Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

Linear Quantifier Elimination

Tobias Nipkow

Received: date / Accepted: date

Abstract This paper presents verified quantifier elimination procedures for dense linear orders (two of them novel), for real and for integer linear arithmetic. All procedures are defined and verified in the theorem prover Isabelle/HOL, are executable and can be applied to HOL formulae themselves (by reflection). The formalization of the different theories is highly modular.

1 Introduction

This paper is about the concise implementation of *quantifier elimination* (QE) procedures (QEPs) for linear arithmetics. QE is a venerable logical technique which yields decision procedures if ground atoms are decidable. The focus of our work is the compact implementation of QEPs (for linear arithmetics) inside a theorem prover. All our QEPs have been defined and verified in Isabelle/HOL [22]. We do not discuss these formal proofs here. They are detailed, mostly structured and available online at afp.sf.net, together with the QEPs themselves. Because the informal proofs of these QEPs can be found in the literature, they need not be discussed either. The exception are our novel QEPs, for which informal correctness proofs are given.

The main contributions of this paper are:

- Two novel QEPs for dense linear orders (DLO) inspired by QEPs for linear real arithmetic.
- Presentation of 8 verified implementations of QEPs: three for DLO, three for linear real arithmetic and two for Presburger arithmetic. We show everything but the most trivial details, providing reference implementations and convincing the reader that nothing has been swept under the carpet.
- Extremely compact formalizations due to the almost excessive use of lists and list comprehensions.
- A common reusable QE framework using Isabelle's structuring facility of locales, thus factoring out the common parts of the different QEPs.

T. Nipkow
Institut für Informatik, Technische Universität München
E-mail: www.in.tum.de/~nipkow

This paper is a contribution to the growing body of verified theorem proving algorithms. In spirit it is close to Harrison's book [13] which presents all algorithms in OCaml.

Why this obsession with executable and verified QEPs? The context of this research is the question of how to implement trustworthy and efficient decision procedures in foundational theorem provers, i.e. without having to trust an external oracle. *Reflection*, originally proposed by Boyer and Moore [2] and used to great effect in systems like Coq (e.g. [10]) and Isabelle (e.g. [4]) has become a standard approach. Suffice it to say that we follow this approach, too, and that all the algorithms in this paper can be used directly on formulae in Isabelle — details can be found elsewhere (e.g. [21]).

It should be emphasized that the presentation is streamlined for succinctness. In particular, we always restrict attention to two of the four relations $=$, $<$, $\leq$, $\neq$. For example, in DLOs it suffices to consider $=$ and $<$ because $x \leq y$ is equivalent with $x < y \vee x = y$ and $x \neq y$ is equivalent with $x < y \vee y < x$. An efficient implementation would always work with all four relations. The corresponding generalization of our code is straightforward.

The paper is structured as follows. In §3 we describe a HOL model of logical formulae parameterized by a language of atoms and present two generic QEPs, one based on NNF, one on DNF. Both are parameterized by a QEP for a single quantifier. The remaining sections present a succession of single-quantifier QEPs for different linear theories: each section starts with a simple DNF-based algorithm and proceeds to more efficient NNF-based ones.

2 Basic Notation

HOL conforms largely to everyday mathematical notation. This section introduces further non-standard notation and in particular a few basic data types with their primitive operations.

The types of truth values, natural numbers, integers and reals are called *bool*, *nat*, *int* and *real*. The space of total functions is denoted by $\Rightarrow$. Type variables are denoted by α, β, etc. The notation $t{::}\tau$ means that term t has type τ.

Sets over type α, type α *set*, follow the usual mathematical conventions.

Lists over type α, type α *list*, come with the empty list $[]$, the infix constructor $\cdot$, the infix @ that appends two lists, and the conversion function *set* from lists to sets. Variable names ending in s usually stand for lists. In addition to the standard functions *map* and *filter*, Isabelle/HOL also supports Haskell-style list comprehension notation. For example, in Haskell `[2*x | x <- xs, x > 0]` denotes the list of all `2*x`, where `x` iterates over all elements of the list `xs` such that `x > 0`. Instead of `[e | x <- xs, ...]` as in Haskell we write $[e.\ x \leftarrow xs, \ldots]$, and $[x{\leftarrow}xs.\ \ldots]$ is short for $[x.\ x{\leftarrow}xs, \ldots]$.

Note that $=$ on type *bool* means "iff" and that *If P then Q* is synonymous with $P \Longrightarrow Q$.

During informal explanations we often switch to everyday mathematical notation where (a,b) can be a pair or an open interval.

3 Logic

Formulae are defined as a recursive datatype with a parameter type α of atoms:

$$\begin{array}{lcl} \textbf{datatype}\ \alpha\ \mathit{fm} & = & \top \mid \bot \mid A\ \alpha \\ & \mid & (\alpha\ \mathit{fm}) \wedge (\alpha\ \mathit{fm}) \mid (\alpha\ \mathit{fm}) \vee (\alpha\ \mathit{fm}) \mid \neg\ (\alpha\ \mathit{fm}) \mid \exists\ (\alpha\ \mathit{fm}) \end{array}$$

The **boldface** symbols $\wedge$, $\vee$, $\neg$ and $\exists$ are ordinary constructors chosen to resemble the logical operators they represent. Constructor A encloses atoms. The type of atoms is left open by making it a parameter α. Variables are represented by de Bruijn indices: quantifiers do not explicitly mention the name of the variable being bound because that is implicit. For example, $\exists\ (\exists \dots \mathit{0} \dots \mathit{1} \dots)$ represents a formula $\exists x_1.\exists x_0.\ \dots x_0 \dots x_1 \dots$. Note that the only place where variables can appear is inside atoms. The only distinction between free and bound variables is that the index of a free variable is larger than the number of enclosing $\exists$.

3.1 Auxiliary Functions

The constructors $\wedge$, $\vee$ and $\neg$ have optimized ("short-circuit") versions *and*, *or* and *neg*:

$$\begin{array}{lll}
\mathit{and}\ \top\ \varphi = \varphi & \mathit{or}\ \top\ \varphi = \top & \mathit{neg}\ \top = \bot \\
\mathit{and}\ \varphi\ \top = \varphi & \mathit{or}\ \varphi\ \top = \top & \mathit{neg}\ \bot = \top \\
\mathit{and}\ \bot\ \varphi = \bot & \mathit{or}\ \bot\ \varphi = \varphi & \mathit{neg}\ \varphi = \neg\ \varphi \\
\mathit{and}\ \varphi\ \bot = \bot & \mathit{or}\ \varphi\ \bot = \varphi & \\
\mathit{and}\ \varphi_1\ \varphi_2 = (\varphi_1 \wedge \varphi_2) & \mathit{or}\ \varphi_1\ \varphi_2 = (\varphi_1 \vee \varphi_2) &
\end{array}$$

Conjunctions and disjunctions of lists of formulae are easily defined:

$$\begin{array}{l}
\mathit{list\text{-}conj}\ [\varphi_1,\dots,\varphi_n] = \mathit{and}\ \varphi_1\ (\mathit{and}\ \dots\ \varphi_n) \\
\mathit{list\text{-}disj}\ [\varphi_1,\dots,\varphi_n] = \mathit{or}\ \varphi_1\ (\mathit{or}\ \dots\ \varphi_n)
\end{array}$$

For convenience the following abbreviation is also introduced:

$$\mathit{Disj}\ \mathit{us}\ f \equiv \mathit{list\text{-}disj}\ (\mathit{map}\ f\ \mathit{us})$$

More interesting is the conversion to DNF:

$$\mathit{dnf} :: \alpha\ \mathit{fm} \Rightarrow \alpha\ \mathit{list}\ \mathit{list}$$

$$\begin{array}{lcl}
\mathit{dnf}\ \top & = & [[]] \\
\mathit{dnf}\ \bot & = & [] \\
\mathit{dnf}\ (A\ \varphi) & = & [[\varphi]] \\
\mathit{dnf}\ (\varphi_1 \vee \varphi_2) & = & \mathit{dnf}\ \varphi_1\ @\ \mathit{dnf}\ \varphi_2 \\
\mathit{dnf}\ (\varphi_1 \wedge \varphi_2) & = & [d_1\ @\ d_2.\ d_1 \leftarrow \mathit{dnf}\ \varphi_1,\ d_2 \leftarrow \mathit{dnf}\ \varphi_2]
\end{array}$$

The resulting list of lists represents the disjunction of conjunctions of atoms. Working with lists rather than type *fm* has the advantage of a well-developed library and notation.

Note that *dnf* assumes that its argument contains neither quantifiers nor negations. Most of our work will be concerned with quantifier-free formulae where all negations have not just been pushed right in front of atoms but actually into them. This is easy for linear orders because $\neg(x < y)$ is equivalent with $y \leq x$. This conversion will be described later on because it depends on the type of atoms. The (trivial to define) predicates

$$\mathit{qfree},\ \mathit{nqfree} :: \alpha\ \mathit{fm} \Rightarrow \mathit{bool}$$

check whether their argument is free of quantifiers (*qfree*), and free of negations and quantifiers (*nqfree*).

There are also two mapping functionals

$$\begin{array}{l}
\mathit{map}_{\mathit{fm}}\ \ :: (\alpha \Rightarrow \beta) \Rightarrow \alpha\ \mathit{fm} \Rightarrow \beta\ \mathit{fm} \\
\mathit{amap}_{\mathit{fm}} :: (\alpha \Rightarrow \beta\ \mathit{fm}) \Rightarrow \alpha\ \mathit{fm} \Rightarrow \beta\ \mathit{fm}
\end{array}$$

where $\mathit{map}_{\mathit{fm}}\ f$ is the canonical one that simply replaces $A\ a$ by $A\ (f\ a)$, whereas $\mathit{amap}_{\mathit{fm}}$ may also simplify the formula via *and*, *or* and *neg*:

$$\begin{aligned}
&\mathit{amap}_{\mathit{fm}}\ h\ \top &&= \top\\
&\mathit{amap}_{\mathit{fm}}\ h\ \bot &&= \bot\\
&\mathit{amap}_{\mathit{fm}}\ h\ (A\ a) &&= h\ a\\
&\mathit{amap}_{\mathit{fm}}\ h\ (\varphi_1 \wedge \varphi_2) &&= \mathit{and}\ (\mathit{amap}_{\mathit{fm}}\ h\ \varphi_1)\ (\mathit{amap}_{\mathit{fm}}\ h\ \varphi_2)\\
&\mathit{amap}_{\mathit{fm}}\ h\ (\varphi_1 \vee \varphi_2) &&= \mathit{or}\ (\mathit{amap}_{\mathit{fm}}\ h\ \varphi_1)\ (\mathit{amap}_{\mathit{fm}}\ h\ \varphi_2)\\
&\mathit{amap}_{\mathit{fm}}\ h\ (\neg\ \varphi) &&= \mathit{neg}\ (\mathit{amap}_{\mathit{fm}}\ h\ \varphi)
\end{aligned}$$

Both mapping functionals are only defined and needed for *qfree* formulae.

