Calculemus 2005 Preliminary Version

An Even Closer Integration of Linear Arithmetic into Inductive Theorem Proving

Tobias Schmidt-Samoa [1]

FB Informatik, Tech. Univ. Kaiserslautern, Germany

Abstract

To broaden the scope of decision procedures for linear arithmetic, they have to be integrated into theorem provers. Successful approaches e.g. in NQTHM or ACL2 suggest a close integration scheme which augments the decision procedures with lemmas about user-defined operators. We propose an even closer integration providing feedback about the state of the decision procedure in terms of entailed formulas for three reasons: First, to provide detailed proof objects for proof checking and archiving. Second, to analyze and improve the interaction between the decision procedure and the theorem prover. Third, to investigate whether the communication of the state of a failed proof attempt to the human user with the comprehensible standard GUI mechanisms of the theorem prover can enhance the speculation of auxiliary lemmas.

Key words: Decision Procedures, Human-Oriented Theorem Proving, Integration Scheme, Lemma Speculation, Linear Arithmetic, Proof Objects

1 Introduction

In comparison to theorem provers based on heuristic search strategies, decision procedures are very efficient in deciding formulas over their dedicated domain. But this domain is usually rather small. Many formulas just fall outside the theory of the decision procedure. Therefore, two different approaches have been studied by many researchers for at least three decades to overcome these limitations: First, the *combination* of different decision procedures over *disjunctive* domains; second, the incorporation of decision procedures into heuristic theorem provers using *augmentation*. Research about the first approach has been initiated by fundamental work from Nelson & Oppen [17] and Shostak [19]. In this paper, we are concerned with the second approach. Seminal work on this topic has been done by Boyer & Moore [7] integrating

[1] Email: schmidt@informatik.uni-kl.de

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

a decision procedure for rational numbers based on Hodes [9] (credited to Fourier in [14]) into their inductive theorem prover `NQTHM` [6].

By *linear arithmetic* we mean the universally quantified first-order theory over predicate symbols $<$, $\leq$, $=$, $\neq$, $\geq$, $>$ for order relations over numbers, and function symbols 0, s and $+$ for constant zero, unary successor function and binary addition.[2] According to the underlying domain, these theories are called *Presburger rational arithmetic* (PRA), *Presburger integer arithmetic* (PIA), and *Presburger natural arithmetic* (PNA) in [10]. We are interested in a semi-decision procedure for an *extended theory* of PNA containing additional predicate or function symbols. These symbols are *uninterpreted* for the decision procedure but may be constrained by the theorem prover.

Hodes' procedure can be used as a decision procedure for PRA and as a semi-decision procedure for PIA and PNA. It checks for unsatisfiability of a set of inequalities. The key idea is to "cross-multiply and add" [7] two inequalities to eliminate a common variable. We call this a *variable elimination step*. Variable elimination steps can be restricted to the heaviest variables in an inequality w.r.t. a fixed wellfounded order. In previous work, the inequalities are stored in an internal state of the decision procedure called *linear arithmetic data base* in [7] or *constraint store* in [2,3]. If an unsatisfiable (ground) inequality is derived, the original inequalities are unsatisfiable over rationals, integers and naturals. Otherwise, if the set is closed under variable elimination steps, the original inequalities are satisfiable over the rationals but may be unsatisfiable over integers or naturals.

If we try to prove theorems in a suitably extended theory, we have to use additional facts about the extension provided by the theorem prover. These facts are usually given in the form of (conditional) lemmas. The conclusion of a lemma can be applied if the conditions of the lemma can be proved valid. These proofs may be performed by the decision procedure or the theorem prover. Thus, we get *mutual dependencies*. According to [13], the integration of linear arithmetic into `ACL2` leads to four dependencies between the simplifier and the arithmetic package. Therefore, it is questionable whether the integration of linear arithmetic as a separate module is sensible.

Following [15], we call a lemma a *rewrite rule* if the conclusion of the lemma is an equation $s{=}t$; we call it a *linear rule* if the conclusion is an inequality $u{\leq}v$. The application of a rewrite rule replaces an instance of s with the same instance of t. In contrast to this, the application of a linear rule adds an instance of $u{\leq}v$ to the state of the decision procedure so that it can benefit from the new inequality. This *augmentation* mechanism was introduced in [7].

The situation gets worse if the required lemmas are not present. To speculate rewrite rules automatically, successful approaches have been proposed e.g. in [16,8]. But for the automatic speculation of linear rules no general

[2] For ease of use, we will also consider constant symbols for all numbers, $-$ for subtraction, and $\cdot$ for multiplication with constants ($n \cdot x$ abbreviates the sum that contains n times x).

approach has been proposed. There only exist approaches for nonlinear arithmetic [12,1]. As a linear rule contains an estimate, it is more difficult than for rewrite rules to speculate lemmas that are both—*valid* and *useful*. In our opinion, lemma speculation is a very creative task that has to be done by humans in most cases. But this task must be supported as far as possible. Therefore, we require an appropriate interaction scheme providing the human user with all the information needed. Previous approaches lack information for two reasons: First of all, they do not explicitly present the state of the decision procedure to the user. Instead, the information is hidden in the internal state of the decision procedure. Furthermore, the decision procedure only eliminates the heaviest terms.

In this paper, we present a new approach to incorporate a decision procedure for PNA closely into our inductive theorem prover QuodLibet [4]. We strictly distinguish the logic part of the decision procedure from its control aspects: Each elementary step of the decision procedure is represented by a new *inference rule* providing the state of the decision procedure explicitly in the clauses that result from its application. A variable elimination step e.g. introduces one new literal that combines two inequalities eliminating the considered variable. Local properties of the inference rules guarantee the *soundness* of our approach. They may be automatically applied with *tactics* written in an adapted imperative programming language QML.

Our approach provides the following advantages: The fragmentation of the decision procedure into elementary steps—realized with inference rules—provides us with detailed *proof objects* that can be easily checked with an external proof checker. It also enables a *uniform* and *flexible* integration into our simplification process. This allows us to evaluate different integration schemes that are defined on a much more fine-grained level than in previous approaches. It also gives us the opportunity to implement different strategies: The integration into the simplification process `simplify` uses the heuristics known from the literature to automate the decision procedure and to guide the augmentation mechanism. Furthermore, we have implemented a special purpose tactic `leq-var-elim` that performs all possible variable elimination steps (but no other steps). We call this tactic only if the simplification process fails and we need more information to speculate an auxiliary linear lemma. Although this tactic may lead to an exponential blow-up of inequalities, this does not seem to be a severe problem in practice. Firstly, we remove redundant inequalities. Secondly, an experienced user often finds the required inequalities soon concentrating on the simpler ones.

