Quantifier Elimination for Linear Arithmetic

Tobias Nipkow

March 17, 2025

Abstract

This article formalizes quantifier elimination procedures for dense linear orders, linear real arithmetic and Presburger arithmetic. In each case both a DNF-based non-elementary algorithm and one or more (doubly) exponential NNF-based algorithms are formalized, including the well-known algorithms by Ferrante and Rackoff and by Cooper. The NNF-based algorithms for dense linear orders are new but based on Ferrante and Rackoff and on an algorithm by Loos and Weispfenning which simulates infinitesimals.

All algorithms are directly executable. In particular, they yield reflective quantifier elimination procedures for HOL itself.

The formalization makes heavy use of locales and is therefore highly modular.

For an exposition of the DNF-based procedures see [5], for the NNF-based procedures see [4].

Contents

1 Logic

theory *Logic*
imports *Main HOL−Library.FuncSet*
begin

We start with a generic formalization of quantified logical formulae using de Bruijn notation. The syntax is parametric in the type of atoms.

declare *Let-def*[*simp*]

datatype (*atoms*: 'a) *fm* =
  TrueF | *FalseF* | *Atom* 'a | *And* 'a *fm* 'a *fm* | *Or* 'a *fm* 'a *fm* |
  Neg 'a *fm* | *ExQ* 'a *fm*

notation *map-fm* (‹map_{fm}›)

abbreviation *Imp* **where** *Imp* φ_1 φ_2 ≡ *Or* (*Neg* φ_1) φ_2
abbreviation *AllQ* **where** *AllQ* φ ≡ *Neg*(*ExQ*(*Neg* φ))

definition *neg* **where**
neg φ = (*if* φ=*TrueF* *then* *FalseF* *else* *if* φ=*FalseF* *then* *TrueF* *else* *Neg* φ)

definition *and* :: 'a *fm* ⇒ 'a *fm* ⇒ 'a *fm* **where**
and φ_1 φ_2 =
 (*if* φ_1=*TrueF* *then* φ_2 *else* *if* φ_2=*TrueF* *then* φ_1 *else*
  if φ_1=*FalseF* ∨ φ_2=*FalseF* *then* *FalseF* *else* *And* φ_1 φ_2)

definition *or* :: 'a *fm* ⇒ 'a *fm* ⇒ 'a *fm* **where**
or φ_1 φ_2 =
 (*if* φ_1=*FalseF* *then* φ_2 *else* *if* φ_2=*FalseF* *then* φ_1 *else*

if φ_1=*TrueF* ∨ φ_2=*TrueF then TrueF else Or* φ_1 φ_2)

definition *list-conj* :: *'a fm list* ⇒ *'a fm* **where**
list-conj fs = *foldr and fs TrueF*

definition *list-disj* :: *'a fm list* ⇒ *'a fm* **where**
list-disj fs = *foldr or fs FalseF*

abbreviation *Disj is f* ≡ *list-disj* (*map f is*)

lemmas *atoms-map-fm*[*simp*] = *fm.set-map*

fun *amap-fm* :: (*'a* ⇒ *'b fm*) ⇒ *'a fm* ⇒ *'b fm* (‹$amap_{fm}$›) **where**
$amap_{fm}$ *h TrueF* = *TrueF* |
$amap_{fm}$ *h FalseF* = *FalseF* |
$amap_{fm}$ *h* (*Atom a*) = *h a* |
$amap_{fm}$ *h* (*And* φ_1 φ_2) = *and* ($amap_{fm}$ *h* φ_1) ($amap_{fm}$ *h* φ_2) |
$amap_{fm}$ *h* (*Or* φ_1 φ_2) = *or* ($amap_{fm}$ *h* φ_1) ($amap_{fm}$ *h* φ_2) |
$amap_{fm}$ *h* (*Neg* φ) = *neg* ($amap_{fm}$ *h* φ)

lemma *amap-fm-list-disj*:
$amap_{fm}$ *h* (*list-disj fs*) = *list-disj* (*map* ($amap_{fm}$ *h*) *fs*)
⟨*proof*⟩

fun *qfree* :: *'a fm* ⇒ *bool* **where**
qfree(*ExQ f*) = *False* |
qfree(*And* φ_1 φ_2) = (*qfree* φ_1 ∧ *qfree* φ_2) |
qfree(*Or* φ_1 φ_2) = (*qfree* φ_1 ∧ *qfree* φ_2) |
qfree(*Neg* φ) = (*qfree* φ) |
qfree φ = *True*

lemma *qfree-and*[*simp*]: ⟦ *qfree* φ_1; *qfree* φ_2 ⟧ ⟹ *qfree*(*and* φ_1 φ_2)
⟨*proof*⟩

lemma *qfree-or*[*simp*]: ⟦ *qfree* φ_1; *qfree* φ_2 ⟧ ⟹ *qfree*(*or* φ_1 φ_2)
⟨*proof*⟩

lemma *qfree-neg*[*simp*]: *qfree*(*neg* φ) = *qfree* φ
⟨*proof*⟩

lemma *qfree-foldr-Or*[*simp*]:
qfree(*foldr Or fs* φ) = (*qfree* φ ∧ (∀ φ ∈ *set fs*. *qfree* φ))
⟨*proof*⟩

lemma *qfree-list-conj*[*simp*]:
assumes ∀ φ ∈ *set fs*. *qfree* φ **shows** *qfree*(*list-conj fs*)
⟨*proof*⟩

lemma *qfree-list-disj*[*simp*]:

assumes $\forall \varphi \in$ *set fs. qfree* φ **shows** *qfree*(*list-disj fs*)
⟨*proof*⟩

lemma *qfree-map-fm*: *qfree* (map_{fm} *f* φ) = *qfree* φ
⟨*proof*⟩

lemma *atoms-list-disjE*:
  $a \in$ *atoms*(*list-disj fs*) $\Longrightarrow a \in (\bigcup \varphi \in$ *set fs. atoms* $\varphi)$
⟨*proof*⟩

lemma *atoms-list-conjE*:
  $a \in$ *atoms*(*list-conj fs*) $\Longrightarrow a \in (\bigcup \varphi \in$ *set fs. atoms* $\varphi)$
⟨*proof*⟩

fun *dnf* :: *'a fm* ⇒ *'a list list* **where**
dnf TrueF = [[]] |
dnf FalseF = [] |
dnf (*Atom* φ) = [[φ]] |
dnf (*And* φ_1 φ_2) = [*d1* @ *d2*. *d1* ← *dnf* φ_1, *d2* ← *dnf* φ_2] |
dnf (*Or* φ_1 φ_2) = *dnf* φ_1 @ *dnf* φ_2

fun *nqfree* :: *'a fm* ⇒ *bool* **where**
nqfree(*Atom a*) = *True* |
nqfree TrueF = *True* |
nqfree FalseF = *True* |
nqfree (*And* φ_1 φ_2) = (*nqfree* φ_1 ∧ *nqfree* φ_2) |
nqfree (*Or* φ_1 φ_2) = (*nqfree* φ_1 ∧ *nqfree* φ_2) |
nqfree φ = *False*

lemma *nqfree-qfree*[*simp*]: *nqfree* $\varphi \Longrightarrow$ *qfree* φ
⟨*proof*⟩

lemma *nqfree-map-fm*: *nqfree* (map_{fm} *f* φ) = *nqfree* φ
⟨*proof*⟩

fun *interpret* :: (*'a* ⇒ *'b list* ⇒ *bool*) ⇒ *'a fm* ⇒ *'b list* ⇒ *bool* **where**
interpret h TrueF xs = *True* |
interpret h FalseF xs = *False* |
interpret h (*Atom a*) *xs* = *h a xs* |
interpret h (*And* φ_1 φ_2) *xs* = (*interpret h* φ_1 *xs* ∧ *interpret h* φ_2 *xs*) |
interpret h (*Or* φ_1 φ_2) *xs* = (*interpret h* φ_1 *xs* ∨ *interpret h* φ_2 *xs*) |
interpret h (*Neg* φ) *xs* = (¬ *interpret h* φ *xs*) |
interpret h (*ExQ* φ) *xs* = (∃ *x*. *interpret h* φ (*x*#*xs*))

