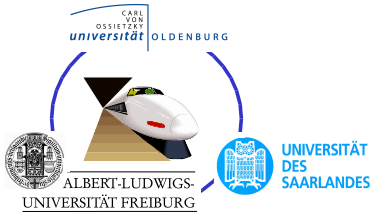

AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Superposition Modulo Linear Arithmetic SUP(LA)

by

Ernst Althaus Evgeny Kruglov

Christoph Weidenbach

AVACS Technical Report No. 53

December 2009

ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Author(s) contact: Christoph Weidenbach (`cw@avacs.org`).

Superposition Modulo Linear Arithmetic SUP(LA)

Ernst Althaus, Evgeny Kruglov, and Christoph Weidenbach

Max Planck 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 and refine 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., [BGW93,JMW98,HSG04,ABRS09], and is therefore a good basis for actually proving decidability of fragments and obtaining terminating implementations. There are now several 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], DPLL(T) extended with saturation [dMB08,BLdM09], and model evolution extended with linear arithmetic [BFT08]. In this paper we instantiate the hierarchic theorem proving approach [BGW94] to linear arithmetic SUP(LA). It offers an abstract completeness result for the combination via a sufficient completeness criterion that goes beyond ground problems as needed for DPLL(T) completeness, or specific theories, as studied for example in local theory extensions [GSSW06,IJSS08]. Furthermore, 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 addition, we fix a bug in [BGW94] concerning the definition of sufficient completeness (see Definition 9). 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 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).

This paper is a significantly extended version of [AKW09].

2 Definitions and Notations

For the presentation of the superposition calculus, we refer to the notation and notions of [Wei01], where the specific notions serving the hierarchical combination are adaptions from [BGW94] and will be eventually instantiated to the case of a linear arithmetic base specification.

A *signature* is a pair $\Sigma = (\mathcal{S}, \mathcal{F})$ where $\mathcal{S}$ is a finite set of sort symbols and $\mathcal{F}$ a set of function symbols over $\mathcal{S}$. The set of well-sorted terms over a set of sorted variables $\mathcal{X}$ is denoted by $\mathcal{T}_\Sigma(\mathcal{F}, \mathcal{X})$ for all terms and $\mathcal{T}_\Sigma(\mathcal{F})$ or for short $\mathcal{T}_\Sigma$ for the ground terms, respectively. Terms of a specific sort S are denoted by $\mathcal{T}_{\Sigma,S}(\mathcal{F}, \mathcal{X})$ and $\mathcal{T}_{\Sigma,S}(\mathcal{F})$ for ground terms, respectively.

As usual in the superposition context, all considered non-base atoms (literals) are (dis)equations $t \approx s$ where s and t are of the same sort. 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 *Σ-algebra* $\mathcal{A}$ consists of a Σ-sorted family of non-empty carrier sets $S_\mathcal{A}$ for each $S \in \mathcal{S}$ 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 $f \colon S_1 \ldots S_n \to S \in \mathcal{F}$. 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.

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

A *hierarchic specification* is a pair $(\mathsf{Spec}, \mathsf{Spec}')$ of specifications, where $\mathsf{Spec} = (\Sigma, \mathbb{C})$, $\Sigma = (\mathcal{S}, \mathcal{F})$, is called the *base specification* and $\mathsf{Spec}' = (\Sigma', N')$, $\Sigma' = (\mathcal{S}', \mathcal{F}')$, the *body* of the hierarchic specification with $\mathcal{S} \subseteq \mathcal{S}'$ and $\mathcal{F} \subseteq \mathcal{F}'$. A term, literal or a clause that solely consists of base functions is called a *base term*, *base literal* or *base clause*, respectively.

A base sort may contain further terms build over function symbols from $\mathcal{F}'$ and in particular $\mathcal{F}' \setminus \mathcal{F}$ 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 function and a non-base function symbol.

If $\mathcal{A}'$ is a Σ'-algebra of a hierarchic specification, the restriction of $\mathcal{A}'$ to Σ, written $\mathcal{A}' \mid_\Sigma$ is the Σ-algebra that is obtained from $\mathcal{A}'$ by removing all carrier sets $S_{\mathcal{A}'}$ for $S \in \mathcal{S}' \setminus \mathcal{S}$ and all functions $f_{\mathcal{A}'}$ for $f \in \mathcal{F}' \setminus \mathcal{F}$. The *weak models* of a hierarchic specification $(\mathsf{Spec}, \mathsf{Spec}')$ are all models $\mathcal{A}'$ of Spec' such that for all $\mathcal{C} \in \mathbb{C}$ there exists a homomorphism $h_\mathcal{C} \colon \mathcal{C} \to \mathcal{A}'$ such that $h_\mathcal{C}(S_\mathcal{C}) \subseteq S_{\mathcal{A}'}$ for all $S \in \mathcal{S}$ and $f_{\mathcal{A}'}(h_\mathcal{C}((t_1)_\mathcal{C}), \ldots, h_\mathcal{C}((t_n)_\mathcal{C})) = h_\mathcal{C}(f_\mathcal{C}((t_1)_\mathcal{C}, \ldots, (t_n)_\mathcal{C}))$, $t_i \in \mathcal{T}_{\Sigma,S_i}$, and $f \colon S_1 \ldots S_n \to S$. The *models* for a hierarchic specification $(\mathsf{Spec}, \mathsf{Spec}')$ are those models $\mathcal{A}'$ of Spec' that extend some model of $\mathbb{C}$ and neither collapse any of its sorts nor add new elements to them, i.e., $\mathcal{A}' \mid_\Sigma \in \mathbb{C}$. Note that the set of all models is a subset of the set of all weak models, where the corresponding homomorphism is the identity.