The rest of the paper is organized as follows: After a short overview over QuodLibet in Section 2, we illustrate the augmentation mechanism as well as our new integration scheme with a simple example in Section 3. We describe our approach in Section 4 and evaluate it in Section 5 concentrating on its ability to speculate new linear lemmas. After a survey of related work in Section 6, we conclude with further work in Section 7.

2 The Inductive Theorem Prover QuodLibet

QUODLIBET [4] is an equality-based inductive theorem prover for clausal first-order logic with implicitly universally quantified variables. It admits *partial* definitions of operators over *free constructors* using (possibly *non-terminating*) conditional equations as well as *constructor*, *destructor*, and *mutual* recursion. *Inductive validity* is defined as validity in the class of so-called *data models*, the models that do not equalize any different constructor ground terms.

More precisely, a specification $\texttt{spec} = (\Sigma, E)$ is given by a signature Σ and *positive/negative*-conditional equations E. A signature $\Sigma = (S, C, F)$ consists of a set of sorts S, a set of free constructors $C \subseteq F$, and a set of function symbols F. Given a set of variables V, the set of terms $\mathrm{Term}(F, V)$ is defined as usual. Let $\mathrm{top}(t)$ be the *toplevel operator* of term t. Atoms are constructed using one of the predefined predicate symbols for equality (symbol $=$), definedness ($\texttt{def}$) and order relations ($<$ or $\leq$), respectively.

We use a sequent calculus to prove clauses—termed *goals* in QUODLIBET. A clause $\{l_1, \ldots, l_n\}$ consists of disjunctively combined literals l_i.[3] An *inference rule* reduces a goal to a (possibly empty) sequence of new subgoals. A proof is represented by a *proof state tree* consisting of *goal* and *inference nodes*. The root goal node of a proof state tree consists of the clause to be proved. An inference node represents the inference rule applied to its parent which is a goal node. Its n children ($n \geq 0$) are again goal nodes and represent the new subgoals created by the inference rule. A proof state tree is *closed* if all leaves are inference nodes. In this case, the clause of the root goal node is inductively valid provided that this holds true for the applied lemmas.

3 A Simple Example

First, we illustrate Hodes' procedure with a simple example.

Example 3.1 (derived from [7]) *We want to prove the validity of Formula (1) over the naturals. Therefore, we check its negation for unsatisfiability.*

(1) $\forall K, L, Max, Min.(L \leq Min \wedge 0 < K \wedge Min \leq Max \rightarrow L < Max + K)$

After normalizing the negation of (1), we get the following conjunctively combined inequalities. Note that we use the integral property of the naturals in Inequality (3): The difference of two unequal naturals is at least one.

(2) $L \leq Min$ (3) $1 \leq K$ (4) $Min \leq Max$ (5) $Max + K \leq L$

We restrict variable elimination steps using an alphabetic order on variable names. Thus, we derive the following inequalities with an unsatisfiable ground Inequality (8):

[3] More precisely, a clause is a list of literals. The order of the literals is relevant for the automatic proof control. We write $\Delta \cup \Gamma$ to append the literals of clauses Δ and Γ.

(6) $L \leq Max$ *from* (2) *and* (4) *eliminating* Min

(7) $K \leq 0$ *from* (6) *and* (5) *eliminating* Max

(8) $1 \leq 0$ *from* (3) *and* (7) *eliminating* K □

Example 3.1 falls into the decidable theory of PNA. But if we replace variables Min and Max with function calls $MIN(A)$ and $MAX(A)$ as well as the third condition $Min \leq Max$ with $A \neq nil$, then the formula is no longer valid in pure PNA. Therefore, we use the augmentation mechanism [7].

Example 3.2 (derived from [7]) *We want to prove the validity of Formula* (9) *over the naturals. We assume that* (10) *is valid in the extended theory.*

(9) $\forall A, K, L.(L \leq MIN(A) \wedge 0 < K \wedge A \neq nil \rightarrow L < MAX(A) + K)$

(10) $\forall A.(A \neq nil \rightarrow MIN(A) \leq MAX(A))$

The decision procedure can make use of Inequalities (11)–(13) *derived from the negation of* (9)*. We assume that the decision procedure handles terms starting with uninterpreted function symbols just like variables over the naturals. Therefore, we omit explicit generalizations.*

(11) $L \leq MIN(A)$ (12) $1 \leq K$ (13) $MAX(A) + K \leq L$

As the decision procedure only tries to eliminate the heaviest terms in an inequality, it does not perform a single step (assuming the same order on the terms as in Example 3.1). But Lemma (10) *may be applied as it contains additional information about the heaviest term of* (11)*. The condition of the lemma can be relieved because the same literal occurs in Formula* (9)*. Therefore, we can augment the data base of the decision procedure with the conclusion of the lemma, namely* $MIN(A) \leq MAX(A)$*. Then we can replay the proof from Example 3.1.* □

Example 3.2 can be complicated further by replacing the condition $A \neq nil$ in Formula (9) with $length(A) > 0$, introducing another uninterpreted function symbol. Then, the condition of Lemma (10) is not directly present in Formula (9) but has to be relieved by recursively calling the simplifier of the theorem prover and the decision procedure.

In the following example, we want to indicate how our integration scheme supports the speculation of auxiliary lemmas required for the augmentation mechanism if these lemmas are not present.

Example 3.3 *We consider Formula* (9) *from Example 3.2 (in clausal form) and want to derive Lemma* (10)*. Figure 1 illustrates our derivation in form of a proof state tree with the root goal node displayed at the top.*