The set of atoms in a formula is computed by the function $\mathit{atoms} :: \alpha\ \mathit{fm} \Rightarrow \alpha\ \mathit{set}$.

3.2 Interpretation

The interpretation or semantics of a *fm* is defined via the obvious homomorphic mapping to an HOL formula: $\wedge$ becomes $\wedge$, $\vee$ becomes $\vee$, etc. The interpretation of atoms is a parameter of this mapping. Atoms may refer to variables and are thus interpreted w.r.t. a valuation. Since variables are represented as natural numbers, the valuation is naturally represented as a list: variable i refers to the ith entry in the list (starting with 0). This leads to the following interpretation function $\mathit{interpret} :: (\alpha \Rightarrow \beta\ \mathit{list} \Rightarrow \mathit{bool}) \Rightarrow \alpha\ \mathit{fm} \Rightarrow \beta\ \mathit{list} \Rightarrow \mathit{bool}$:

$$\begin{aligned}
&\mathit{interpret}\ h\ \top\ \mathit{xs} &&= \mathit{True}\\
&\mathit{interpret}\ h\ \bot\ \mathit{xs} &&= \mathit{False}\\
&\mathit{interpret}\ h\ (A\ a)\ \mathit{xs} &&= h\ a\ \mathit{xs}\\
&\mathit{interpret}\ h\ (\varphi_1 \wedge \varphi_2)\ \mathit{xs} &&= \mathit{interpret}\ h\ \varphi_1\ \mathit{xs} \wedge \mathit{interpret}\ h\ \varphi_2\ \mathit{xs}\\
&\mathit{interpret}\ h\ (\varphi_1 \vee \varphi_2)\ \mathit{xs} &&= \mathit{interpret}\ h\ \varphi_1\ \mathit{xs} \vee \mathit{interpret}\ h\ \varphi_2\ \mathit{xs}\\
&\mathit{interpret}\ h\ (\neg\ \varphi)\ \mathit{xs} &&= \neg\ \mathit{interpret}\ h\ \varphi\ \mathit{xs}\\
&\mathit{interpret}\ h\ (\exists\ \varphi)\ \mathit{xs} &&= \exists x.\ \mathit{interpret}\ h\ \varphi\ (x \cdot \mathit{xs})
\end{aligned}$$

In the equation for $\exists$ the value of the bound variable x is added at the front of the valuation. De Bruijn indexing ensures that in the body 0 refers to x and $i+1$ refers to bound variable i further up.

3.3 Atoms

Atoms are more than a type parameter α. They come with an *interpretation* (their semantics), and a few other specific functions. These functions are also parameters of the generic part of quantifier elimination. Thus the further development will be like a module parameterized with the type of atoms and some functions on atoms. These parameters will be instantiated later on when applying the framework to various linear arithmetics.

In Isabelle this parameterization is achieved by means of a **locale** [1], a named context of types, functions and assumptions about them. We call this context *ATOM*. It provides the following functions

$$\begin{aligned}
&I_a &&:: \alpha \Rightarrow \beta\ \mathit{list} \Rightarrow \mathit{bool}\\
&\mathit{aneg} &&:: \alpha \Rightarrow \alpha\ \mathit{fm}\\
&\mathit{depends}_0 &&:: \alpha \Rightarrow \mathit{bool}\\
&\mathit{decr} &&:: \alpha \Rightarrow \alpha
\end{aligned}$$

with the following intended meaning:

$I_a\ a\ \mathit{xs}$ is the interpretation of atom a w.r.t. valuation *xs*, where variable i (note $i :: \mathit{nat}$ because of de Bruijn) is assigned the ith element of *xs*.

aneg negates an atom. It returns a formula which should be free of negations. This is strictly for convenience: it means we can eliminate all negations from a formula. In the worst case we would have to introduce negated versions of all atoms, but in the case of linear orders this is not necessary because we can turn, for example, $\neg(x < y)$ into $(y < x) \vee (y = x)$.

$depends_0$ a checks if atom *a* contains (depends on) variable 0.

decr a decrements every variable in *a* by 1.

Within context *ATOM* we introduce the abbreviation $I \equiv \mathit{interpret}\ I_a$. The assumptions on the parameters of *ATOM* can now be stated quite succinctly:

$$\begin{aligned}
&I\ (\mathit{aneg}\ a)\ \mathit{xs} = (\neg\ I_a\ a\ \mathit{xs})\\
&\mathit{nqfree}\ (\mathit{aneg}\ a)\\
&\neg\ \mathit{depends}_0\ a \Longrightarrow I_a\ a\ (x \cdot \mathit{xs}) = I_a\ (\mathit{decr}\ a)\ \mathit{xs}
\end{aligned}$$

Function *aneg* must return a quantifier and negation-free formula whose interpretation is the negation of the input. And when interpreting an atom not containing variable 0 we can drop the head of the valuation and decrement the variables without changing the interpretation.

These assumptions must be discharged when the locale is instantiated. We do not show this in the text because the proofs are straightforward in all cases.

In the context of *ATOM* we define two auxiliary functions: $\mathit{atoms}_0\ \varphi$ computes the list of all atoms in φ that depend on variable 0. The *negation normal form* (NNF) of a *qfree* formula is defined in the customary manner by pushing negations inwards. We show only a few representative equations:

$$\begin{aligned}
\mathit{nnf}\ (\neg\ (A\ a)) &= \mathit{aneg}\ a\\
\mathit{nnf}\ (\varphi_1 \vee \varphi_2) &= \mathit{nnf}\ \varphi_1 \vee \mathit{nnf}\ \varphi_2\\
\mathit{nnf}\ (\neg\ (\varphi_1 \vee \varphi_2)) &= \mathit{nnf}\ (\neg\ \varphi_1) \wedge \mathit{nnf}\ (\neg\ \varphi_2)\\
\mathit{nnf}\ (\neg\ (\varphi_1 \wedge \varphi_2)) &= \mathit{nnf}\ (\neg\ \varphi_1) \vee \mathit{nnf}\ (\neg\ \varphi_2)
\end{aligned}$$

The first equation differs from the usual definition and gets rid of negations altogether — see the explanation of *aneg* above.

3.4 Quantifier Elimination

The elimination of all quantifiers from a formula is achieved by eliminating them one by one in a bottom-up fashion. Thus each step needs to deal merely with the elimination of a single quantifier in front of a quantifier-free formula. This step is theory-dependent and hard. The lifting to arbitrary formulae is simple and can be done once and for all. We assume we are given a function $\mathit{qe} :: \alpha\ \mathit{fm} \Rightarrow \alpha\ \mathit{fm}$ for the elimination of a single $\exists$, i.e. $I\ (\mathit{qe}\ \varphi) = I\ (\exists\ \varphi)$ if $\mathit{qfree}\ \varphi$. Note that *qe* is not applied to $\exists\ \varphi$ but just to φ, $\exists$ remains implicit. Lifting *qe* is straightforward:

$$\mathit{lift\text{-}nnf\text{-}qe} :: (\alpha\ \mathit{fm} \Rightarrow \alpha\ \mathit{fm}) \Rightarrow \alpha\ \mathit{fm} \Rightarrow \alpha\ \mathit{fm}$$

$$\begin{aligned}
\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ (\varphi_1 \wedge \varphi_2) &= \mathit{and}\ (\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ \varphi_1)\ (\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ \varphi_2)\\
\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ (\varphi_1 \vee \varphi_2) &= \mathit{or}\ (\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ \varphi_1)\ (\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ \varphi_2)\\
\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ (\neg\ \varphi) &= \mathit{neg}\ (\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ \varphi)\\
\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ (\exists\ \varphi) &= \mathit{qe}\ (\mathit{nnf}\ (\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ \varphi))\\
\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ \varphi &= \varphi
\end{aligned}$$

Note that *qe* is called with an argument already in NNF.

3.4.1 DNF

We can go even further and put the argument of *qe* into DNF. This can lead to non-elementary complexity but allows particularly straightforward algorithms because then we can push existential quantifiers through disjunctions as follows (using customary logical notation):

$$(\exists x. \bigvee_i \bigwedge_j a_{ij}) \;=\; (\bigvee_i \exists x. \bigwedge_j a_{ij})$$

where a_{ij} are the atoms of the DNF. Thus *qe* can be applied directly to a conjunction of atoms. Using

$$(\exists x.\, A \wedge B(x)) \;=\; (A \wedge (\exists x.\, B(x)))$$

where A does not depend on x, we can push the quantifier right in front of a conjunction of atoms all of which depend on x. This simplifies matters for *qe* as much as possible.

Now we look at the formalization of this second lifting procedure:

$\textit{lift-dnf-qe} :: (\alpha\ \textit{list} \Rightarrow \alpha\ \textit{fm}) \Rightarrow \alpha\ \textit{fm} \Rightarrow \alpha\ \textit{fm}$

Because we represent the DNF via lists of lists of atoms, the first argument of *lift-dnf-qe* takes a list rather than a conjunction of atoms.

The separation of a list (conjunction) of atoms into those that do contain 0 and those that do not, and the application of *qe* to the former is performed by an auxiliary function:

```
$\textit{qelim qe as} = (\mathsf{let}\ \textit{qf} = \textit{qe}\ [a \leftarrow \textit{as}.\ \textit{depends}_0\ a];$
                      $\textit{indep} = [A(\textit{decr}\ a).\ a \leftarrow \textit{as},\ \neg\ \textit{depends}_0\ a]$
                  $\mathsf{in}\ \textit{and qf}\ (\textit{list-conj indep}))$
```

Because the innermost quantifier is eliminated, all references to other quantifiers need to be decremented. For the atoms independent of the innermost quantifier this needs to be done explicitly, for the other atoms this must happen inside *qe*.