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 a the linear arithmetic base specification over the rationals $\mathbb{Q}$ considered here we use the signature $\{+, -, \leq, <, \approx, >, \geq\} \subset \mathcal{F}$ 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 literals, called the *clause constraint*. The sequences Γ, Δ of first-order atoms only contain signature symbols from the body of the hierarchic specification, also called the free first-order theory, i.e., equations over terms from $\mathcal{T}_{\Sigma'}(\mathcal{F}' \setminus \mathcal{F}, \mathcal{X})$. 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.

We assume a hierarchic specification $(\mathsf{Spec}, \mathsf{Spec}')$. Any given disjunction of literals can be transformed into a clause of the form $\Lambda \parallel \Gamma \to \Delta$, where Λ only contains base literals and all base terms in Γ, Δ are variables by the following transformation [BGW94]. Whenever a subterm t, whose top symbol is a base symbol from $\mathcal{F}$, occurs immediately below a non-base function 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 from $\mathcal{F}' \setminus \mathcal{F}$, occurs immediately below a base function symbol, it is replaced by a 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 base 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

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

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. We do not explicitly mention this rule as it is obvious. It was not needed in the previous formulation of the calculus [BGW94] because there no strict separation between base terms and free terms of a clause was made. However, it is very useful to separate the base from the non-base literals in order to explore the reasoning mechanisms needed for the base specification.

Definition 1 (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 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 [Wei01]. Note that we also do not consider the terms in the base 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 $\mathcal{F}$ are smaller in the precedence than all symbols in $\mathcal{F}' \setminus \mathcal{F}$. 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 in SPASS(LA) exactly in the way described below.

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*

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

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

Standard factoring of non-equality literals is a special variant of the equality factoring rule where $r = r' = \mathit{true}$ and hence the literal $r \approx r'$ is true and can be eliminated.

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

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

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

Definition 5 (Constraint Refutation). *The constraint refutation rule is*

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

where $\Lambda_1 \parallel \rightarrow \wedge \ldots \wedge \Lambda_n \parallel \rightarrow$ is inconsistent in the base specification.

Actually the above condition means that we have to show unsatisfiability of $\forall \boldsymbol{v}[\neg \Lambda_1 \wedge \ldots \wedge \neg \Lambda_n]$ in the base specification, where $\boldsymbol{v}$ are the different variables of the Λ_i and the Λ_i are conjunctions of base literals. The completeness theorem will require compactness of the base specification 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, thereby reducing the condition of the rule to simply testing whether the universal closure of Λ_1 is unsatisfiable in the base specification. For the single model LA case this corresponds to showing satisfiablity of $\exists \boldsymbol{v} \Lambda_1$.

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

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

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

Definition 7 (Subsumption Deletion). *The hierarchic subsumption deletion rule is*

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

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 specification.

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 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 specification 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. In general, this may be not tractable for a typical run of SPASS(LA) where it generates several thousand new clauses. 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 (weakly) 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 specification. A decision procedure for the above implication test, potentially modulo a theory substitution is needed. This amounts to decide validity of the formula $\forall \boldsymbol{u} \exists \boldsymbol{v}[\Lambda_2 \rightarrow \Lambda_1\delta]$ for the standard matcher δ, where $\boldsymbol{v}$ are the variables from $dom(\tau)$.

Definition 8 (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 9 (Sufficient Completeness). *A set N of clauses is called* sufficiently complete with respect to simple instances*, if for every weak 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 $\mathcal{A}'$.*

The above definition is different from the one given in [BGW94] as we consider weak models and not just models of N. The problem with only considering models is that any clause set having only weak models that collapse or extend the base sorts have no model at all and hence in such a case sufficient completeness trivially holds if defined with respect to models. For example, the below clauses are sufficiently complete with respect to the definition given in [BGW94], but no refutation is possible. Note that if the base sorts solely contains base terms, then any clause set N over over the base and free sorts of a hierarchic specification 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 that is not sufficiently complete and where the hierarchical calculus is therefore 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 not a model of linear arithmetic. The equality to any f-term to a base term can, e.g., be forced by the additional clause

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

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

$$u < 0 \parallel w \approx u \rightarrow$$

and a equality resolution together with a constraint refutation application of the respectively resulting clause yields the contradiction $\square$.

4 Linear Arithmetic Constraint Solving

As discussed in Section 3, we have to provide procedures for (un)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 algebra and linear programming theory and assume that the reader is familiar with these topics. A standard reference is [Sch89].

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 $\boldsymbol{a}$, $\boldsymbol{a}_i$ stands for the ith element of $\boldsymbol{a}$. We write $\boldsymbol{a} \leq b$ to show that $\boldsymbol{a}_i \leq b$ for every entry $\boldsymbol{a}_i$ of $\boldsymbol{a}$; $\leq$ can alternatively be $<, \approx, >$, or $\geq$. Given an arbitrary variable vector $\boldsymbol{x}$, we assume that all entries $\boldsymbol{x}_i$ of $\boldsymbol{x}$ are pairwise distinct. In the following, we often resign to give the dimensions of the defining vectors and matrices.

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 $\boldsymbol{a}^T\boldsymbol{x} \approx c$ to $\boldsymbol{a}^T\boldsymbol{x} \leq c \wedge \boldsymbol{a}^T\boldsymbol{x} \geq c$ and $\boldsymbol{a}^T\boldsymbol{x} \geq c$ to $-\boldsymbol{a}^T\boldsymbol{x} \leq -c$.

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_i is used to denote the ith row of the matrix A.

A *system of linear (in)equalities* (*SLI* – for short) is the conjunction of a set of linear (in)equalities. Let $A \in \mathbb{Q}^{m \times n}$ be the matrix with m rows $A_i \in \mathbb{Q}^n$, $\boldsymbol{c}$ be the vector with entries $\boldsymbol{c}_i \in \mathbb{Q}$ and $\bullet$ be the vector with entries $\bullet_i \in \{\leq, <, \approx, >, \geq\}$. The *feasible region* of an SLI (the set of all points satisfying the system of (in)equalities) is typically written as $\{\bar{\boldsymbol{x}} \in \mathbb{Q}^n \mid A_i\bar{\boldsymbol{x}} \bullet_i \boldsymbol{c}_i, \text{ for all } 1 \leq i \leq m\}$, where m is the number of (in)equalities. For short, the feasible region can be written as $\{\bar{\boldsymbol{x}} \in \mathbb{Q}^n \mid A\bar{\boldsymbol{x}} \bullet \boldsymbol{c}\}$.

