A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic

Philipp Rümmer

Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
philipp@chalmers.se

Abstract. First-order logic modulo the theory of integer arithmetic is the basis for reasoning in many areas, including deductive software verification and software model checking. While satisfiability checking for ground formulae in this logic is well understood, it is still an open question how the general case of quantified formulae can be handled in an efficient and systematic way. As a possible answer, we introduce a sequent calculus that combines ideas from free-variable constraint tableaux with the Omega quantifier elimination procedure. The calculus is complete for theorems of first-order logic (without functions, but with arbitrary uninterpreted predicates), can decide Presburger arithmetic, and is complete for a substantial fragment of the combination of both.

1 Introduction

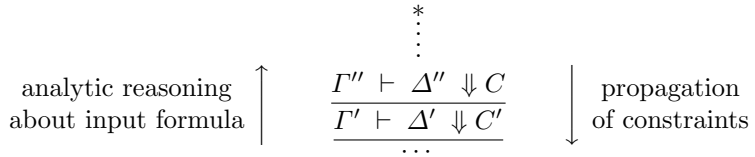
One of the main challenges in automated theorem proving is to combine reasoning about full first-order logic (FOL), including quantifiers, with reasoning about theories like the integers. At the time, there are efficient provers for handling formulae in first-order logic, as well as SMT-solvers that can efficiently handle ground problems modulo many theories, but the support for the combination of both is typically weak. In this paper, we develop a novel calculus for reasoning about first-order logic modulo linear integer arithmetic that is complete for both the first-order part and the theory part, and that can handle a substantial fragment of the combination of both. Because the calculus is close to the DPLL(T) architecture, techniques and optimisations used in SMT-solvers are readily applicable when working on ground problems, but can be combined with free-variable techniques to treat quantifiers more systematically.

We start from two existing approaches: free-variable tableaux with incremental closure, following the work by Martin Giese [1], and the Omega quantifier elimination procedure [2] for deciding Presburger arithmetic (PA) [3]. From the former method, our calculus inherits the concept of generating *constraints* that describe valuations of free variables for which a formula is satisfied. The latter method provides the basic rules for dealing with linear integer arithmetic,

and the concept of recursive application of a calculus in order to handle nested and alternating quantifiers. The resulting calculus accepts arbitrary formulae of PA enriched with arbitrary uninterpreted predicates as input. Uninterpreted functions are not directly supported, but can be treated by a translation to uninterpreted predicates and functionality and totality axioms.

Our calculus operates on *constrained sequents* $\Gamma \vdash \Delta \Downarrow C$, which consist of two sets Γ, Δ of formulae (the antecedent and the succedent) and one further formula C (the constraint). In this paper, C will always be a formula of PA. The semantics of a constrained sequent is the same as of the implication $C \Rightarrow (\Gamma \vdash \Delta)$, i.e., we call the sequent valid if the constraint C implies the ordinary sequent $\Gamma \vdash \Delta$ (and the ordinary sequent holds iff the formula $\bigwedge \Gamma \to \bigvee \Delta$ holds). In this sense, we can say that the constraint C is an approximation of the sequent $\Gamma \vdash \Delta$. The sequent $\forall x.(x \dot{\geq} 0 \to p(x)) \vdash p(c) \Downarrow c \dot{\geq} 0$ is valid, for instance, as are the sequents $\forall x.(x \dot{\geq} 0 \to p(x)) \vdash p(c) \Downarrow c \doteq 3$ and $\Gamma \vdash \Delta \Downarrow \mathit{false}$.

In practice, the constraints of sequents will be unknown during the construction of a proof. Proving thus consists of two or more phases: starting with a problem $\Gamma \vdash \Delta \Downarrow ?$ with unknown constraint, a proof procedure will first apply analytic rules to the antecedent and succedent and build a proof tree, similarly as in a normal Gentzen-style sequent calculus. At some point when it seems appropriate, the procedure will start to close branches by synthesising constraints, which are subsequently propagated downwards from the leaves to the root of the tree. If the constraint that reaches the root is found to be valid, the validity of the input problem $\Gamma \vdash \Delta$ has been shown; otherwise, the procedure will continue to expand the proof tree and later update the resulting constraints.



If the input problem $\Gamma \vdash \Delta$ does not contain uninterpreted predicates (i.e., corresponds to a PA formula), it is always possible to find proofs such that the resulting constraint is equivalent to $\Gamma \vdash \Delta$ (we will call such proofs *exhaustive*). This allows us to use the calculus as a quantifier elimination procedure for PA.

Our main contributions are: the introduction of the calculus, completeness results for a number of fragments (including FOL and PA), a complete and terminating proof strategy for the PA fragment, and the result that fair proof construction is complete for formulae that are provable at all. Proofs for all theorems in the paper are given in [4].

The paper is organised as follows: After giving basic definitions in Sect. 2, we introduce our calculus in three steps: Sect. 3 gives a version for pure first-order logic, Sect. 4 a minimalist version for first-order logic modulo integer arithmetic, together with completeness results, and Sect. 5 an equivalent but more refined calculus. Sect. 6 contains the result that fair proof strategies are complete. Finally, Sect. 7 summarises related work and Sect. 8 concludes.

2 Preliminaries

We assume that the reader is familiar with classical first-order logic and Gentzen-style sequent calculi, see [5] for an introduction. Assuming that $x \in X$ ranges over an infinite set of variables, $c \in A$ over an infinite set of constants, $p \in P$ over a set of uninterpreted predicates with fixed arity, and $\alpha \in \mathbb{Z}$ over integers, the syntactic categories of terms t and formulae ϕ are defined by:

$$t \;::=\; \alpha \;\big|\; x \;\big|\; c \;\big|\; \alpha t + \cdots + \alpha t$$
$$\phi \;::=\; \phi \wedge \phi \;\big|\; \phi \vee \phi \;\big|\; \neg\phi \;\big|\; \forall x.\phi \;\big|\; \exists x.\phi \;\big|\; t \doteq 0 \;\big|\; t \dot{\geq} 0 \;\big|\; t \dot{\leq} 0 \;\big|\; \alpha \mid t \;\big|\; p(t, \ldots, t)$$

For reasons of simplicity, we only allow 0 as right-hand side of equations and inequalities, although we deviate from this convention in some places for sake of clarity. The explicit divisibility operator $\alpha \mid t$ is added for presentation purposes only and does not add any expressiveness (divisibility can also be expressed with an existentially quantified equation). Further, we use the abbreviations *true*, *false* for the equations $0 \doteq 0$, $1 \doteq 0$ and $\phi \rightarrow \psi$ as abbreviation for $\neg\phi \vee \psi$.