The main function *lift-dnf-qe* recurses down the formula (we omit the obvious equations) until it finds an $\exists\ \varphi$, removes the quantifiers from φ, puts the result into NNF and DNF, and applies *qelim qe* to each disjunct:

$\textit{lift-dnf-qe qe}\ (\exists\ \varphi) = \textit{Disj}\ (\textit{dnf}\ (\textit{nnf}\ (\textit{lift-dnf-qe qe}\ \varphi)))\ (\textit{qelim qe})$

3.4.2 Equality

We can generalize quantifier elimination via DNF even further based on the predicate calculus law

$$(\exists x.\, x = t \wedge \phi) = \phi[t/x] \tag{1}$$

provided x does not occur in t. In some theories this enables us to remove equalities completely: in linear real arithmetic, any equation containing variable x is either independent of the value of x (e.g. $x = x$ or $x = x + 1$) or can be brought into the form $x = t$ with x not in t. But even if one cannot remove all equalities, as in most non-linear theories, it is useful to deal with $x = t$ separately for obvious efficiency reasons. Hence we extend locale *ATOM* to locale *ATOM-EQ* containing the following additional parameters

$\textit{solvable}_0$	$:: \alpha \Rightarrow \textit{bool}$
$\textit{trivial}$	$:: \alpha \Rightarrow \textit{bool}$
$\textit{subst}_0$	$:: \alpha \Rightarrow \alpha \Rightarrow \alpha$

with the following intended meaning expressed by the corresponding assumptions:

- For solvable atoms, any valuation of the variables > 0 can be extended to a satisfying valuation: $\mathit{solvable}_0\ \mathit{eq} \Longrightarrow \exists x.\ I_a\ \mathit{eq}\ (x \cdot \mathit{xs})$.
- Trivial atoms satisfy every valuation: $\mathit{trivial}\ \mathit{eq} \Longrightarrow I_a\ \mathit{eq}\ \mathit{xs}$.
- Function subst_0 substitutes its first argument, a solvable equality, into its second argument. This is expressed by requiring that subst_0 satisfies the substitution lemma under certain conditions: *If* $\mathit{solvable}_0\ \mathit{eq}$ *and* $\neg\ \mathit{trivial}\ \mathit{eq}$ *and* $I_a\ \mathit{eq}\ (x \cdot \mathit{xs})$ *and* $\mathit{depends}_0\ a$ *then* $I_a\ (\mathit{subst}_0\ \mathit{eq}\ a)\ \mathit{xs} = I_a\ a\ (x \cdot \mathit{xs})$. And substituting a solvable atom into itself results in a trivial atom: $\mathit{solvable}_0\ \mathit{eq} \Longrightarrow \mathit{trivial}\ (\mathit{subst}_0\ \mathit{eq}\ \mathit{eq})$.

Now we can define a lifting function that takes a quantifier elimination procedure *qe* on lists of atoms and extends it to lists containing trivial atoms (by filtering them out) and solvable atoms (by substituting them in):

```
lift-eq-qe qe as =
(let as = [a←as. ¬ trivial a]
 in case [a←as. solvable₀ a] of
      [] ⇒ qe as
    | eq · eqs ⇒ (let ineqs = [a←as. ¬ solvable₀ a]
                  in list-conj (map (A ∘ subst₀ eq) (eqs @ ineqs))))
```

3.5 Correctness

Correctness of these lifting functions is roughly expressed as follows: if *qe* eliminates one existential while preserving the interpretation, then *lift qe* eliminates all quantifiers while preserving the interpretation.

For compactness we employ a set theoretic language for expressing properties of functions: $A \rightarrow B$ is the set of functions from A to B. Note that sets and predicates are identified in HOL.

First we look at *lift-nnf-qe*. Elimination of all quantifiers is easy:

Lemma 1 *If* $\mathit{qe} \in \mathit{nqfree} \rightarrow \mathit{qfree}$ *then* $\mathit{qfree}\ (\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ \varphi)$.

Preservation of the interpretation is slightly more involved:

Lemma 2 *If* $\mathit{qe} \in \mathit{nqfree} \rightarrow \mathit{qfree}$ *and* $\forall \varphi {\in} \mathit{nqfree}.\ \forall \mathit{xs}.\ I\ (\mathit{qe}\ \varphi)\ \mathit{xs} = (\exists x.\ I\ \varphi\ (x \cdot \mathit{xs}))$ *then* $I\ (\mathit{lift\text{-}nnf\text{-}qe}\ \mathit{qe}\ \varphi)\ \mathit{xs} = I\ \varphi\ \mathit{xs}$.

For *lift-dnf-qe* the statements are a bit more involved, but essentially analogous to those for *lift-nnf-qe*. The only difference is that *qe* applies to lists of atoms *as* instead of a formula φ. Function *lists* yields the set of lists over a given set.

Lemma 3 *If* $\mathit{qe} \in \mathit{lists}\ \mathit{depends}_0 \rightarrow \mathit{qfree}$ *then* $\mathit{qfree}\ (\mathit{lift\text{-}dnf\text{-}qe}\ \mathit{qe}\ \varphi)$.

Lemma 4 *If* $\mathit{qe} \in \mathit{lists}\ \mathit{depends}_0 \rightarrow \mathit{qfree}$ *and* $\forall \mathit{as} {\in}\ \mathit{lists}\ \mathit{depends}_0.\ \mathit{is\text{-}dnf\text{-}qe}\ \mathit{qe}\ \mathit{as}$, *then* $I\ (\mathit{lift\text{-}dnf\text{-}qe}\ \mathit{qe}\ \varphi)\ \mathit{xs} = I\ \varphi\ \mathit{xs}$.

where $\mathit{is\text{-}dnf\text{-}qe}\ \mathit{qe}\ \mathit{as} \equiv \forall \mathit{xs}.\ I\ (\mathit{qe}\ \mathit{as})\ \mathit{xs} = (\exists x.\ \forall a {\in} \mathit{set}\ \mathit{as}.\ I_a\ a\ (x \cdot \mathit{xs}))$. The right-hand side is equal to $\exists x.\ I\ (\mathit{list\text{-}conj}\ (\mathit{map}\ A\ \mathit{as}))\ (x \cdot \mathit{xs})$.

Finally we consider the extension to equality. From the assumptions of locale *ATOM-EQ* it is not hard to prove that if *qe* performs quantifier elimination on any list of unsolvable atoms depending on variable 0, then *lift-eq-qe qe* is a quantifier elimination procedure on any list of atoms depending on 0:

Lemma 5 *If* $\forall as \in list(depends_0 \cap -solvable_0).\ is\text{-}dnf\text{-}qe\ qe\ as$ *then* $\forall as \in list\ depends_0.\ is\text{-}dnf\text{-}qe\ (lift\text{-}eq\text{-}qe\ qe)\ as.$

In our instantiations, the unsolvable atoms will be the inequalities ($<$) and *qe* will only need to deal with them; $=$ is taken care of completely by this lifting process.

Finally we compose *lift-dnf-qe* and *lift-eq-qe*

$$lift\text{-}dnfeq\text{-}qe = lift\text{-}dnf\text{-}qe \circ lift\text{-}eq\text{-}qe$$

and obtain a corollary to lemmas 4 and 5:

Corollary 1 *If* $qe \in lists\ depends_0 \to qfree$ *and* $\forall as \in lists(depends_0 \cap -solvable_0).\ is\text{-}dnf\text{-}qe\ qe\ as$ *then* $I\ (lift\text{-}dnfeq\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs.$

In the same manner we obtain

Corollary 2 *If* $qe \in list\ depends_0 \to qfree$ *then* $qfree\ (lift\text{-}dnfeq\text{-}qe\ qe\ \varphi).$

All proofs in this subsection are straightforward inductions using a number of simple additional lemmas.

In the following sections we define a number of quantifier elimination functions called $qe\text{-}f_1$ (for different names f) that eliminate a single $\exists$. In each case

- we have proved that $qe\text{-}f_1$ satisfies the assumptions of lemmas 1 and 2 (or lemmas 3 and 4, or corollaries 1 and 2), with $qe\text{-}f_1$ for qe,
- we define $qe\text{-}f = lift\text{-}nnf\text{-}qe\ qe\text{-}f_1$ (or via *lift-dnf-qe*, or via *lift-dnfeq-qe*),
- we obtain $qfree\ (qe\text{-}f\ \varphi)$ and $I\ (qe\text{-}f\ \varphi)\ xs = I\ \varphi\ xs$ as corollaries of the above lemmas and corollaries.

Because of this uniformity and because the correctness proofs are either discussed informally beforehand or are well-known from the literature, we suppress all of this in the presentation. Thus it may look as if we merely present code, but the proofs are all there.

4 Dense Linear Orders

The theory of dense linear orders (without endpoints) is an extension of the theory of linear orders with the axioms

$$x < z \Longrightarrow \exists y.\ x < y \wedge y < z \qquad \exists u.\ x < u \qquad \exists l.\ l < x$$

It is the canonical example of quantifier elimination [14]. The equivalence $(\exists y.\ x < y \wedge y < z) = (x < z)$ is an easy consequence of the axioms and the essence of Fourier's elimination method. It generalizes to arbitrary conjunctions of inequalities containing the quantified variable: partition the inequalities into those of the form $l_i < x$ and those of the form $x < u_j$ and combine all pairs:

$$(\exists x.\ (\bigwedge_i l_i < x) \wedge (\bigwedge_j x < u_j)) \ = \ (\bigwedge_{ij} l_i < u_j) \tag{2}$$

We formalize this DNF-based (and thus non-elementary) quantifier elimination procedure in §4.2.

More interesting are two new NNF-based algorithms developed in §4.3–4.5. They are based on the *test point* method (originally due to Cooper [5] and Ferrante and Rackoff [8] and later generalized by Weispfenning [26]). The idea is to find a finite set of test points T (depending on φ) such that $(\exists x.\ \varphi(x)) = (\bigvee_{t \in T} \varphi(t))$. The complication is that (conceptually) T may contain values like infinity, infinitesimals or intermediate points, values that are not representable in the given term language. The challenge is to define special versions of substitution for these values.

4.1 Atoms

There are just the two relations $<$ and $=$ and no function symbols. Thus atomic formulae can be represented by the following datatype:

datatype $atom = nat \mathbin{\boldsymbol{<}} nat \mid nat \mathbin{\boldsymbol{=}} nat$

Note the **bold** infix constructors **<** and **=**. Because there are no function symbols, the arguments of the relations must be variables. For example, $i \mathbin{\boldsymbol{<}} j$ represents the atom $x_i < x_j$ in de Bruijn notation.