1.1 Atoms

The locale ATOM of atoms provides a minimal framework for the generic formulation of theory-independent algorithms, in particular quantifier elimination.

locale $ATOM$ =
fixes $aneg :: 'a \Rightarrow 'a\ fm$
fixes $anormal :: 'a \Rightarrow bool$
assumes $nqfree\text{-}aneg$: $nqfree(aneg\ a)$
assumes $anormal\text{-}aneg$: $anormal\ a \Longrightarrow \forall\, b{\in}atoms(aneg\ a).\ anormal\ b$

fixes $I_a :: 'a \Rightarrow 'b\ list \Rightarrow bool$
assumes $I_a\text{-}aneg$: $interpret\ I_a\ (aneg\ a)\ xs = (\neg\ I_a\ a\ xs)$

fixes $depends_0 :: 'a \Rightarrow bool$
and $decr :: 'a \Rightarrow 'a$
assumes $not\text{-}dep\text{-}decr$: $\neg depends_0\ a \Longrightarrow I_a\ a\ (x\#xs) = I_a\ (decr\ a)\ xs$
assumes $anormal\text{-}decr$: $\neg\ depends_0\ a \Longrightarrow anormal\ a \Longrightarrow anormal(decr\ a)$

begin

fun $atoms_0 :: 'a\ fm \Rightarrow 'a\ list$ **where**
$atoms_0\ TrueF = []\ |$
$atoms_0\ FalseF = []\ |$
$atoms_0\ (Atom\ a) = (if\ depends_0\ a\ then\ [a]\ else\ [])\ |$
$atoms_0\ (And\ \varphi_1\ \varphi_2) = atoms_0\ \varphi_1\ @\ atoms_0\ \varphi_2\ |$
$atoms_0\ (Or\ \varphi_1\ \varphi_2) = atoms_0\ \varphi_1\ @\ atoms_0\ \varphi_2\ |$
$atoms_0\ (Neg\ \varphi) = atoms_0\ \varphi$

abbreviation I **where** $I \equiv interpret\ I_a$

fun $nnf :: 'a\ fm \Rightarrow 'a\ fm$ **where**
$nnf\ (And\ \varphi_1\ \varphi_2) = And\ (nnf\ \varphi_1)\ (nnf\ \varphi_2)\ |$
$nnf\ (Or\ \varphi_1\ \varphi_2) = Or\ (nnf\ \varphi_1)\ (nnf\ \varphi_2)\ |$
$nnf\ (Neg\ TrueF) = FalseF\ |$
$nnf\ (Neg\ FalseF) = TrueF\ |$
$nnf\ (Neg\ (Neg\ \varphi)) = (nnf\ \varphi)\ |$
$nnf\ (Neg\ (And\ \varphi_1\ \varphi_2)) = (Or\ (nnf\ (Neg\ \varphi_1))\ (nnf\ (Neg\ \varphi_2)))\ |$
$nnf\ (Neg\ (Or\ \varphi_1\ \varphi_2)) = (And\ (nnf\ (Neg\ \varphi_1))\ (nnf\ (Neg\ \varphi_2)))\ |$
$nnf\ (Neg\ (Atom\ a)) = aneg\ a\ |$
$nnf\ \varphi = \varphi$

lemma $nqfree\text{-}nnf$: $qfree\ \varphi \Longrightarrow nqfree(nnf\ \varphi)$
$\langle proof \rangle$

lemma $qfree\text{-}nnf[simp]$: $qfree(nnf\ \varphi) = qfree\ \varphi$
$\langle proof \rangle$

lemma *I-neg*[*simp*]: $I\ (neg\ \varphi)\ xs = I\ (Neg\ \varphi)\ xs$
⟨*proof*⟩

lemma *I-and*[*simp*]: $I\ (and\ \varphi_1\ \varphi_2)\ xs = I\ (And\ \varphi_1\ \varphi_2)\ xs$
⟨*proof*⟩

lemma *I-list-conj*[*simp*]:
$I\ (list\text{-}conj\ fs)\ xs = (\forall\,\varphi \in set\ fs.\ I\ \varphi\ xs)$
⟨*proof*⟩

lemma *I-or*[*simp*]: $I\ (or\ \varphi_1\ \varphi_2)\ xs = I\ (Or\ \varphi_1\ \varphi_2)\ xs$
⟨*proof*⟩

lemma *I-list-disj*[*simp*]:
$I\ (list\text{-}disj\ fs)\ xs = (\exists\,\varphi \in set\ fs.\ I\ \varphi\ xs)$
⟨*proof*⟩

lemma *I-nnf*: $I\ (nnf\ \varphi)\ xs = I\ \varphi\ xs$
⟨*proof*⟩

lemma *I-dnf*:
$nqfree\ \varphi \Longrightarrow (\exists\,as{\in}set\ (dnf\ \varphi).\ \forall\,a{\in}set\ as.\ I_a\ a\ xs) = I\ \varphi\ xs$
⟨*proof*⟩

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

lemma *normal-simps*[*simp*]:
$normal\ TrueF$
$normal\ FalseF$
$normal\ (Atom\ a) \longleftrightarrow anormal\ a$
$normal\ (And\ \varphi_1\ \varphi_2) \longleftrightarrow normal\ \varphi_1 \wedge normal\ \varphi_2$
$normal\ (Or\ \varphi_1\ \varphi_2) \longleftrightarrow normal\ \varphi_1 \wedge normal\ \varphi_2$
$normal\ (Neg\ \varphi) \longleftrightarrow normal\ \varphi$
$normal\ (ExQ\ \varphi) \longleftrightarrow normal\ \varphi$
⟨*proof*⟩

lemma *normal-aneg*[*simp*]: $anormal\ a \Longrightarrow normal\ (aneg\ a)$
⟨*proof*⟩

lemma *normal-and*[*simp*]:
$normal\ \varphi_1 \Longrightarrow normal\ \varphi_2 \Longrightarrow normal\ (and\ \varphi_1\ \varphi_2)$
⟨*proof*⟩

lemma *normal-or*[*simp*]:
$normal\ \varphi_1 \Longrightarrow normal\ \varphi_2 \Longrightarrow normal\ (or\ \varphi_1\ \varphi_2)$
⟨*proof*⟩

lemma *normal-list-disj*[*simp*]:
$\forall\,\varphi{\in}set\ fs.\ normal\ \varphi \Longrightarrow normal\ (list\text{-}disj\ fs)$

⟨*proof*⟩

lemma *normal-nnf*: $normal\ \varphi \Longrightarrow normal(nnf\ \varphi)$
⟨*proof*⟩

lemma *normal-map-fm*:
$\forall a.\ anormal(f\ a) = anormal(a) \Longrightarrow normal\ (map_{fm}\ f\ \varphi) = normal\ \varphi$
⟨*proof*⟩

lemma *anormal-nnf*:
$qfree\ \varphi \Longrightarrow normal\ \varphi \Longrightarrow \forall\, a{\in}atoms(nnf\ \varphi).\ anormal\ a$
⟨*proof*⟩

lemma *atoms-dnf*: $nqfree\ \varphi \Longrightarrow as \in set(dnf\ \varphi) \Longrightarrow a \in set\ as \Longrightarrow a \in atoms\ \varphi$
⟨*proof*⟩

lemma *anormal-dnf-nnf*:
$as \in set(dnf(nnf\ \varphi)) \Longrightarrow qfree\ \varphi \Longrightarrow normal\ \varphi \Longrightarrow a \in set\ as \Longrightarrow anormal\ a$
⟨*proof*⟩

end

end

2 Quantifier elimination

theory *QE*
imports *Logic*
begin

The generic, i.e. theory-independent part of quantifier elimination. Both DNF and an NNF-based procedures are defined and proved correct.