Simultaneous substitution of terms $t_1, \ldots, t_n$ for variables $x_1, \ldots, x_n$ is denoted by $[x_1/t_1, \ldots, x_n/t_n]\phi$, whereby we assume that variable capture is avoided by renaming bound variables when necessary. As short-hand notations, we sometimes also substitute terms for constants (as in $[c/t]\phi$), quantify over constants (as in $\forall c.\phi$), or quantify over sets of constants (as in $\forall U.\phi$).

Semantics. The only universe considered for evaluation are the integers $\mathbb{Z}$ (an exception is Sect. 3, where we treat normal first-order logic). A variable assignment $\beta : X \rightarrow \mathbb{Z}$ is a mapping from variables to integers, a constant assignment $\delta : A \rightarrow \mathbb{Z}$ a mapping from constants to integers, and an interpretation $I : P \rightarrow \mathcal{P}(\mathbb{Z}^*)$ a mapping from predicates to sets of $\mathbb{Z}$-tuples. The evaluation function $val_{I,\beta,\delta}$ for terms and formulae is then defined as is common and gives the arithmetic operations their normal meaning. We call a formula ϕ valid if $val_{I,\beta,\delta}(\phi)$ is true for all I, β, δ.

Sequents. If Γ, Δ are finite sets of formulae and C is a formula, all of which do not contain free variables, then $\Gamma \vdash \Delta$ is an (ordinary) sequent and $\Gamma \vdash \Delta \Downarrow C$ is a (constrained) sequent. We sometimes identify sequents with the formulae $\bigwedge \Gamma \rightarrow \bigvee \Delta$ (resp., $\bigwedge \Gamma \wedge C \rightarrow \bigvee \Delta$). A calculus rule is a binary relation between finite sets of constrained sequents (the premisses) and constrained sequents (the conclusion). A sequent calculus rule is called sound, iff, for all instances

$$\frac{\Gamma_1 \vdash \Delta_1 \Downarrow C_1 \quad \cdots \quad \Gamma_n \vdash \Delta_n \Downarrow C_n}{\Gamma \vdash \Delta \Downarrow C}$$

it holds that: if all premisses $\Gamma_1 \vdash \Delta_1 \Downarrow C_1$, $\ldots$, $\Gamma_n \vdash \Delta_n \Downarrow C_n$ are valid, then $\Gamma \vdash \Delta \Downarrow C$ is valid. Proof trees are defined as is common as trees growing upwards in which each node is labelled with a constrained sequent, and in which each node that is not a leaf is related with the nodes directly above through an instance of a calculus rule. A proof is closed if it is finite, and if all leaves are justified by a rule instance without premisses.

Simplification. We denote elementary simplification steps on terms and atomic formulae in a proof with SIMP, without showing more details about the applied transformation (in an implementation, SIMP might be a part of the datastructures for formulae). SIMP normalises terms to the form $\alpha_1 t_1 + \cdots + \alpha_n t_n$, in which $\alpha_1, \ldots, \alpha_n$ are non-zero integers and $t_1, \ldots, t_n$ are pairwise distinct variables, constants, or 1 (possibly 0 as the empty sum). Further, terms are put into a canonical form by sorting summands according to a well-founded ordering $<_r$:

- on variables, constants and integers, $<_r$ is an arbitrary well-ordering such that variables are bigger than constants, constants are bigger than integers, and: $0 <_r 1 <_r -1 <_r 2 <_r -2 <_r 3 <_r \cdots$.
- on terms with coefficients, $<_r$ is defined by $\alpha t <_r \alpha' t'$ if and only if $t <_r t'$ or $t = t'$ and $\alpha <_r \alpha'$.
- on linear combinations, $<_r$ is defined by $\alpha_1 t_1 + \cdots + \alpha_n t_n <_r \alpha'_1 t'_1 + \cdots + \alpha'_k t'_k$ if and only if $\{\!\{\alpha_1 t_1, \ldots, \alpha_n t_n\}\!\} <_r \{\!\{\alpha'_1 t'_1, \ldots, \alpha'_n t'_n\}\!\}$ (in the multiset extension of $<_r$, cf. [6]).

Atomic formulae $t \doteq 0$, $t \dot{\geq} 0$, $t \dot{\leq} 0$ are normalised by SIMP such that the coefficients of non-constant terms in t are coprime (do not have non-trivial factors in common), and such that the leading coefficient is non-negative. This also detects that equations like $2y - 6c + 1 \doteq 0$ are unsolvable and equivalent to *false*, and that an inequality like $2y - 6c + 1 \dot{\leq} 0$ can be simplified and rounded to $y - 3c + 1 \dot{\leq} 0$ thanks to the discreteness of the integers. All inequalities in the succedent are moved to the antecedent. A divisibility judgement $\alpha \mid t$ is normalised like an equation $\alpha x + t \doteq 0$, and it is ensured that α and the leading coefficient of t are positive.

3 A Constraint Sequent Calculus for First-Order Logic

We first introduce a very restricted calculus for pure first-order logic, in order to illustrate how the framework of constrained sequents is related to normal free-variable tableau calculi. This section is exceptional in that we do *not* assume evaluation of formulae over the universe $\mathbb{Z}$ of integers, and that we allow equations $s \doteq t$ whose right-hand side is not 0. The rules from Fig. 1, together with the following closure rule, form the calculus Pred^C:

$$\frac{*}{\Gamma, p(s_1, \ldots, s_n) \;\vdash\; p(t_1, \ldots, t_n), \Delta \;\Downarrow \bigwedge_i s_i \doteq t_i} \text{ PRED-CLOSE}$$

Instead of unifying complementary literals, a conjunction of equations about the predicate arguments is generated and propagated as a constraint.