We first try to prove the clause automatically by calling tactic `simplify`*. This automatic proof attempt starts by normalizing all literals with inference rule* `la-norm`*. The literals (or terms) that are used by an inference rule are framed in Figure 1. Since tactic* `simplify` *uses the heuristics to eliminate only heaviest terms, the proof attempt fails after the three normalization steps.*

$$\{\ \boxed{L < \texttt{+}(\texttt{MAX}(A), K)},\ \ \neg(L \le \texttt{MIN}(A)),\ \ \neg(\texttt{0} < K),\ \ A = \texttt{nil}\ \}$$

$$\texttt{la-norm}$$

$$\{\ \texttt{+}(\texttt{1}, L) \le \texttt{+}(K, \texttt{MAX}(A)),\ \ \boxed{\neg(L \le \texttt{MIN}(A))},\ \ \neg(\texttt{0} < K),\ \ A = \texttt{nil}\ \}$$

$$\texttt{la-norm}$$

$$\{\ \texttt{+}(\texttt{1}, L) \le \texttt{+}(K, \texttt{MAX}(A)),\ \ \texttt{+}(\texttt{1}, \texttt{MIN}(A)) \le L,\ \ \boxed{\neg(\texttt{0} < K)},\ \ A = \texttt{nil}\ \}$$

$$\texttt{la-norm}$$

$$\{\ \texttt{+}(\texttt{1}, \boxed{L}) \le \texttt{+}(K, \texttt{MAX}(A)),\ \ \texttt{+}(\texttt{1}, \texttt{MIN}(A)) \le \boxed{L},\ \ K \le \texttt{0},\ \ A = \texttt{nil}\ \}$$

$$\le\texttt{-var-elim}$$

$$\{\ \texttt{+}(\texttt{1}, \texttt{MIN}(A)) \le \texttt{+}(\boxed{K}, \texttt{MAX}(A)),\ \ \texttt{+}(\texttt{1}, L) \le \texttt{+}(K, \texttt{MAX}(A)),\ \ \texttt{+}(\texttt{1}, \texttt{MIN}(A)) \le L,\ \ \boxed{K} \le \texttt{0},\ \ A = \texttt{nil}\ \}$$

$$\le\texttt{-var-elim}$$

$$\{\ \texttt{MIN}(A) \le \texttt{MAX}(A),\ \ \texttt{+}(\texttt{1}, \texttt{MIN}(A)) \le \texttt{+}(K, \texttt{MAX}(A)),\ \ \texttt{+}(\texttt{1}, L) \le \texttt{+}(\boxed{K}, \texttt{MAX}(A)),\ \ \texttt{+}(\texttt{1}, \texttt{MIN}(A)) \le L,\ \ \boxed{K} \le \texttt{0},\ \ A = \texttt{nil}\ \}$$

$$\le\texttt{-var-elim}$$

$$\{\ L \le \texttt{MAX}(A),\ \ \boxed{\texttt{MIN}(A) \le \texttt{MAX}(A)},\ \ \texttt{+}(\texttt{1}, \texttt{MIN}(A)) \le \texttt{+}(K, \texttt{MAX}(A)),\ \ \texttt{+}(\texttt{1}, L) \le \texttt{+}(K, \texttt{MAX}(A)),\ \ \texttt{+}(\texttt{1}, \texttt{MIN}(A)) \le L,\ \ K \le \texttt{0},\ \ \boxed{A = \texttt{nil}}\ \}$$

Fig. 1. Derivation of Lemma (10) from Formula (9)

Calling the special purpose tactic `leq-var-elim`*, all variable elimination steps with inference rule* $\le$`-var-elim` *are performed. This results in a last goal node that contains the needed auxiliary lemma as subformula, marked with frames in Figure 1. Therefore, we get a hypothesis for an auxiliary lemma.* □

4 Close Integration of Linear Arithmetic

In this section, we present a simplified version of our integration scheme for PNA into the inductive theorem prover QuodLibet. Due to lack of space, we have to leave out many technical details. These can be found in [18]. Instead, we try to explain the basics of our approach intuitively.

To guarantee a consistent integration of PNA, we assume a base specification $\texttt{spec}_0$ that consists of a sort `Nat` with constructors `0` and `s` and defined function symbols `+`, `-` and `*`. Constructor ground terms $\texttt{s}^i(\texttt{0})$ will be written as i. The defined function symbols are specified by the axioms:

(14) $\{\ \texttt{+}(x, \texttt{0}) = x\ \}$ (15) $\{\ \texttt{+}(x, \texttt{s}(y)) = \texttt{s}(\texttt{+}(x, y))\ \}$

(16) $\{\ \texttt{-}(x, \texttt{0}) = x\ \}$ (17) $\{\ \texttt{-}(\texttt{0}, \texttt{s}(y)) = \texttt{0}\ \}$ (18) $\{\ \texttt{-}(\texttt{s}(x), \texttt{s}(y)) = \texttt{-}(x, y)\ \}$

(19) $\{\ \texttt{*}(x, \texttt{0}) = \texttt{0}\ \}$ (20) $\{\ \texttt{*}(x, \texttt{s}(y)) = \texttt{+}(\texttt{*}(x, y), x)\ \}$

QuodLibet's admissibility condition guarantees that the class of data models DMod(`spec`) for each extended specification `spec` of $\texttt{spec}_0$ is not empty. Thus, the semantics is welldefined. The soundness proofs of our inference rules for PNA are based on inductively valid lemmas for the base specification $\texttt{spec}_0$. Note that the naturals provide one of the data models for $\texttt{spec}_0$.

4.1 Inference Rules for PNA

A decision procedure for PNA can be divided into the following steps: normalization, variable elimination and a check of ground instances. For the integration into QuodLibet, we transform the procedure sketched in Section 1 into a semi-decision procedure for inductive validity using negation. The state of the decision procedure is represented directly within the goal clauses in form of new literals added by the inference rules. Note that we only have to add inference rules for the decision procedure. The augmentation mechanism can be realized by a lemma application mechanism already present in QuodLibet.

4.1.1 Inference Rule `la-norm`

To support the variable elimination steps, we have to determine the number of occurrences of each term in an inequality. Therefore, we define *polynomials* and *normalized inequalities*. We assume $<_{\mathrm{Term}}$ to be a fixed, total, wellfounded order on terms.

Definition 4.1 (Polynomials / Factors / Multiplicands / Constants) *p is a* polynomial *if $p \equiv c + \sum_{i=1}^{n} c_i t_i$ and for all $i, j \in \{1, \ldots, n\}$: $c, c_i \in \mathbb{N}$, $c_i \neq 0$, $t_i \in \mathit{Term}(F, V)$, $\mathit{top}(t_i) \neq$ `+` and $t_i <_{\mathit{Term}} t_j$ for $i < j$. c_i is called* factor*, t_i* multiplicand*, and c* constant *of the polynomial p.*

A polynomial can be easily represented as a term if we construct the sum with operators `+` and `*` with parenthesis associating to the right. Additionally, we eliminate factors with value 1 and constants with value 0. We identify polynomials with their term representation. Thus, we can use them in (in)equalities.