notation (*input*) *Collect* (‹|-|›)

2.1 No Equality

context *ATOM*
begin

2.1.1 DNF-based

Taking care of atoms independent of variable 0:

definition
$qelim\ qe\ as =$
$(let\ qf = qe\ [a{\leftarrow}as.\ depends_0\ a];$
$\quad indep = [Atom(decr\ a).\ a{\leftarrow}as, \neg\ depends_0\ a]$
$\ in\ and\ qf\ (list\text{-}conj\ indep))$

abbreviation *is-dnf-qe* :: $('a\ list \Rightarrow 'a\ fm) \Rightarrow 'a\ list \Rightarrow bool$ **where**
$is\text{-}dnf\text{-}qe\ qe\ as \equiv \forall xs.\ I(qe\ as)\ xs = (\exists x.\forall a{\in}set\ as.\ I_a\ a\ (x\#xs))$

Note that the exported abbreviation will have as a first parameter the type $'b$ of values xs ranges over.

lemma *I-qelim*:
assumes *qe*: $\bigwedge as.\ (\forall a \in set\ as.\ depends_0\ a) \Longrightarrow is\text{-}dnf\text{-}qe\ qe\ as$
shows $is\text{-}dnf\text{-}qe\ (qelim\ qe)\ as$ (**is** $\forall xs.\ ?P\ xs$)
⟨*proof*⟩

The generic DNF-based quantifier elimination procedure:

fun *lift-dnf-qe* :: $('a\ list \Rightarrow 'a\ fm) \Rightarrow 'a\ fm \Rightarrow 'a\ fm$ **where**
$lift\text{-}dnf\text{-}qe\ qe\ (And\ \varphi_1\ \varphi_2) = and\ (lift\text{-}dnf\text{-}qe\ qe\ \varphi_1)\ (lift\text{-}dnf\text{-}qe\ qe\ \varphi_2)\ |$
$lift\text{-}dnf\text{-}qe\ qe\ (Or\ \varphi_1\ \varphi_2) = or\ (lift\text{-}dnf\text{-}qe\ qe\ \varphi_1)\ (lift\text{-}dnf\text{-}qe\ qe\ \varphi_2)\ |$
$lift\text{-}dnf\text{-}qe\ qe\ (Neg\ \varphi) = neg(lift\text{-}dnf\text{-}qe\ qe\ \varphi)\ |$
$lift\text{-}dnf\text{-}qe\ qe\ (ExQ\ \varphi) = Disj\ (dnf(nnf(lift\text{-}dnf\text{-}qe\ qe\ \varphi)))\ (qelim\ qe)\ |$
$lift\text{-}dnf\text{-}qe\ qe\ \varphi = \varphi$

lemma *qfree-lift-dnf-qe*: $(\bigwedge as.\ (\forall a{\in}set\ as.\ depends_0\ a) \Longrightarrow qfree(qe\ as))$
$\Longrightarrow qfree(lift\text{-}dnf\text{-}qe\ qe\ \varphi)$
⟨*proof*⟩

lemma *qfree-lift-dnf-qe2*: $qe \in lists\ |depends_0| \rightarrow |qfree|$
$\Longrightarrow qfree(lift\text{-}dnf\text{-}qe\ qe\ \varphi)$
⟨*proof*⟩

lemma *lem*: $\forall P\ A.\ (\exists x{\in}A.\ \exists y.\ P\ x\ y) = (\exists y.\ \exists x{\in}A.\ P\ x\ y)$ ⟨*proof*⟩

lemma *I-lift-dnf-qe*:
assumes $\bigwedge as.\ (\forall a \in set\ as.\ depends_0\ a) \Longrightarrow qfree(qe\ as)$
and $\bigwedge as.\ (\forall a \in set\ as.\ depends_0\ a) \Longrightarrow is\text{-}dnf\text{-}qe\ qe\ as$
shows $I\ (lift\text{-}dnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
⟨*proof*⟩

lemma *I-lift-dnf-qe2*:
assumes $qe \in lists\ |depends_0| \rightarrow |qfree|$
and $\forall as \in lists\ |depends_0|.\ is\text{-}dnf\text{-}qe\ qe\ as$
shows $I\ (lift\text{-}dnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
⟨*proof*⟩

Quantifier elimination with invariant (needed for Presburger):

lemma *I-qelim-anormal*:
assumes *qe*: $\bigwedge xs\ as.\ \forall a \in set\ as.\ depends_0\ a \wedge anormal\ a \Longrightarrow is\text{-}dnf\text{-}qe\ qe\ as$
and *nm*: $\forall a \in set\ as.\ anormal\ a$
shows $I\ (qelim\ qe\ as)\ xs = (\exists x.\ \forall a{\in}set\ as.\ I_a\ a\ (x\#xs))$
⟨*proof*⟩

context notes [[*simp-depth-limit = 5*]]

begin

lemma *anormal-atoms-qelim*:
$(\bigwedge as.\ \forall\, a \in set\ as.\ depends_0\ a \wedge anormal\ a \Longrightarrow normal(qe\ as)) \Longrightarrow$
$\forall\, a \in set\ as.\ anormal\ a \Longrightarrow a \in atoms(qelim\ qe\ as) \Longrightarrow anormal\ a$
$\langle proof \rangle$

lemma *normal-lift-dnf-qe*:
assumes $\bigwedge as.\ \forall\, a \in set\ as.\ depends_0\ a \Longrightarrow qfree(qe\ as)$
and $\bigwedge as.\ \forall\, a \in set\ as.\ depends_0\ a \wedge anormal\ a \Longrightarrow normal(qe\ as)$
shows $normal\ \varphi \Longrightarrow normal(lift\text{-}dnf\text{-}qe\ qe\ \varphi)$
$\langle proof \rangle$

end

context notes [[*simp-depth-limit* = *9*]]
begin

lemma *I-lift-dnf-qe-anormal*:
assumes $\bigwedge as.\ \forall\, a \in set\ as.\ depends_0\ a \Longrightarrow qfree(qe\ as)$
and $\bigwedge as.\ \forall\, a \in set\ as.\ depends_0\ a \wedge anormal\ a \Longrightarrow normal(qe\ as)$
and $\bigwedge xs\ as.\ \forall\, a \in set\ as.\ depends_0\ a \wedge anormal\ a \Longrightarrow is\text{-}dnf\text{-}qe\ qe\ as$
shows $normal\ f \Longrightarrow I\ (lift\text{-}dnf\text{-}qe\ qe\ f)\ xs = I\ f\ xs$
$\langle proof \rangle$

end

lemma *I-lift-dnf-qe-anormal2*:
assumes $qe \in lists\ |depends_0| \to |qfree|$
and $qe \in lists\ (\ |depends_0| \cap |anormal|\ ) \to |normal|$
and $\forall\, as \in lists(\ |depends_0| \cap |anormal|\ ).\ is\text{-}dnf\text{-}qe\ qe\ as$
shows $normal\ f \Longrightarrow I\ (lift\text{-}dnf\text{-}qe\ qe\ f)\ xs = I\ f\ xs$
$\langle proof \rangle$