Example 1. We show a proof for the sequent $\forall x.\exists y.p(x, y) \;\vdash\; \exists z.p(a, z)$. In order to instantiate existential and universal quantifiers, fresh constants c, d, e are introduced. The constraints on the right-hand side are practically filled in *after* applying PRED-CLOSE. Because $\exists x.\forall y.\exists z.(x \doteq a \wedge y \doteq z)$ is valid, also the

$$\frac{\Gamma \vdash \phi, \Delta \Downarrow C \qquad \Gamma \vdash \psi, \Delta \Downarrow D}{\Gamma \vdash \phi \wedge \psi, \Delta \Downarrow C \wedge D}\ \text{AND-RIGHT}$$

$$\frac{\Gamma, \phi \vdash \Delta \Downarrow C \qquad \Gamma, \psi \vdash \Delta \Downarrow D}{\Gamma, \phi \vee \psi \vdash \Delta \Downarrow C \wedge D}\ \text{OR-LEFT}$$

$$\frac{\Gamma, \phi, \psi \vdash \Delta \Downarrow C}{\Gamma, \phi \wedge \psi \vdash \Delta \Downarrow C}\ \text{AND-LEFT} \qquad \frac{\Gamma \vdash \phi, \psi, \Delta \Downarrow C}{\Gamma \vdash \phi \vee \psi, \Delta \Downarrow C}\ \text{OR-RIGHT}$$

$$\frac{\Gamma \vdash \phi, \Delta \Downarrow C}{\Gamma, \neg\phi \vdash \Delta \Downarrow C}\ \text{NOT-LEFT} \qquad \frac{\Gamma, \phi \vdash \Delta \Downarrow C}{\Gamma \vdash \neg\phi, \Delta \Downarrow C}\ \text{NOT-RIGHT}$$

$$\frac{\Gamma \vdash [x/c]\phi, \exists x.\phi, \Delta \Downarrow [x/c]C}{\Gamma \vdash \exists x.\phi, \Delta \Downarrow \exists x.C}\ \text{EX-RIGHT} \qquad \frac{\Gamma, [x/c]\phi, \forall x.\phi \vdash \Delta \Downarrow [x/c]C}{\Gamma, \forall x.\phi \vdash \Delta \Downarrow \exists x.C}\ \text{ALL-LEFT}$$

$$\frac{\Gamma \vdash [x/c]\phi, \Delta \Downarrow [x/c]C}{\Gamma \vdash \forall x.\phi, \Delta \Downarrow \forall x.C}\ \text{ALL-RIGHT} \qquad \frac{\Gamma, [x/c]\phi \vdash \Delta \Downarrow [x/c]C}{\Gamma, \exists x.\phi \vdash \Delta \Downarrow \forall x.C}\ \text{EX-LEFT}$$

Fig. 1. The rules for first-order predicate logic (without equality). In all rules, c is a constant that does not occur in the conclusion: in contrast to the usage of Skolem functions and free variables in tableaux, the same kinds of symbols (constants) are used to handle both existential and universal quantifiers. Arbitrary renaming of bound variables is allowed in the constraints when necessary to avoid variable capture.

validity of the original problem is proven.

$$\cfrac{\cfrac{\cfrac{\cfrac{*}{\ldots, p(c,d) \vdash \ldots, p(a,e) \Downarrow c \doteq a \wedge d \doteq e}\ \text{PRED-CLOSE}}{\ldots, p(c,d) \vdash \exists z.p(a,z) \Downarrow \exists z.(c \doteq a \wedge d \doteq z)}\ \text{EX-RIGHT}}{\ldots, \exists y.p(c,y) \vdash \exists z.p(a,z) \Downarrow \forall y.\exists z.(c \doteq a \wedge y \doteq z)}\ \text{EX-LEFT}}{\forall x.\exists y.p(x,y) \vdash \exists z.p(a,z) \Downarrow \exists x.\forall y.\exists z.(x \doteq a \wedge y \doteq z)}\ \text{ALL-LEFT}$$

It is easy to see that a constraint C produced by a proof can only consist of equations over variables and constants, conjunctions, and quantifiers (because these are the only constructs that are introduced in constraints by the rules of Pred^C). The validity of constraints/formulae of this kind is decidable and corresponds to simultaneous unification, which makes the calculus effective.

Lemma 2 (Soundness). *If a sequent $\Gamma \vdash \Delta \Downarrow C$ is provable in Pred^C, then it is valid (holds in all first-order structures).*

Lemma 3 (Completeness). *Suppose ϕ is closed, valid (holds in all first-order structures), and does not contain constants. Then there is a valid constraint C such that $\vdash \phi \Downarrow C$ is provable in Pred^C.*

4 Adding Integer Arithmetic

Relatively few changes to the calculus Pred^C from the previous section are necessary to reason about problems in integer arithmetic. In this section, we

describe a minimalist approach in which all integer reasoning happens during the constraint solving and investigate fragments on which the resulting method is complete. Later in the paper, the calculus is refined and optimised. From now on and in contrast to the previous section, assume that formulae and terms are evaluated over first-order structures with the universe $\mathbb{Z}$ as described in Sect. 2.

In contrast to the previous section, to handle integer arithmetic disjunctive constraints also need to be considered. We thus split the rule PRED-CLOSE into two new rules, one of which (PRED-UNIFY) generates unification conditions for complementary pairs, while the other one (CLOSE) allows to synthesise a constraint from arbitrary formulae in a sequent:

$$\frac{\Gamma, p(s_1, \ldots, s_n) \vdash p(t_1, \ldots, t_n), \bigwedge_i s_i - t_i \doteq 0, \Delta \Downarrow C}{\Gamma, p(s_1, \ldots, s_n) \vdash p(t_1, \ldots, t_n), \Delta \Downarrow C}\ \text{PRED-UNIFY}$$

$$\frac{*}{\Gamma, \phi_1, \ldots, \phi_n \vdash \psi_1, \ldots, \psi_m, \Delta \Downarrow \neg\phi_1 \vee \cdots \vee \neg\phi_n \vee \psi_1 \vee \cdots \vee \psi_m}\ \text{CLOSE}$$

($\phi_1, \ldots, \phi_n, \psi_1, \ldots, \psi_m$ do not contain uninterpreted predicates)

Besides these two rules, PresPred_S^C contains all rules given in Fig. 1. It is obvious that any proof in Pred^C can be translated to a proof in PresPred_S^C by replacing applications of PRED-CLOSE with applications of PRED-UNIFY, followed by CLOSE, which means that PresPred_S^C is complete for first-order logic.

Because uninterpreted predicates are excluded in CLOSE, the constraint resulting from a proof is always a formula in Presburger arithmetic and can in principle be handled using any decision procedure for PA (e.g. [2], also see Sect. 5.3). We come back to this issue later in the paper and assume for the time being that some procedure is available for deciding the validity of constraints.