Any system of linear (in)equalities can be transformed into the standard form $\Lambda = (A'\boldsymbol{x} \leq \boldsymbol{c}', A''\boldsymbol{x} < \boldsymbol{c}'')$ with the corresponding feasible region of the SLI $\bar{\Lambda} = \{\bar{\boldsymbol{x}} \in \mathbb{Q}^n \mid A'\bar{\boldsymbol{x}} \leq \boldsymbol{c}', A''\bar{\boldsymbol{x}} < \boldsymbol{c}''\}$.

We say that an (in)equality $\boldsymbol{a}^T\boldsymbol{x} \geq c$ or an SLI Λ *holds* (or *is valid*) for a point $\bar{\boldsymbol{x}}$, meaning that the respective object is consistent when its variable vector $\boldsymbol{x}$ is substituted with assignment $\bar{\boldsymbol{x}}$, for instance: $1x_1 - 2x_2 \leq 3$ holds for $\bar{\boldsymbol{x}} = (1, 2)$.

A *linear programming* (*LP*) *problem* is one of maximizing or minimizing a linear function subject to linear inequality constraints. A general form of an LP problem is

$$\begin{array}{rr} \text{maximize} & b_1x_1 + \cdots + b_nx_n \\ \text{subject to} & a_{11}x_1 + \cdots + a_{1n}x_n \leq c_1, \\ & \vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \leq c_m, \end{array}$$

where $x_1, \ldots, x_n$ are variables and the remaining elements are rational coefficients: $b_i, a_{ij}, c_j \in \mathbb{Q}$, for all $1 \leq i \leq n$, $1 \leq j \leq m$. The "maximize" can alternatively be "minimize".

The linear function $b_1x_1 + \cdots + b_nx_n$ that we seek to maximize (minimize) is called the *objective function*, and $b_1, \ldots, b_n$ are called the *objective coefficients*. The inequalities $a_{i1}x_1 + \cdots + a_{in}x_n \leq c_i$ (for $i = 1, \ldots, m$) are referred to as the *constraints*, and the values $c_1, \ldots, c_m$ are called the *right-hand side* values.

It is convenient to express an LP problem in matrix form:

$$\begin{array}{rr} \text{maximize} & \boldsymbol{b}^T\boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} \leq \boldsymbol{c}. \end{array}$$

The matrix A is called the *constraint matrix*.

Every linear programming problem, referred to as a *primal problem*, can be converted into a *dual problem*, which provides an upper bound to the optimal value of the primal problem. If the primal problem is expressed in matrix form as above, then the corresponding dual problem is:

$$\begin{array}{rr} \text{minimize} & \boldsymbol{y}^T\boldsymbol{c} \\ \text{subject to} & \boldsymbol{y}^TA \approx \boldsymbol{b}, \\ & \boldsymbol{y} \geq 0. \end{array}$$

Recall the following variant of the Strong Duality Theorem:

Theorem 2 (Strong Duality Theorem [Sch89]). *Let A be a matrix, and let $\boldsymbol{b}$ and $\boldsymbol{c}$ be vectors. Then*

$$\max\{\boldsymbol{b}^T\boldsymbol{x} \mid A\boldsymbol{x} \leq \boldsymbol{c}\} = \min\{\boldsymbol{y}^T\boldsymbol{c} \mid \boldsymbol{y}^TA \approx \boldsymbol{b}; \boldsymbol{y} \geq 0\}$$

provided that both sets are nonempty.

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 Λ of a clause 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, given as the SLI $\Lambda = A\boldsymbol{x} \bullet \boldsymbol{c}$, for $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{c} \in \mathbb{Q}^m$, $\bullet \in \{\leq, <, \approx, >, \geq\}^m$, and m – number of inequalities in Λ, with the subset of $\mathbb{Q}^n$ of satisfying assignments, i.e. the set $\bar{\Lambda} = \{\bar{\boldsymbol{x}} \in \mathbb{Q}^n \mid A\bar{\boldsymbol{x}} \bullet \boldsymbol{c}\}$. 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. If $\Lambda = A\boldsymbol{x} \leq \boldsymbol{c}$, i.e. Λ contains no strict inequalities (for conciseness, we assume that Λ is given in the standard form), the corresponding linear program is defined on the set of inequalities in Λ with an arbitrary objective function:

$$\begin{array}{rr} \text{maximize} & 0 \\ \text{subject to} & A\boldsymbol{x} \leq \boldsymbol{c}. \end{array}$$

If $\Lambda = (A'\boldsymbol{x} \leq \boldsymbol{c}', A''\boldsymbol{x} < \boldsymbol{c}'')$, the corresponding linear program is constructed in the following way: we add one extra variable $\delta \geq 0$ to each strict inequality and turn the strict inequalities to non-strict: $\Lambda' = (A'\boldsymbol{x} \leq \boldsymbol{c}', A''\boldsymbol{x} + \delta\mathbb{1} \leq \boldsymbol{c}'', \delta \geq 0)$, – and maximize δ over Λ':

$$\begin{array}{rr} \text{maximize} & \delta \\ \text{subject to} & A'\boldsymbol{x} \leq \boldsymbol{c}', \\ & A''\boldsymbol{x} + \delta\mathbb{1} \leq \boldsymbol{c}'', \\ & \delta \geq 0, \end{array}$$

where $\mathbb{1}$ is the all ones vector with appropriate dimension, – and if the objective function value is strictly positive, then Λ is satisfiable.

