Reflecting Quantifier Elimination for Linear Arithmetic

Tobias NIPKOW
Institut für Informatik, TU München

Abstract. This paper formalizes and verifies quantifier elimination procedures for dense linear orders and for real and integer linear arithmetic in the theorem prover Isabelle/HOL. It is a reflective formalization because it can be applied to HOL formulae themselves. In particular we obtain verified executable decision procedures for linear arithmetic. The formalization for the various theories is modularized with the help of *locales*, a structuring facility in Isabelle.

1. Introduction

This research is about adding decision procedures to theorem provers in a reliable manner, i.e. without having to trust the decision procedure. The traditional LCF approach [16] involves programming the decision procedure in the implementation language of the theorem prover using the basic inference rules of the logic. This is safe but tricky to write and maintain. There are two alternatives: checking externally generated certificates (for an example see [22]), and *reflection*, i.e. the formalization and verification of the decision procedure in the logic itself. The focus of this paper is reflection, partly because the theories we consider do not lend themselves to certificate checking: there are no short certificates, i.e. checking the certificates is as expensive as generating them in the first place.

The mathematical subject matter of the paper is quantifier elimination, i.e. the process of computing a quantifier-free equivalent of a quantified formula, yielding in particular a decision procedure for closed formulae. Many numeric theories enjoy quantifier elimination. The most celebrated instance is quantifier elimination for real closed fields, i.e. $(\mathbb{R}, +, *)$, due to Tarski [31]. We reflect quantifier elimination procedures for dense linear orders, and linear real and integer arithmetic.

Everything has been formalized and verified in the logic HOL (higher-order logic) of the theorem prover Isabelle [27] and is available online in the Archive of Formal Proofs at afp.sf.net. In particular we have made use of *locales*, a structuring facility akin to parameterized theories. Locales are a fairly recent addition to Isabelle [1,2] and this article demonstrates locales in a serious application.

In summary, the article makes the following contributions:

1. A detailed exposition of a formalization of quantifier elimination for linear real and integer arithmetics in HOL.
2. Reflective implementations of quantifier elimination.

3. A modular development based on locales.

Note that our presentation aims for simplicity and minimality of concepts, not for practical efficiency. For example, we restrict to as few atomic propositions as possible, typically by having only one of $\leq$ and $<$. In practice one would avoid this as it tends to lead to inefficiencies due to coding, e.g. if $s = t$ is replaced by $s \leq t \wedge t \leq s$. Nevertheless our presentation provides a convenient starting point for more efficient implementations, as demonstrated elsewhere [8], where the reflective implementation is two orders of magnitude faster than the LCF approach.

The core of the article is structured as follows: We start with an abstract generic account of logical formulae (§4); quantifier elimination is given as a locale parametric in the specific logical theory of interest. This locale is instantiated four times: for dense linear orders (§5), for linear real arithmetic (Fourier-Motzkin elimination in §6.2 and Ferrante and Rackoff's procedure in §6.4), and for linear integer arithmetic (§7).

2. Reflection, Informally

Reflection means to perform a proof step by computation inside the logic rather than inside some external programming language (ML). Inside the logic it is not possible to write functions by pattern matching over the syntax of formulae because two syntactically distinct formulae may be logically equivalent. Hence the relevant fragment of formulae must be represented (*reflected*) inside the logic as a datatype. We call it *rep*, the representation.

The two levels of formulae must be connected by two functions:

I (a HOL function) maps an element of *rep* to the formula it represents, and
$reify$ (an ML function) maps a formula to its representation.

The two functions should be inverses of each other. Informally $I(\mathit{reify}(P)) = P$ should hold. More precisely, taking the ML representation of a formula P and applying *reify* to it yields an ML representation of a term p of type *rep* such that $I(p) = P$ holds.

Typically, the formalized proof step is some equivalence $P \leftrightarrow P'$ where P is given and P' is some simplified version of P (e.g. the elimination of quantifiers). This transformation is now expressed as a recursive function *simp* of type *rep* $\rightarrow$ *rep*. We prove (typically by induction on *rep*) that *simp* preserves the interpretation: $I(\mathit{simp}(p)) \leftrightarrow I(p)$. To apply this theorem to a given formula P we proceed as follows:

1. Create a *rep*-term p from P using *reify*. This *reification* step must be performed in ML.
2. Prove $P \leftrightarrow I(p)$. Usually this is trivial by rewriting with the definition of I.
3. Instantiate *simp*'s correctness theorem $I(\mathit{simp}(p)) \leftrightarrow I(p)$, compute the result p' of evaluating $\mathit{simp}(p)$ and obtain the theorem $I(p') \leftrightarrow I(p)$ (and by symmetry $I(p) \leftrightarrow I(p')$). This is the *evaluation* step.
4. Simplify $I(p')$, again by rewriting with the definition of I, yielding a theorem $I(p') \leftrightarrow P'$

The final theorem $P \leftrightarrow P'$ holds by transitivity.

The evaluation step is crucial for efficiency as all other steps are typically linear-time. We employ Isabelle's recent code generator [18] for compiling and evaluating

simp(p) in ML. Other approaches include evaluation via LISP (Boyer and Moore's "metafunctions" [5], the mother of all reflections) and the use of an internal λ-calculus evaluator [17] as in Coq.

There is also the practical issue of where *reify* comes from. In general, the implementor of the reflected proof procedure must program it in ML and link it into the above chain of deductions. But because *reify* must be the inverse of I, it is often possible to automate this step. Isabelle implements a sufficiently general inversion scheme for I such that for all of the examples in this paper, *reify* is performed automatically.

In principle the reader may now forget about the details of reflection and merely keep in mind that all the algorithms in this paper, although expressed on some representation of formulae, carry over to HOL formulae automatically.

3. 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 basic 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 convention.

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, with minor differences: instead of `[e | x <- xs, ...]` we write $[e.\ x \leftarrow xs, \ldots]$, and $[x{\leftarrow}xs.\ \ldots]$ is short for $[x.\ x{\leftarrow}xs, \ldots]$.

Finally note that $=$ on type *bool* means "iff".

Although all our algorithms and formal theorems conform to HOL syntax, we frequently switch to everyday mathematical notation during informal explanations.

4. Logic

The data type of formulae is defined in the usual manner:

$$\begin{array}{rcl} \alpha\ fm & = & \mathit{TrueF} \mid \mathit{FalseF} \mid \mathit{Atom}\ \alpha \\ & \mid & \mathit{And}\ (\alpha\ fm)\ (\alpha\ fm) \\ & \mid & \mathit{Or}\ (\alpha\ fm)\ (\alpha\ fm) \\ & \mid & \mathit{Neg}\ (\alpha\ fm) \\ & \mid & \mathit{ExQ}\ (\alpha\ fm) \end{array}$$

This representation provides the customary logical operators but leaves the type of atoms 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, *ExQ* (*ExQ* $\ldots$ *0* $\ldots$ *1* $\ldots$) represents a formula $\exists x_1.\exists x_0.\ \ldots x_0 \ldots x_1 \ldots$. 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 binders.