As an implication of a more general result (Lem. 13), it can be observed that PresPred_S^C is proof-confluent: if ϕ is provable, then every partial proof of $\vdash \phi \Downarrow ?$ can be extended to a closed proof of a sequent $\vdash \phi \Downarrow C$ with valid constraint C.

Example 4. We show a proof for the following sequent (Fig. 2):

$$\forall x.p(2x), \forall x.\neg p(2x+1) \vdash \forall y.(p(y) \rightarrow p(y+10))$$

The sequent is proven by first building the "main proof" (upwards) to a point where CLOSE can be applied. The constraints $C_1, \ldots, C_4$ are then filled in and propagated downwards. Because C_4 is valid, we have proven the validity of the original formula. The constraint simplification is explained in more detail later.

Completeness on fragments. Two fragments on which PresPred_S^C is complete are the classes of purely universal and of purely existential formulae. We call positions in the antecedent/succedent of a sequent *positive* if they are underneath an odd/even number of negations. All other positions are called *negative*.

Lemma 5. *If $\Gamma \vdash \Delta$ is a valid sequent in which $\exists$ only occurs in negative and $\forall$ only in positive positions, then there is a valid PA constraint C such that $\Gamma \vdash \Delta \Downarrow C$ has a proof in the calculus PresPred_S^C.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\ldots \vdash \ldots, 2d-c-10 \doteq 0, c-2e-1 \doteq 0 \Downarrow C_1}\,\text{CLOSE}}{p(2d), \ldots, p(c) \vdash p(c+10), p(2e+1) \Downarrow C_1}\,\text{PRED-UNIFY} \times 2}{\ldots, p(2d), \forall x.\neg p(2x+1), p(c) \vdash p(c+10) \Downarrow C_2}\,\text{ALL-LEFT, NOT-LEFT}}{\forall x.p(2x), \forall x.\neg p(2x+1), p(c) \vdash p(c+10) \Downarrow C_3}\,\text{ALL-LEFT}}{\forall x.p(2x), \forall x.\neg p(2x+1) \vdash \neg p(c) \vee p(c+10) \Downarrow C_3}\,\text{OR-RIGHT, NOT-RIGHT}}{\forall x.p(2x), \forall x.\neg p(2x+1) \vdash \forall y.(p(y) \rightarrow p(y+10)) \Downarrow C_4}\,\text{ALL-RIGHT}$$

The constraints are:

$$\begin{array}{lllll} C_1 & = & & & 2d-c-10 \doteq 0 \vee c-2e-1 \doteq 0 \\ C_2 & = & \exists y.[e/y]C_1 & = & \exists y.(2d-c-10 \doteq 0 \vee c-2y-1 \doteq 0) \\ C_3 & = & \exists x.[d/x]C_2 & = & \exists x.\exists y.(2x-c-10 \doteq 0 \vee c-2y-1 \doteq 0) \\ & & & \equiv & 2 \mid (c+10) \vee 2 \mid (c-1) \\ C_4 & = & \forall x.[c/x]C_3 & = & \forall x.(2 \mid (x+10) \vee 2 \mid (x-1)) \\ & & & \equiv & \mathit{true} \end{array}$$

Fig. 2. An example proof in the calculus PresPred_S^C.

Lemma 6. *If $\Gamma \vdash \Delta$ is a valid sequent (without constants) in which $\exists$ only occurs in positive and $\forall$ only in negative positions, then there is a valid PA constraint C such that $\Gamma \vdash \Delta \Downarrow C$ has a proof in the calculus PresPred_S^C.*

Comparison with $\mathcal{ME}$(LIA). We can also show that the calculus PresPred_S^C is complete on the logic that can be handled by Model Evolution modulo linear integer arithmetic [7]. Ignoring minor syntactic issues and the fact that $\mathcal{ME}$(LIA) works on clauses, $\mathcal{ME}$(LIA) is a sound and complete calculus for proving the unsatisfiability of formulae of the shape $\exists \bar{a}.(\phi \wedge \psi)$, where $\bar{a} = (a_1, \ldots, a_m)$ is a vector of existentially quantified variables, ϕ is a PA formula over $\bar{a}$ that only has finitely many solutions, and ψ is an arbitrary formula over $\bar{a}$ in which $\exists/\forall$ only occurs in negative/positive positions.

Lemma 7. *If $\exists \bar{a}.(\phi \wedge \psi)$ as above is an unsatisfiable formula that does not contain constants or free variables, then there is a valid constraint C such that the sequent $\exists \bar{a}.(\phi \wedge \psi) \vdash \Downarrow C$ has a proof in PresPred_S^C.*

5 Built-In Handling of Presburger Arithmetic

Although the calculus from the previous section is in principle usable, it practically has a number of shortcomings: the handling of arithmetic in constraints provides little guidance for the construction of proofs, so that large constraints are produced in a very indeterministic manner that cannot be solved efficiently. Moreover, constraints are even needed to handle ground problems, for which branch-local reasoning should be sufficient. The main goal when refining the calculus is, therefore, to reduce the usage of constraints as far as possible.

In this section, we define built-in rules for handling linear integer arithmetic that can be interleaved with the rules from the previous section. The rules make it possible to handle ground problems branch-locally: proof trees for ground problems can be constructed depth-first (non-iteratively), similarly to the way in which SMT-solvers work. It can be achieved that the only constraints that can result from a subproof in case of ground problems are *true* or *false* (more details are given in [4]). Branch-local reasoning is also possible for innermost $\forall$-quantifiers in positive and $\exists$ in negative positions. The arithmetic rules also yield a decision procedure for PA that can be used to decide constraints (Sect. 5.3).

The rules in detail. The calculus $\mathrm{PresPred}^{C}$ consists of the rules given in Fig. 3, together with all rules from the calculus $\mathrm{PresPred}^{C}_{S}$ and the simplification rule SIMP. We introduce new rules EX-RIGHT-D, ALL-LEFT-D that instantiate quantified formulae destructively, because formulae that do not contain uninterpreted predicates never have to be instantiated twice (also see Lem. 13 below).

The equality handling follows the calculus given in [8] and can solve arbitrary equations in the antecedent, in the sense that the equations are rewritten until the leading coefficients are all 1 and the leading terms of equations occur in exactly one place. Speaking in terms of matrices, RED is the rule for performing row operations, while COL-RED(-SUBST) is responsible for column operations. We define a suitable strategy for guiding the rules below.