2.1.2 NNF-based

fun $lift\text{-}nnf\text{-}qe :: ('a\ fm \Rightarrow 'a\ fm) \Rightarrow 'a\ fm \Rightarrow 'a\ fm$ **where**
$lift\text{-}nnf\text{-}qe\ qe\ (And\ \varphi_1\ \varphi_2) = and\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi_1)\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi_2)\ |$
$lift\text{-}nnf\text{-}qe\ qe\ (Or\ \varphi_1\ \varphi_2) = or\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi_1)\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi_2)\ |$
$lift\text{-}nnf\text{-}qe\ qe\ (Neg\ \varphi) = neg(lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ |$
$lift\text{-}nnf\text{-}qe\ qe\ (ExQ\ \varphi) = qe(nnf(lift\text{-}nnf\text{-}qe\ qe\ \varphi))\ |$
$lift\text{-}nnf\text{-}qe\ qe\ \varphi = \varphi$

lemma *qfree-lift-nnf-qe*: $(\bigwedge \varphi.\ nqfree\ \varphi \Longrightarrow qfree(qe\ \varphi))$
$\Longrightarrow qfree(lift\text{-}nnf\text{-}qe\ qe\ \varphi)$
$\langle proof \rangle$

lemma *qfree-lift-nnf-qe2*:
$qe \in |nqfree| \to |qfree| \Longrightarrow qfree(lift\text{-}nnf\text{-}qe\ qe\ \varphi)$

$\langle proof \rangle$

lemma *I-lift-nnf-qe*:
assumes $\bigwedge\varphi.\ nqfree\ \varphi \Longrightarrow qfree(qe\ \varphi)$
and $\bigwedge xs\ \varphi.\ nqfree\ \varphi \Longrightarrow I\ (qe\ \varphi)\ xs = (\exists\, x.\ I\ \varphi\ (x\#xs))$
shows $I\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
$\langle proof \rangle$

lemma *I-lift-nnf-qe2*:
assumes $qe \in |nqfree| \to |qfree|$
and $\forall\, \varphi \in |nqfree|.\ \forall\, xs.\ I\ (qe\ \varphi)\ xs = (\exists\, x.\ I\ \varphi\ (x\#xs))$
shows $I\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
$\langle proof \rangle$

lemma *normal-lift-nnf-qe*:
assumes $\bigwedge\varphi.\ nqfree\ \varphi \Longrightarrow qfree(qe\ \varphi)$
and $\bigwedge\varphi.\ nqfree\ \varphi \Longrightarrow normal\ \varphi \Longrightarrow normal(qe\ \varphi)$
shows $normal\ \varphi \Longrightarrow normal(lift\text{-}nnf\text{-}qe\ qe\ \varphi)$
$\langle proof \rangle$

lemma *I-lift-nnf-qe-normal*:
assumes $\bigwedge\varphi.\ nqfree\ \varphi \Longrightarrow qfree(qe\ \varphi)$
and $\bigwedge\varphi.\ nqfree\ \varphi \Longrightarrow normal\ \varphi \Longrightarrow normal(qe\ \varphi)$
and $\bigwedge xs\ \varphi.\ normal\ \varphi \Longrightarrow nqfree\ \varphi \Longrightarrow I\ (qe\ \varphi)\ xs = (\exists\, x.\ I\ \varphi\ (x\#xs))$
shows $normal\ \varphi \Longrightarrow I\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
$\langle proof \rangle$

lemma *I-lift-nnf-qe-normal2*:
assumes $qe \in |nqfree| \to |qfree|$
and $qe \in |nqfree| \cap |normal| \to |normal|$
and $\forall\, \varphi \in |normal| \cap |nqfree|.\ \forall\, xs.\ I\ (qe\ \varphi)\ xs = (\exists\, x.\ I\ \varphi\ (x\#xs))$
shows $normal\ \varphi \Longrightarrow I\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
$\langle proof \rangle$

end

2.2 With equality

DNF-based quantifier elimination can accommodate equality atoms in a generic fashion.

locale $ATOM\text{-}EQ = ATOM\ +$
fixes $solvable_0 :: {}'a \Rightarrow bool$
and $trivial :: {}'a \Rightarrow bool$
and $subst_0 :: {}'a \Rightarrow {}'a \Rightarrow {}'a$
assumes $subst_0$:
$[\![\ solvable_0\ eq;\ \ \neg trivial\ eq;\ \ I_a\ eq\ (x\#xs);\ \ depends_0\ a\ ]\!]$
$\Longrightarrow I_a\ (subst_0\ eq\ a)\ xs = I_a\ a\ (x\#xs)$
and *trivial*: $trivial\ eq \Longrightarrow I_a\ eq\ xs$
and *solvable*: $solvable_0\ eq \Longrightarrow \exists\, x.\ I_a\ eq\ (x\#xs)$

and *is-triv-self-subst*: $solvable_0\ eq \Longrightarrow trivial\ (subst_0\ eq\ eq)$

begin

definition *lift-eq-qe* :: $('a\ list \Rightarrow 'a\ fm) \Rightarrow 'a\ list \Rightarrow 'a\ fm$ **where**
lift-eq-qe qe as =
 $(let\ as = [a{\leftarrow}as.\ \neg\ trivial\ a]$
  $in\ case\ [a{\leftarrow}as.\ solvable_0\ a]\ of$
    $[] \Rightarrow qe\ as$
  $|\ eq\ \#\ eqs \Rightarrow$
       $(let\ ineqs = [a{\leftarrow}as.\ \neg\ solvable_0\ a]$
        $in\ list\text{-}conj\ (map\ (Atom \circ (subst_0\ eq))\ (eqs\ @\ ineqs))))$

theorem *I-lift-eq-qe*:
assumes *dep*: $\forall\, a{\in}set\ as.\ depends_0\ a$
assumes *qe*: $\bigwedge as.\ (\forall\, a \in set\ as.\ depends_0\ a \wedge \neg\ solvable_0\ a) \Longrightarrow$
  $I\ (qe\ as)\ xs = (\exists\, x.\ \forall\, a \in set\ as.\ I_a\ a\ (x\#xs))$
shows $I\ (lift\text{-}eq\text{-}qe\ qe\ as)\ xs = (\exists\, x.\ \forall\, a \in set\ as.\ I_a\ a\ (x\#xs))$
 (**is** $?L = ?R$)
⟨*proof*⟩

definition *lift-dnfeq-qe* = *lift-dnf-qe* ∘ *lift-eq-qe*

lemma *qfree-lift-eq-qe*:
 $(\bigwedge as.\ \forall\, a{\in}set\ as.\ depends_0\ a \Longrightarrow qfree\ (qe\ as)) \Longrightarrow$
  $\forall\, a{\in}set\ as.\ depends_0\ a \Longrightarrow qfree(lift\text{-}eq\text{-}qe\ qe\ as)$
⟨*proof*⟩

lemma *qfree-lift-dnfeq-qe*: $(\bigwedge as.\ (\forall\, a{\in}set\ as.\ depends_0\ a) \Longrightarrow qfree(qe\ as))$
 $\Longrightarrow qfree(lift\text{-}dnfeq\text{-}qe\ qe\ \varphi)$
⟨*proof*⟩

lemma *I-lift-dnfeq-qe*:
 $(\bigwedge as.\ (\forall\, a \in set\ as.\ depends_0\ a) \Longrightarrow qfree(qe\ as)) \Longrightarrow$
  $(\bigwedge as.\ (\forall\, a \in set\ as.\ depends_0\ a \wedge \neg\ solvable_0\ a) \Longrightarrow is\text{-}dnf\text{-}qe\ qe\ as) \Longrightarrow$
  $I\ (lift\text{-}dnfeq\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
⟨*proof*⟩

lemma *I-lift-dnfeq-qe2*:
 $qe \in lists\ |depends_0| \to |qfree| \Longrightarrow$
  $(\forall\, as \in lists(\ |depends_0| \cap -\ |solvable_0|\ ).\ is\text{-}dnf\text{-}qe\ qe\ as) \Longrightarrow$
  $I\ (lift\text{-}dnfeq\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
⟨*proof*⟩

end

end

3 DLO

theory *DLO*
imports *QE Complex-Main*
begin

3.1 Basics

class *dlo* = *linorder* +
assumes *dense*: $x < z \Longrightarrow \exists\, y.\ x < y \wedge y < z$
and *no-ub*: $\exists\, u.\ x < u$ **and** *no-lb*: $\exists\, l.\ l < x$