Now we can instantiate locale *ATOM*. Type parameter α becomes type *atom*. The interpretation function I_a becomes I_{dlo} where

$$I_{dlo}\ (i \mathbin{\boldsymbol{=}} j)\ xs = (xs_{[i]} = xs_{[j]})$$
$$I_{dlo}\ (i \mathbin{\boldsymbol{<}} j)\ xs = (xs_{[i]} < xs_{[j]})$$

The notation $xs_{[i]}$ means selection of the ith element of xs. The type of I_{dlo} is explicitly restricted such that xs must be a list of elements over a dense linear order, where the latter is formalized as a type class [11] with the axioms shown at the start of this section. Thus all valuations in this section are over dense linear orders. Parameter *aneg* becomes neg_{dlo}:

$$neg_{dlo}\ (i \mathbin{\boldsymbol{<}} j) = (A\ (j \mathbin{\boldsymbol{<}} i) \vee A\ (i \mathbin{\boldsymbol{=}} j))$$
$$neg_{dlo}\ (i \mathbin{\boldsymbol{=}} j) = (A\ (i \mathbin{\boldsymbol{<}} j) \vee A\ (j \mathbin{\boldsymbol{<}} i))$$

The parameters *adepends* and *adecr* are instantiated with $depends_{dlo}$ and $decr_{dlo}$:

$$depends_{dlo}\ (i \mathbin{\boldsymbol{=}} j) = (i = 0 \vee j = 0)$$
$$depends_{dlo}\ (i \mathbin{\boldsymbol{<}} j) = (i = 0 \vee j = 0)$$

$$decr_{dlo}\ (i \mathbin{\boldsymbol{<}} j) = (i - 1 \mathbin{\boldsymbol{<}} j - 1)$$
$$decr_{dlo}\ (i \mathbin{\boldsymbol{=}} j) = (i - 1 \mathbin{\boldsymbol{=}} j - 1)$$

This instantiation satisfies all the axioms of *ATOM*.

4.2 DNF-Based Quantifier Elimination

First we consider the special case of a list (conjunction) of **<** atoms *as*. We may assume they all depend on variable 0, the variable to be eliminated:

```
$qe\text{-}dlo_1\ as =$
  $($if $(0 \mathbin{\boldsymbol{<}} 0) \in set\ as$ then $\bot$ else
   let $lbs = [i.\ (Suc\ i \mathbin{\boldsymbol{<}} 0) \leftarrow as];$
       $ubs = [j.\ (0 \mathbin{\boldsymbol{<}} Suc\ j) \leftarrow as];$
       $pairs = [A(i \mathbin{\boldsymbol{<}} j).\ i \leftarrow lbs, j \leftarrow ubs]$
   in $list\text{-}conj\ pairs)$
```

This is an executable version of (2), except that we also take care of the unsatisfiable atom $x_0 < x_0$ and we decrement the variables to compensate for the eliminated quantifier. Instead of detecting only the contradiction $x_0 < x_0$ one could (and should) return $\bot$ upon finding any $x_i < x_i$.

The extension with equality is provided by instantiating *ATOM-EQ* (see §3.4.2): *solvable*$_0$ becomes $\lambda(i=j) \Rightarrow i{=}0 \vee j{=}0 \mid a \Rightarrow \mathit{False}$,[1] *trivial* becomes $\lambda(i=j) \Rightarrow i{=}j \mid a \Rightarrow \mathit{False}$ and *subst*$_0$ is defined as follows:

$$\begin{aligned} &\mathit{subst}_0\ (i=j)\ (m < n) = (\mathit{subst}\ i\ j\ m < \mathit{subst}\ i\ j\ n) \\ &\mathit{subst}_0\ (i=j)\ (m = n) = (\mathit{subst}\ i\ j\ m = \mathit{subst}\ i\ j\ n) \end{aligned}$$

$$\mathit{subst}\ i\ j\ k = (\mathsf{if}\ k = 0\ \mathsf{then}\ \mathsf{if}\ i = 0\ \mathsf{then}\ j\ \mathsf{else}\ i\ \mathsf{else}\ k) - 1$$

Discharging the assumptions of *ATOM-EQ* is straightforward.

4.3 The Interior Point Method

Ferrante and Rackoff [8] realized (for linear real arithmetic) that when eliminating x from ϕ it (essentially) suffices to collect all lower bounds l of x (i.e. $l < x$ occurs in ϕ) and all upper bounds u of x (i.e. $x < u$ occurs in ϕ) and try all such $(l+u)/2$ as test points. This method is implemented in §5.3.

Now we present a novel quantifier elimination method for DLOs based on Ferrante and Rackoff's idea. The problem with DLOs is that one cannot name any point between two variables x and y. Hence a special form of substitution must be defined that behaves as if some intermediate point was substituted without requiring such a point. We use the symbolic notation $x{\downarrow}y$ to denote some arbitrary but fixed point in the interval (x,y). Substitution with $x{\downarrow}y$ is defined as follows:

$$\begin{array}{llll} (x{\downarrow}y < z) & = (y \le z) & (x{\downarrow}y = z) & = \mathit{False} \\ (z < x{\downarrow}y) & = (z \le x) & (z = x{\downarrow}y) & = \mathit{False} \\ (x{\downarrow}y < x{\downarrow}y) & = \mathit{False} & (x{\downarrow}y = x{\downarrow}y) & = \mathit{True} \end{array}$$

Note that $x{\downarrow}y = z$ is false because we can always choose $x{\downarrow}y$ to be different from z. Note also that these definitions only work as expected if $x < y$.

We also need the fictitious values $-\infty$ and ∞ first used by Cooper. Then we can formulate the interior point method as a logical equivalence in test point form, where ϕ must be quantifier-free and in NNF:

$$(\exists x.\ \phi(x)) \ = \ (\phi(-\infty) \vee \phi(\infty) \vee \bigvee_{y \in E} \phi(y) \vee \bigvee_{y \in L, z \in U} (y < z \wedge \phi(y{\downarrow}z))) \tag{3}$$

E is the set of y such that $x = y$ or $y = x$ occur in $\phi(x)$, L is the set of y such that $y < x$ occurs in $\phi(x)$, U is the set of y such that $x < y$ occurs in $\phi(x)$, where x is the bound variable and y is different from x.

We sketch a proof of (3), details can be found in the Isabelle proof. The if-direction is easy as in each case a witness is given. Except that $-\infty$, ∞ and $y{\downarrow}z$ are not proper values. But by induction on ϕ one can show that $\phi(-\infty)$ etc imply $\phi(x)$ for suitable x:

$$\begin{gathered} \exists x.\forall y \le x.\ \phi(-\infty) = \phi(y) \\ \exists x.\forall y \ge x.\ \phi(\infty) = \phi(y) \\ y < z \wedge \phi(y{\downarrow}z) \Longrightarrow \forall x \in (y,z).\ \phi(x) \end{gathered}$$

[1] The notation $\lambda\ \mathit{pat} \Rightarrow e \mid \dots$ is short for $\lambda v.\ \mathsf{case}\ v\ \mathsf{of}\ \mathit{pat} \Rightarrow e \mid \dots$.

For the only-if-direction assume $\phi(x)$ and not $\phi(-\infty) \vee \phi(\infty) \vee \bigvee_{y\in E} \phi(y)$. We have to show that $\phi(y{\downarrow}z)$ for some $y \in L$ and $z \in U$. From the assumptions it follows by induction on ϕ that there must be $y_0 \in L$ and $z_0 \in U$ such that $x \in (y_0, z_0)$. Now we show (by induction on ϕ) the lemma that innermost intervals (y,z) completely satisfy ϕ:

Lemma 6 *If $x \in (y,z)$, $x \notin E$, $(y,x) \cap L = \emptyset$ and $(x,z) \cap U = \emptyset$, then $\phi(x)$ implies that $\forall u \in (y,z).\ \phi(u)$.*

Given $x \in (y_0, z_0)$ we define $y = \max\{y \in L \mid y < x\}$ and $z = \min\{z \in U \mid x < z\}$. It is easy to see that this satisfies the premises of the lemma and hence $\forall u \in (y,z).\ \phi(u)$. Again by induction on ϕ one can show that this actually implies $\phi(y{\downarrow}z)$:

Lemma 7 *If $x \in (y,z)$, $x \notin E$, $(y,x) \cap L = \emptyset$ and $(x,z) \cap U = \emptyset$, then $\forall x \in (y,z).\ \phi(x)$ implies $\phi(y{\downarrow}z)$.*

4.4 A Verified Implementation of the Interior Point Method

The executable version of (3) is short

$$\begin{array}{l}
\mathit{qe\text{-}interior}_1\ \varphi = \\
(\mathsf{let}\ \mathit{as} = \mathit{atoms}_0\ \varphi;\ \mathit{lbs} = \mathit{lbounds\ as};\ \mathit{ubs} = \mathit{ubounds\ as};\ \mathit{ebs} = \mathit{ebounds\ as}; \\
\quad \mathit{intrs} = [\,A(l < u) \wedge (\mathit{subst}_2\ l\ u\ \varphi).\ l{\leftarrow}\mathit{lbs},\ u{\leftarrow}\mathit{ubs}] \\
\ \mathsf{in}\ \mathit{list\text{-}disj}\ (\mathit{inf}_-\ \varphi \cdot \mathit{inf}_+\ \varphi \cdot \mathit{intrs}\ @\ \mathit{map}\ (\mathit{subst}\ \varphi)\ \mathit{ebs}))
\end{array}$$

but requires some auxiliary functions:

The implementation of substituting $l{\downarrow}u$ in atoms is given below. Please note that substitution must not just substitute for variable 0 but must also decrement the other variables.

$$\begin{array}{lcl}
\mathit{asubst}_2\ l\ u\ (0 < 0) & = & \bot \\
\mathit{asubst}_2\ l\ u\ (0 < \mathit{Suc}\ j) & = & A\ (u < j) \vee A\ (u = j) \\
\mathit{asubst}_2\ l\ u\ (\mathit{Suc}\ i < 0) & = & A\ (i < l) \vee A\ (i = l) \\
\mathit{asubst}_2\ l\ u\ (\mathit{Suc}\ i < \mathit{Suc}\ j) & = & A\ (i < j) \\
\mathit{asubst}_2\ l\ u\ (0 = 0) & = & \top \\
\mathit{asubst}_2\ l\ u\ (0 = \mathit{Suc}\ v) & = & \bot \\
\mathit{asubst}_2\ l\ u\ (\mathit{Suc}\ v = 0) & = & \bot \\
\mathit{asubst}_2\ l\ u\ (\mathit{Suc}\ i = \mathit{Suc}\ j) & = & A\ (i = j)
\end{array}$$