The rules DIV-RIGHT and DIV-LEFT translate divisibility statements to equations, while DIV-CLOSE synthesises divisibility statements from equations. The formula C' in DIV-CLOSE can be found through pseudo-division (multiplying equations, inequalities or divisibility statements in C with non-zero factors). For $C = (c + d \doteq 0)$ and $\alpha = 3$, for instance, we would choose $C' = (x + 3d \doteq 0)$.

Inequalities are handled based on the Omega test [2], which is an extension of the Fourier-Motzkin variable elimination method (cf. [9]) for integer problems. The central rule is OMEGA-ELIM for replacing a conjunction of inequalities with a disjunction over simpler cases (OMEGA-ELIM is directly based on the main theorem underlying the Omega test [2]). The literal m_i in the rule is defined by:

$$m = \max_j \beta_j, \qquad m_i = \left\lfloor \frac{m\alpha_i - \alpha_i - m}{m} \right\rfloor$$

In case there are no upper bounds, we define $m = m_i = -1$. The application of OMEGA-ELIM is only meaningful if c does not occur in formulae other than inequalities. Note, that if there are no lower or no upper bounds, the rule will replace all inequalities whose leading term is c with *true*.

Because we avoid the application of OMEGA-ELIM in certain common situations (for instance, whenever the constant c occurs as argument of uninterpreted predicates), we also introduce a rule FM-ELIM for normal Fourier-Motzkin elimination. FM-ELIM can be applied with higher priority than OMEGA-ELIM and is often able to close proofs faster than OMEGA-ELIM, reducing the need to resort to the more complex rule. Further, we define two rules to convert between equations and inequalities. While the rule SPLIT-EQ is strictly necessary for certain problems, ANTI-SYMM is introduced only for reasons of efficiency.

$$\frac{\Gamma \vdash [x/c]\phi, \Delta \Downarrow [x/c]C}{\Gamma \vdash \exists x.\phi, \Delta \Downarrow \exists x.C}\ \textsc{ex-right-d} \qquad \frac{\Gamma, [x/c]\phi \vdash \Delta \Downarrow [x/c]C}{\Gamma, \forall x.\phi \vdash \Delta \Downarrow \exists x.C}\ \textsc{all-left-d}$$

(c a constant that does not occur in the conclusion, ϕ does not contain uninterpreted predicates)

$$\frac{\Gamma, t \doteq 0 \vdash \phi[s + \alpha \cdot t], \Delta \Downarrow C}{\Gamma, t \doteq 0 \vdash \phi[s], \Delta \Downarrow C}\ \textsc{red}$$

(α a literal, or t a literal and α an arbitrary term)

$$\frac{\Gamma, \alpha(u + c') + t \doteq 0, c - u - c' \doteq 0 \vdash \Delta \Downarrow [x/c']C}{\Gamma, \alpha c + t \doteq 0 \vdash \Delta \Downarrow \forall x.C}\ \textsc{col-red}$$

(c' a constant that does not occur in the conclusion or in u)

$$\frac{\Gamma, \alpha(u + c') + t \doteq 0, c - u - c' \doteq 0 \vdash \Delta \Downarrow [x/c']C}{\Gamma, \alpha c + t \doteq 0 \vdash \Delta \Downarrow [x/c - u]C}\ \textsc{col-red-subst}$$

(c' a constant that does not occur in the conclusion or in u)

$$\frac{\Gamma, \exists x.\alpha x + t \doteq 0 \vdash \Delta \Downarrow C}{\Gamma, \alpha \mid t \vdash \Delta \Downarrow C}\ \textsc{div-left}$$

(x an arbitrary variable)

$$\frac{\Gamma, (\alpha \mid t + 1) \vee \cdots \vee (\alpha \mid t + \alpha - 1) \vdash \Delta \Downarrow C}{\Gamma \vdash \alpha \mid t, \Delta \Downarrow C}\ \textsc{div-right} \qquad (\alpha > 0)$$

$$\frac{\Gamma, \alpha c - t \doteq 0 \vdash \Delta \Downarrow C}{\Gamma, \alpha c - t \doteq 0 \vdash \Delta \Downarrow [x/t]C' \vee \alpha \nmid t}\ \textsc{div-close}$$

(c does not occur in t or in C', C' a PA formula such that $C \Leftrightarrow [x/\alpha c]C'$)

$$\frac{\Gamma \vdash t \dot{\leq} 0, \Delta \Downarrow C \quad \Gamma \vdash t \dot{\geq} 0, \Delta \Downarrow D}{\Gamma \vdash t \doteq 0, \Delta \Downarrow C \wedge D}\ \textsc{split-eq}$$

$$\frac{\Gamma, t \doteq 0 \vdash \Delta \Downarrow C}{\Gamma, t \dot{\leq} 0, t \dot{\geq} 0 \vdash \Delta \Downarrow C}\ \textsc{anti-symm}$$

$$\frac{\Gamma, \alpha c + s \dot{\geq} 0, \beta c + t \dot{\leq} 0, \beta s - \alpha t \dot{\geq} 0 \vdash \Delta \Downarrow C}{\Gamma, \alpha c + s \dot{\geq} 0, \beta c + t \dot{\leq} 0 \vdash \Delta \Downarrow C}\ \textsc{fm-elim}$$

$(\alpha > 0,\ \beta > 0)$

$$\frac{\Gamma,\ \begin{array}{c} \bigwedge_{i,j} \alpha_i b_j - a_i \beta_j - (\alpha_i - 1)(\beta_j - 1) \dot{\geq} 0 \\ \vee \\ \bigvee_i \bigvee_{k=0}^{m_i} \left(\begin{array}{c} \alpha_i c - a_i - k \doteq 0 \wedge \\ \bigwedge_i \alpha_i c - a_i \dot{\geq} 0 \wedge \bigwedge_j \beta_j c - b_j \dot{\leq} 0 \end{array} \right) \end{array} \vdash \Delta \Downarrow C}{\Gamma, \{\alpha_i c - a_i \dot{\geq} 0\}_i, \{\beta_j c - b_j \dot{\leq} 0\}_j \vdash \Delta \Downarrow C}\ \textsc{omega-elim}$$

$(\alpha_i > 0,\ \beta_j > 0)$

Fig. 3. Rules for equations, inequalities, and divisibility judgements. In RED, we write $\phi[s]$ in the succedent to denote that s occurs in an arbitrary formula in the sequent, which can in particular also be in the antecedent. m_i in OMEGA-ELIM as on page 8.