Further logical operators can be introduced as abbreviations, in particular *AllQ* $\varphi \equiv$ *Neg* (*ExQ* (*Neg* φ)).

4.1. Auxiliary Functions

The set of atoms is computed by the (easy to define) function

$$\mathit{atoms} :: \alpha\ \mathit{fm} \Rightarrow \alpha\ \mathit{set}.$$

Conjunctions and disjunctions of lists of formulae are created by the functions

$$\begin{array}{ll} \mathit{list\text{-}conj} & :: \alpha\ \mathit{fm}\ \mathit{list} \Rightarrow \alpha\ \mathit{fm} \\ \mathit{list\text{-}disj} & :: \alpha\ \mathit{fm}\ \mathit{list} \Rightarrow \alpha\ \mathit{fm} \end{array}$$

Their definition is straightforward:

list-conj $[\varphi_1,\ldots,\varphi_n] =$ *and* φ_1 (*and* $\ldots$ φ_n)

where *and* is an intelligent version of *And*:

$$\begin{array}{ll} \mathit{and\ FalseF}\ \varphi & = \mathit{FalseF} \\ \mathit{and}\ \varphi\ \mathit{FalseF} & = \mathit{FalseF} \\ \mathit{and\ TrueF}\ \varphi & = \varphi \\ \mathit{and}\ \varphi\ \mathit{TrueF} & = \varphi \\ \mathit{and}\ \varphi_1\ \varphi_2 & = \mathit{And}\ \varphi_1\ \varphi_2 \end{array}$$

Similar for *list-disj* and *or*, an optimized version of *Or*. For convenience the following abbreviation is introduced:

Disj us f $\equiv$ *list-disj* (*map f us*)

More interesting is the conversion to DNF:

$$\begin{array}{l} \mathit{dnf} :: \alpha\ \mathit{fm} \Rightarrow \alpha\ \mathit{list}\ \mathit{list} \\ \\ \mathit{dnf\ TrueF} = [[]] \\ \mathit{dnf\ FalseF} = [] \\ \mathit{dnf}\ (\mathit{Atom}\ \varphi) = [[\varphi]] \\ \mathit{dnf}\ (\mathit{Or}\ \varphi_1\ \varphi_2) = \mathit{dnf}\ \varphi_1\ @\ \mathit{dnf}\ \varphi_2 \\ \mathit{dnf}\ (\mathit{And}\ \varphi_1\ \varphi_2) = [d_1\ @\ d_2.\ d_1 \leftarrow \mathit{dnf}\ \varphi_1, d_2 \leftarrow \mathit{dnf}\ \varphi_2] \end{array}$$

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 (easy to define) predicates

$$\begin{array}{ll} \mathit{qfree} & :: \alpha\ \mathit{fm} \Rightarrow \mathit{bool} \\ \mathit{nqfree} & :: \alpha\ \mathit{fm} \Rightarrow \mathit{bool} \end{array}$$

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

There is also a mapping functional

$$map_{fm} :: (\alpha \Rightarrow \beta) \Rightarrow \alpha\ fm \Rightarrow \beta\ fm$$

which recurses down a formula, e.g.

$map_{fm}\ h\ (And\ \varphi_1\ \varphi_2) = And\ (map_{fm}\ h\ \varphi_1)\ (map_{fm}\ h\ \varphi_2)$

until it finds an atom: $map_{fm}\ h\ (Atom\ a) = Atom\ (h\ a)$.

4.2. Interpretation

The interpretation or semantics of a *fm* is defined via the obvious homomorphic mapping to an HOL formula: *And* becomes $\wedge$, *Or* 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:

$interpret :: (\alpha \Rightarrow \beta\ list \Rightarrow bool) \Rightarrow \alpha\ fm \Rightarrow \beta\ list \Rightarrow bool$

$interpret\ h\ TrueF\ xs = True$
$interpret\ h\ FalseF\ xs = False$
$interpret\ h\ (Atom\ a)\ xs = h\ a\ xs$
$interpret\ h\ (And\ \varphi_1\ \varphi_2)\ xs = (interpret\ h\ \varphi_1\ xs \wedge interpret\ h\ \varphi_2\ xs)$
$interpret\ h\ (Or\ \varphi_1\ \varphi_2)\ xs = (interpret\ h\ \varphi_1\ xs \vee interpret\ h\ \varphi_2\ xs)$
$interpret\ h\ (Neg\ \varphi)\ xs = (\neg\ interpret\ h\ \varphi\ xs)$
$interpret\ h\ (ExQ\ \varphi)\ xs = (\exists x.\ interpret\ h\ \varphi\ (x{\cdot}xs))$

In the equation for *ExQ* 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.

4.3. 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

$$\begin{array}{ll} I_a & :: \alpha \Rightarrow \beta\ list \Rightarrow bool \\ aneg & :: \alpha \Rightarrow \alpha\ fm \\ depends_0 & :: \alpha \Rightarrow bool \\ decr & :: \alpha \Rightarrow \alpha \end{array}$$

with the following intended meaning:

I_a *a xs* is the interpretation of atom *a* w.r.t. valuation *xs*, where variable *i* (*i* :: *nat*!) is assigned the *i*th 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, and *decr a* decrements every variable in *a* by 1.

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

$$I\ (aneg\ a)\ xs = (\neg\ I_a\ a\ xs) \qquad nqfree\ (aneg\ a)$$
$$\neg\ depends_0\ a \Longrightarrow I_a\ a\ (x{\cdot}xs) = I_a\ (decr\ a)\ xs$$

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.

The *negation normal form* (NNF) of a formula is defined in the customary manner by pushing negations inwards. We show only a few representative equations:

$$nnf :: \alpha\ fm \Rightarrow \alpha\ fm$$
$$nnf\ (Neg\ (Atom\ a)) = aneg\ a$$
$$nnf\ (Or\ \varphi_1\ \varphi_2) = Or\ (nnf\ \varphi_1)\ (nnf\ \varphi_2)$$
$$nnf\ (Neg\ (Or\ \varphi_1\ \varphi_2)) = And\ (nnf\ (Neg\ \varphi_1))\ (nnf\ (Neg\ \varphi_2))$$
$$nnf\ (Neg\ (And\ \varphi_1\ \varphi_2)) = Or\ (nnf\ (Neg\ \varphi_1))\ (nnf\ (Neg\ \varphi_2))$$

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

The fact that *nnf* preserves interpretations is a trivial inductive consequence of the assumptions about the locale parameters: $I\ (nnf\ \varphi)\ xs = I\ \varphi\ xs$.