Definition 4.2 (Normalized Inequalities) *$p_1 \leq p_2$ is a* normalized inequality *if p_1 and p_2 are polynomials that do not share any multiplicand, one of the constants is equal to 0, and the set of factors of p_1 and p_2 is coprime.*

Every (in)equality l over terms of sort `Nat` can be transformed into an equivalent formula containing only normalized inequalities: Both sides are transformed into polynomials and common occurrences of multiplicands and constants are canceled. Coprimality is achieved by dividing the inequality through the greatest common divisor g of all factors. If g does not divide the constant of p_1 (resp. p_2), the result can be rounded up (resp. down) without changing the set of integer solutions. We assume a *normalization function* 'norm' that performs the whole transformation. In fact, we use different implementations obeying Definition 4.2. On the one hand, we want to simplify the multiplicands as far as possible; on the other hand, we want to reduce the number of case splits (see Section 4.1.4). A normalization function may leave the toplevel occurrences of operator '`-`' untouched, or may eliminate them according to Axioms (16) to (18). In this case, the normalization function has to introduce a case split w.r.t. the *linearization hypothesis* [7] whether the minuend is greater or equal than the subtrahend. In general, a normalization function has the form $\mathrm{norm}(l) = ((\mathrm{lh}_1, \mathrm{nf}_1), \ldots, (\mathrm{lh}_k, \mathrm{nf}_k))$ such that for each $i \in \{1, \ldots, k\}$: lh_i

is a sequence of normalized inequalities, nf_i is a normalized inequality and the literal l is equivalent to $(\bigvee \mathrm{lh}_1 \vee \mathrm{nf}_1) \wedge \cdots \wedge (\bigvee \mathrm{lh}_k \vee \mathrm{nf}_k)$. The sets lh_i are the *linearization hypotheses* and the inequalities nf_i the *normal forms* of l. Our concrete implementations can be found in [18] based on ideas from [7].

The application of the inference rule `la-norm`(j) creates one subgoal for each normal form of literal l_j: It replaces l_j with its normal form nf_i and adds the linearization hypotheses lh_i to the front of the subgoal.

`la-norm`(j):

$$\frac{\{l_1, \ldots, l_{j-1}, l_j, l_{j+1}, \ldots, l_n\}}{\mathrm{lh}_1 \cup \{l_1, \ldots, l_{j-1}, \mathrm{nf}_1, l_{j+1}, \ldots, l_n\} \quad \ldots \quad \mathrm{lh}_k \cup \{l_1, \ldots, l_{j-1}, \mathrm{nf}_k, l_{j+1}, \ldots, l_n\}}$$

where $\mathrm{norm}(l_j) = ((\mathrm{lh}_1, \mathrm{nf}_1), \ldots, (\mathrm{lh}_k, \mathrm{nf}_k))$.

4.1.2 Inference Rule $\leq$`-var-elim`

Given two normalized inequalities $l_{j_1} \equiv p_1 \leq p_2$ and $l_{j_2} \equiv p_3 \leq p_4$ with $p_k \equiv c^{(k)} + \sum_{i=1}^{n_k} c_i^{(k)} t_i^{(k)}$, we want to eliminate a common term $t \equiv t_u^{(1)} \equiv t_v^{(4)}$. First, we negate both inequalities. The result is $1 + p_2 \leq p_1$ and $1 + p_4 \leq p_3$, respectively. Then, we multiply the first inequality with $c_v^{(4)}$ and the second one with $c_u^{(1)}$. Let $p_1' \equiv c_v^{(4)} p_1$, $p_2' \equiv c_v^{(4)} + c_v^{(4)} p_2$, $p_3' \equiv c_u^{(1)} p_3$ and $p_4' \equiv c_u^{(1)} + c_u^{(1)} p_4$. The addition of the two inequalities results in $p_2' + p_4' \leq p_1' + p_3'$. Thus, we may add its negation $1 + p_1' + p_3' \leq p_2' + p_4'$ preserving soundness. This inequality contains $c_u^{(1)} c_v^{(4)} t$ on both sides which will be eliminated by normalization.

The application of inference rule $\leq$`-var-elim`(j_1, j_2, t) adds the normal forms of the negation of the last inequality to the front of the clause.

$\leq$`-var-elim`(j_1, j_2, t):

$$\frac{\{l_1, \ldots, l_{j_1}, \ldots, l_{j_2}, \ldots, l_n\}}{\mathrm{lh}_1 \cup \{\mathrm{nf}_1, l_1, \ldots, l_{j_1}, \ldots, l_{j_2}, \ldots, l_n\} \quad \ldots \quad \mathrm{lh}_k \cup \{\mathrm{nf}_k, l_1, \ldots, l_{j_1}, \ldots, l_{j_2}, \ldots, l_n\}}$$

where $\mathrm{norm}(1 + p_1' + p_3' \leq p_2' + p_4') = ((\mathrm{lh}_1, \mathrm{nf}_1), \ldots, (\mathrm{lh}_k, \mathrm{nf}_k))$, p_i' defined as above.

Usually, the normalization of the sum of normalized inequalities results in one normal form without linearization hypotheses. In this case, the application of the inference rule just adds one new subgoal that contains one new literal.

Note that we do not restrict the inference rule to heaviest terms since this is not important for its soundness. Instead, our inference rule is more general. Its automatic application is restricted by heuristics implemented in tactics.

4.1.3 Inference Rule $\leq$`-taut`

As inequalities are only defined over terms of sort `Nat`, each normalized inequality $0 \leq p$ is (inductively) valid. Thus, we get a simple tautological inference rule that does not create any new subgoals.

$\leq$`-taut`(j):

$$\frac{\{l_1, \ldots, l_j, \ldots, l_n\}}{}$$

where $l_j \equiv 0 \leq p$.

4.1.4 Further Refinements

Our current integration of a decision procedure for PNA consists of a couple of further inference rules:

- `≤-case-split` can be used to turn the semi-decision procedure for PNA into a decision procedure, see Section 6.
- `≤-removal` and `≤-subs-removal` eliminate redundant literals that do not contain additional information, i.e. inequalities of the form $c + t \leq 0$ with $1 \leq c$; or $c_1 + t_1 \leq c_2 + t_2$ if a stronger inequality $d_1 + t_1 \leq d_2 + t_2$ is present with $c_2 + d_1 \leq c_1 + d_2$. This reduces the complexity of the goal clause.
- `≠-var-elim` and `la-const-rewrite` allow one to directly eliminate one variable in a negated equation with rewriting techniques [14].
- `la-term-norm` replaces a term of sort `Nat` with its polynomial w.r.t. Def. 4.1.

To reduce the number of case splits, we offer three different normalization functions, resulting in three different *normalization levels* for literals. Only the third level converts an equality into two inequalities. Only the second and third level eliminate toplevel occurrences of the operator '`-`' in multiplicands. On the first level, we do not split at all w.r.t. equalities and operator '`-`'.