Lemma 8 (Soundness). *If a sequent $\Gamma \vdash \Delta \Downarrow C$ is provable in PresPred^C, then it is valid.*

5.1 Exhaustive Proofs

The existence of a closed proof for a sequent $\Gamma \vdash \Delta \Downarrow C$ guarantees that the implication $C \Rightarrow (\Gamma \vdash \Delta)$ holds (this is the soundness of the calculus, Lem. 8). In the special case that the sequent $\Gamma \vdash \Delta$ does not contain uninterpreted predicates, it is possible to distinguish particular closed proofs that also guarantee the opposite implication $(\Gamma \vdash \Delta) \Rightarrow C$, and thus $(\Gamma \vdash \Delta) \Leftrightarrow C$. While this can be achieved in a trivial way by always applying CLOSE such that *all* formulae in a sequent are selected, it is sufficient to impose a weaker condition on proof trees that leads to smaller constraints and also makes it possible to eliminate quantifiers (Sect. 5.3). To this end, it is necessary to remember whether a constant was introduced by an existential rule (like EX-RIGHT) or a universal rule (like ALL-RIGHT). A generalisation of the condition is described in [4].

Assume that a PresPred^C-proof is given. We annotate the sequents in the proof with sets U of "universal" constants that the calculus attempts to eliminate. More formally, the proof is called *exhaustive* iff there is a mapping from proof nodes (constrained sequents) to sets U of constants that satisfies:

1. The rules AND-*, OR-*, NOT-*, PRED-UNIFY, RED, DIV-*, SPLIT-EQ, ANTISYMM, FM-ELIM, and SIMP keep or reduce the set: if the conclusion is annotated with U, the premisses are annotated with arbitrary subsets of U.
2. The rules EX-RIGHT(-D), ALL-LEFT(-D) erase the set: the premiss is annotated with $\emptyset$.
3. The rules EX-LEFT and ALL-RIGHT may add the introduced constant c to the set: if the conclusion is annotated with U, then the premiss is annotated with a subset of $U \cup \{c\}$.
4. The rule COL-RED is only applied if the conclusion is annotated with U such that $c \in U$. In this case, the premiss is annotated with a subset of $U \cup \{c'\}$.
5. The rule COL-RED-SUBST is only applied if the conclusion is annotated with U such that $c \notin U$, and if u does not contain any constants from U. In this case, the premiss is annotated with a subset of U.
6. The rule OMEGA-ELIM is only applied if the conclusion is annotated with U such that $c \in U$ and if c does not occur in Γ or Δ. In this case, the premiss is annotated with an arbitrary subset of U.
7. The rule DIV-CLOSE is only applied if the conclusion is annotated with U such that $c \in U$. In this case, the premiss is annotated with a subset of U.
8. The rule CLOSE is always applied such that all formulae without uninterpreted predicates are selected, apart from (possibly) those equations in the succedent that contain constants from U that exclusively occur in equations in the succedent.

Lemma 9 (Constraint completeness). *Suppose that a PresPred^C-proof is closed and exhaustive. For each sequent $\Gamma \vdash \Delta \Downarrow C$ in the tree, let Γ_p, Δ_p denote the subsets of PA formulae in Γ, Δ. Then, for each sequent $\Gamma \vdash \Delta \Downarrow C$ that is annotated with a set U, the implication $\forall U.\ (\Gamma_p \vdash \Delta_p) \Rightarrow \forall U.\ C$ holds.*

Example 10. The formula $\neg\exists x.\exists y.(2x - c - 10 \doteq 0 \vee 2y - c + 1 \doteq 0)$ from Example 4 is simplified by constructing a proof. To see that the proof is exhaustive, the sequent with constraint D_5 is annotated with $\emptyset$, the sequent with D_1 with $\{e\}$, the sequent with D_3 with $\{d\}$, and all other sequents with the set $\{d, e\}$. This implies that the original formula is equivalent to D_5.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{2d - c - 10 \doteq 0 \vdash \quad \Downarrow D_1}\,\text{CLOSE}}{2d - c - 10 \doteq 0 \vdash \quad \Downarrow D_2}\,\text{DIV-CLOSE} \qquad \dfrac{\dfrac{\dfrac{*}{2e - c + 1 \doteq 0 \vdash \quad \Downarrow D_3}\,\text{CLOSE}}{2e - c + 1 \doteq 0 \vdash \quad \Downarrow D_4}\,\text{DIV-CLOSE}}{2d - c - 10 \doteq 0 \vee 2e - c + 1 \doteq 0 \vdash \quad \Downarrow D_2 \wedge D_4}\,\text{OR-LEFT}}{\exists x.\exists y.(2x - c - 10 \doteq 0 \vee 2y - c + 1 \doteq 0) \vdash \quad \Downarrow D_5}\,\text{EX-LEFT} \times 2$$

The constraints resulting from the proof are:

$$\begin{array}{lcll}
D_1 & = & & 2d - c - 10 \not\doteq 0 \\
D_2 & = & [2d/c + 10]D_1 \vee 2 \nmid (c + 10) & (c + 10) - c - 10 \not\doteq 0 \vee 2 \nmid (c + 10) \\
 & & & \equiv \; 2 \nmid (c + 10) \\
D_3 & = & & 2e - c + 1 \not\doteq 0 \\
D_4 & = & [2e/c - 1]D_3 \vee 2 \nmid (c - 1) & (c - 1) - c + 1 \not\doteq 0 \vee 2 \nmid (c - 1) \\
 & & & \equiv \; 2 \nmid (c - 1) \\
D_5 & = & \exists x.[d/x]\exists y.[e/y](D_2 \wedge D_4) & \exists x.\exists y.(2 \nmid (c + 10) \wedge 2 \nmid (c - 1)) \\
 & & & \equiv \; 2 \nmid (c + 10) \wedge 2 \nmid (c - 1)
\end{array}$$

5.2 The Construction of Exhaustive Proofs for PA Problems

We define a strategy to apply the PresPred^C-rules to a sequent $\Gamma \vdash \Delta \Downarrow ?$ that only contains PA formulae. The strategy is guaranteed to terminate and to produce a closed and exhaustive proof, and it is deterministic in the sense that no search is required, every ordering of rule applications (that is consistent with given priorities) leads to an exhaustive proof. In order to guide the proof construction, the strategy maintains a set U of constants (which is initially empty) and a term ordering $<_r$ (as in Sect. 2) that are updated when new constants are introduced or existing constants need to be reordered. The ordering $<_r$ is always chosen such that the constants in U are bigger than all constants that are not in U. Both U and $<_r$ are branch-local: different branches in a proof tree can be built using different Us and $<_r$s.