4.4. 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 defined once and for all. We assume we are given a function $qe :: \alpha\ fm \Rightarrow \alpha\ fm$ for the elimination of a single *ExQ*, i.e. $I\ (qe\ \varphi) = I\ (ExQ\ \varphi)$ if $qfree\ \varphi$. Note that *qe* is not applied to $ExQ\ \varphi$ but just to φ, *ExQ* remains implicit. Lifting *qe* is straightforward:

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

$$\begin{array}{l}
\textit{lift-nnf-qe qe}\ (\textit{And}\ \varphi_1\ \varphi_2) = \textit{and}\ (\textit{lift-nnf-qe qe}\ \varphi_1)\ (\textit{lift-nnf-qe qe}\ \varphi_2) \\
\textit{lift-nnf-qe qe}\ (\textit{Or}\ \varphi_1\ \varphi_2) = \textit{or}\ (\textit{lift-nnf-qe qe}\ \varphi_1)\ (\textit{lift-nnf-qe qe}\ \varphi_2) \\
\textit{lift-nnf-qe qe}\ (\textit{Neg}\ \varphi) = \textit{neg}\ (\textit{lift-nnf-qe qe}\ \varphi) \\
\textit{lift-nnf-qe qe}\ (\textit{ExQ}\ \varphi) = \textit{qe}\ (\textit{nnf}\ (\textit{lift-nnf-qe qe}\ \varphi)) \\
\textit{lift-nnf-qe qe}\ \varphi = \varphi
\end{array}$$

To simplify life for *qe* we put its argument into NNF.

We can go even further and put the argument of *qe* into DNF because then we can pull the disjunction out of the existential quantifier 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:

$$\begin{array}{l}
\textit{qelim qe as} = (\textsf{let}\ \textit{qf} = \textit{qe}\ [a \leftarrow \textit{as}.\ \textit{depends}_0\ a]; \\
\qquad\qquad \textit{indep} = [\textit{Atom}(\textit{decr}\ a).\ a \leftarrow \textit{as}, \neg\ \textit{depends}_0\ a] \\
\qquad \textsf{in}\ \textit{and qf}\ (\textit{list-conj indep}))
\end{array}$$

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 *ExQ* φ, removes the quantifiers from φ, puts the result into NNF and DNF, and applies *qelim qe* to each disjunct:

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

4.4.1. 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: $A \to B$ is the set of functions from A to B, *lists* A the set of lists over A, $-A$ the complement of A, and $|P| \equiv \{x \mid P\,x\}$.

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

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

Preservation of the interpretation is slightly more involved:

Lemma 2 *If* $qe \in |nqfree| \to |qfree|$ *and* $nqfree\ \varphi \Longrightarrow I\ (qe\ \varphi)\ xs = (\exists x.\ I\ \varphi\ (x{\cdot}xs))$ *for all* φ *and xs, then* $I\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$.

For *lift-dnf-qe* the statements are a bit more involved still, 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.

Lemma 3 If $qe \in lists\ |depends_0| \to |qfree|$ then $qfree\ (lift\text{-}dnf\text{-}qe\ qe\ \varphi)$.

Lemma 4 *If* $qe \in lists\ |depends_0| \to |qfree|$ *and* $\forall as{\in}\ lists\ |depends_0|$. *is-dnf-qe qe as, then* $I\ (lift\text{-}dnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$.

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

All proofs are straightforward inductions using a number of additional lemmas.

4.4.2. Complexity

Conversion to DNF may (unavoidably) cause exponential blowup. Since this can happen every time a quantifier is eliminated, even if *qe* runs in linear time, the worst case running time of *lift-dnf-qe qe* is non-elementary in the size of the formula, i.e. a tower of exponents $2^{\cdot^{\cdot^{\cdot^{2}}}}$ whose height is the size of the formula. In contrast, conversion to NNF is linear. This leads to more reasonable upper bounds. For example, if *qe* takes quadratic time, the worst case running time of *lift-nnf-qe qe* is only doubly exponential. Thus we have the choice between an essentially infeasible lifting function *lift-dnf-qe* which allows each quantifier elimination step to focus on conjunctions of atoms, or a potentially feasible lifting function *lift-nnf-qe* which requires each quantifier elimination step to deal with arbitrary combinations of conjunctions and disjunctions.

4.4.3. 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 two of our theories this will enable 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 nonlinear 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

$$\begin{array}{ll} \mathit{solvable}_0 & :: \alpha \Rightarrow \mathit{bool} \\ \mathit{trivial} & :: \alpha \Rightarrow \mathit{bool} \\ \mathit{subst}_0 & :: \alpha \Rightarrow \alpha \Rightarrow \alpha \end{array}$$

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: $\mathit{solvable}_0\ \mathit{eq} \Longrightarrow \exists\, e.\ I_a\ \mathit{eq}\ (e{\cdot}\mathit{xs})$.
- 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 the substitution lemma must hold 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):

$\mathit{lift\text{-}eq\text{-}qe}\ \mathit{qe}\ \mathit{as} =$
$(\mathsf{let}\ \mathit{as} = [a{\leftarrow}\mathit{as}.\ \neg\ \mathit{trivial}\ a]$
$\ \mathsf{in}\ \mathsf{case}\ [a{\leftarrow}\mathit{as}.\ \mathit{solvable}_0\ a]\ \mathsf{of}$
$\quad [] \Rightarrow \mathit{qe}\ \mathit{as}$
$\ \mid \mathit{eq} \cdot \mathit{eqs} \Rightarrow (\mathsf{let}\ \mathit{ineqs} = [a{\leftarrow}\mathit{as}.\ \neg\ \mathit{solvable}_0\ a]$
$\qquad\qquad \mathsf{in}\ \mathit{list\text{-}conj}\ (\mathit{map}\ (\mathit{Atom} \circ \mathit{subst}_0\ \mathit{eq})\ (\mathit{eqs}\ @\ \mathit{ineqs}))))$

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* $\forall\, \mathit{as} \in \mathit{list}(|\mathit{depends}_0| \cap -|\mathit{solvable}_0|).$ *is-dnf-qe qe as then* $\forall\, \mathit{as} \in \mathit{list}|\mathit{depends}_0|.$ *is-dnf-qe* $(\mathit{lift\text{-}eq\text{-}qe}\ \mathit{qe})$ *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*:

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

and obtain a corollary to lemmas 4 and 5:

Corollary 1 *If* $\mathit{qe} \in \mathit{lists}\ |\mathit{depends}_0| \rightarrow |\mathit{qfree}|$ *and* $\forall\, \mathit{as} \in \mathit{lists}(|\mathit{depends}_0| \cap -|\mathit{solvable}_0|).$ *is-dnf-qe qe as then* $I\ (\mathit{lift\text{-}dnfeq\text{-}qe}\ \mathit{qe}\ \varphi)\ \mathit{xs} = I\ \varphi\ \mathit{xs}.$

In the same manner we obtain

Corollary 2 *If* $\mathit{qe} \in \mathit{list}\ |\mathit{depends}_0| \rightarrow |\mathit{qfree}|$ *then qfree* $(\mathit{lift\text{-}dnfeq\text{-}qe}\ \mathit{qe}\ \varphi).$

5. Dense Linear Orders

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

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

It is the canonical example of quantifier elimination [23] and the basis for the arithmetic theories to come. The equivalence $(\exists x.\ y < x \wedge x < z) = (y < z)$ is an easy consequence of the axioms. 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:

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

The *only-if* direction holds by transitivity. The *if* direction follows because the right-hand formula is just another way of saying that the maximum of the l_i is less than the minimum of the u_j. By denseness there must exists a value in between, which is the witness for the existential formula.

Now we formalize this theory and its quantifier elimination procedure. We concentrate on quantifier elimination via DNF, thus obtaining a non-elementary procedure.

5.1. Atoms

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

$$\mathit{atom} = \mathit{Less\ nat\ nat} \mid \mathit{Eq\ nat\ nat}$$

Because there are no function symbols, the arguments of the relations must be variables. For example, *Less i j* represents the atom $x_i < x_j$ in de Bruijn notation. We define two auxiliary predicates *is-Less* and *is-Eq* which do what their name suggests.

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

$I_{dlo}\ (\mathit{Eq\ i\ j})\ \mathit{xs} = (\mathit{xs}_{[i]} = \mathit{xs}_{[j]})$
$I_{dlo}\ (\mathit{Less\ i\ j})\ \mathit{xs} = (\mathit{xs}_{[i]} < \mathit{xs}_{[j]})$

The notation $\mathit{xs}_{[i]}$ means selection of the *i*th 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 [19] 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}:

$\mathit{neg}_{dlo}\ (\mathit{Less\ i\ j}) = \mathit{Or}\ (\mathit{Atom}\ (\mathit{Less\ j\ i}))\ (\mathit{Atom}\ (\mathit{Eq\ i\ j}))$
$\mathit{neg}_{dlo}\ (\mathit{Eq\ i\ j}) = \mathit{Or}\ (\mathit{Atom}\ (\mathit{Less\ i\ j}))\ (\mathit{Atom}\ (\mathit{Less\ j\ i}))$

The instantiation of the parameters *adepends* and *adecr* is obvious:

$depends_{dlo}\ (Eq\ i\ j) = (i = 0 \vee j = 0)$
$depends_{dlo}\ (Less\ i\ j) = (i = 0 \vee j = 0)$

$decr_{dlo}\ (Less\ i\ j) = Less\ (i - 1)\ (j - 1)$
$decr_{dlo}\ (Eq\ i\ j) = Eq\ (i - 1)\ (j - 1)$

It is straightforward to show that this instantiation satisfies all the axioms of *ATOM*. The extension to *ATOM-EQ* (see §4.4.3) is easy: $solvable_0$ becomes $\lambda Eq\ i\ j \Rightarrow i{=}0 \vee j{=}0 \mid a \Rightarrow False$, *trivial* becomes $\lambda Eq\ i\ j \Rightarrow i{=}j \mid a \Rightarrow False$ and $subst_0$ is defined as follows:

$subst_0\ (Eq\ i\ j)\ (Less\ m\ n) = Less\ (subst\ i\ j\ m)\ (subst\ i\ j\ n)$
$subst_0\ (Eq\ i\ j)\ (Eq\ m\ n) = Eq\ (subst\ i\ j\ m)\ (subst\ i\ j\ n)$

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

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

5.2. Quantifier Elimination

The quantifier elimination procedure sketched above assumes that it is given a list, i.e. conjunction of atoms. Variable 0, the innermost one, is to be eliminated. Because *lift-dnfeq-qe* already takes care of equalities, we can concentrate on the case where all atoms are *Less*:

qe-less as =
  (*if Less 0 0* ∈ *set as* **then** *FalseF* **else**
   let *lbs* = [*i*. *Less* (*Suc i*) *0* ← *as*];
       ubs = [*j*. *Less 0* (*Suc j*) ← *as*];
       pairs = [*Atom*(*Less i j*). *i* ← *lbs*, *j* ← *ubs*]
   in *list-conj pairs*)

This is exactly the above informal algorithm, 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 *FalseF* upon finding any $x_i < x_i$.

5.3. Correctness

Theorem 1 $\forall\, a{\in}set\ as.\ is\text{-}Less\ a \wedge depends_{dlo}\ a \Longrightarrow is\text{-}dnf\text{-}qe\ qe\text{-}less\ as$

Remember that *is-dnf-qe* abbreviates an equivalence (see §4.4.1). The proof of the $\rightarrow$-direction of the equivalence distinguishes whether *lbs* or *ubs* are empty (in which case the lack of endpoints guarantees the existence of *x*) or not (in which case density comes to the rescue). The other direction follows via transitivity.

Defining *dlo-qe* = *lift-dnfeq-qe qe-less* we obtain the main result

Corollary 3 $I\ (dlo\text{-}qe\ \varphi)\ xs = I\ \varphi\ xs$

as a consequence of Corollary 1, Theorem 1 and the lemma *qfree* (*qe-less as*).

6. 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 number work just as well. In fact, any ordered, divisible, torsion free, Abelian group will do.

This time we will present two quantifier elimination procedures: one resembling the one for DLO, so called Fourier-Motzkin elimination, and a clever algorithm due to Ferrante and Rackoff [14] which brings the complexity down from non-elementary to doubly exponential.

6.1. Atoms

Type *atom* formalizes the normal forms explained above:

$$\mathit{atom} = \mathit{Less\ real}\ (\mathit{real\ list}) \mid \mathit{Eq\ real}\ (\mathit{real\ list})$$

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 on two 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 [28]). We can prove all the algebraic properties we need, like $\langle xs + ys,zs\rangle = \langle xs,zs\rangle + \langle ys,zs\rangle$.