Assume the base theory parts Λ_1, Λ_2 of two clauses have the same set of variables. Λ_2 implies Λ_1 if the feasible region of Λ_2 is contained in the feasible region of Λ_1 (all satisfying assignments for Λ_2 satisfy Λ_1). In Section 4.1, we show how this containment problem can be reduced to testing the feasibility of a linear program. We call the algorithm used to establish the containment for such SLIs *Identical-Space Implication Test*.

For the implication test, where the sets of variables of Λ_1 and Λ_2 are not the same, we have, in addition, to compute a matcher τ such that Λ_2 implies $\Lambda_1\tau$ for given Λ_1 and Λ_2. 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*, i.e. matchers substituting variables that solely occur in Λ_1 (and do not appear in Λ_2) with linear combinations of variables of Λ_2, bringing thereby variable domains of Λ_1 and Λ_2 to the same set. We call the algorithm used to establish the containment for such SLIs *Different-Space Implication Test*.

4.1 Identical-Space Implication Test

In this section we discuss our approach to solve the following problem: given $\Lambda_1 = (A'\boldsymbol{x} \leq \boldsymbol{c}', A''\boldsymbol{x} < \boldsymbol{c}'')$ and $\Lambda_2 = (B'\boldsymbol{x} \leq \boldsymbol{d}', B''\boldsymbol{x} < \boldsymbol{d}'')$, tell if Λ_2 implies Λ_1, provided that the given SLIs are defined on the same set of variables $\boldsymbol{x}$, or the set of variables occurring in Λ_1 is contained in the one of Λ_2.

We first shortly discuss the case if all inequalities are non-strict, which is based on the well known Farkas' Lemma (see e.g. [Sch89]). 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 ([Sch89]).

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

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 $\bar{\boldsymbol{x}} \in \bar{\Lambda}$.

For the other direction, let $\boldsymbol{a}^T\boldsymbol{x} \leq c$ be an inequality that holds for all $\bar{\boldsymbol{x}} \in \bar{\Lambda}$. Consider the linear program $\max\{\boldsymbol{a}^T\boldsymbol{x} \mid \Lambda\}$ 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 $\bar{\boldsymbol{p}}$ of the dual with value at most c.

From Lemma 1 we conclude:

Corollary 1. *A set $\{\bar{\boldsymbol{x}} \in \mathbb{Q}^n \mid A\bar{\boldsymbol{x}} \leq \boldsymbol{c}\}$ contains a non-empty set $\{\bar{\boldsymbol{x}} \in \mathbb{Q}^n \mid B\bar{\boldsymbol{x}} \leq \boldsymbol{d}\}$, 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 for each inequality $A_i\boldsymbol{x} \leq \boldsymbol{c}_i$ there is a nonnegative vector $\bar{\boldsymbol{p}}$ with $\bar{\boldsymbol{p}}^T B = A_i$ and $\bar{\boldsymbol{p}}^T\boldsymbol{d} \leq \boldsymbol{c}_i$.*

Rewriting the rows $\bar{\boldsymbol{p}}^T$, defined above for each inequality $A_i\boldsymbol{x} \leq \boldsymbol{c}_i$, as matrix $\bar{P}$ yields the existence of a matrix $\bar{P}$ with $\bar{P}B = A$, $\bar{P} \geq 0$ and $\bar{P}\boldsymbol{d} \leq \boldsymbol{c}$. Such a matrix $\bar{P}$ 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 the Different-Space Implication 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.

The result can be extended to strict inequalities as follows:

Lemma 2 (Farkas' Lemma for strict inequalities). *Let Λ be an SLI given as $(B'\boldsymbol{x} \leq \boldsymbol{d}', B''\boldsymbol{x} < \boldsymbol{d}'')$ and its feasible region $\bar{\Lambda}$ be non-empty. We assume that $B''\boldsymbol{x} < \boldsymbol{d}''$ includes the trivial inequality $0 < 1$. Then:*

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

Proof. Let $Q(c') = \{(\bar{\boldsymbol{p}}', \bar{\boldsymbol{p}}'') \mid \bar{\boldsymbol{p}}'^T B' + \bar{\boldsymbol{p}}''^T B'' = \boldsymbol{a}^T, \bar{\boldsymbol{p}}'^T\boldsymbol{d}' + \bar{\boldsymbol{p}}''^T\boldsymbol{d}'' \leq c', \bar{\boldsymbol{p}}', \bar{\boldsymbol{p}}'' \geq 0\}$ (i.e. $Q(c')$ is the set of all multipliers certifying that $\boldsymbol{a}^T\boldsymbol{x} \leq c'$ holds for all $\bar{\boldsymbol{x}} \in \bar{\Lambda}$).

One direction of the both above statements can easily be shown as follows: if $Q(c)$ is non-empty, then by Lemma 1, taking the matrix B to be the composure matrix out of B' and B'', the vector $\boldsymbol{d}$ – the composure vector out of $\boldsymbol{d}'$ and $\boldsymbol{d}''$, the vector $\bar{\boldsymbol{p}}$ – the composure of $\bar{\boldsymbol{p}}'$ and $\bar{\boldsymbol{p}}''$, the inequality $\boldsymbol{a}^T\boldsymbol{x} \leq c$ holds for all

$\bar{\boldsymbol{x}} \in \bar{\Lambda}$ as we can obtain it as a non-negative linear combination of inequalities of the given system. If furthermore $\bar{\boldsymbol{p}}'' \neq 0$ for a point $(\bar{\boldsymbol{p}}', \bar{\boldsymbol{p}}'') \in Q(c)$, the multiplier of at least one strict inequality is non-zero and hence the sum $\bar{\boldsymbol{p}}'^T \boldsymbol{d}' + \bar{\boldsymbol{p}}''^T \boldsymbol{d}''$ is strictly less than c, therefore $\boldsymbol{a}^T \boldsymbol{x} < c$ holds for $\bar{\Lambda}$ as well.