From atoms to formulae is a short step: $\mathit{subst}_2\ l\ u\ \varphi \equiv \mathit{amap}_{\mathit{fm}}\ (\mathit{asubst}_2\ l\ u)\ \varphi$

Plain old substitution of a variable k for 0 is defined first on variables, then on atoms and finally on formulae:

$$\begin{array}{l}
\mathit{isubst}\ k\ 0 = k \\
\mathit{isubst}\ k\ (\mathit{Suc}\ i) = i
\end{array}$$

$$\begin{array}{l}
\mathit{asubst}\ k\ (i < j) = (\mathit{isubst}\ k\ i < \mathit{isubst}\ k\ j) \\
\mathit{asubst}\ k\ (i = j) = (\mathit{isubst}\ k\ i = \mathit{isubst}\ k\ j)
\end{array}$$

$$\mathit{subst}\ \varphi\ k \equiv \mathit{map}_{\mathit{fm}}\ (\mathit{asubst}\ k)\ \varphi$$

Substituting $-\infty$ for 0 is implemented by *amin-inf* and inf_-:

$$\begin{array}{lcl}
\textit{amin-inf}\ (\_ < 0) & = & \bot \\
\textit{amin-inf}\ (0 < \textit{Suc}\ \_) & = & \top \\
\textit{amin-inf}\ (\textit{Suc}\ i < \textit{Suc}\ j) & = & A\ (i < j) \\
\textit{amin-inf}\ (0 = 0) & = & \top \\
\textit{amin-inf}\ (0 = \textit{Suc}\ \_) & = & \bot \\
\textit{amin-inf}\ (\textit{Suc}\ \_ = 0) & = & \bot \\
\textit{amin-inf}\ (\textit{Suc}\ i = \textit{Suc}\ j) & = & A\ (i = j)
\end{array}$$

$$\textit{inf}_-\ \varphi \equiv \textit{amap}_{fm}\ \textit{amin-inf}\ \varphi$$

Dually there is $\textit{inf}_+$ for substituting ∞. Lower bounds, upper bounds and equalities are conveniently collected from a list of atoms by list comprehension:

$$\begin{array}{l}
\textit{lbounds as} = [i.\ (\textit{Suc}\ i < 0) \leftarrow \textit{as}] \\
\textit{ubounds as} = [i.\ (0 < \textit{Suc}\ i) \leftarrow \textit{as}] \\
\textit{ebounds as} = [i.\ (\textit{Suc}\ i = 0) \leftarrow \textit{as}]\ @\ [i.\ (0 = \textit{Suc}\ i) \leftarrow \textit{as}]
\end{array}$$

4.5 The Method of Infinitesimals

Loos and Weispfenning [15] proposed a quantifier elimination procedure for linear real arithmetic (see §5.4) where test points are $x+\varepsilon$ (for x a lower bound) or $y-\varepsilon$ (for y an upper bound) where ε is an infinitesimal. That is, the test points are arbitrarily close to the lower or upper bounds of the eliminated variable. In particular, it is not necessary to pair all lower and upper bounds but one can choose either set, typically the smaller one. For succinctness we ignore this duality and concentrate on the lower bounds only.

In this section we adapt the idea of infinitesimals to derive a new quantifier elimination procedure for DLOs. We merely need to explain what substitution of $x+\varepsilon$ means:

$$\begin{array}{llll}
(x+\varepsilon < y) & = (x < y) & (x+\varepsilon = y) & = \textit{False} \\
(y < x+\varepsilon) & = (y \le x) & (y = x+\varepsilon) & = \textit{False} \\
(x+\varepsilon < x+\varepsilon) & = \textit{False} & (x+\varepsilon = x+\varepsilon) & = \textit{True}
\end{array}$$

where x and y are different variables.

The test point method with infinitesimals is justified by the following equivalence, where, as usual, ϕ is quantifier free and in NNF:

$$(\exists x.\ \phi(x)) \;=\; (\phi(-\infty) \vee \bigvee_{y \in E} \phi(y) \vee \bigvee_{y \in L} \phi(y+\varepsilon)) \tag{4}$$

where E and L are defined as in (3). The proof is also similar. The main differences are: For the if-direction we need to show (by induction on ϕ) that $y+\varepsilon$ represents a proper witness:

$$\phi(y+\varepsilon) \Longrightarrow \exists y' > y.\ \forall x \in (y,y').\ \phi(x)$$

The two lemmas for the only-if-direction become

Lemma 8 *If $y < x$, $x \notin E$, $(y,x) \cap L = \emptyset$ and $\phi(x)$, then $\forall u \in (y,x].\ \phi(u)$.*

Lemma 9 *If $y < x$, $x \notin E$, $(y,x) \cap L = \emptyset$ and $\forall u \in (y,x].\ \phi(u)$, then $\phi(y+\varepsilon)$.*

Our verified implementation of (4)

$$\begin{array}{l}
\textit{qe-eps}_1\ \varphi = (\textsf{let}\ \textit{as} = \textit{atoms}_0\ \varphi;\ \textit{lbs} = \textit{lbounds as};\ \textit{ebs} = \textit{ebounds as} \\
\qquad\qquad \textsf{in}\ \textit{list-disj}\ (\textit{inf}_-\ \varphi \cdot \textit{map}\ (\textit{subst}_+\ \varphi)\ \textit{lbs}\ @\ \textit{map}\ (\textit{subst}\ \varphi)\ \textit{ebs}))
\end{array}$$

requires only one new concept, *subst*$_+$ φ *y*, the substitution $\phi(y+\varepsilon)$:

$$\begin{array}{lcl}
\mathit{asubst}_+\ k\ (0 < 0) & = & \bot \\
\mathit{asubst}_+\ k\ (0 < \mathit{Suc}\ j) & = & A\ (k < j) \\
\mathit{asubst}_+\ k\ (\mathit{Suc}\ i < 0) & = & \textsf{if}\ i = k\ \textsf{then}\ \top\ \textsf{else}\ A\ (i < k) \lor A\ (i = k) \\
\mathit{asubst}_+\ k\ (\mathit{Suc}\ i < \mathit{Suc}\ j) & = & A\ (i < j) \\
\mathit{asubst}_+\ k\ (0 = 0) & = & \top \\
\mathit{asubst}_+\ k\ (0 = \mathit{Suc}\ \_) & = & \bot \\
\mathit{asubst}_+\ k\ (\mathit{Suc}\ \_ = 0) & = & \bot \\
\mathit{asubst}_+\ k\ (\mathit{Suc}\ i = \mathit{Suc}\ j) & = & A\ (i = j)
\end{array}$$

$$\mathit{subst}_+\ \varphi\ k \equiv \mathit{amap}_{\mathit{fm}}\ (\mathit{asubst}_+\ k)\ \varphi$$

4.6 Complexity

We analyze the complexity of the interior point and the infinitesimal method. A formula of size n can contain at most n variables. The set of variables decreases by one in each step. In the worst case all of them are bound and need to be eliminated. In each step of the quantifier elimination processes (3) and (4) the sets E, L and U are at most as large as k, the current number of variables.

The interior point method makes at most $(k-1)^2$ copies of the formula in each step. Hence the size of the output formula and also the amount of working space required is $O(n\cdot(n-1)^2\cdots 1^2) = O(n\cdot(n-1)!^2)$. The method of infinitesimals, however, only makes at most $k-1$ copies, thus requiring only $O(n\cdot(n-1)\cdots 1) = O(n!)$ space. The time complexity of both algorithms is linear in their space complexity, i.e. time and space coincide.

5 Linear Real Arithmetic

Linear real arithmetic is concerned with terms built up from variables, constants, addition, and multiplication with constants. Relations between such terms can be put into a normal form $r \bowtie c_0 * x_0 + \cdots c_n * x_n$ with $\bowtie\ \in \{=,<\}$ and $r, c_0, \ldots, c_n \in \mathbb{R}$. It is this normal form we work with in this section. Note that although we phrase everything in terms of the real numbers, the rational numbers work just as well. In fact, any ordered, divisible, torsion free, Abelian group will do.

We present verified implementations of three quantifier elimination procedures due to Fourier [9], Ferrante and Rackoff [8] and Loos and Weispfenning [15].

5.1 Atoms

Type *atom* formalizes the normal forms explained above:

datatype *atom* = *real* < (*real list*) | *real* = (*real list*)

The second constructor argument is the list of coefficients $[c_0,\ldots,c_n]$ of the variables *0* to n — remember de Bruijn! Coefficient lists should be viewed as vectors and we define the usual vector operations on them:

$x *_s xs$ is the componentwise multiplication of a scalar x with a vector xs.

$xs + ys$ and $xs - ys$ are componentwise addition and subtraction of vectors.

$\langle xs,ys\rangle = (\sum (x,y) \leftarrow zip\ xs\ ys.\ x{*}y)$ is the inner product of two vectors, i.e. the sum over the componentwise products.

If the two vectors involved in an operation are of different length, the shorter one is padded with 0s (as in Obua's treatment of matrices [24]). Although only the set of vectors of the same length form a vector space, we can prove all the algebraic properties we need, for example $\langle xs + ys,zs\rangle = \langle xs,zs\rangle + \langle ys,zs\rangle$.

Now we instantiate locale *ATOM* just like for DLO in §4.1. The main function is the interpretation I_R of atoms, which is straightforward:

$$I_R\ (r < cs)\ xs = (r < \langle cs,xs\rangle)$$
$$I_R\ (r = cs)\ xs = (r = \langle cs,xs\rangle)$$

5.2 Fourier-Motzkin Elimination

Fourier-Motzkin Elimination is a procedure discovered by Fourier [9]. Motzkin [19] (but also Farkas [7]) cited Fourier but were concerned with the algebraic background, not the algorithm. You put the formula into DNF and for each conjunct the inequalities are split into those of the form $l < x$ and those of the form $x < u$, and then you "multiply out" exactly as in (2) for DLOs. Except that one has to transform the inequalities into the form $l < x$ and $x < u$ explicitly and the l and u can be proper terms, not just variables.