As QuodLibet can handle partially defined operators, the inference rules may have to create additional definedness subgoals for terms starting with a defined operator. Due to lack of space, we do not present further details here. In our examples, we hide the definedness subgoals created. Nevertheless, definedness subgoals are created in the case studies: They are responsible for some of the inference rules applied as well as for a slice of the runtime needed.

4.1.5 Properties of the Inference Rules

The following lemma states two important local properties of our new inference rules: soundness and safeness. The *soundness* of all inference rules guarantees the inductive validity of a lemma with a closed proof state tree provided that all non-inductively applied lemmas are inductively valid. Vice versa, if a goal with a non-valid clause, like the empty clause, is derived in a proof attempt of a lemma, then this lemma (or one of the applied lemmas) cannot be inductively valid because of the *safeness* property of the inference rules. A proof of Lemma 4.3 can be found in [18].

Lemma 4.3 *All inference rules for PNA are sound and safe.* □

4.2 Automation

According to `NQTHM` [6] and `ACL2` [15], we have implemented a *waterfall* model that divides the simplification process into *phases*. The idea of a waterfall model is to start with the cheapest phases that promise the highest profit. The control of our waterfall is simple but flexible and can be configured dynamically. Its simplicity allows the easy integration of new *operations* to handle additional proof patterns. Its flexibility is sufficient to fix the order for applying the inference rules in a suitable way with a table-based configuration. Each operation can be additionally influenced by optional parameters.

Our implementation uses a separate tactic to control the recursive structure of the waterfall. For each open goal, each phase is applied in succession until the first phase succeeds. During the application of a phase, each literal in the goal is handled in succession. A literal is handled by calling those operations of the phase that are associated with the type of the literal. Thus, each phase contains for each type of literal a list of operations to be checked for applicability. The *type* of a literal consists of the predicate symbol of the literal ($=$, `def`, $<$, $\leq$) and a flag whether the literal is negated.

An *operation* is called with the considered goal and literal as arguments. It is intended to apply a number of inference rules to handle a certain proof pattern. If the operation fails, it has to restore the former state before it was called. Thus, it has to delete all proof steps applied by itself. If the operation succeeds, it returns all open subgoals that should be handled by a recursive call of the simplification process. The proof steps of a successful operation will not be deleted anymore. Thus, the operation is responsible to check whether the proof pattern applies. To achieve this, the operation may recursively call the simplification process. But these calls should be restricted to perform complete proofs of certain subgoals.

If an operation succeeds, the simplification process is usually started from the beginning for the resulting subgoals. To provide more flexibility, we may choose a different behavior. For each phase we may specify two lists of phases: The phases of the first list are called for each new subgoal and the (rewritten) literal for which the operation succeeded. The phases of the second list are called for each new subgoal and the new literals that were added by the successful operation. This allows us to define a specific behavior as response to a successful operation. Furthermore, we may choose to complete the phase for all the literals that were already present in the original goal, before we start the simplification process from the beginning. In this way, we can realize a fair handling of every literal in the goal clause. We may even choose to continue the simplification process with the next phase. This is sensible if we know that the first phases will fail anyway so that we do not have to check them once again. This is the case e.g. for a phase that only removes literals.

Before we integrated a decision procedure for PNA, our waterfall consisted of the following five phases: The first proved simple tautologies; the second removed redundant literals; the third applied directly applicable lemmas (i.e. lemmas whose condition literals are directly present in the goal clause); the fourth decomposed literals and applied lemmas even if they were not directly applicable; the fifth used equalities for *cross-fertilization* [6]. For the integration of the decision procedure, for each normalization level, we add one phase to normalize (in)equalities over terms of sort `Nat`; one phase to eliminate variables in normalized negated equations and inequalities, respectively; one phase to replace a term with its polynomial; for negated equations and inequalities, one phase to implement the augmentation mechanism; and one phase to introduce a Cut with a tautological literal $0 \leq v$ to allow the elimination of v on the right-hand side of inequalities. Furthermore, we extend the first two phases of the old waterfall to prove simple tautologies for inequalities and to remove redundant inequalities. We also split the third and fourth phase into two parts: Definedness atoms are handled before we normalize (in)equalities. For the other types of literals, these phases are applied after the variable elimination steps. The other new phases as well as the normalization of literals in the third normalization level are inserted between the fourth and fifth phase of the old waterfall. In this way, we realize in tactic `simplify` a simplification process that interleaves the decision procedure with the previous phases of the theorem prover on a fine-grained level. This interleaving is based on our intuition as well as on the experiments we have performed. Nevertheless, our integration is not optimal, yet. We will improve e.g. our heuristics to restrict the augmentation mechanism furthermore. The flexibility of our integration scheme supports us in this task.

Independently from our simplification process, we have implemented a special purpose tactic `leq-var-elim` to facilitate the speculation of auxiliary lemmas for the augmentation mechanism. This tactic performs all variable elimination steps possible, without considering the heuristics to eliminate only heaviest terms. To guarantee termination, the tactic performs all variable elimination steps for a term only once. It starts with the heaviest multiplicand w.r.t. $<_{\mathrm{Term}}$ that occurs in an inequality. The soundness of the tactics is guaranteed by Lemma 4.3.

5 Case Studies

With our case studies, we want to demonstrate that our integration scheme is sensible. Therefore, we evaluate it on five problems taken from the literature [7,14]. These problems were used as benchmarks for the incorporation of the *extended proof method* `EPM` into `Clam` [10], and *constraint contextual rewriting* `CCR(X)` into `RDL` [2], respectively. Problem 6 is a little bit more challenging. It entails that the square root of 2 is irrational. The proof is performed with induction based on geometric ideas of Hippasos of Metapont. The induction scheme that can hardly be guessed algebraically is supplied manually. The rest of the proof is done automatically. Table 1 contains for each problem the auxiliary lemmas L that are available to prove goal G. The last three columns contain the runtime in seconds for the systems `Clam`, `RDL` and QuodLibet (`QL`), respectively. An entry '—' means that the test was not performed with the system, '?' means that the goal was not proved. Note that we did not perform the experiments for `Clam` and `RDL` on our own. Instead we quote the results mentioned for `Clam` in [10] and for `RDL` in [2]. The tests for `Clam` were performed on a 433 Mhz PC, whereas the tests for `RDL` and QuodLibet were made on a 1 Ghz PC. Note that our measurements contain the output of a