For the other direction, consider $\Lambda^{\leq} = (B'\boldsymbol{x} \leq \boldsymbol{d}', B''\boldsymbol{x} \leq \boldsymbol{d}'')$ with its feasible region $\bar{\Lambda}^{\leq}$.

We show the two statements in turn. Assume $\boldsymbol{a}^T \boldsymbol{x} \leq c$ is valid for $\bar{\Lambda}$. We first show that $\boldsymbol{a}^T \boldsymbol{x} \leq c$ is valid for $\bar{\Lambda}^{\leq}$. Assume it is not valid, then there is a point $\tilde{\boldsymbol{x}}$ in $\bar{\Lambda}^{\leq}$ with $\boldsymbol{a}^T \tilde{\boldsymbol{x}} > c$. Let $\bar{\boldsymbol{x}}$ be an arbitrary point of $\bar{\Lambda}$. As any point $\lambda \bar{\boldsymbol{x}} + (1-\lambda)\tilde{\boldsymbol{x}}$ with $0 < \lambda < 1$ lies in $\bar{\Lambda}$ (because $\bar{\Lambda}$ is convex and dense), there is a point $\hat{\boldsymbol{x}} \in \bar{\Lambda}$ with $\boldsymbol{a}^T \hat{\boldsymbol{x}} > c$, a contradiction. Hence $\boldsymbol{a}^T \boldsymbol{x} \leq c$ is valid for $\bar{\Lambda}^{\leq}$.

From Lemma 1, we conclude that there is a $(\bar{\boldsymbol{p}}', \bar{\boldsymbol{p}}'') \in Q(c)$.

We now show the second item. First assume that $\boldsymbol{a}^T \boldsymbol{x} < c$ is also valid for $\bar{\Lambda}^{\leq}$. Let $\boldsymbol{p}''_{0<1}$ be the multiplier for the strict inequality $0 < 1$. Let $c' = \max\{\boldsymbol{a}^T \bar{\boldsymbol{x}} \mid \bar{\Lambda}^{\leq}\}$, which is attained by $\bar{\Lambda}^{\leq}$. Hence $c' < c$ (as for otherwise our assumption that $\boldsymbol{a}^T \boldsymbol{x} < c$ is valid for $\bar{\Lambda}^{\leq}$ would be wrong) and we have that $\boldsymbol{a}^T \boldsymbol{x} \leq c'$ is also valid for $\bar{\Lambda}^{\leq}$. So we find some $(\bar{\boldsymbol{p}}', \bar{\boldsymbol{p}}'') \in Q(c')$. Increasing $\bar{\boldsymbol{p}}''_{0<1}$ by $c - c'$ gives a vector in $Q(c)$ with $\bar{\boldsymbol{p}}'' \neq 0$.

Finally assume $\boldsymbol{a}^T \boldsymbol{x} < c$ is valid for $\bar{\Lambda}$ but not for $\bar{\Lambda}^{\leq}$. Consider the linear program $\max\{\delta \mid B'\boldsymbol{x} \leq \boldsymbol{d}', B''\boldsymbol{x} + \delta \mathbb{1} \leq \boldsymbol{d}'', \boldsymbol{a}^T \boldsymbol{x} \geq c\}$, where $\mathbb{1}$ is the all ones vector with appropriate dimension. The linear program asks for a point $\bar{\boldsymbol{x}}$ in $\bar{\Lambda}^{\leq}$ with $\boldsymbol{a}^T \bar{\boldsymbol{x}} \geq c$ that maximizes the distance to the strict inequalities. As there is a point $\bar{\boldsymbol{x}} \in \bar{\Lambda}^{\leq}$ with $\boldsymbol{a}^T \bar{\boldsymbol{x}} \geq c$, there is a solution of this linear program with $\delta = 0$, namely $(\bar{x}, 0)$. Hence the objective function value of the linear program is at least 0. Furthermore there can not be a solution $(\tilde{\boldsymbol{x}}, \tilde{\delta})$ of the linear program with $\tilde{\delta} > 0$ as $\tilde{\boldsymbol{x}}$ would be a solution for $\bar{\Lambda}$ with $\boldsymbol{a}^T \tilde{\boldsymbol{x}} \geq c$. Hence δ can not be strictly larger than 0 and the optimal solution of the linear program has objective function value 0. Consider now its dual $\min\{\boldsymbol{p}'^T \boldsymbol{d}' + \boldsymbol{p}''^T \boldsymbol{d}'' + qc \mid \boldsymbol{p}'^T B' + \boldsymbol{p}''^T B'' + q\boldsymbol{a}^T = 0, \boldsymbol{p}''^T \mathbb{1} = 1, \boldsymbol{p}' \geq 0, \boldsymbol{p}'' \geq 0, q \leq 0\}$. From strong duality, we conclude that there is a solution $(\bar{\boldsymbol{p}}', \bar{\boldsymbol{p}}'', \bar{q})$ for the dual with objective value 0. We claim that $\bar{q} < 0$. Assume otherwise and let again $(\bar{x}, 0)$ be a solution of the primal. We would have $0 = \bar{\boldsymbol{p}}'^T \boldsymbol{d}' + \bar{\boldsymbol{p}}''^T \boldsymbol{d}'' \leq \bar{\boldsymbol{p}}'^T B' \bar{\boldsymbol{x}} + \bar{\boldsymbol{p}}''^T B'' \bar{\boldsymbol{x}} < 0$, a contradiction. The last inequality is strict as $\bar{\boldsymbol{p}}'' \neq 0$. Hence $\bar{q} < 0$ and therefore $(-\bar{\boldsymbol{p}}'/\bar{q}, -\bar{\boldsymbol{p}}''/\bar{q}) \in Q(c)$, $\bar{\boldsymbol{p}}''/\bar{q} \neq 0$.