Quantifier elimination for the special case of a list of $<$ atoms *as*, all of which depend on variable 0, is a one-liner

$$qe\text{-}lin_1\ as = list\text{-}conj\ [A(combine\ p\ q).\ p{\leftarrow}lbounds\ as,\ q{\leftarrow}ubounds\ as]$$

where *lbounds* and *ubounds* select the inequalities where variable 0 has respectively a positive and a negative coefficient

$$lbounds\ as = [(r/c,\ (-1/c) *_s cs).\ (r < c{\cdot}cs) \leftarrow as,\ c{>}0]$$
$$ubounds\ as = [(r/c,\ (-1/c) *_s cs).\ (r < c{\cdot}cs) \leftarrow as,\ c{<}0]$$

and they are combined as explained above:

$$combine\ (r_1,\ cs_1)\ (r_2,\ cs_2) = (r_1 - r_2 < cs_2 - cs_1)$$

Instead of transforming the inequalities into $l < x$ and $x < u$ by dividing by the coefficient of x one can combine $r_1 < c_1x + t_1$ and $r_2 < c_2x + t_2$ into $c_1r_2 - c_2r_1 < c_1t_2 - c_2t_1$ provided $c_1 > 0$, $c_2 < 0$, and x does not occur in the t_i.

The extension with equality is provided by instantiating *ATOM-EQ* (see §3.4.2, and see earlier footnote for λ-notation): $solvable_0$ is any $=$ atom whose head coefficient is nonzero ($\lambda(r = c{\cdot}cs) \Rightarrow c \neq 0 \mid a \Rightarrow False$), *trivial* is any $=$ atome where both sides are zero ($\lambda(r = cs) \Rightarrow r{=}0 \wedge (\forall c \in set\ cs.\ c{=}0) \mid a \Rightarrow False$), and $subst_0$ is defined as follows:

$$subst_0\ (r = c{\cdot}cs)\ (s < d{\cdot}ds) = (s - r * d\ /\ c < ds - (d\ /\ c) *_s cs)$$
$$subst_0\ (r = c{\cdot}cs)\ (s = d{\cdot}ds) = (s - r * d\ /\ c = ds - (d\ /\ c) *_s cs)$$

5.3 Ferrante and Rackoff

Ferrante and Rackoff [8], inspired by Cooper [5], avoided DNF conversions by the test point method explained in §4. We have already explained the key idea of Ferrante and Rackoff in

§4.3. If you replace $y{\downarrow}z$ in (3) by $(y+z)/2$ you almost obtain their algorithm. In principle any point between y and z works but $(y+z)/2$ also takes care of equalities: they lump E, L and U together (to be avoided in an implementation) but because $(y+y)/2 = y$ this recovers E. As their algorithm is well-known, we present its optimized and verified implementation right away:

$$\begin{array}{l}
\mathit{qe\text{-}FR}_1\ \varphi = \\
(\mathsf{let}\ \mathit{as} = \mathit{atoms}_0\ \varphi;\ \mathit{lbs} = \mathit{lbounds}\ \mathit{as};\ \mathit{ubs} = \mathit{ubounds}\ \mathit{as};\ \mathit{ebs} = \mathit{ebounds}\ \varphi; \\
\quad \mathit{intrs} = [\mathit{subst}\ \varphi\ (\mathit{between}\ l\ u)\ .\ l \leftarrow \mathit{lbs},\ u \leftarrow \mathit{ubs}]; \\
\ \mathsf{in}\ \mathit{list\text{-}disj}\ (\mathit{inf}_-\ \varphi \cdot \mathit{inf}_+\ \varphi \cdot \mathit{intrs}\ @\ \mathit{map}\ (\mathit{subst}\ \varphi)\ \mathit{ebs}))
\end{array}$$

Except for the definition of *intrs* this looks identical to the definition of $\mathit{qe\text{-}interior}_1$ in §4.4. However, all auxiliary functions are different: they operate on pairs (r, cs) which, under a valuation xs, represent the value $r + \langle cs, xs\rangle$. First the various bounds are extracted:

$$\begin{array}{l}
\mathit{lbounds}\ \mathit{as} = [(r/c,\ (-1/c) *_s cs).\ (r < c \cdot cs) \leftarrow \mathit{as},\ c{>}0] \\
\mathit{ubounds}\ \mathit{as} = [(r/c,\ (-1/c) *_s cs).\ (r < c \cdot cs) \leftarrow \mathit{as},\ c{<}0] \\
\mathit{ebounds}\ \mathit{as} = [(r/c,\ (-1/c) *_s cs).\ (r = c \cdot cs) \leftarrow \mathit{as},\ c{\neq}0]
\end{array}$$

The intermediate point between two such points is easy:

$$\mathit{between}\ (r,\ cs)\ (s,\ ds) = ((r + s)\ /\ 2,\ (1\ /\ 2) *_s (cs + ds))$$

We also need both ordinary substitution of (r, cs) pairs

$$\begin{array}{l}
\mathit{asubst}\ (r,\ cs)\ (s < d \cdot ds) = (s - d * r < d *_s cs + ds) \\
\mathit{asubst}\ (r,\ cs)\ (s = d \cdot ds) = (s - d * r = d *_s cs + ds) \\
\mathit{asubst}\ \mathit{rcs}\ a = a \\
\mathit{subst}\ \varphi\ \mathit{rcs} \equiv \mathit{map}_{\mathit{fm}}\ (\mathit{asubst}\ \mathit{rcs})\ \varphi
\end{array}$$

and substitution inf_- of $-\infty$ (and the analogous version inf_+ for ∞):

$$\begin{array}{l}
\mathit{inf}_-\ (\varphi_1 \wedge \varphi_2) = \mathit{and}\ (\mathit{inf}_-\ \varphi_1)\ (\mathit{inf}_-\ \varphi_2) \\
\mathit{inf}_-\ (\varphi_1 \vee \varphi_2) = \mathit{or}\ (\mathit{inf}_-\ \varphi_1)\ (\mathit{inf}_-\ \varphi_2) \\
\mathit{inf}_-\ (A\ (r < c \cdot cs)) = (\mathsf{if}\ c < 0\ \mathsf{then}\ \top\ \mathsf{else}\ \mathsf{if}\ 0 < c\ \mathsf{then}\ \bot\ \mathsf{else}\ A\ (r < cs)) \\
\mathit{inf}_-\ (A\ (r = c \cdot cs)) = (\mathsf{if}\ c = 0\ \mathsf{then}\ A\ (r = cs)\ \mathsf{else}\ \bot) \\
\mathit{inf}_-\ \varphi = \varphi
\end{array}$$

This concludes the definition of the auxiliary functions.

5.4 Loos and Weispfenning

The method of infinitesimals described in §4.5 was inspired by the analogous method for linear real arithmetic proposed by Loos and Weispfenning [15] who also showed practical examples where it outperforms Ferrante and Rackoff. For a recent comparison of related QEPs see [18]. Our implementation $\mathit{qe\text{-}eps}_1$ is textually identical to the one for DLOs in §4.5. But the auxiliary functions differ. Luckily we have seen all of them already, except subst_+:

$$\begin{array}{l}
\mathit{asubst}_+\ (r,\ cs)\ (s < d \cdot ds) = \\
(\mathsf{if}\ d = 0\ \mathsf{then}\ A\ (s < ds) \\
\ \mathsf{else}\ \mathsf{let}\ u = s - d * r;\ v = d *_s cs + ds;\ \mathit{lessa} = A\ (u < v) \\
\qquad \mathsf{in}\ \mathsf{if}\ d < 0\ \mathsf{then}\ \mathit{lessa}\ \mathsf{else}\ \mathit{lessa} \vee A\ (u = v)) \\
\mathit{asubst}_+\ \mathit{rcs}\ (r = d \cdot ds) = (\mathsf{if}\ d = 0\ \mathsf{then}\ A\ (r = ds)\ \mathsf{else}\ \bot) \\
\mathit{asubst}_+\ \mathit{rcs}\ a = A\ a \\
\mathit{subst}_+\ \varphi\ \mathit{rcs} \equiv \mathit{amap}_{\mathit{fm}}\ (\mathit{asubst}_+\ \mathit{rcs})\ \varphi
\end{array}$$

6 Presburger Arithmetic

Presburger Arithmetic is linear integer arithmetic. Presburger [25] showed that this theory has quantifier elimination. In contrast to linear real arithmetic we need an additional predicate to obtain quantifier elimination: there is no quantifier-free equivalent of $\exists x.\ x+x=y$ if we restrict to linear arithmetic. The way out is to allow the divisibility predicate as well, but only of the form $d \mid t$ where d is a constant. Now $\exists x.\ x+x=y$ is equivalent with $2 \mid x$. Alternatively one can introduce congruence relations $s \equiv t \ (mod\, d)$ instead of divisibility. On the other hand we do not need both $<$ and $=$ on the integers but can restrict our attention to $\leq$. Thus we may assume all atoms are of the form $i \leq k_0 * x_0 + \cdots + k_n * x_n$ or $d \parallel i + k_0 * x_0 + \cdots k_n * x_n$, where $\parallel$ is $\mid$ or $\nmid$, and $d, i, k_0, \ldots, k_n \in \mathbb{Z}$ and $d > 0$. The negated atom $i \not\leq j$ is equivalent with $j + 1 \leq i$.

6.1 Atoms

The above language of atoms is formalized as follows:

datatype $atom = Le\ int\ (int\ list) \mid Dvd\ int\ int\ (int\ list) \mid NDvd\ int\ int\ (int\ list)$

We have avoided infix constructors because they work less well for ternary operations. Atoms are interpreted w.r.t. a list of variables as usual:

$$I_Z\ (Le\ i\ ks)\ xs = (i \leq \langle ks, xs \rangle)$$
$$I_Z\ (Dvd\ d\ i\ ks)\ xs = d \mid (i + \langle ks, xs \rangle)$$
$$I_Z\ (NDvd\ d\ i\ ks)\ xs = (\neg\ d \mid (i + \langle ks, xs \rangle))$$

Note that we reuse the polymorphic vector, i.e. list operations like $\langle .,. \rangle$ introduced for linear real arithmetic: they are defined for arbitrary types with 0, $+$ and $*$.

The parameters of locale *ATOM* are instantiated as follows. The interpretation of atoms is given by function I_Z above, their negation by

$$neg_Z\ (Le\ i\ ks) = A\ (Le\ (1 - i)\ (-\ ks))$$
$$neg_Z\ (Dvd\ d\ i\ ks) = A\ (NDvd\ d\ i\ ks)$$
$$neg_Z\ (NDvd\ d\ i\ ks) = A\ (Dvd\ d\ i\ ks)$$

and their decrementation by

$$decr_Z\ (Le\ i\ ks) = Le\ i\ (tl\ ks)$$
$$decr_Z\ (Dvd\ d\ i\ ks) = Dvd\ d\ i\ (tl\ ks)$$
$$decr_Z\ (NDvd\ d\ i\ ks) = NDvd\ d\ i\ (tl\ ks)$$

where *tl* is the tail of a list: $tl\ [] = []$ and $tl\ (x \cdot xs) = xs$.