Prob#		Problem	Clam	RDL	QL
1 [7]	G	$\{\ L < \mathtt{+}(\mathtt{MAX}(A), K),\ \neg(L \leq \mathtt{MIN}(A)),\ \neg(\mathtt{0} < K),\ A = \mathtt{nil}\ \}$	0.14	—	0.02
	L	(i) $\{\ \mathtt{MIN}(A) \leq \mathtt{MAX}(A),\ A = \mathtt{nil}\ \}$			
2 [7]	G	$\{\ \mathtt{+}(I, \mathtt{DELTA1}(PAT, LP, C)) \leq MAXINT,\ \neg(\mathtt{+}(LP, LT) \leq MAXINT),\ \neg(I \leq LT)\ \}$	0.23	0.01	0.02
	L	(i) $\{\ \mathtt{DELTA1}(PAT, LP, C) \leq LP\ \}$			
3 [7]	G	$\{\ \mathtt{+}(W, \mathtt{LEN}(\mathtt{DEL}(Z, A))) < \mathtt{+}(K, V),\ \mathtt{MEMB}(Z, A) \neq \mathtt{true},\ \neg(\mathtt{+}(W, \mathtt{LEN}(A)) \leq K)\ \}$	—	0.01	0.02
	L	(i) $\{\ \mathtt{LEN}(\mathtt{DEL}(X, S)) < \mathtt{LEN}(S),\ \mathtt{MEMB}(X, S) \neq \mathtt{true}\ \}$			
4 [7]	G	$\{\ \mathtt{+}(\mathtt{+}(\mathtt{MS}(c), \mathtt{*}(\mathtt{MS}(a), \mathtt{MS}(a))), \mathtt{*}(\mathtt{MS}(b), \mathtt{MS}(b))) < \mathtt{+}(\mathtt{+}(\mathtt{+}(\mathtt{MS}(c), \mathtt{*}(\mathtt{MS}(b), \mathtt{MS}(b))),$ $\mathtt{*}(\mathtt{2}, \mathtt{*}(\mathtt{MS}(a), \mathtt{*}(\mathtt{MS}(a), \mathtt{MS}(b))))), \mathtt{*}(\mathtt{MS}(a), \mathtt{*}(\mathtt{MS}(a), \mathtt{*}(\mathtt{MS}(a), \mathtt{MS}(a)))))\ \}$	5.73	0.03	0.52
	L	(i) $\{\ J \leq \mathtt{*}(I, J),\ \neg(\mathtt{0} < I)\ \}$ (ii) $\{\ \mathtt{0} < \mathtt{MS}(x)\ \}$			
5 [14]	G	$\{\ z < \mathtt{+}(\mathtt{g}(x), y),\ \mathtt{p}(x) \neq \mathtt{true},\ \neg(z \leq \mathtt{f}(\mathtt{max}(x, y))),\ \neg(\mathtt{0} < \mathtt{min}(x, y)),$ $\neg(x \leq \mathtt{max}(x, y)),\ \neg(\mathtt{max}(x, y) \leq x)\ \}$	?	0.06	0.10
	L	(i) $\{\ \mathtt{f}(x) \leq \mathtt{g}(x),\ \mathtt{p}(x) \neq \mathtt{true}\ \}$ (ii) $\{\ \mathtt{min}(x, y) = y,\ \mathtt{max}(x, y) \neq x\ \}$			
6	G	$\{\ \mathtt{*}(\mathtt{2}, \mathtt{*}(y, y)) \neq \mathtt{*}(x, x),\ y = \mathtt{0}\ \}$	—	—	1.72
	L	(i) $\{\ \mathtt{*}(w, x) \leq \mathtt{*}(y, z),\ \mathtt{+}(\mathtt{1}, y) \leq w,\ \mathtt{+}(\mathtt{1}, z) \leq x\ \}$ (ii) $\{\ \mathtt{*}(y, y) \neq \mathtt{0},\ y = \mathtt{0}\ \}$			

Table 1
Benchmark Problems

P#	T.	Literals (Framed Literals are Important for Lemma Speculation)	S.L.
1	S	$\mathtt{+}(\mathtt{1}, L) \leq \mathtt{+}(K, \mathtt{MAX}(A)),\ \mathtt{+}(\mathtt{1}, \mathtt{MIN}(A)) \leq L,\ K \leq \mathtt{0},\ \boxed{A = \mathtt{nil}}$	—
	E	$L \leq \mathtt{MAX}(A),\ \boxed{\mathtt{MIN}(A) \leq \mathtt{MAX}(A)},\ \mathtt{+}(\mathtt{1}, \mathtt{MIN}(A)) \leq \mathtt{+}(K, \mathtt{MAX}(A)),\ \ldots$	(i)
2	S	$\mathtt{+}(I, \mathtt{DELTA1}(PAT, LP, C)) \leq MAXINT,\ \mathtt{+}(\mathtt{1}, MAXINT) \leq \mathtt{+}(LP, LT),\ \mathtt{+}(\mathtt{1}, LT) \leq I$	—
	E	$\mathtt{+}(LT, \mathtt{DELTA1}(PAT, LP, C)) \leq MAXINT,\ \mathtt{+}(MAXINT, \mathtt{DELTA1}(PAT, LP, C)) \leq \mathtt{+}(\mathtt{*}(\mathtt{2}, LP), LT),$ $\mathtt{+}(\mathtt{1}, MAXINT) \leq \mathtt{+}(I, LP),\ \boxed{\mathtt{DELTA1}(PAT, LP, C) \leq LP},\ \mathtt{+}(I, \mathtt{DELTA1}(PAT, LP, C)) \leq \mathtt{+}(LP, LT),\ \ldots$	(i)
3	S	$\mathtt{+}(\mathtt{1}, \mathtt{+}(W, \mathtt{LEN}(\mathtt{DEL}(Z, A)))) \leq \mathtt{+}(K, V),\ \boxed{\mathtt{MEMB}(Z, A) \neq \mathtt{true}},\ \mathtt{+}(\mathtt{1}, K) \leq \mathtt{+}(W, \mathtt{LEN}(A))$	—
	E	$\boxed{\mathtt{+}(\mathtt{1}, \mathtt{LEN}(\mathtt{DEL}(Z, A))) \leq \mathtt{+}(V, \mathtt{LEN}(A))},\ \ldots$	(i)
4	S	$\mathtt{+}(\mathtt{1}, \mathtt{*}(\mathtt{MS}(a), \mathtt{MS}(a))) \leq \mathtt{+}(\mathtt{*}(\mathtt{2}, \mathtt{*}(\mathtt{MS}(a), \mathtt{*}(\mathtt{MS}(a), \mathtt{MS}(b)))), \mathtt{*}(\mathtt{MS}(a), \mathtt{*}(\mathtt{MS}(a), \mathtt{*}(\mathtt{MS}(a), \mathtt{MS}(a)))))$	(i)
	S	$\mathtt{*}(\mathtt{MS}(a), \mathtt{*}(\mathtt{MS}(a), \mathtt{MS}(a))) \neq \mathtt{*}(\mathtt{MS}(a), \mathtt{MS}(a)),\ \boxed{\mathtt{*}(\mathtt{MS}(a), \mathtt{MS}(b)) \neq \mathtt{0}}$	(ii)
5	S	$\boxed{\mathtt{max}(x, y) \neq x},\ \mathtt{+}(\mathtt{1}, z) \leq \mathtt{+}(y, \mathtt{g}(x)),\ \mathtt{p}(x) \neq \mathtt{true},\ \mathtt{+}(\mathtt{1}, \mathtt{f}(x)) \leq z,\ \boxed{\mathtt{min}(x, y) \leq \mathtt{0}}$	(ii)
	S	$\mathtt{max}(x, y) \neq x,\ \mathtt{+}(\mathtt{1}, z) \leq \mathtt{+}(y, \mathtt{g}(x)),\ \boxed{\mathtt{p}(x) \neq \mathtt{true}},\ \mathtt{+}(\mathtt{1}, \mathtt{f}(x)) \leq z,\ y \leq \mathtt{0}$	—
	E	$z \leq \mathtt{g}(x),\ \boxed{\mathtt{f}(x) \leq \mathtt{g}(x)},\ \mathtt{+}(\mathtt{1}, \mathtt{f}(x)) \leq \mathtt{+}(y, \mathtt{g}(x)),\ \ldots$	(i)
6	S	$\boxed{\mathtt{+}(\mathtt{1}, y) \leq x},\ \boxed{\mathtt{*}(\mathtt{2}, \mathtt{*}(y, y)) \neq \mathtt{*}(x, x)},\ y = \mathtt{0}$	(i)
	S	$\boxed{\mathtt{*}(y, y) \neq \mathtt{0}},\ \mathtt{+}(\mathtt{1}, y) \leq x,\ \mathtt{0} \neq \mathtt{*}(x, x),\ \boxed{y = \mathtt{0}}$	(ii)