instance *real* :: *dlo*
⟨*proof*⟩

datatype *atom* = *Less nat nat* | *Eq nat nat*

fun *is-Less* :: *atom* ⇒ *bool* **where**
is-Less (*Less i j*) = *True* |
is-Less f = *False*

abbreviation *is-Eq* ≡ *Not* ∘ *is-Less*

lemma *is-Less-iff*: *is-Less a* = (∃ *i j*. *a* = *Less i j*)
⟨*proof*⟩
lemma *is-Eq-iff*: (∀ *i j*. *a* ≠ *Less i j*) = (∃ *i j*. *a* = *Eq i j*)
⟨*proof*⟩
lemma *not-is-Eq-iff*: (∀ *i j*. *a* ≠ *Eq i j*) = (∃ *i j*. *a* = *Less i j*)
⟨*proof*⟩

fun neg_{dlo} :: *atom* ⇒ *atom fm* **where**
neg_{dlo} (*Less i j*) = *Or* (*Atom*(*Less j i*)) (*Atom*(*Eq i j*)) |
neg_{dlo} (*Eq i j*) = *Or* (*Atom*(*Less i j*)) (*Atom*(*Less j i*))

fun I_{dlo} :: *atom* ⇒ *'a::dlo list* ⇒ *bool* **where**
I_{dlo} (*Eq i j*) *xs* = (*xs*!*i* = *xs*!*j*) |
I_{dlo} (*Less i j*) *xs* = (*xs*!*i* < *xs*!*j*)

fun $depends_{dlo}$ :: *atom* ⇒ *bool* **where**
$depends_{dlo}$(*Eq i j*) = (*i*=*0* | *j*=*0*) |
$depends_{dlo}$(*Less i j*) = (*i*=*0* | *j*=*0*)

fun $decr_{dlo}$ :: *atom* ⇒ *atom* **where**
$decr_{dlo}$ (*Less i j*) = *Less* (*i* − *1*) (*j* − *1*) |
$decr_{dlo}$ (*Eq i j*) = *Eq* (*i* − *1*) (*j* − *1*)

definition [*code del*]: *nnf* = *ATOM.nnf* neg_{dlo}
definition [*code del*]: *qelim* = *ATOM.qelim* $depends_{dlo}$ $decr_{dlo}$
definition [*code del*]: *lift-dnf-qe* = *ATOM.lift-dnf-qe* neg_{dlo} $depends_{dlo}$ $decr_{dlo}$

definition [*code del*]: *lift-nnf-qe* = *ATOM.lift-nnf-qe* neg_{dlo}

hide-const *nnf qelim lift-dnf-qe lift-nnf-qe*

lemmas *DLO-code-lemmas* = *nnf-def qelim-def lift-dnf-qe-def lift-nnf-qe-def*

interpretation *DLO*:
ATOM neg_{dlo} ($\lambda a.$ *True*) I_{dlo} $depends_{dlo}$ $decr_{dlo}$
⟨*proof*⟩

lemmas [*folded DLO-code-lemmas*, *code*] =
DLO.nnf.simps DLO.qelim-def DLO.lift-dnf-qe.simps DLO.lift-dnf-qe.simps

⟨*ML*⟩

definition *lbounds* **where** *lbounds as* = [*i*. *Less* (*Suc i*) *0* ← *as*]
definition *ubounds* **where** *ubounds as* = [*i*. *Less 0* (*Suc i*) ← *as*]
definition *ebounds* **where**
ebounds as = [*i*. *Eq* (*Suc i*) *0* ← *as*] @ [*i*. *Eq 0* (*Suc i*) ← *as*]

lemma *set-lbounds*: *set*(*lbounds as*) = {*i*. *Less* (*Suc i*) *0* ∈ *set as*}
⟨*proof*⟩
lemma *set-ubounds*: *set*(*ubounds as*) = {*i*. *Less 0* (*Suc i*) ∈ *set as*}
⟨*proof*⟩
lemma *set-ebounds*:
set(*ebounds as*) = {*k*. *Eq* (*Suc k*) *0* ∈ *set as* ∨ *Eq 0* (*Suc k*) ∈ *set as*}
⟨*proof*⟩

abbreviation *LB f xs* ≡ {*xs*!*i*|*i*. *Less* (*Suc i*) *0* ∈ *set*(*DLO*.$atoms_0$ *f*)}
abbreviation *UB f xs* ≡ {*xs*!*i*|*i*. *Less 0* (*Suc i*) ∈ *set*(*DLO*.$atoms_0$ *f*)}
definition *EQ f xs* = {*xs*!*k*|*k*.
Eq (*Suc k*) *0* ∈ *set*(*DLO*.$atoms_0$ *f*) ∨ *Eq 0* (*Suc k*) ∈ *set*(*DLO*.$atoms_0$ *f*)}

lemma *EQ-And*[*simp*]: *EQ* (*And f g*) *xs* = (*EQ f xs* ∪ *EQ g xs*)
⟨*proof*⟩

lemma *EQ-Or*[*simp*]: *EQ* (*Or f g*) *xs* = (*EQ f xs* ∪ *EQ g xs*)
⟨*proof*⟩

lemma *EQ-conv-set-ebounds*:
x ∈ *EQ f xs* = (∃ *e*∈*set*(*ebounds*(*DLO*.$atoms_0$ *f*)). *x* = *xs*!*e*)
⟨*proof*⟩

fun *isubst* **where** *isubst k 0* = *k* | *isubst k* (*Suc i*) = *i*

fun *asubst* :: *nat* ⇒ *atom* ⇒ *atom* **where**

$asubst\ k\ (Less\ i\ j) = Less\ (isubst\ k\ i)\ (isubst\ k\ j)|$
$asubst\ k\ (Eq\ i\ j) = Eq\ (isubst\ k\ i)\ (isubst\ k\ j)$

abbreviation $subst\ \varphi\ k \equiv map_{fm}\ (asubst\ k)\ \varphi$

lemma *I-subst*:
$qfree\ f \Longrightarrow DLO.I\ (subst\ f\ k)\ xs = DLO.I\ f\ (xs!k\ \#\ xs)$
$\langle proof \rangle$

fun *amin-inf* :: *atom* ⇒ *atom fm* **where**
$amin\text{-}inf\ (Less\ \text{-}\ 0) = FalseF\ |$
$amin\text{-}inf\ (Less\ 0\ \text{-}) = TrueF\ |$
$amin\text{-}inf\ (Less\ (Suc\ i)\ (Suc\ j)) = Atom(Less\ i\ j)\ |$
$amin\text{-}inf\ (Eq\ 0\ 0) = TrueF\ |$
$amin\text{-}inf\ (Eq\ 0\ \text{-}) = FalseF\ |$
$amin\text{-}inf\ (Eq\ \text{-}\ 0) = FalseF\ |$
$amin\text{-}inf\ (Eq\ (Suc\ i)\ (Suc\ j)) = Atom(Eq\ i\ j)$

abbreviation *min-inf* :: *atom fm* ⇒ *atom fm* ($‹inf_-›$) **where**
$inf_- \equiv amap_{fm}\ amin\text{-}inf$

fun *aplus-inf* :: *atom* ⇒ *atom fm* **where**
$aplus\text{-}inf\ (Less\ 0\ \text{-}) = FalseF\ |$
$aplus\text{-}inf\ (Less\ \text{-}\ 0) = TrueF\ |$
$aplus\text{-}inf\ (Less\ (Suc\ i)\ (Suc\ j)) = Atom(Less\ i\ j)\ |$
$aplus\text{-}inf\ (Eq\ 0\ 0) = TrueF\ |$
$aplus\text{-}inf\ (Eq\ 0\ \text{-}) = FalseF\ |$
$aplus\text{-}inf\ (Eq\ \text{-}\ 0) = FalseF\ |$
$aplus\text{-}inf\ (Eq\ (Suc\ i)\ (Suc\ j)) = Atom(Eq\ i\ j)$