Now we instantiate locale *ATOM* just like for DLO in §5.1. The main function is the interpretation I_R of atoms, which is straightforward:

$I_R\ (\mathit{Less\ r\ cs})\ xs = (r < \langle cs,xs\rangle)$
$I_R\ (\mathit{Eq\ r\ cs})\ xs = (r = \langle cs,xs\rangle)$

Although this is irrelevant in our context, note that our lists do not form a vector space. Clearly the 0 vector would have to be [], but then there are almost no inverses: $[x] + [-x]$ is $[0]$ but not []. For a vectors space we would need to remove trailing 0s after each operation. And certain laws like $xs + 0 = xs$ would only hold for vectors without trailing 0s. A proper treatment of vectors spaces requires lists of different lengths to have different types. A solution along these lines is given by Harrison [21].

It is easy to extend the instantiation of *ATOM* to *ATOM-EQ* (see §4.4.3): $\mathit{solvable}_0$ is any *Eq* whose head coefficient is nonzero ($\lambda \mathit{Eq\ r\ (c{\cdot}cs)} \Rightarrow c \neq 0 \mid a \Rightarrow \mathit{False}$), *trivial* is any *Eq* where both sides are zero ($\lambda \mathit{Eq\ r\ cs} \Rightarrow r{=}0 \wedge (\forall\, c \in \mathit{set\ cs}.\ c{=}0) \mid a \Rightarrow \mathit{False}$), and subst_0 is defined as follows:

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

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

6.2. Fourier-Motzkin Elimination

Fourier-Motzkin Elimination is a procedure discovered by Fourier [15].[1] Essentially, it works like for dense linear orders. 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). 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 are of the form *Less*, is a one-liner

qe-less as = *list-conj* [*Atom*(*combine p q*). *p*←*lbounds as*, *q*←*ubounds as*]

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

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

and they are combined as explained above:

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

The correctness theorem

Theorem 2 $\forall a{\in}set\ as.\ is\text{-}Less\ a \wedge depends_R\ a \Longrightarrow is\text{-}dnf\text{-}qe\ qe\text{-}less\ as$

is proved along the same lines as its counterpart Theorem 1, except that linear arithmetic reasoning is necessary now.

The extension with equality is provided by locale *ATOM-EQ*. Defining *lin-qe* = *lift-dnfeq-qe qe-less* we obtain the main result

Corollary 4 $I\ (lin\text{-}qe\ \varphi)\ xs = I\ \varphi\ xs$

as a consequence of Corollary 1, Theorem 2 and the lemma *qfree* (*qe-less as*).

Above we transformed inequalities into $l < x$ and $x < u$ by dividing with the coefficient of x. Alternatively 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.

[1] Motzkin [26] and Farkas [13] cited Fourier but were concerned with the algebraic background, not the algorithm.

6.3. An Optimization

The above code is correct but produces horribly bloated results: even if the initial formula is closed, the result will not just be *TrueF* or *FalseF* but some complicated formula equivalent to that. As a trivial example take $\exists x.1 < x \wedge x < 2$. It is converted to $1 < 2$. To be able to cope with larger inputs, it is essential to simplify intermediate results as much as possible: at the very least, unsatisfiable atoms should be replaced by *FalseF* and tautological ones by *TrueF*. This is very easy to spot: *Less r cs* is unsatisfiable/tautological iff all elements of *cs* are 0 and $r \geq 0 / r < 0$. Here is a corresponding function which simplifies individual atoms to *TrueF* or *FalseF* whenever it can:

asimp (*Less r cs*) =
(*if* $\forall$ *c*∈*set cs*. *c* = *0* *then if* *r* < *0* *then* *TrueF* *else* *FalseF* *else* *Atom* (*Less r cs*))
asimp (*Eq r cs*) =
(*if* $\forall$ *c*∈*set cs*. *c* = *0* *then if* *r* = *0* *then* *TrueF* *else* *FalseF* *else* *Atom* (*Eq r cs*))

This simplification is applied when lower and upper bounds are combined:

qe-less′ as = *list-conj* [*asimp*(*combine p q*). *p*←*lbounds as*, *q*←*ubounds as*]

The definition of *list-conj* via *and* ensures that any *TrueF* is dropped and any *FalseF* propagates to the output.

It is not hard to prove that *I* (*qe-less′ as*) *xs* = *I* (*qe-less as*) *xs*, from which the analogous version of Corollary 4 for *lin-qe′* = *lift-dnfeq-qe qe-less′* instead of *lin-qe* follows easily.

6.4. Ferrante and Rackoff

Fourier-Motzkin elimination has non-elementary complexity because of the repeated DNF conversions. Ferrante and Rackoff [14], inspired by Cooper [11], avoid putting the formula explicitly into DNF but still capitalize on the fact that it has a DNF. Below, let ϕ be some quantifier-free formula with a free variable x. Substituting x by some r is written $\phi(r)$.

When eliminating x from ϕ, we can partition the atoms of ϕ that depend on x into 3 categories: $l < x$, $x < u$ and $x = t$. Let $LB(\phi)$ denote the set of all such l in ϕ, $UB(\phi)$ the set of such u, and $EQ(\phi)$ the set of such t. The DNF of a formula over these atoms can be seen as a finite union of finite intersections of half-open intervals (l, ∞) and $(-\infty, u)$ and points t. Each such intersection is equivalent to either a single interval (l, ∞), $(-\infty, u)$ or (l, u), or to a point t — or it is empty, in which case we can silently forget about it. Thus there are 4 possibilities why ϕ can hold: $\phi(x)$ holds for any sufficiently large x (case (l, ∞)), $\phi(x)$ holds for any sufficiently small x (case $(-\infty, u)$), $\phi(x)$ holds for all $x \in (l, u)$ for some $l \in LB(\phi)$ and $u \in UB(\phi)$, or $\phi(t)$ for some $t \in EQ(\phi)$. This leads to the following optimized version of the equivalence due to Ferrante and Rackoff: [2]

[2]The special treatment of equality is missing in Ferrante and Rackoff's work, probably to simplify matters. The asymptotic complexity remains unaffected.

$$(\exists x.\phi(x)) \;=\; (\phi(-\infty) \vee \phi(\infty) \vee \bigvee_{\substack{l\in LB(\phi)\\ u\in UB(\phi)}} \phi(\frac{l+u}{2}) \vee \bigvee_{t\in EQ(\phi)} \phi(t)) \tag{3}$$

The choice of $(l+u)/2$ is arbitrary: any value in (l,u) will do.

Notation $\phi(-\infty)$ and $\phi(\infty)$ is merely suggestive syntax for the following form of "substitution":

$$\begin{array}{ll} (-\infty < u) = True & (\infty < u) = False \\ (l < -\infty) = False & (l < \infty) = True \\ (-\infty = t) = False & (\infty = t) = False \end{array}$$

Ferrante and Rackoff only sketch the proof of (3). We examine some of the delicate details. The proof of the $\leftarrow$-direction is obvious in case the witness is $(l+u)/2$ or t. For $-\infty$ and ∞, the following lemmas provide the witness:

$$\exists x.\forall y \leq x.\ \phi(-\infty) = \phi(y) \qquad \exists x.\forall y \geq x.\ \phi(\infty) = \phi(y)$$

They are proved by induction on ϕ.