We list the rules that the strategy applies to a proof goal with descending priority: step 2 will only be carried out if step 1 is impossible, etc.

1. apply SIMP (if possible).
2. apply RED if an α exists such that $s + \alpha \cdot t <_r s$
   (and if $s \neq t$ or $\phi[s]$ is not an equation in the antecedent).
3. if the antecedent contains an equation $\alpha c + t \doteq 0$ with $\alpha > 1$, then:
   - if $c \notin U$, apply COL-RED-SUBST. The fresh constant c' is inserted in the term ordering $<_r$ such that it becomes minimal, and u is chosen such that $(\alpha u + t) = \min_{<_r}\{\alpha u' + t \mid u' \text{ a term}\}$.

- if $c \in U$ and t contains at least one further constant from U whose coefficient is not a multiple of α, apply COL-RED. The fresh constant c' is added to U and is inserted in the term ordering $<_r$ such that it becomes smaller than all other constants in U, but bigger than all constants not in U. u is again chosen such that $(\alpha u + t) = \min_{<_r} \{\alpha u' + t \mid u' \text{ a term}\}$.

4. if the antecedent contains an equation $\alpha c + t \doteq 0$ with $c \in U$, apply DIV-CLOSE, remove c from U, and update $<_r$ such that c becomes minimal. (This is also possible for $\alpha = 1$)
5. if possible, apply any of the following rules:
   - ANTI-SYMM.
   - FM-ELIM, if the result is not subsumed by an inequality in the antecedent.
   - any of the rules AND-*, OR-*, NOT-*.
6. if possible, apply any of the following rules:
   - SPLIT-EQ: if an equation can be split that contains a constant $c \in U$ that also occurs as leading term of an inequality in the antecedent.
   - OMEGA-ELIM: if inequalities $\{\alpha_i c - a_i \dot{\geq} 0\}_i$, $\{\beta_j c - b_j \dot{\leq} 0\}_j$ occur in the antecedent and $c \in U$, and if c does not occur in any other formula.
   - ALL-RIGHT, EX-LEFT: add the fresh constant c to U and insert it into $<_r$ such that it becomes maximal.
   - EX-RIGHT-D, ALL-LEFT-D: set U to $\emptyset$ and insert c arbitrarily into $<_r$.
   - DIV-LEFT, DIV-RIGHT.
7. apply CLOSE and select exactly those formulae that do not contain constants from U or uninterpreted predicates.

The steps 1–4 of the strategy work by eliminating all U-constants that occur in equations in the antecedent. Similarly as in [8], in the antecedent only equations will be left whose leading coefficient is 1 and whose leading term does not occur in other places in the sequent anymore. The steps 5–6 handle inequalities by first applying the Fourier-Motzkin rule exhaustively, and by eliminating constants using the Omega rule whenever possible. Also quantifiers, propositional connectives and divisibility judgements are treated in step 5–6. A proof that is constructed using this procedure is shown in Example 10.

Lemma 11 (Termination and exhaustiveness). *If a sequent $\Gamma \vdash \Delta \Downarrow ?$ does not contain uninterpreted predicates, the strategy from above terminates and produces a closed exhaustive proof.*

5.3 Deciding Presburger Arithmetic by Recursive Proving

The anticipated way to decide constraints in proofs is to eliminate quantifiers already during the constraint propagation, i.e., at the points where the rules EX-RIGHT(-D), ALL-LEFT(-D), ALL-RIGHT, EX-LEFT or COL-RED are applied and cause quantifiers to occur in constraints. By eliminating such quantifiers right away, each subproof of the proof can be annotated with a constraint that is a quantifier-free PA formula. When building proofs incrementally, this makes it possible to easily distinguish between unsatisfiable subproofs (i.e., subproofs

with an unsatisfiable constraint) that need to be expanded further, and satisfiable subproofs whose expansion can be postponed. Besides, due to Lem. 11 and as only quantifier-free constraints occur, the resulting procedure decides PA.

To eliminate quantifiers, the calculus $\mathrm{PresPred}^{C}$ can be used (Example 10):

Lemma 12 (Quantifier elimination). *Suppose a formula ϕ does not contain uninterpreted predicates and $\forall$ occurs in ϕ only in positive positions and $\exists$ only in negative positions. The strategy from the previous section produces a proof with root $\vdash \phi \Downarrow C$ in which C does not contain quantifiers (more precisely, if C contains a quantified subformula $Qx.\psi$, then x does not occur in ψ).*

6 Fair Construction of Proofs

We now compare the calculus $\mathrm{PresPred}^{C}$ with the more restricted calculus $\mathrm{PresPred}^{C}_{S}$ from Sect. 4. Because the former calculus is a superset of the latter, it is a trivial observation that any sequent provable in $\mathrm{PresPred}^{C}_{S}$ is also provable in $\mathrm{PresPred}^{C}$. It can also be shown that $\mathrm{PresPred}^{C}$ cannot prove more sequents than $\mathrm{PresPred}^{C}_{S}$ [4], which means that the two calculi are equivalent.

Proofs in $\mathrm{PresPred}^{C}$ can be found by a backtracking-free fair application strategy. To define the notion "fair," it has to be observed that formulae in a $\mathrm{PresPred}^{C}$-proof can be rewritten by applying RED or SIMP. When this happens, it is possible to identify a unique successor of the modified formula in the premiss of the rule application (vice versa, a formula can have multiple predecessors because distinct formulae could become equal when applying a rule).

A *fair $\mathrm{PresPred}^{C}$-proof* for a sequent $\Gamma \vdash \Delta \Downarrow ?$ is a possibly infinite proof in $\mathrm{PresPred}^{C}$ in which all constraints are ? and all branches have the properties:

- *Fair treatment of formulae with uninterpreted predicates:* whenever at some point on the branch one of the rules in Fig. 1 is applicable to a formula that contains uninterpreted predicates, the rule is applied to the formula or to a successor of the formula at some later point on the branch. (This implies that ALL-LEFT and EX-RIGHT are applied infinitely often to each universally quantified formula with uninterpreted predicates).
- *Fair unification of complementary literals:* if there is a sequent on the branch of the shape $\Gamma, p(\bar{t}) \vdash p(\bar{s}), \Delta \Downarrow ?$, the rule PRED-UNIFY is applied at least once on the branch to the pair $p(\bar{t})$, $p(\bar{s})$ or to successors of these formulae.
- *Exhaustiveness:* all proof nodes can be annotated with sets U as in Sect. 5.1.