From Lemma 2 we obtain the following result for the Identical-Space Implication problem:

Corollary 2. *An SLI $\Lambda_1 = (A'\boldsymbol{x} \leq \boldsymbol{c}', A''\boldsymbol{x} < \boldsymbol{c}'')$ is implied by an SLI $\Lambda_2 = (B'\boldsymbol{x} \leq \boldsymbol{d}', B''\boldsymbol{x} < \boldsymbol{d}'')$, iff each inequality of the first system can be obtained by a non-negative linear combination of the inequalities of the second system and the trivial inequality $0 < 1$, where at least one multiplier of a strict inequality has to be different from zero if the inequality of the first system is strict, provided that the feasible regions of the given SLIs are non-empty.*

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

$$\begin{aligned}(&P_1, P_2, P_3, P_4 \geq 0,\\ &P_1B' + P_2B'' \approx A',\\ &P_3B' + P_4B'' \approx A'',\\ &P_4\mathbb{1} > 0,\\ &P_1\boldsymbol{d}' + P_2\boldsymbol{d}'' \leq \boldsymbol{c}',\\ &P_3\boldsymbol{d}' + P_4\boldsymbol{d}'' \leq \boldsymbol{c}''),\end{aligned}$$

where $\mathbb{1}$ 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 rows in A' and A''.

To illustrate Corollary 2, let us consider the following SLIs:

$$\Lambda_1 = \begin{pmatrix} x_1 + 2x_2 + 3x_3 \leq 5 \\ 2x_1 + 4x_2 + x_3 < 4 \end{pmatrix}$$

$$\Lambda_2 = \begin{pmatrix} x_1 + 2x_2 + 3x_3 \leq 4 \\ 0.5x_1 + x_2 - x_3 < 0 \end{pmatrix}$$

An easy computation shows that Λ_2 implies Λ_1. Indeed, every inequality in Λ_1 can be obtained as a nonnegative linear combination of inequalities in Λ_2: $1 \cdot (x_1 + 2x_2 + 3x_3) + 0 \cdot (0.5x_1 + x_2 - x_3) = x_1 + 2x_2 + 3x_3$, $1 \cdot 4 + 0 \cdot 0 = 4 \leq 5$, and $1 \cdot (x_1 + 2x_2 + 3x_3) + 2 \cdot (0.5x_1 + x_2 - x_3) = 2x_1 + 4x_2 + x_3$, $1 \cdot 4 + 2 \cdot 0 = 4 \leq 4$.

4.2 Different-Space Implication Test

We discuss our approach for the Different-Space Implication Test when all inequalities are non-strict. The extension to strict inequalities is straight-forward with the arguments of Corollary 2.

Assume $\Lambda_1 = (A'\boldsymbol{x} + A''\boldsymbol{y} \leq \boldsymbol{c})$ and $\Lambda_2 = (B'\boldsymbol{x} + B''\boldsymbol{z} \leq \boldsymbol{d})$, where the vectors $\boldsymbol{y}$ and $\boldsymbol{z}$ do not share any variable. Since $\boldsymbol{y}$ is non-empty and different from $\boldsymbol{z}$, implication from Λ_2 to Λ_1 can never hold, as for any assignment $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{z}})$ satisfying Λ_2, we can pick an assignment $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, for which Λ_1 does not hold. Thus, our goal is to transform Λ_1 in such a way, that its variable space gets the same as the one of Λ_2, whereupon the implication test can take a place. For this to happen we apply affine transformations to the variables $\boldsymbol{y}$ by mapping them to linear combinations of the variables $\boldsymbol{x}$ and $\boldsymbol{z}$. Please, note that application of affine transformations is sufficient to establish the implication from Λ_2 to Λ_1, but it is not necessary, as in the general case the variables $\boldsymbol{y}$ can be actually mapped to arbitrary functions over $\boldsymbol{x}$ and $\boldsymbol{z}$. Here we exploit affine transformations because in this case we can provide an effective algorithm to solve the implication problem.

Let Λ_1 and Λ_2 be as given above. Let m_A be the number of rows in A' and A'', and m_B be the number of rows in B' and B''. Let n_x, n_y, n_z equal the number of entries in the respective vectors. Consider arbitrary affine transformations $\boldsymbol{y}_i := \bar{S}_i\boldsymbol{x} + \bar{T}_i\boldsymbol{z} + \bar{\boldsymbol{\beta}}_i$ with $\bar{S}_i \in \mathbb{Q}^{n_x}$, $\bar{T}_i \in \mathbb{Q}^{n_z}$, $\bar{\boldsymbol{\beta}}_i \in \mathbb{Q}$ for $1 \leq i \leq n_y$. Let

S be the matrix with rows $S_1, \ldots S_{n_y}$, let T be the matrix with rows $T_1, \ldots T_{n_y}$ and $\boldsymbol{\beta}$ be the vector with entries $\boldsymbol{\beta}_1, \ldots \boldsymbol{\beta}_{n_y}$. Let $\tau(\bar{S}, \bar{T}, \bar{\boldsymbol{\beta}})$ be the substitution corresponding to these affine transformations. Hence $\boldsymbol{y}\tau(\bar{S}, \bar{T}, \bar{\boldsymbol{\beta}}) = \bar{S}\boldsymbol{x} + \bar{T}\boldsymbol{z} + \bar{\boldsymbol{\beta}}$ and $\Lambda_1\tau(\bar{S}, \bar{T}, \bar{\boldsymbol{\beta}}) = (A'\boldsymbol{x} + A''\bar{S}\boldsymbol{x} + A''\bar{T}\boldsymbol{z} \leq \boldsymbol{c} - A''\bar{\boldsymbol{\beta}})$.