The proof of the $\rightarrow$-direction is more subtle. We have $\phi(x)$. Assuming $x \notin EQ(\phi)$, $\neg\phi(-\infty)$ and $\neg\phi(\infty)$, we have to show that $\phi((l+u)/2)$ for some $l \in LB(\phi)$ and $u \in UB(\phi)$. In fact, we show that there are l and u such that $l < u$ and $\phi(y)$ for all $y \in (l,u)$. From the assumptions it follows by induction on ϕ that there must be $l_0 \in LB(\phi)$ and $u_0 \in UB(\phi)$ such that $x \in (l_0,u_0)$. Now we show (by induction on ϕ) the lemma that "innermost" intervals (l,u) completely satisfy ϕ:

Lemma 6 *If* $\phi(x)$, $x \in (l,u)$, $x \notin EQ(\phi)$, $(l,x) \cap LB(\phi) = \emptyset$ *and* $(x,u) \cap UB(\phi) = \emptyset$, *then* $\forall y \in (l,u).\ \phi(y)$.

Given $x \in (l_0,u_0)$ we define $l = \max\{l \in LB(\phi) \mid l < x\}$ and $u = \min\{u \in UB(\phi) \mid x < u\}$. It is easy to see that this satisfies the premises of the lemma and the desired conclusion follows.

Now we describe the implementation of Ferrante and Rackoff's procedure, starting at the top with (3):

```
FR1 φ = (let as = atoms0 φ; lbs = lbounds as; ubs = ubounds as;
             intvs = [subst φ (between p q) . p ← lbs, q ← ubs];
             eqs = [subst φ rcs . rcs ← ebounds as]
         in list-disj (inf_ φ · inf+ φ · intvs @ eqs))
```

Function FR_1 expects a formula φ in NNF. Function $atoms_0$ collects the atoms of φ that depend on variable 0. Functions LB, UB and EQ are realized by *lbounds*, *ubounds* (see above) and *ebounds*:

$$ebounds\ as = [(r/c,\ (-1/c) *_s cs).\ Eq\ r\ (c{\cdot}cs) \leftarrow as,\ c{\neq}0]$$

Function *between* picks the mid-point between two points:

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

Substitution, as usual for variable 0, is first defined for atoms

$$\begin{array}{l}
\mathit{asubst}\ (r, cs)\ (\mathit{Less}\ s\ (d{\cdot}ds)) = \mathit{Less}\ (s - d * r)\ (d *_s cs + ds) \\
\mathit{asubst}\ (r, cs)\ (\mathit{Eq}\ s\ (d{\cdot}ds)) = \mathit{Eq}\ (s - d * r)\ (d *_s cs + ds) \\
\mathit{asubst}\ (r, cs)\ (\mathit{Less}\ s\ []) = \mathit{Less}\ s\ [] \\
\mathit{asubst}\ (r, cs)\ (\mathit{Eq}\ s\ []) = \mathit{Eq}\ s\ []
\end{array}$$

and then lifted to formulae: $\mathit{subst}\ \varphi\ rcs \equiv \mathit{map}_{fm}\ (\mathit{asubst}\ rcs)\ \varphi$. The characteristic lemma is

$$\mathit{qfree}\ \varphi \Longrightarrow I\ (\mathit{subst}\ \varphi\ (r, cs))\ xs = I\ \varphi\ ((r + \langle cs{,}xs\rangle){\cdot}xs)$$

It remains to define the substitution of $-\infty$ for 0:

$$\begin{array}{l}
\mathit{inf}_-\ (\mathit{And}\ \varphi_1\ \varphi_2) = \mathit{and}\ (\mathit{inf}_-\ \varphi_1)\ (\mathit{inf}_-\ \varphi_2) \\
\mathit{inf}_-\ (\mathit{Or}\ \varphi_1\ \varphi_2) = \mathit{or}\ (\mathit{inf}_-\ \varphi_1)\ (\mathit{inf}_-\ \varphi_2) \\
\mathit{inf}_-\ (\mathit{Atom}\ (\mathit{Less}\ r\ (c{\cdot}cs))) = \\
(\mathit{if}\ c < 0\ \mathit{then}\ \mathit{TrueF}\ \mathit{else}\ \mathit{if}\ 0 < c\ \mathit{then}\ \mathit{FalseF}\ \mathit{else}\ \mathit{Atom}\ (\mathit{Less}\ r\ cs)) \\
\mathit{inf}_-\ (\mathit{Atom}\ (\mathit{Eq}\ r\ (c{\cdot}cs))) = (\mathit{if}\ c = 0\ \mathit{then}\ \mathit{Atom}\ (\mathit{Eq}\ r\ cs)\ \mathit{else}\ \mathit{FalseF})
\end{array}$$

The remaining cases are the identity. The definition of inf_+ is dual.

The proof of the main correctness theorem

$$\mathit{nqfree}\ \varphi \Longrightarrow I\ (\mathit{FR}_1\ \varphi)\ xs = (\exists x.\ I\ \varphi\ (x{\cdot}xs))$$

is essentially the proof of (3). Defining $\mathit{FR} = \mathit{lift\text{-}nnf\text{-}qe}\ \mathit{FR}_1$ we obtain the overall correctness as a corollary to Lemma 2: $I\ (\mathit{FR}\ \varphi)\ xs = I\ \varphi\ xs$.

Ferrante and Rackoff show that their procedure executes in space $O(2^{cn})$ and hence time $O(2^{2^{dn}})$ where n is the size of the input. This significant improvement over the non-elementary complexity of Fourier's procedure becomes relevant in the context of deeply nested and alternating quantifiers because that is the situation where conversion to DNF can blow up repeatedly.

7. Presburger Arithmetic

Presburger Arithmetic is linear integer arithmetic. Presburger [29] 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 $=$ (or $\leq$) on the integers because $i < j$ is equivalent with $i + 1 \leq j$. Hence we restrict our attention to $\leq$. All atoms are assumed to be 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 $j + 1 \leq i$.

7.1. Atoms

The above language of atoms is formalized as follows:

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

Atoms are interpreted w.r.t. a list of variables as usual:

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

where *dvd* is HOL's divisibility predicate. Note that we can reuse the polymorphic vector, i.e. list operations like $\langle .,. \rangle$ introduced for linear real arithmetic.

There is a slight complication here: We want to exclude the atoms *Dvd 0 i ks* and *NDvd 0 i ks* because they behave anomalously and the algorithm does not generate them either. Catering for them would complicate the algorithm with case distinctions. 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.