abbreviation *plus-inf* :: *atom fm* ⇒ *atom fm* ($‹inf_+›$) **where**
$inf_+ \equiv amap_{fm}\ aplus\text{-}inf$

lemma *min-inf*:
$nqfree\ f \Longrightarrow \exists x.\ \forall y \le x.\ DLO.I\ (inf_-\ f)\ xs = DLO.I\ f\ (y\ \#\ xs)$
(**is** - $\Longrightarrow \exists x.\ ?P\ f\ x$)
$\langle proof \rangle$

lemma *plus-inf*:
$nqfree\ f \Longrightarrow \exists x. \forall y \ge x.\ DLO.I\ (inf_+\ f)\ xs = DLO.I\ f\ (y\ \#\ xs)$
(**is** - $\Longrightarrow \exists x.\ ?P\ f\ x$)
$\langle proof \rangle$

context notes [[*simp-depth-limit=2*]]
begin

lemma *LBex*:
$[\![\ nqfree\ f;\ DLO.I\ f\ (x\#xs);\ \neg DLO.I\ (inf_-\ f)\ xs;\ x \notin EQ\ f\ xs\ ]\!]$

$\Longrightarrow \exists\, l \in LB\ f\ xs.\ l < x$
⟨*proof*⟩

lemma *UBex*:
$[\![$ *nqfree f*; *DLO.I f* (*x#xs*); $\neg$*DLO.I* (*inf*$_+$ *f*) *xs*; $x \notin EQ\ f\ xs$ $]\!]$
$\Longrightarrow \exists\, u \in UB\ f\ xs.\ x < u$
⟨*proof*⟩

end

lemma *finite-LB*: *finite*(*LB f xs*)
⟨*proof*⟩

lemma *finite-UB*: *finite*(*UB f xs*)
⟨*proof*⟩

lemma *qfree-amin-inf*: *qfree* (*amin-inf a*)
⟨*proof*⟩

lemma *qfree-min-inf*: *nqfree* $\varphi \Longrightarrow$ *qfree*(*inf*$_-$ φ)
⟨*proof*⟩

lemma *qfree-aplus-inf*: *qfree* (*aplus-inf a*)
⟨*proof*⟩

lemma *qfree-plus-inf*: *nqfree* $\varphi \Longrightarrow$ *qfree*(*inf*$_+$ φ)
⟨*proof*⟩

end

theory *QEdlo*
imports *DLO*
begin

3.2 DNF-based quantifier elimination

definition *qe-dlo*$_1$ :: *atom list* $\Rightarrow$ *atom fm* **where**
qe-dlo$_1$ *as* =
(*if Less 0 0* $\in$ *set as then FalseF else*
let lbs = [*i*. *Less* (*Suc i*) *0* $\leftarrow$ *as*]; *ubs* = [*j*. *Less 0* (*Suc j*) $\leftarrow$ *as*];
pairs = [*Atom*(*Less i j*). *i* $\leftarrow$ *lbs*, *j* $\leftarrow$ *ubs*]
in list-conj pairs)

theorem *I-qe-dlo*$_1$:
assumes *less*: $\forall\, a \in$ *set as*. *is-Less a* **and** *dep*: $\forall\, a \in$ *set as*. *depends*$_{dlo}$ *a*
shows *DLO.I* (*qe-dlo*$_1$ *as*) *xs* = ($\exists\, x.\ \forall\, a \in$ *set as*. I_{dlo} *a* (*x#xs*))
(**is** *?L* = *?R*)

$\langle proof \rangle$

lemma *I-qe-dlo₁-pretty*:

$\forall a \in set\ as.\ is\text{-}Less\ a \wedge depends_{dlo}\ a \Longrightarrow DLO.is\text{-}dnf\text{-}qe\ \text{-}\ qe\text{-}dlo_1\ as$

$\langle proof \rangle$

definition $subst :: nat \Rightarrow nat \Rightarrow nat \Rightarrow nat$ **where**
$subst\ i\ j\ k = (if\ k{=}0\ then\ if\ i{=}0\ then\ j\ else\ i\ else\ k) - 1$
fun $subst_0 :: atom \Rightarrow atom \Rightarrow atom$ **where**
$subst_0\ (Eq\ i\ j)\ a = (case\ a\ of$
$\quad Less\ m\ n \Rightarrow Less\ (subst\ i\ j\ m)\ (subst\ i\ j\ n)$
$\mid Eq\ m\ n \Rightarrow Eq\ (subst\ i\ j\ m)\ (subst\ i\ j\ n))$

lemma $subst_0$*-pretty*:
$subst_0\ (Eq\ i\ j)\ (Less\ m\ n) = Less\ (subst\ i\ j\ m)\ (subst\ i\ j\ n)$
$subst_0\ (Eq\ i\ j)\ (Eq\ m\ n) = Eq\ (subst\ i\ j\ m)\ (subst\ i\ j\ n)$
$\langle proof \rangle$
interpretation DLO_e:
$ATOM\text{-}EQ\ neg_{dlo}\ (\lambda a.\ True)\ I_{dlo}\ depends_{dlo}\ decr_{dlo}$
$\quad (\lambda Eq\ i\ j \Rightarrow i{=}0 \vee j{=}0 \mid a \Rightarrow False)$
$\quad (\lambda Eq\ i\ j \Rightarrow i{=}j \mid a \Rightarrow False)\ subst_0$
$\langle proof \rangle$
$\langle ML \rangle$

definition $qe\text{-}dlo = DLO_e.lift\text{-}dnfeq\text{-}qe\ qe\text{-}dlo_1$
lemma *qfree-qe-dlo₁*: $qfree\ (qe\text{-}dlo_1\ as)$
$\langle proof \rangle$

theorem *I-qe-dlo*: $DLO.I\ (qe\text{-}dlo\ \varphi)\ xs = DLO.I\ \varphi\ xs$
$\langle proof \rangle$

theorem *qfree-qe-dlo*: $qfree\ (qe\text{-}dlo\ \varphi)$
$\langle proof \rangle$

end

theory *QEdlo-ex* **imports** *QEdlo*
begin

definition $interpret :: atom\ fm \Rightarrow {}'a{::}dlo\ list \Rightarrow bool$ **where**
$interpret = Logic.interpret\ I_{dlo}$

lemma *interpret-Atoms*:
$interpret\ (Atom\ (Eq\ i\ j))\ xs = (xs!i = xs!j)$
$interpret\ (Atom\ (Less\ i\ j))\ xs = (xs!i < xs!j)$
$\langle proof \rangle$

lemma *interpret-others*:
  interpret (*Neg*(*ExQ* (*Neg f*))) *xs* = ($\forall$ *x*. *interpret f* (*x*#*xs*))
  interpret (*Or* (*Neg f1*) *f2*) *xs* = (*interpret f1 xs* $\longrightarrow$ *interpret f2 xs*)
⟨*proof*⟩

lemmas *reify-eqs* =
  Logic.interpret.simps(*1*,*2*,*4*−*7*)[*of* I_{dlo}, *folded interpret-def*]
  interpret-others interpret-Atoms

⟨*ML*⟩

declare I_{dlo}.*simps*(*1*)[*code*]
declare *Logic.interpret.simps*[*code del*]
declare *Logic.interpret.simps*(*1*−*2*)[*code*]