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

With the approach described above these are $\bar{S}, \bar{T}, \bar{\boldsymbol{\beta}}$ such that there are $\bar{P}_1, \ldots, \bar{P}_{m_A} \in \mathbb{Q}^{m_B}$ with $\bar{P}_1, \ldots, \bar{P}_{m_A} \geq 0$, $\bar{P}_i B' = (A' + A''\bar{S})_i$, $\bar{P}_i B'' = (A''\bar{T})_i$, and $\bar{P}_i \boldsymbol{d} \leq \boldsymbol{c}_i - A''\boldsymbol{\beta}$, 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 SLI:

$$(PB' \approx A' + A''S, PB'' \approx A''T, P \geq 0 \text{ and } Pd \leq \boldsymbol{c} - A''\boldsymbol{\beta}),$$

where the total number of unknowns in the matrices P, S, T and vector $\boldsymbol{\beta}$ is $n = m_A \times m_B + n_x \times n_y + n_z \times n_y + n_y$

Let us summarize the non-strict case of the Different-Space Implication Test in the following Lemma:

Lemma 3. *Let $\Lambda_1 = (A'\boldsymbol{x} + A''\boldsymbol{y} \leq \boldsymbol{c})$ and $\Lambda_2 = (B'\boldsymbol{x} + B''\boldsymbol{z} \leq \boldsymbol{d})$, and let $\tau(\bar{S}, \bar{T}, \bar{\boldsymbol{\beta}})$ be the substitution representing an arbitrary affine transformation $\boldsymbol{y}_i := \bar{S}_i\boldsymbol{x} + \bar{T}_i\boldsymbol{z} + \bar{\boldsymbol{\beta}}_i$ for every variable $\boldsymbol{y}_i$ in $\boldsymbol{y}$. Then Λ_2 implies $\Lambda_1\tau(\bar{S}, \bar{T}, \bar{\boldsymbol{\beta}})$, if and only if the following SLI is feasible:*

$$\begin{aligned} (&PB' \approx A' + A''S,\\ &PB'' \approx A''T,\\ &Pd \leq \boldsymbol{c} - A''\boldsymbol{\beta},\\ &P \geq 0), \end{aligned}$$

where the matrices P, S, T, and the vector $\boldsymbol{\beta}$ consist of unknowns.

If the given SLIs contain strict inequalities, we have to find a solution of an SLI that contains strict inequalities.

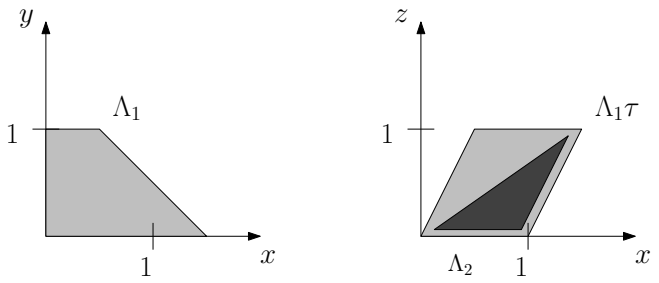

Fig. 1. Different-Space SLIs Containment.

Consider the example depicted in Figure 4.2. In the left figure, we show the feasible region of the SLI $\Lambda_1 = (x \geq 0, y \geq 0, y \leq 1, 2x + 2y \leq 3)$, choosing

$\tau(y) = -x + z/2 + 1$ gives the SLI $\Lambda_1\tau = (x \geq 0, -x + z/2 \geq -1, -x + z/2 \leq 0, z \leq 1)$, whose feasible region is shown in light gray in the right figure. $\Lambda_1\tau$ is implied by $\Lambda_2 = (z \geq 0, z - x \leq 0, -x + z/2 \geq -1)$ shown in dark gray as its feasible region is contained in $\Lambda_1\tau$.

4.3 Complexity

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[1] we rely on QSopt [2].

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 [3] would lead to a large increase in running-time. Currently we are integrating a

[1] See `http://spass-prover.org/prototypes`.
[2] See `http://www2.isye.gatech.edu/~wcook/qsopt/`.
[3] See `http://www.dii.uchile.cl/~daespino/ESolver_doc/main.html`.

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.

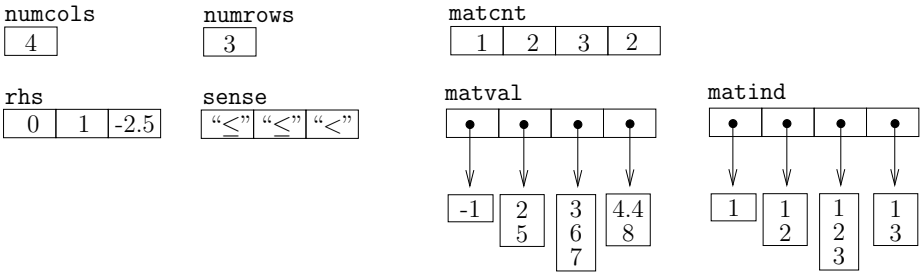

Fig. 2. Constraint Data Structures

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 [4].

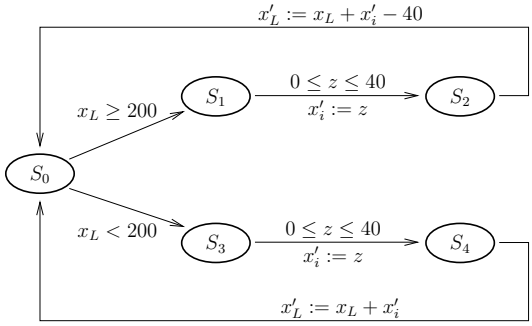

Fig. 3. Water Tank Controller

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

[4] See `http://spass-prover.org/prototypes/`.

$$\forall u, v, w\,((S_1(u, v) \wedge w \leq 40 \wedge w \geq 0) \rightarrow S_2(u, w))$$

and after normal form translation and abstraction the clause

$$w \leq 40, w \geq 0 \;\|\; S_1(u, v) \rightarrow S_2(u, w)$$

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

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

The conjecture is translated into the formula

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

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 and the minimal model property of superposition with respect to existentially quantified conjunctions of atoms [HW08] 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].