Table 2
Speculation of Auxiliary Lemmas

detailed proof log. From the results in Table 1, we can see that our integration scheme is competitive.

Next, we want to investigate how our close integration into QuodLibet supports the speculation of auxiliary lemmas. Thus, we consider the problems from Table 1 once again but without any auxiliary lemmas. We sketch the process of deriving these lemmas with QuodLibet in Table 2. For each problem, we list the tactics (T.) we call: tactic `simplify` is represented with `S`, tactic `leq-var-elim` with `E`. The literals of the resulting subgoal are given in the next column. For tactic `leq-var-elim` we only display the new literals;

the dots stand for the literals after the last execution of tactic `simplify` given in the previous line. Literals that are used to speculate a lemma are framed. The last column contains the speculated lemmas (S.L.) w.r.t. Table 1. The number of literals measures the complexity of the goal: The more literals are present, the more difficult it is to find the important literals, and thus an auxiliary lemma. We believe that their identification has to be done with human expertise. We assume that this task is rather easy if the auxiliary lemma of the considered problem in Table 1 is a subformula of the resulting subgoal clause in Table 2.

The first two problems do not cause any difficulties. After applying both tactics, the auxiliary lemmas are subformulas of the resulting goal clauses. This is, however, not the case if we only call tactic `simplify`. For Problem 1, the derivation of the subgoal can be found in Figure 1 on Page 6. The first important literal can be identified if we look for a literal that contains at least one uninterpreted function symbol but as few extra variables as possible. In this context, by an *extra* variable of a literal we mean a variable that does not occur in a subterm of the literal with an uninterpreted function symbol as toplevel symbol. The last goal node in Figure 1 contains the extra variable L for the first and fifth literal; K for the third and sixth literal; L and K for the fourth literal; and A for the seventh literal. But a lemma that only consists of the second literal $\mathtt{MIN}(A) \leq \mathtt{MAX}(A)$ is not inductively valid. Instead, a human expert has to add the last literal $A = \mathtt{nil}$ to get an inductively valid lemma. For Problem 3, the resulting subgoal contains an additional variable V. This can be eliminated if we use the fact that we only deal with naturals. But at the moment, this is done automatically by tactic `simplify` only if this seems to be advantageous. Nevertheless, if only one lemma is missing, our close integration facilitates the speculation of auxiliary lemmas quite well.

For the remaining problems, two auxiliary lemmas are missing. For Problem 4 and 6, our tactics do not provide any additional information for the first lemma. This is not very surprising since in these examples no variable elimination steps can be performed at all. Note that the first important literal for Problem 6 is introduced by the manual inductive case split. With these two important literals for Problem 6, it is not difficult for a human user with domain knowledge about multiplication to guess the required monotonicity property as auxiliary lemma. To speculate the first auxiliary lemma for Problem 5, an experienced user only needs to know the first important literal and the left-hand side of the second one. Then, the relationship between `max` and `min` is obvious. Note that in the original goal clause presented in Table 1, the first important literal is not present. This complicates the speculation of the required auxiliary lemma for the original goal. Only for Problem 4, the second auxiliary lemma is not a subformula of the considered subgoal clause. But the important literal of this problem suggests an auxiliary lemma that may be used as well.

To conclude, seven of nine lemmas can be speculated easily with our integration scheme. Four of them require an additional call to `leq-var-elim` because `simplify` does not provide enough information. 14 of 42 literals are important for the speculation. In the presented case studies there is no exponential blow-up of the inequalities.

6 Related Work

In the literature, there exist many other decision procedures for PRA (PIA or PNA) besides that of Hodes [9]. But as explained in [7], the efficiency of the decision procedure itself does not matter when using it in an extended theory. In the cited case study with NQTHM, an instantaneous oracle for linear inequalities would reduce the overall runtime by less than 3%. Instead, the interaction between the decision procedure and the theorem prover is more important. Therefore, we have not investigated other decision procedures.

Our work is essentially influenced by [7], where Boyer & Moore describe many helpful heuristics to restrict the search space. But their description is sometimes hard to read as they use internal data structures special to their theorem prover NQTHM. Instead, we use inference rules for the incorporation.