Parameter $depends_0$ becomes $\lambda a.\ hd\text{-}coeff\ a \neq 0$ where

$$hd\text{-}coeff\ (Le\ i\ ks) = (\mathsf{case}\ ks\ \mathsf{of}\ [] \Rightarrow 0 \mid k \cdot x \Rightarrow k)$$
$$hd\text{-}coeff\ (Dvd\ d\ i\ ks) = (\mathsf{case}\ ks\ \mathsf{of}\ [] \Rightarrow 0 \mid k \cdot x \Rightarrow k)$$
$$hd\text{-}coeff\ (NDvd\ d\ i\ ks) = (\mathsf{case}\ ks\ \mathsf{of}\ [] \Rightarrow 0 \mid k \cdot x \Rightarrow k)$$

There is a slight complication here: We want to exclude atoms of the form *Dvd 0 i ks* and *NDvd 0 i ks* because they behave anomalously and the algorithms do not generate them either. In order to restrict attention to a subset of atoms, locale *ATOM* in fact has another parameter not mentioned so far: $anormal :: \alpha \Rightarrow bool$ with the axioms

$$anormal\ a \Longrightarrow \forall b \in atoms\ (aneg\ a).\ anormal\ b$$
$$\neg\ depends_0\ a \Longrightarrow anormal\ a \Longrightarrow anormal\ (decr\ a)$$

In words: negation and decrementation do not lead outside the normal atoms. With the help of these axioms the following refined versions of lemmas 2 and 4 can be proved, where *normal* $\varphi = (\forall a{\in}atoms\ \varphi.\ anormal\ a)$:

Lemma 10 *If* $qe \in nqfree \to qfree$ *and* $qe \in nqfree \cap normal \to normal$ *and* $\forall\,\varphi{\in}normal \cap nqfree.\ \forall xs.\ I\ (qe\ \varphi)\ xs = (\exists x.\ I\ \varphi\ (x{\cdot}xs))$*, then normal* φ *implies* $I\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$.

Lemma 11 *If* $qe \in lists\ depends_0 \to qfree$ *and* $qe \in lists\ (depends_0 \cap anormal) \to normal$ *and* $\forall as \in lists(depends_0 \cap anormal).\ is\text{-}dnf\text{-}qe\ qe\ as$*, then normal* φ *implies* $I\ (lift\text{-}dnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$.

The analogous versions of lemmas 1 and 3 are straightforward and omitted.

For Presburger arithmetic, parameter *anormal* becomes $\lambda a.\ divisor\ a \neq 0$ where

$divisor\ (Le\ i\ ks) = 1$
$divisor\ (Dvd\ d\ i\ ks) = d$
$divisor\ (NDvd\ d\ i\ ks) = d$

6.2 DNF-based Algorithm

The algorithm we are going to describe differs from Presburger's original algorithm because that one covers only $=$ (and congruence) — Presburger merely states that it can be extended to $<$. Our algorithm resembles Enderton's version [6], except that the main case split is different: Enderton distinguishes if there are congruences or not, we distinguish if there are lower bounds or not.

Input to the algorithm is $P(x)$, a conjunction of atoms, all depending on x. The algorithm consists of the following two steps:

1. Set all coefficients of x to the positive least common multiple (lcm) m of all coefficients of x in $P(x)$. Call the result $Q(m * x)$. Let $R(x) = Q(x) \wedge m \mid x$.
2. Let δ be the lcm of all divisors d in $R(x)$ and let L be the set of lower bounds for x in $R(x)$. If $L \neq \emptyset$ then return $\bigvee_{t \in T} R(t)$ where $T = \{l + n \mid l \in L \wedge 0 \leq n < \delta\}$. If $L = \emptyset$ return $\bigvee_{t \in T} R'(t)$ where R' is R without $\leq$-atoms and $T = \{n \mid 0 \leq n < \delta\}$.

As usual, upper bounds work just as well as lower bounds.

For example, if $P(x) = (l \leq 2x \wedge 3x \leq u)$, then $Q(6 * x) = (3l \leq 6x \wedge 6x \leq 2u)$, $R(x) = (3l \leq x \wedge x \leq 2u \wedge 6 \mid x)$, and the result is $\bigvee_{0 \leq n < 6} R(3l + n)$.

The first step of the algorithm is clearly equivalence preserving. Now we have a conjunction $R(x)$ of atoms where x has coefficient 1 everywhere. Equivalence preservation of the second step is proved in both directions separately as follows.

First we assume the returned formula and show $R(t)$ for some t. If $L \neq \emptyset$ then $R(t)$ for some $t \in T$ and we are done. Now assume $L = \emptyset$. By assumption there must be some $0 \leq n < \delta$ such that $R'(n)$. If there are no upper bounds for x in $R(x)$ either, then $R(x)$ contains no $\leq$-atoms, $R' = R$, and hence $R(n)$. Otherwise let U be the set of all upper bounds of x in $R(x)$, let m be the minimum of U and let $t = n - ((n - m)\ div\ \delta + 1)\delta$. We show $R(t)$. From $R'(n)$ and the definition of R' and t, $R'(t)$ follows. All $\leq$-atoms must be upper bound constraints $x \leq u$ and hence $m \leq u$. Because $(n - m)\ mod\ \delta < \delta$ we obtain $t \leq m \leq u$. Thus t satisfies all $\leq$-atoms, and hence $R(t)$.

Now assume that $R(z)$ for some z. In this direction it is important to note that (non)divisibility atoms $a(x)$ are cyclic in their divisor d, i.e. $a(x)$ is equivalent with $a(x\ mod\ d)$ because the coefficient of x is 1. This carries over to any multiple of d, in particular δ. If $L = \emptyset$ we obtain $R'(z\,mod\,\delta)$ with $0 \leq z\,mod\,\delta < \delta$ as required because $R'(x)$ consists only of (non)divisibility atoms. If $L \neq \emptyset$ we show $R(t)$ where $t = m+n$ where m is the maximum of L and $n = (z-m)\,mod\,\delta$. Let $a(x)$ be some atom in $R(x)$. If a is a lower bound atom for x, $a(t)$ follows because $t \geq m$ and m is the maximum of L. If a is an upper bound atom for x, $a(t)$ follows because $t \leq z$ and $a(z)$. If a is a (non)divisibility atom, $a(t)$ follows from $a(z)$ because $t\,mod\,\delta = z\,mod\,\delta$.

6.3 Formalization

First, head coefficients are set to 1 or -1 (-1 is needed because variables may only appear on the lhs of $\leq$). This is achieved by multiplying each atom a with m/k, where m is the lcm of all head coefficients and k is a's head coefficient (assuming $k > 0$ for simplicity).

```
$hd\text{-}coeff1\ m\ (Le\ i\ (k \cdot ks)) =$
$(\mathsf{if}\ k = 0\ \mathsf{then}\ Le\ i\ (k \cdot ks)$
  $\mathsf{else}\ \mathsf{let}\ m' = m\ div\ |k|\ \mathsf{in}\ Le\ (m' * i)\ (sgn\ k \cdot m' *_s\ ks))$

$hd\text{-}coeff1\ m\ (Dvd\ d\ i\ (k \cdot ks)) =$
$(\mathsf{if}\ k = 0\ \mathsf{then}\ Dvd\ d\ i\ (k \cdot ks)$
  $\mathsf{else}\ \mathsf{let}\ m' = m\ div\ k\ \mathsf{in}\ Dvd\ (m' * d)\ (m' * i)\ (1 \cdot m' *_s\ ks))$

$hd\text{-}coeff1\ m\ (NDvd\ d\ i\ (k \cdot ks)) =$
$(\mathsf{if}\ k = 0\ \mathsf{then}\ NDvd\ d\ i\ (k \cdot ks)$
  $\mathsf{else}\ \mathsf{let}\ m' = m\ div\ k\ \mathsf{in}\ NDvd\ (m' * d)\ (m' * i)\ (1 \cdot m' *_s\ ks))$

$hd\text{-}coeff1\ m\ a = a$

$sgn\ i = (\mathsf{if}\ i = 0\ \mathsf{then}\ 0\ \mathsf{else}\ \mathsf{if}\ 0 < i\ \mathsf{then}\ 1\ \mathsf{else}\ -1)$
```

Note that *hd-coeff1* is well-defined even if the head coefficient is 0. The DNF-based quantifier elimination framework guarantees that all atoms under consideration have non-0 head coefficients, but below we reuse *hd-coeff1* in a context where this is not the case.

Function *hd-coeffs1* sets the head coefficients of a list of atoms *as* to 1 or -1 and adds the divisibility constraint $m \mid x_0$:

```
$hd\text{-}coeffs1\ as =$
$(\mathsf{let}\ m = zlcms\ (map\ hd\text{-}coeff\ as)\ \mathsf{in}\ Dvd\ m\ 0\ [1] \cdot map\ (hd\text{-}coeff1\ m)\ as)$
```

Remember that *hd-coeff* extracts the head coefficient from an atom (see §6.1); *zlcms* computes the positive lcm of a list of integers.

We prove that *hd-coeffs1* leaves the interpretation unchanged in the following sense:

Lemma 12 *If* $\forall a{\in}set\ as.\ hd\text{-}coeff\ a \neq 0$ *then* $I\ (\exists\ (list\text{-}conj\ (map\ A\ (hd\text{-}coeffs1\ as)))) = I\ (\exists\ (list\text{-}conj\ (map\ A\ as)))$.