A formula is defined as normal iff all its atoms are:

$normal\ \varphi = (\forall\, a{\in}atoms\ \varphi.\ anormal\ a)$

With the help of the above axioms the following modified version of Lemma 4 can be proved:

Lemma 7 *If* $qe \in lists\ |depends_0| \rightarrow |qfree|$ *and* $qe \in lists\ (|depends_0| \cap |anormal|) \rightarrow |normal|$ *and* $\forall\, as \in lists(\ |depends_0| \cap |anormal|\ ).$ *is-dnf-qe qe as then normal* φ *implies I* $(lift\text{-}dnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$.

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

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

and their decrementation by

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

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

hd-coeff (*Le i ks*) = (*case ks of* [] ⇒ *0* | *k·x* ⇒ *k*)
hd-coeff (*Dvd d i ks*) = (*case ks of* [] ⇒ *0* | *k·x* ⇒ *k*)
hd-coeff (*NDvd d i ks*) = (*case ks of* [] ⇒ *0* | *k·x* ⇒ *k*)

and parameter *anormal* becomes λa. *divisor a* $\neq$ *0* where

divisor (*Le i ks*) = *1*
divisor (*Dvd d i ks*) = *d*
divisor (*NDvd d i ks*) = *d*

7.2. Algorithm

In this section we describe and formalize a DNF-based algorithm. It 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 [12], 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. As an example we pick $l \leq 2x \wedge 3x \leq u$. The algorithm consists of the following steps:

1. Set all coefficients of x to the positive least common multiple (lcm) of all coefficients of x in $P(x)$. Call the result $Q(m * x)$.
   Example: $Q(6 * x) = (3l \leq 6x \wedge 6x \leq 2u)$.
2. Set $R(x) = Q(x) \wedge m \mid x$.
   Example: $R(x) = (3l \leq x \wedge x \leq 2u \wedge 6 \mid x)$.
3. 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 $T = \{n \mid 0 \leq n < \delta\}$.
   Example: $\delta = 6$, $L = \{3l\}$ and the result is $\bigvee_{0 \leq n < 6} R(3l + n)$.

Instead of lower bounds, one may just as well choose upper bounds. In fact, as a local optimization one typically picks the smaller of the two sets.

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

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 $t \, mod \, \delta = z \, mod \, \delta$.

7.3. Formalization

The above algorithm consist of two steps which we implement separately. First the head coefficients of a list of atoms are set to 1 or -1 and the divisibility predicate is added:

hd-coeffs1 as =
(*let m* = *zlcms* (*map hd-coeff as*) *in Dvd m 0* [*1*]·*map* (*hd-coeff1 m*) *as*)

where *zlcms* computes the positive lcm of a list of integers, *hd-coeff* extracts the head coefficient from an atom (see §7.1), and *hd-coeff1* sets the head coefficient of one atom to 1 or -1:

hd-coeff1 m (*Le i* (*k*·*ks*)) =
(*let m'* = *m div* |*k*| *in Le* (*m'* ∗ *i*) (*sgn k*·*m'* ∗$_s$ *ks*))
hd-coeff1 m (*Dvd d i* (*k*·*ks*)) =
(*let m'* = *m div k in Dvd* (*m'* ∗ *d*) (*m'* ∗ *i*) (*1*·*m'* ∗$_s$ *ks*))
hd-coeff1 m (*NDvd d i* (*k*·*ks*)) =
(*let m'* = *m div k in NDvd* (*m'* ∗ *d*) (*m'* ∗ *i*) (*1*·*m'* ∗$_s$ *ks*))

sgn i ≡ *if i* = *0 then 0 else if 0* < *i then 1 else* − *1*

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

Lemma 8 If ∀ *a*∈*set as*. *hd-coeff a* ≠ *0* then (∃ *x*. ∀ *a*∈*set as*. I_Z *a* (*x*·*e*)) = (∃ *x*. ∀ *a*∈*set* (*hd-coeffs1 as*). I_Z *a* (*x*·*e*)).

In the second step the actual quantifier elimination is performed:

qe-pres as =
  (*let ds* = [*a*←*as*. *is-dvd a*]; *d* = *zlcms*(*map divisor ds*); *ls* = *lbounds as*
   in if ls = []
     then Disj [*0..d*−*1*] (λ*n*. *list-conj*(*map* (*Atom* ∘ *asubst n* []) *ds*))
     else Disj ls (λ(*i*,*ks*).
          Disj [*0..d*−*1*] (λ*n*. *list-conj*(*map* (*Atom* ∘ *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*·*ks*) ← *as*, *k*>*0*], and *asubst* is substitution:

asubst i' ks' (*Le i* (*k*·*ks*)) = *Le* (*i* − *k* ∗ *i'*) (*k* ∗$_s$ *ks'* + *ks*)
asubst i' ks' (*Dvd d i* (*k*·*ks*)) = *Dvd d* (*i* + *k* ∗ *i'*) (*k* ∗$_s$ *ks'* + *ks*)
asubst i' ks' (*NDvd d i* (*k*·*ks*)) = *NDvd d* (*i* + *k* ∗ *i'*) (*k* ∗$_s$ *ks'* + *ks*)

The following lemma shows that *asubst* is indeed 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:

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

7.4. Correctness

The main correctness theorem is

Theorem 3 *If* $\forall\, a{\in}set\ as.\ divisor\ a \neq 0$ *and* $\forall\, a{\in}set\ as.\ hd\text{-}coeff\text{-}is1\ a$ *then* $I\ (qe\text{-}pres\ as)\ xs = (\exists\, x.\ \forall\, a{\in}set\ as.\ I_Z\ a\ (x\cdot xs))$.

Its proof was given in §7.2. Predicate *hd-coeff-is1 a* is true iff the head coefficient of a is 1 or -1. Combining this theorem with Lemma 8 (and the lemma that *hd-coeff1* establishes *hd-coeff-is1*) yields: If $\forall\, a{\in}set\ as.\ divisor\ a \neq 0$ and $\forall\, a{\in}set\ as.\ hd\text{-}coeff\ a \neq 0$ then $I\ ((qe\text{-}pres \circ hd\text{-}coeffs1)\ as)\ e = (\exists\, x.\ \forall\, a{\in}set\ as.\ I_Z\ a\ (x\cdot e))$. Because $depends_0\ a = (hd\text{-}coeff\ a \neq 0)$ and $anormal\ a = (divisor\ a \neq 0)$, Lemma 7 yields as a corollary:

$$normal\ \varphi \Longrightarrow I\ (pres\text{-}qe\ \varphi)\ xs = I\ \varphi\ xs$$

This requires an easy $(qfree\ ((qe\text{-}pres \circ hd\text{-}coeffs1)\ as)$ and a tedious lemma:
if $\forall\, a{\in}set\ as.\ hd\text{-}coeff\ a \neq 0 \wedge divisor\ a \neq 0$ then $normal((qe\text{-}pres{\circ}hd\text{-}coeffs1)\ as)$.