In [14], Kapur & Nie extend the approach from [7] at least in two ways: They do not convert equalities into two inequalities but use equalities directly to eliminate variables. This handling of equality information is more efficient than that in [7]. Furthermore, they extend the decision procedure for PRA in such a way that it is also a decision procedure for PIA and PNA: At first, the closure under the variable elimination steps is calculated. If no unsatisfiable inequality can be found, then there exists a rational solution which is determined by the inequalities. This solution has to be checked for a solution over integers (or naturals). We have implemented three inference rules to handle these improvements (see Section 4.1.4). Our automatic proof control does not realize the whole decision procedure for PNA as this may result in a huge case distinction. Instead, we use this method only if the intervals that have to be checked are small. Otherwise, we have to use other proof techniques e.g. induction.

Janicic, Bundy & Green [11] formalize and generalize the approach from [7]. Their presentation is independent from the theory and the decision procedure to be used. Instead, they assume that the decision procedure can be divided into two steps: the elimination of variables and a check on ground instances. This approach is further developed in [10] taking into account the combination of decision procedures. Our fragmentation of the decision procedure into inference rules is influenced by [11]. As a third major step of a decision procedure, we identify the normalization of literals.

The approach proposed by Armando & Ranise [2] is similar to that in [11]. But they combine the decision procedure more closely with rewriting. They pose additional demands on the rewriting mechanism and the decision procedure. This allows them to prove soundness and termination properties for their approach. The soundness of our approach is guaranteed by local properties of our inference rules. Termination properties may be proved by constraining the control in a similar way as in [2].

In [5], Berezin, Ganesh & Dill also propose an inference system for their integration scheme. Our inference rules are on a higher level. Therefore, they can be easier applied manually. Although their inference rules can be easier checked with an external proof checker this is also possible for ours.

[16] and [8] contain proposals for the speculation of rewrite rules. For nonlinear equalities, Armando, Rusinowitch & Stratulat propose the use of Buchberger's algorithm based on Gröbner basis in [3]. In [1] and [12], approaches are described to extend the integration of linear arithmetic to nonlinear arithmetic. These extensions almost only effect the heuristics to choose or speculate lemmas for the augmentation mechanism. Therefore, their integration into our approach should be easy. This is subject of further research.

7 Conclusion

In this paper, we have presented a close integration scheme for the incorporation of a decision procedure for PNA into the inductive theorem prover QuodLibet. Our approach strictly separates the logic engine given by an inference system from its control. This allows us to easily prove the soundness of our integration. Furthermore, it enables the creation of detailed proof objects that can be easily checked by implementing a proof checker based on our inference rules. In spite of these detailed proofs, our case studies illustrate that our integration scheme is competitive with other approaches as e.g. `EPM` and `CCR(X)`. We have also demonstrated how our approach can be used to facilitate the speculation of auxiliary lemmas. For this, we use a special purpose tactic that does not obey some of the heuristic restrictions inherently used in other approaches. The evaluation and improvement of our heuristics is ongoing work. Our flexible integration scheme supports us in this task.

Acknowledgement

I would like to thank R. Rondot for working out the technical details of the inference rules in his diploma thesis [18]. In this paper, we present our ideas underlying [18], and describe an improved control. Furthermore, I would like to thank J. Avenhaus, B. Löchner, C.-P. Wirth, and the anonymous referees for helpful comments on earlier drafts of this paper.

References

[1] A. Armando and S. Ranise. *A Practical Extension Mechanism for Decision Procedures: the Case Study of Universal Presburger Arithmetic.* J.UCS, 7(2):124–140, February 2001.

[2] A. Armando and S. Ranise. *Constraint contextual rewriting.* J. Symb. Comp., 36(1–2):193–216, 2003.

[3] A. Armando, M. Rusinowitch, and S. Stratulat. *Incorporating decision procedures in implicit induction.* J. Symb. Comput., 34(4):241–258, 2002.

[4] J. Avenhaus, U. Kühler, T. Schmidt-Samoa, and C.-P. Wirth. *How to prove inductive theorems? QuodLibet!* 19th CADE, LNAI 2741, pp. 328–333, 2003.

[5] S. Berezin, V. Ganesh, and D. L. Dill. *An online proof-producing decision procedure for mixed-integer linear arithmetic.* 9th TACAS, pp. 521–536, 2003.

[6] R. S. Boyer and J S. Moore. *A Computational Logic Handbook.* Acad. Press, 1988.

[7] R. S. Boyer and J S. Moore. *Integrating decision procedures into heuristic theorem provers: a case study of linear arithmetic.* Machine intelligence 11, pp. 83–124. Oxford University Press, 1988.

[8] J. Giesl and D. Kapur. *Deciding inductive validity of equations.* 19th CADE, LNAI 2741, pp. 17–31. Springer, 2003.

[9] L. Hodes. *Solving problems by formula manipulation in logic and linear inequalities.* 2nd IJCAI, pp. 553–559, London, 1971.

[10] P. Janicic and A. Bundy. *A general setting for combining and integrating decision procedures into theorem provers.* J. Autom. Reas., 28:257–305, 2002.

[11] P. Janicic, A. Bundy, and I. Green. *A framework for the flexible integration of a class of decision procedures into theorem provers.* 16th CADE, LNAI 1632, pp. 127–141, 1999.

[12] W. A. Hunt Jr., R. B. Krug, and J S. Moore. *Linear and nonlinear arithmetic in ACL2.* CHARME, LNCS 2860, pp. 319–333. Springer, 2003.

[13] W. A. Hunt Jr., R. B. Krug, and J S. Moore. *Integrating nonlinear arithmetic into ACL2.* ACL2-2004.

[14] D. Kapur and X. Nie. *Reasoning about numbers in Tecton.* 8th ISMIS, LNCS 869, pp. 57–70. Springer, 1994.

[15] M. Kaufmann, and P. Manolios, J S. Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.

[16] D. Kapur and M. Subramaniam. *Automatic generation of simple lemmas from recursive definitions using decision procedures - preliminary report.* 8th ASIAN, pp. 125–145, 2003.

[17] G. Nelson and D. C. Oppen. *Simplification by cooperating decision procedures.* ACM Trans. Program. Lang. Syst., 1(2):245–257, 1979.

[18] R. Rondot. *Integration von Entscheidungsverfahren in den induktiven Theorembeweiser QuodLibet.* Diploma thesis (in German), FB Informatik, TU Kaiserslautern, Germany, 2004. www-avenhaus.informatik.uni-kl.de/quodlibet/LADARondot.ps.gz

[19] R. E. Shostak. *Deciding combinations of theories.* J. ACM, 31(1):1–12, 1984.