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.

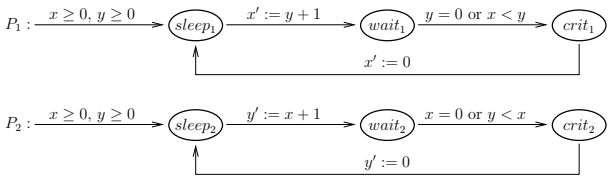

Fig. 4. Bakery Protocol

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

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

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 $\mathit{car}(\mathit{nil})$ is not well-sorted anymore. Therefore, the axiom set using this extended sort structure is sufficiently complete.

The term $\mathit{car}(\mathit{nil})$ 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 $\mathit{car}(\mathit{nil})$ to some LA constant, e.g. 0. We add $\rightarrow \mathit{car}(\mathit{nil}) \approx 0$ to the above axioms making it sufficiently complete. Then in order to preserve the intended list semantics, we replace the third clause by $\rightarrow \mathit{cons}(\mathit{car}(x), \mathit{cdr}(x)) \approx x, x \approx \mathit{nil}$. The resulting set is sufficiently complete and preserves the list semantics. In general, additional atoms $t \approx \mathit{nil}$ have to be added to any clause containing a $\mathit{car}(t)$ 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). 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, implication, and refutation test. In particular, for the base specification $\mathsf{Spec} = (\Sigma, \mathbb{C})$ we need (semi) decision procedures for the problems

Tautology	$\mathbb{C} \models \exists \boldsymbol{v} \Lambda$	see Definition 6
Redundancy	$\mathbb{C} \models \forall \boldsymbol{u} \exists \boldsymbol{v} [\Lambda_2 \to \Lambda_1]$	see Definition 7
Refutation	$\mathbb{C} \models \exists \boldsymbol{v} [\Lambda_1 \vee \ldots \vee \Lambda_n]$	see Definition 5

where the Λ_i are conjunctions of literals from the base specification.

Acknowledgements. The authors are supported by the German Transregional Collaborative Research Center SFB/TR 14 AVACS.

References

[ABRS09] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 10(1):129–179, 2009.

[AD09] Ernst Althaus and Daniel Dumitriu. Fast and accurate bounds on linear programs. In Jan Vahrenhold, editor, *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA 2009)*, volume 5526 of *LNCS*, pages 40–50. Springer, 2009.

[AKW09] Ernst Althaus, Evgeny Kruglov, and Christoph Weidenbach. Superposition modulo linear arithmetic SUP(LA). In Silvio Ghilardi and Roberto Sebastiani, editors, *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings*, volume 5749 of *LNCS*, pages 84–99. Springer, 2009.

[BFT08] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. $\mathcal{ME}$(LIA) - model evolution with linear integer arithmetic constraints. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings*, volume 5330 of *LNCS*, pages 258–273. Springer, 2008.

[BGW93] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Computational Logic and Proof Theory, Third Kurt Gödel Colloquium*, volume 713 of *LNCS*, pages 83–96. Springer, August 1993.

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

[BLdM09] Maria Paola Bonacina, Christopher Lynch, and Leonardo Mendonça de Moura. On deciding satisfiability by DPLL($\Gamma + \mathcal{T}$) and unsound theorem proving. In Renate A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *LNCS*, pages 35–50. Springer, 2009.

[Dim09] Dilyana Dimova. On the translation of timed automata into first-order logic, 2009. Supervisors: A. Fietzke, C. Weidenbach.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Engineering DPLL(T) + saturation. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008*, volume 5195 of *LNCS*, pages 475–490. Springer, 2008.

[FNORC08] Germain Faure, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. SAT modulo the theory of linear arithmetic: Exact, inexact and commercial solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008*, volume 4996 of *LNCS*, pages 77–90. Springer, 2008.

[GSSW06] Harald Ganzinger, Viorica Sofronie-Stokkermans, and Uwe Waldmann. Modular proof systems for partial functions with Evans equality. *Information and Computation*, 204(10):1453–1492, 2006.

[HSG04] Ullrich Hustadt, Renate A. Schmidt, and Lilia Georgieva. A survey of decidable first-order fragments and description logics. *Journal of Relational Methods in Computer Science*, 1:251–276, 2004.

[HW08] Matthias Horbach and Christoph Weidenbach. Superposition for fixed domains. In Michael Kaminski and Simone Martini, editors, *22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL*, volume 5213 of *LNCS*, pages 293–307. Springer, 2008.

[IJSS08] Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. On local reasoning in verification. In C. R. Ramakrishnan and Jakob Rehof, editors, *Proceedings of TACAS 2008*, volume 4963 of *LNCS*, pages 265–281. Springer, 2008.

[JMW98] Florent Jacquemard, Christoph Meyer, and Christoph Weidenbach. Unification in extensions of shallow equational theories. In Tobias Nipkow, editor, *Rewriting Techniques and Applications, 9th International Conference, RTA-98*, volume 1379 of *LNCS*, pages 76–90. Springer, 1998.

[KV07] Konstantin Korovin and Andrei Voronkov. Integrating linear arithmetic into superposition calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL 2007*, volume 4646 of *LNCS*, pages 223–237. Springer, 2007.

[Non00] Andreas Nonnengart. Hybrid systems verification by location elimination. In Nancy A. Lynch and Bruce H. Krogh, editors, *HSCC 2000*, volume 1790 of *Lecture Notes in Computer Science*, pages 352–365. Springer, 2000.

[Sch89] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., 1989.

[Wal01] Uwe Waldmann. Superposition and chaining for totally ordered divisible abelian groups. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *IJCAR 2001*, volume 2083 of *LNAI*, pages 226–241. Springer, 2001.

[Wei01] Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 27, pages 1965–2012. Elsevier, 2001.