8. Related Work

This paper is an outgrowth of [9]. One of the many differences of the two papers is the replacement of Cooper's NNF-based algorithm [11] for Presburger arithmetic by a DNF-based one. These two algorithms are related to each other like Ferrante and Rackoff's is to Fourier's. One can also view this article as translating some of the programs in Harrison's forthcoming textbook [20] from OCaml to HOL and verifying them (and a number of additional ones).

Another popular quantifier elimination method for Presburger arithmetic is due to Pugh [30] and takes Fourier's method as a starting point. Linear arithmetic over both reals and integers also admits quantifier elimination [32]. Chaieb reflected this algorithm in Isabelle [7]. The decision problem for first-order arithmetic theories can also be solved by automata theoretic methods. Büchi [6] initiated this approach for Presburger arithmetic. It was later extended to mixed integer and real arithmetic [4].

An LCF-style quantifier elimination procedure for real closed fields has been implemented by McLaughlin [25], a reflective version of Collin's CAD method [10] has been implemented but only partly verified by Mahboubi [24].

The special case of decision procedures for quantifier free linear real arithmetic has received an enormous amount of attention for its practical relevance and because it is solvable in polynomial time. In particular it is possible to generate short certificates that can be checked quickly (e.g. [3]).

Acknowledgements

Amine Chaieb helped me to understand quantifier elimination. He and John Harrison were constant sources of ideas. Clemens Ballarin, Florian Haftmann and Makarius Wenzel conceived and implemented locales and helped me to use them.

References

[1] C. Ballarin. Locales and locale expressions in Isabelle/Isar. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs (TYPES 2003)*, volume 3085 of *Lect. Notes in Comp. Sci.*, pages 34–50. Springer-Verlag, 2004.

[2] C. Ballarin. Interpretation of locales in Isabelle: Theories and proof contexts. In J. Borwein and W. Farmer, editors, *Mathematical Knowledge Management (MKM 2006)*, volume 4108 of *Lect. Notes in Comp. Sci.*, pages 31–43. Springer-Verlag, 2006.

[3] F. Besson. Fast reflexive arithmetic tactics the linear case and beyond. In T. Altenkirch and C. McBride, editors, *Types for Proofs and Programs (TYPES 2006)*, volume 4502 of *Lect. Notes in Comp. Sci.*, pages 48–62. Springer-Verlag, 2007.

[4] B. Boigelot, S. Jodogne, and P. Wolper. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Trans. Comput. Log.*, 6:614–633, 2005.

[5] R. S. Boyer and J. S. 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.

[6] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

[7] A. Chaieb. Verifying mixed real-integer quantifier elimination. In U. Furbach and N. Shankar, editors, *Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lect. Notes in Comp. Sci.*, pages 528–540. Springer-Verlag, 2006.

[8] A. Chaieb and T. 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 *Lect. Notes in Comp. Sci.*, pages 367–380. Springer-Verlag, 2005.

[9] A. Chaieb and T. Nipkow. Proof synthesis and reflection for linear arithmetic. Technical report, Institut für Informatik, Technische Universität München, 2006. Submitted for publication.

[10] G. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Second GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lect. Notes in Comp. Sci.*, pages 134–183. Springer-Verlag, 1976.

[11] D. 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.

[12] H. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[13] J. Farkas. Theorie der einfachen Ungleichungen. *Journal für die reine und angewandte Mathematik*, 124:1–27, 1902.

[14] J. Ferrante and C. Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM J. Computing*, 4:69–76, 1975.

[15] J. B. J. Fourier. Solution d'une question particulière du calcul des inégalités. In G. Darboux, editor, *Joseph Fourier - Œuvres complétes*, volume 2, pages 317–328. Gauthier-Villars, 1888–1890.

[16] M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF: a Mechanised Logic of Computation*, volume 78 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1979.

[17] B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In *Int. Conf. Functional Programming*, pages 235–246. ACM Press, 2002.

[18] F. Haftmann and T. Nipkow. A code generator framework for Isabelle/HOL. In K. Schneider and J. Brandt, editors, *Theorem Proving in Higher Order Logics: Emerging Trends*. Department of Computer Science, University of Kaiserslautern, 2007.

[19] F. Haftmann and M. Wenzel. Constructive type classes in Isabelle. In T. Altenkirch and C. McBride, editors, *Types for Proofs and Programs (TYPES 2006)*, volume 4502 of *Lect. Notes in Comp. Sci.*, pages 160–174. Springer-Verlag, 2007.

[20] J. Harrison. *Introduction to Logic and Automated Theorem Proving*. Cambridge University Press. Forthcoming.

[21] J. Harrison. A HOL theory of Eucledian space. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics, TPHOLs 2005*, volume 3603 of *Lect. Notes in Comp. Sci.*, pages 114–129. Springer-Verlag, 2005.

[22] J. Harrison. Verifying nonlinear real formulas via sums of squares. In K. Schneider and J. Brandt, editors, *Theorem Proving in Higher Order Logics, TPHOLs 2007*, volume 4732 of *Lect. Notes in Comp. Sci.*, pages 102–118. Springer-Verlag, 2007.

[23] C. Langford. Some theorems on deducibility. *Annals of Mathematics (2nd Series)*, 28:16–40, 1927.

[24] A. Mahboubi. *Contributions à la certification des calculs sur $\mathbb{R}$: théorie, preuves, programmation*. PhD thesis, Université de Nice, 2006.

[25] S. McLaughlin and J. Harrison. A proof-producing decision procedure for real arithmetic. In R. Nieuwenhuis, editor, *Automated Deduction — CADE-20*, volume 3632 of *Lect. Notes in Comp. Sci.*, pages 295–314. Springer-Verlag, 2005.

[26] T. Motzkin. *Beiträge zur Theorie der linearen Ungleichungen*. PhD thesis, Universität Basel, 1936.

[27] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 2002. http://www.in.tum.de/~nipkow/LNCS2283/.

[28] S. 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 *Lect. Notes in Comp. Sci.*, pages 227–244. Springer-Verlag, 2005.

[29] M. 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.

[30] W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proc. 1991 ACM/IEEE Conference on Supercomputing*, pages 4–13. ACM Press, 1991.

[31] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, 1951.

[32] V. Weispfenning. Mixed real-integer linear quantifier elimination. In *International Symposium Symbolic and Algebraic Computation (ISSAC)*, pages 129–136. ACM Press, 1999.