3.3 Examples

lemma $\forall$ *x*::*real*. $\neg$ *x* < *x*
⟨*proof*⟩

lemma $\forall$ *x y*::*real*. $\exists$ *z*. *x* < *y* $\longrightarrow$ *x* < *z* $\wedge$ *z* < *y*
⟨*proof*⟩

lemma $\exists$ *x*::*real*. *a*+*b* < *x* $\wedge$ *x* < *c*∗*d*
⟨*proof*⟩

lemma $\forall$ *x*::*real*. $\neg$ *x* < *x*
⟨*proof*⟩

lemma $\forall$ *x y*::*real*. $\exists$ *z*. *x* < *y* $\longrightarrow$ *x* < *z* $\wedge$ *z* < *y*
⟨*proof*⟩

lemma $\neg$($\exists$ *x y z*. $\forall$ *u*::*real*. *x* < *x* $\vee$ $\neg$ *x*<*u* $\vee$ *x*<*y* $\wedge$ *y*<*z* $\wedge$ $\neg$ *x*<*z*)
⟨*proof*⟩

lemma *qe-dlo*(*AllQ* (*Imp* (*Atom*(*Less 0 1*)) (*Atom*(*Less 1 0*)))) = *FalseF*
⟨*proof*⟩

lemma *qe-dlo*(*AllQ*(*AllQ* (*Imp* (*Atom*(*Less 0 1*)) (*Atom*(*Less 0 1*))))) = *TrueF*
⟨*proof*⟩

lemma
  qe-dlo(*AllQ*(*ExQ*(*AllQ* (*And* (*Atom*(*Less 2 1*)) (*Atom*(*Less 1 0*)))))) = *FalseF*
⟨*proof*⟩

lemma *qe-dlo*(*AllQ*(*ExQ*(*ExQ* (*And* (*Atom*(*Less 1 2*)) (*Atom*(*Less 2 0*)))))) =

TrueF
⟨*proof*⟩

lemma
qe-dlo(*AllQ*(*AllQ*(*ExQ* (*And* (*Atom*(*Less 1 0*)) (*Atom*(*Less 0 2*)))))) = *FalseF*
⟨*proof*⟩

lemma *qe-dlo*(*AllQ*(*AllQ*(*ExQ* (*Imp* (*Atom*(*Less 1 2*)) (*And* (*Atom*(*Less 1 0*)) (*Atom*(*Less 0 2*))))))) = *TrueF*
⟨*proof*⟩

value *qe-dlo*(*AllQ* (*Imp* (*Atom*(*Less 0 1*)) (*Atom*(*Less 0 2*))))

end

theory *QEdlo-fr*
imports *DLO*
begin

3.4 Interior Point Method

This section formalizes a new quantifier elimination procedure based on the idea of Ferrante and Rackoff [2] (see also §4.3) of taking a point between each lower and upper bound as a test point. For dense linear orders it is not obvious how to realize this because we cannot name any intermediate point directly.

fun $asubst_2$ :: *nat* ⇒ *nat* ⇒ *atom* ⇒ *atom fm* **where**
$asubst_2$ *l u* (*Less 0 0*) = *FalseF* |
$asubst_2$ *l u* (*Less 0* (*Suc j*)) = *Or* (*Atom*(*Less u j*)) (*Atom*(*Eq u j*)) |
$asubst_2$ *l u* (*Less* (*Suc i*) *0*) = *Or* (*Atom*(*Less i l*)) (*Atom*(*Eq i l*)) |
$asubst_2$ *l u* (*Less* (*Suc i*) (*Suc j*)) = *Atom*(*Less i j*) |
$asubst_2$ *l u* (*Eq 0 0*) = *TrueF* |
$asubst_2$ *l u* (*Eq 0 -*) = *FalseF* |
$asubst_2$ *l u* (*Eq - 0*) = *FalseF* |
$asubst_2$ *l u* (*Eq* (*Suc i*) (*Suc j*)) = *Atom*(*Eq i j*)

abbreviation $subst_2$ *l u* ≡ $amap_{fm}$ ($asubst_2$ *l u*)

lemma *I-*$subst_2$*1*:
nqfree f ⟹ *xs*!*l* < *xs*!*u* ⟹ *DLO.I* ($subst_2$ *l u f*) *xs*
⟹ *xs*!*l* < *x* ⟹ *x* < *xs*!*u* ⟹ *DLO.I f* (*x*#*xs*)
⟨*proof*⟩

definition
nolub f xs l x u ⟷ (∀ *y*∈{*l*<..<*x*}. *y* ∉ *LB f xs*) ∧ (∀ *y*∈{*x*<..<*u*}. *y* ∉ *UB f xs*)

lemma *nolub-And*[*simp*]:

$nolub\ (And\ f\ g)\ xs\ l\ x\ u = (nolub\ f\ xs\ l\ x\ u \wedge nolub\ g\ xs\ l\ x\ u)$
⟨proof⟩

lemma *nolub-Or*[*simp*]:
$nolub\ (Or\ f\ g)\ xs\ l\ x\ u = (nolub\ f\ xs\ l\ x\ u \wedge nolub\ g\ xs\ l\ x\ u)$
⟨proof⟩

context notes [[*simp-depth-limit=3*]]
begin

lemma *innermost-intvl*:
$[\![\ nqfree\ f;\ nolub\ f\ xs\ l\ x\ u;\ l < x;\ x < u;\ x \notin EQ\ f\ xs;$
$DLO.I\ f\ (x\#xs);\ l < y;\ y < u]\!]$
$\Longrightarrow DLO.I\ f\ (y\#xs)$
⟨proof⟩

lemma *I-subst$_2$2*:
$nqfree\ f \Longrightarrow xs!l < x \wedge x < xs!u \Longrightarrow nolub\ f\ xs\ (xs!l)\ x\ (xs!u)$
$\Longrightarrow \forall x{\in}\{xs!l <..< xs!u\}.\ DLO.I\ f\ (x\#xs) \wedge x \notin EQ\ f\ xs$
$\Longrightarrow DLO.I\ (subst_2\ l\ u\ f)\ xs$
⟨proof⟩

end

definition
$qe\text{-}interior_1\ \varphi =$
$(let\ as = DLO.atoms_0\ \varphi;\ lbs = lbounds\ as;\ ubs = ubounds\ as;\ ebs = ebounds\ as;$
$intrs = [And\ (Atom(Less\ l\ u))\ (subst_2\ l\ u\ \varphi).\ l{\leftarrow}lbs,\ u{\leftarrow}ubs]$
$in\ list\text{-}disj\ (inf_-\ \varphi\ \#\ inf_+\ \varphi\ \#\ intrs\ @\ map\ (subst\ \varphi)\ ebs))$

lemma *dense-interval*:
assumes $finite\ L\ finite\ U\ l \in L\ u \in U\ l < x\ x < u\ P(x{::}'a{::}dlo)$
and *dense*: $\bigwedge y\ l\ u.\ [\![\ \forall y{\in}\{l{<}..{<}x\}.\ y \notin L;\ \ \forall y{\in}\{x{<}..{<}u\}.\ y \notin U;$
$l{<}x;x{<}u;\ l{<}y;y{<}u\ ]\!] \Longrightarrow P\ y$
shows $\exists l{\in}L.\exists u{\in}U.\ l{<}x \wedge x{<}u \wedge (\forall y{\in}\{l{<}..{<}x\}.\ y{\notin}L) \wedge (\forall y{\in}\{x{<}..{<}u\}.\ y{\notin}U)$
$\wedge (\forall y.\ l{<}y \wedge y{<}u \longrightarrow P\ y)$
⟨proof⟩

theorem *I-interior1*:
assumes $nqfree\ \varphi$ **shows** $DLO.I\ (qe\text{-}interior_1\ \varphi)\ xs = (\exists x.\ DLO.I\ \varphi\ (x\#xs))$
(**is** $?QE = ?EX$)
⟨proof⟩

lemma *qfree-asubst$_2$*: $qfree\ (asubst_2\ l\ u\ a)$
⟨proof⟩

lemma *qfree-subst$_2$*: $nqfree\ \varphi \Longrightarrow qfree\ (subst_2\ l\ u\ \varphi)$
⟨proof⟩

lemma *qfree-interior1*: $nqfree\ \varphi \Longrightarrow qfree(qe\text{-}interior_1\ \varphi)$
⟨*proof*⟩

definition $qe\text{-}interior = DLO.lift\text{-}nnf\text{-}qe\ qe\text{-}interior_1$

lemma *qfree-qe-interior*: $qfree(qe\text{-}interior\ \varphi)$
⟨*proof*⟩

lemma *I-qe-interior*: $DLO.I\ (qe\text{-}interior\ \varphi)\ xs = DLO.I\ \varphi\ xs$
⟨*proof*⟩

end

theory *QEdlo-inf*
imports *DLO*
begin

3.5 Quantifier elimination with infinitesimals

This section presents a new quantifier elimination procedure for dense linear orders based on (the simulation of) infinitesimals. It is a fairly straightforward adaptation of the analogous algorithm by Loos and Weispfenning for linear arithmetic described in §4.4.

