supported by the German Transregional Collaborative Research Center SFB/TR 14 AVACS.

References

[ABRS09] 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)

[AD09] Althaus, E., Dumitriu, D.: Fast and accurate bounds on linear programs. In: Vahrenhold, J. (ed.) SEA 2009. LNCS, vol. 5526, pp. 40–50. Springer, Heidelberg (2009)

[BGW94] Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. Applicable Algebra in Engineering, Communication and Computing, AAECC 5(3/4), 193–212 (1994)

[Dim09] Dimova, D.: On the translation of timed automata into first-order logic. In: Fietzke, A., Weidenbach, C. (supervisors) (2009)

[dMB08] de Moura, L.M., Bjørner, N.: Engineering dpll(t) + saturation. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 475–490. Springer, Heidelberg (2008)

[FNORC08] Faure, G., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Sat modulo the theory of linear arithmetic: Exact, inexact and commercial solvers. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 77–90. Springer, Heidelberg (2008)

[HSG04] Hustadt, U., Schmidt, R.A., Georgieva, L.: A survey of decidable first-order fragments and description logics. Journal of Relational Methods in Computer Science 1, 251–276 (2004)

[KV07] Korovin, K., Voronkov, A.: Integrating linear arithmetic into superposition calculus. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 223–237. Springer, Heidelberg (2007)

[Non00] Nonnengart, A.: Hybrid systems verification by location elimination. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 352–365. Springer, Heidelberg (2000)

[Sch89] Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Inc., Chichester (1989)

[Wal01] Waldmann, U.: Superposition and chaining for totally ordered divisible abelian groups. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 226–241. Springer, Heidelberg (2001)

[Wei01] Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 27, vol. 2, pp. 1965–2012. Elsevier, Amsterdam (2001)