A constraint C is *generated* by a fair proof of $\Gamma \vdash \Delta \Downarrow ?$ if a (finite) proof for $\Gamma \vdash \Delta \Downarrow C$ can be obtained by chopping off all branches of the fair proof at some point, applying CLOSE in some way to the leaves and propagating the resulting constraints through the proof. The next lemma, together with Lem. 9, implies the completeness of a fair rule, formula, and branch selection strategy.

Lemma 13 (Fair construction). *Suppose that a $\mathrm{PresPred}^{C}_{S}$-proof for the sequent $\Gamma \vdash \Delta \Downarrow C$ exists. Every fair $\mathrm{PresPred}^{C}$-proof of $\Gamma \vdash \Delta \Downarrow ?$ whose root is annotated with the set U generates a constraint D with $\forall U.C \Rightarrow \forall U.D$.*

7 Related Work

$\mathcal{ME}$(LIA) [7] is a recently proposed variant of the Model Evolution calculus that is similar to our calculus in that it supports PA enhanced with uninterpreted predicates (and without functions) as input language, and that its architecture resembles tableau calculi. Model Evolution does not use rigid free variables that are shared among different branches in the way tableaux do, however, which means that also constraints can be kept branch-local. Further differences are that $\mathcal{ME}$(LIA) works on clauses, only supports a restricted form of existential quantification, and has a more explicit representation of candidate models.

SMT-solvers based on the DPLL(T) architecture [10] can handle ground problems modulo integer arithmetic (and many other theories) efficiently, but only offer heuristic quantifier handling. Because of the similarity between DPLL and sequent calculi, the work presented in this paper can be seen as an alternative approach to handling quantifiers that should also be applicable to DPLL(T).

The simplification of formulae by the rules in Fig. 3 is roughly comparable with deduction modulo [11]. The concept is here integrated in a setting that resembles free-variable tableaux to treat quantifiers more efficiently.

An approach to embed algebraic constraints in tableau calculi is described in [12], where quantifier elimination tasks in real arithmetic (possibly involving more than one proof goal) are carried out by an external procedure, in a manner comparable to the simultaneous solving of constraints from multiple proof goals described here. Uninterpreted functions or predicates are not handled.

There are a number of approaches to include theories into resolution-based calculi. [13] works with constraints that are solved in a theory, but requires to enumerate the solutions of constraints (whereas it is enough to check the validity of constraints in our work). In [14], while it is enough to check satisfiability of constraints, no uninterpreted functions or predicates are supported. A recent calculus to handle rational arithmetic is given in [15], and is similar to our work in that it has built-in rules to solve systems of equations and inequalities (based on Fourier-Motzkin). The calculus is complete under restrictions that effectively prevent quantification over rationals. It remains to be investigated how this fragment is related to the fragments discussed here.

8 Conclusions and Future Work

We have presented a novel calculus to reason about problems in first-order logic modulo linear integer arithmetic. The calculus is complete for function-free first-order logic (on such problems, proofs in the calculus resemble free-variable tableaux with incremental closure [1]) and can decide Presburger arithmetic (in a manner that is similar to the Omega test [2]). As further results, we have shown that the calculus is at least as complete as the calculus $\mathcal{ME}$(LIA), and allows the fair construction of proofs. An implementation of the calculus is available under `http://www.cs.chalmers.se/~philipp/princess` and is described in [4].

Apart from continuing the implementation and benchmarks, there are a number of concepts that require more research, among others: the encoding and

handling of functions and further theories; the integration of lemma learning; the integration of connectivity conditions to make proof search more directed; the elimination of cuts in proofs; an analysis of the complexity of the calculus as a decision procedure for PA. We also plan to extend our calculus to support nonlinear arithmetic (following the work in [8]), and possibly rational arithmetic.

Acknowledgements. I want to thank Wolfgang Ahrendt, Nikolaj Bjørner, Richard Bubel, Reiner Hähnle, Henrik Johansson, and the anonymous referees for discussions and/or comments during different stages of this work.

References

1. Giese, M.: Incremental closure of free variable tableaux. In Goré, R., Leitsch, A., Nipkow, T., eds.: Proceedings, IJCAR, Siena, Italy. Volume 2083 of LNAI, Springer (2001) 545–560
2. Pugh, W.: The Omega test: a fast and practical integer programming algorithm for dependence analysis. In: Proceedings, 1991 ACM/IEEE conference on Supercomputing, New York, NY, USA, ACM (1991) 4–13
3. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Sprawozdanie z I Kongresu metematyków slowiańskich, Warszawa 1929, Warsaw, Poland (1930) 92–101,395
4. Rümmer, P.: Calculi for Program Incorrectness and Arithmetic. PhD thesis, Chalmers University of Technology (2008) to appear.
5. Fitting, M.C.: First-Order Logic and Automated Theorem Proving. 2nd edn. Springer-Verlag, New York (1996)
6. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM **22** (1979) 465–476
7. Baumgartner, P., Fuchs, A., Tinelli, C.: MELIA – model evolution with linear integer arithmetic constraints. to appear (2008)
8. Rümmer, P.: A sequent calculus for integer arithmetic with counterexample generation. In Beckert, B., ed.: Proceedings, 4th International Verification Workshop. Volume 259 of CEUR (`http://ceur-ws.org/`) (2007)
9. Schrijver, A.: Theory of Linear and Integer Programming. Wiley (1986)
10. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM **53** (2006) 937–977
11. Dowek, G., Hardin, T., Kirchner, C.: Theorem proving modulo. Journal of Automated Reasoning **31** (2003) 33–72
12. Platzer, A.: Differential dynamic logic for hybrid systems. Journal of Automated Reasoning **41** (2008) 143–189
13. Stickel, M.E.: Automated deduction by theory resolution. Journal of Automated Reasoning **1** (1985) 333–355
14. Bürckert, H.J.: A resolution principle for clauses with constraints. In Stickel, M.E., ed.: Proceedings, 10th International Conference on Automated Deduction. Volume 449 of LNCS, Springer (1990) 178–192
15. Korovin, K., Voronkov, A.: Integrating linear arithmetic into superposition calculus. In Duparc, J., Henzinger, T.A., eds.: Proceedings, 21st International Workshop on Computer Science Logic. Volume 4646 of LNCS, Springer (2007) 223–237