In the second step, quantifier elimination is performed:

```
$qe\text{-}pres_1\ as =$
  $(\mathsf{let}\ ds = [a{\leftarrow}as.\ is\text{-}dvd\ a];\ d = zlcms(map\ divisor\ ds);\ ls = lbounds\ as$
   $\mathsf{in}\ \mathsf{if}\ ls = []$
     $\mathsf{then}\ Disj\ [0..d{-}1]\ (\lambda n.\ list\text{-}conj(map\ (A \circ asubst\ n\ [])\ ds))$
     $\mathsf{else}\ Disj\ ls\ (\lambda (i,ks).$
          $Disj\ [0..d{-}1]\ (\lambda n.\ list\text{-}conj(map\ (A \circ asubst\ (i{+}n)\ ({-}ks))\ as))))$
```

where *is-dvd a* is true iff *a* is of the form *Dvd* or *NDvd*, and *lbounds* collects the lower bounds for variable 0

$$lbounds\ as = [(i,ks).\ Le\ i\ (k \cdot ks) \leftarrow as,\ k{>}0]$$

and *asubst* performs substitution for variable 0:

$$\begin{array}{lcl}
asubst\ i'\ ks'\ (Le\ i\ (k \cdot ks)) & = & Le\ (i - k * i')\ (k *_s ks' + ks) \\
asubst\ i'\ ks'\ (Dvd\ d\ i\ (k \cdot ks)) & = & Dvd\ d\ (i + k * i')\ (k *_s ks' + ks) \\
asubst\ i'\ ks'\ (NDvd\ d\ i\ (k \cdot ks)) & = & NDvd\ d\ (i + k * i')\ (k *_s ks' + ks) \\
asubst\ i'\ ks'\ a & = & a
\end{array}$$

The following lemma shows precisely in which sense *asubst* is substitution:

$$I_Z\ (asubst\ i\ ks\ a)\ xs = I_Z\ a\ ((i + \langle ks,xs \rangle) \cdot xs)$$

The actual quantifier elimination procedure is the lifted composition of the two basic steps:

$$qe\text{-}pres = lift\text{-}dnf\text{-}qe\ (qe\text{-}pres_1 \circ hd\text{-}coeffs1)$$

The correctness proof of $qe\text{-}pres_1 \circ hd\text{-}coeffs1$ was given in the previous subsection. In order to apply lemma 11 we also need to show that the algorithm preseves normality of atoms, which is straightforward but tedious and hence omitted.

6.4 Cooper's Algorithm

Cooper's algorithm relies on Cooper's theorem [5] which holds provided all coefficients of x in $\phi(x)$ are 1 or -1 (or 0):

$$(\exists x.\ \phi(x)) \ = \ (\bigvee_{j \in (0,\delta-1)} \phi_{-\infty}(j) \vee \bigvee_{y \in L} \bigvee_{j \in (0,\delta-1)} \phi(y+j))$$

where δ is the lcm of all d such that $d \mid t$ or $d \nmid t$ occurs in $\phi(x)$ and t contains x, L is the set of lower bounds for x in $\phi(x)$, and $\phi_{-\infty}(j)$ is $\phi(x)$ where x has been replaced by $-\infty$ in all inequations and by j in all other atoms.

We start by setting all head coefficients to 1 or -1 (or leaving them at 0):

$$\begin{array}{l}
hd\text{-}coeffs1\ \varphi = \\
(\mathsf{let}\ m = zlcms\ (map\ hd\text{-}coeff\ (atoms_0\ \varphi)) \\
\ \ \mathsf{in}\ A\ (Dvd\ m\ 0\ [1]) \wedge map_{fm}\ (hd\text{-}coeff1\ m)\ \varphi)
\end{array}$$

This function supersedes its namesake in the previous subsection defined on lists of atoms. Remember that $atoms_0$ is the set of atoms that depend on variable 0, i.e. whose head coefficient is non-0.

Now we start to implement Cooper's theorem. The substitution $\phi_{-\infty}(j)$ is implemented by the composition of

$$\begin{array}{l}
inf_-\ (\varphi_1 \wedge \varphi_2) = and\ (inf_-\ \varphi_1)\ (inf_-\ \varphi_2) \\
inf_-\ (\varphi_1 \vee \varphi_2) = or\ (inf_-\ \varphi_1)\ (inf_-\ \varphi_2) \\
inf_-\ (A\ (Le\ i\ (k \cdot ks))) = (\mathsf{if}\ k < 0\ \mathsf{then}\ \top\ \mathsf{else\ if}\ 0 < k\ \mathsf{then}\ \bot\ \mathsf{else}\ A\ (Le\ i\ (0 \cdot ks))) \\
inf_-\ \varphi = \varphi
\end{array}$$

and ordinary substitution:

$$subst\ i\ ks\ \varphi \equiv map_{fm}\ (asubst\ i\ ks)\ \varphi$$

The right-hand side of Cooper's theorem now becomes executable:

```
$qe\text{-}cooper_1\ \varphi =$
$(\mathsf{let}\ as = atoms_0\ \varphi;\ d = zlcms(map\ divisor\ as);$
    $lbs = [(i,ks).\ Le\ i\ (k \cdot ks) \leftarrow as,\ k{>}0]$
 $\mathsf{in}\ or\ (Disj\ [0..d-1]\ (\lambda n.\ subst\ n\ []\ (inf_{-}\ \varphi)))$
       $(Disj\ lbs\ (\lambda(i,ks).\ Disj\ [0..d-1]\ (\lambda n.\ subst\ (i+n)\ (-ks)\ \varphi))))$
```

The two phases of Cooper's algorithm are simply composed and lifted:

$$qe\text{-}cooper = lift\text{-}nnf\text{-}qe\ (qe\text{-}cooper_1 \circ hd\text{-}coeffs1)$$

Our formal correctness proof follows the literature.

7 Related work

The literature on decision procedures for linear arithmetic is vast. We concentrate on formally verified algorithms.

This paper is a revised combination of [20] and [21], which concentrate on NNF-based and DNF-based algorithms, respectively. It is an outgrowth of the work by Chaieb and Nipkow [4] who present a reflective implementations of Cooper's algorithm. They lack the generic framework and they use special purpose data structures for terms instead of relying on lists as we do. As a result some of their functions are more complicated than ours and theorems and proofs are littered with linearity assumptions that are implicit in our list representation. Hence they can only present part of their implementation. Chaieb [3] presents a verified combination of Ferrante-Rackoff and Cooper.

Norrish [23] was the first to implement a proof-producing version of Cooper's algorithm in a theorem prover. Similar implementation of QE for complex numbers and for real closed fields are reported by Harrison [12] and McLaughlin [17]. The CAD QE procedure for real closed fields has been reflected and partly verified by Mahboubi [16].

Acknowledgment Amine Chaieb introduced me to QE and alerted me to the infinitesimal approach [15]. He and Jeremy Avigad discussed the material with me extensively. John Harrison's anonymous comments helped to improve the presentation.

References

1. Clemens Ballarin. Interpretation of locales in Isabelle: Theories and proof contexts. In J. Borwein and W. Farmer, editors, *Mathematical Knowledge Management*, volume 4108 of *LNCS*, pages 31–43. Springer, 2006.
2. Robert S. Boyer and J Strother Moore. Metafunctions: proving them correct and using them efficiently as new proof procedures. In R. Boyer and J Moore, editors, *The Correctness Problem in Computer Science*, pages 103–184. Academic Press, 1981.
3. Amine Chaieb. Verifying mixed real-integer quantifier elimination. In U. Furbach and N. Shankar, editors, *Automated Reasoning (IJCAR 2006)*, volume 4130 of *LNCS*, pages 528–540. Springer, 2006.
4. Amine Chaieb and Tobias Nipkow. Verifying and reflecting quantifier elimination for Presburger arithmetic. In G. Stutcliffe and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2005)*, volume 3835 of *LNCS*, pages 367–380. Springer, 2005.
5. D.C. Cooper. Theorem proving in arithmetic without multiplication. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 7, pages 91–100. Edinburgh University Press, 1972.
6. Herbert Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

7. Julius Farkas. Theorie der einfachen Ungleichungen. *Journal für die reine und angewandte Mathematik*, 124:1–27, 1902.
8. Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM J. Computing*, 4:69–76, 1975.
9. Jean Baptiste Joseph Fourier. Solution d'une question particulière du calcul des inégalités. In Gaston Darboux, editor, *Joseph Fourier - Œuvres complétes*, volume 2, pages 317–328. Gauthier-Villars, 1888–1890.
10. Georges Gonthier. Formal proof—the four-colour theorem. *Notices of the AMS*, 55:1382–1393, 2008.
11. Florian Haftmann and Makarius Wenzel. Constructive type classes in Isabelle. In Th. Altenkirch and C. McBride, editors, *Types for Proofs and Programs (TYPES 2006)*, volume 4502 of *LNCS*, pages 160–174. Springer, 2007.
12. John Harrison. Complex quantifier elimination in HOL. In R. Boulton and P. Jackson, editors, *TPHOLs 2001: Supplemental Proceedings*, pages 159–174. Division of Informatics, University of Edinburgh, 2001.
13. John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.
14. C.H. Langford. Some theorems on deducibility. *Annals of Mathematics (2nd Series)*, 28:16–40, 1927.
15. Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36:450–462, 1993.
16. Assia Mahboubi. *Contributions à la certification des calculs sur $\mathbb{R}$: théorie, preuves, programmation*. PhD thesis, Université de Nice, 2006.
17. Sean McLaughlin and John Harrison. A proof-producing decision procedure for real arithmetic. In R. Nieuwenhuis, editor, *Automated Deduction — CADE-20*, volume 3632 of *LNCS*, pages 295–314. Springer, 2005.
18. David Monniaux. A quantifier elimination algorithm for linear real arithmetic. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2008)*, volume 5330 of *LNCS*, pages 243–257. Springer, 2008.
19. Theodor Motzkin. *Beiträge zur Theorie der linearen Ungleichungen*. PhD thesis, Universität Basel, 1936.
20. Tobias Nipkow. Linear quantifier elimination. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning (IJCAR 2008)*, volume 5195 of *LNCS*, pages 18–33. Springer, 2008.
21. Tobias Nipkow. Reflecting quantifier elimination for linear arithmetic. In O. Grumberg, T. Nipkow, and C. Pfaller, editors, *Formal Logical Methods for System Security and Correctness*, pages 245–266. IOS Press, 2008.
22. Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
23. Michael Norrish. Complete integer decision procedures as derived rules in HOL. In D. Basin and B. Wolff, editors, *Theorem Proving in Higher Order Logics, TPHOLs 2003*, volume 2758 of *LNCS*, pages 71–86. Springer, 2003.
24. Steven Obua. Proving bounds for real linear programs in Isabelle/HOL. In J. Hurd, editor, *Theorem Proving in Higher Order Logics (TPHOLs 2005)*, volume 3603 of *LNCS*, pages 227–244. Springer, 2005.
25. Mojzesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I Congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.
26. Volker Weispfenning. The complexity of linear problems in fields. *J. Symbolic Computation*, 5:3–27, 1988.