fun $asubst\text{-}peps :: nat \Rightarrow atom \Rightarrow atom\ fm\ (\langle asubst_+ \rangle)$ **where**
$asubst\text{-}peps\ k\ (Less\ 0\ 0) = FalseF\ |$
$asubst\text{-}peps\ k\ (Less\ 0\ (Suc\ j)) = Atom(Less\ k\ j)\ |$
$asubst\text{-}peps\ k\ (Less\ (Suc\ i)\ 0) = (if\ i{=}k\ then\ TrueF$
$\quad else\ Or\ (Atom(Less\ i\ k))\ (Atom(Eq\ i\ k)))\ |$
$asubst\text{-}peps\ k\ (Less\ (Suc\ i)\ (Suc\ j)) = Atom(Less\ i\ j)\ |$
$asubst\text{-}peps\ k\ (Eq\ 0\ 0) = TrueF\ |$
$asubst\text{-}peps\ k\ (Eq\ 0\ \text{-}) = FalseF\ |$
$asubst\text{-}peps\ k\ (Eq\ \text{-}\ 0) = FalseF\ |$
$asubst\text{-}peps\ k\ (Eq\ (Suc\ i)\ (Suc\ j)) = Atom(Eq\ i\ j)$

abbreviation $subst\text{-}peps :: atom\ fm \Rightarrow nat \Rightarrow atom\ fm\ (\langle subst_+ \rangle)$ **where**
$subst_+\ \varphi\ k \equiv amap_{fm}\ (asubst_+\ k)\ \varphi$

definition $nolb\ \varphi\ xs\ l\ x = (\forall\, y{\in}\{l{<}..{<}x\}.\ y \notin LB\ \varphi\ xs)$

lemma *nolb-And*[*simp*]:
$\quad nolb\ (And\ \varphi_1\ \varphi_2)\ xs\ l\ x = (nolb\ \varphi_1\ xs\ l\ x \wedge nolb\ \varphi_2\ xs\ l\ x)$
⟨*proof*⟩

lemma *nolb-Or*[*simp*]:
$\quad nolb\ (Or\ \varphi_1\ \varphi_2)\ xs\ l\ x = (nolb\ \varphi_1\ xs\ l\ x \wedge nolb\ \varphi_2\ xs\ l\ x)$
⟨*proof*⟩

context notes [[*simp-depth-limit=3*]]
begin

lemma *innermost-intvl*:
$[\![$ *nqfree* φ; *nolb* φ *xs l x*; $l < x$; $x \notin$ *EQ* φ *xs*; *DLO.I* φ $(x\#xs)$; $l < y$; $y \leq x]\!]$
$\Longrightarrow$ *DLO.I* φ $(y\#xs)$
⟨*proof*⟩

lemma *I-subst-peps2*:
nqfree $\varphi \Longrightarrow xs!l < x \Longrightarrow$ *nolb* φ *xs* $(xs!l)$ $x \Longrightarrow x \notin$ *EQ* φ *xs*
$\Longrightarrow \forall\, y \in \{xs!l <..\ x\}$. *DLO.I* φ $(y\#xs)$
$\Longrightarrow$ *DLO.I* $(subst_+\ \varphi\ l)$ *xs*
⟨*proof*⟩

end

lemma *dense-interval*:
assumes *finite L* $l \in L$ $l < x$ $P(x{::}'a{::}dlo)$
and *dense*: $\bigwedge y\ l.\ [\![\ \forall\, y{\in}\{l{<}..{<}x\}.\ y \notin L;\ l{<}x;\ l{<}y;\ y{\leq}x\ ]\!] \Longrightarrow P\ y$
shows $\exists\, l{\in}L.\ l{<}x \wedge (\forall\, y{\in}\{l{<}..{<}x\}.\ y \notin L) \wedge (\forall\, y.\ l{<}y \wedge y{\leq}x \longrightarrow P\ y)$
⟨*proof*⟩

lemma *I-subst-peps*:
nqfree $\varphi \Longrightarrow$ *DLO.I* $(subst_+\ \varphi\ l)$ *xs* $\longrightarrow$
$(\exists\, leps{>}xs!l.\ \forall\, x.\ xs!l < x \wedge x \leq leps \longrightarrow$ *DLO.I* φ $(x\#xs))$
⟨*proof*⟩

definition
$qe\text{-}eps_1(\varphi)$ =
(*let as* = $DLO.atoms_0\ \varphi$; *lbs* = *lbounds as*; *ebs* = *ebounds as*
in list-disj $(inf_-\ \varphi\ \#\ map\ (subst_+\ \varphi)\ lbs\ @\ map\ (subst\ \varphi)\ ebs)$)

theorem *I-qe-eps1*:
assumes *nqfree* φ **shows** *DLO.I* $(qe\text{-}eps_1\ \varphi)$ *xs* = $(\exists\, x.$ *DLO.I* φ $(x\#xs))$
(**is** *?QE* = *?EX*)
⟨*proof*⟩

lemma *qfree-asubst-peps*: *qfree* $(asubst_+\ k\ a)$
⟨*proof*⟩

lemma *qfree-subst-peps*: *nqfree* $\varphi \Longrightarrow$ *qfree* $(subst_+\ \varphi\ k)$
⟨*proof*⟩

lemma $qfree\text{-}qe\text{-}eps_1$: *nqfree* $\varphi \Longrightarrow$ *qfree*$(qe\text{-}eps_1\ \varphi)$
⟨*proof*⟩

definition *qe-eps* = *DLO.lift-nnf-qe qe-eps*$_1$

lemma *qfree-qe-eps*: *qfree*(*qe-eps* φ)
⟨*proof*⟩

lemma *I-qe-eps*: *DLO.I* (*qe-eps* φ) *xs* = *DLO.I* φ *xs*
⟨*proof*⟩

end

4 Linear real arithmetic

theory *LinArith*
imports *QE HOL−Library.ListVector Complex-Main*
begin

declare *iprod-assoc*[*simp*]

4.1 Basics

4.1.1 Syntax and Semantics

datatype *atom* = *Less real real list* | *Eq real real list*

fun *is-Less* :: *atom* ⇒ *bool* **where**
is-Less (*Less r rs*) = *True* |
is-Less f = *False*

abbreviation *is-Eq* ≡ *Not* ∘ *is-Less*

lemma *is-Less-iff*: *is-Less f* = (∃ *r rs*. *f* = *Less r rs*)
⟨*proof*⟩

lemma *is-Eq-iff*: (∀ *i j*. *a* ≠ *Less i j*) = (∃ *i j*. *a* = *Eq i j*)
⟨*proof*⟩

fun neg_R :: *atom* ⇒ *atom fm* **where**
neg_R (*Less r t*) = *Or* (*Atom*(*Less* (−*r*) (−*t*))) (*Atom*(*Eq r t*)) |
neg_R (*Eq r t*) = *Or* (*Atom*(*Less r t*)) (*Atom*(*Less* (−*r*) (−*t*)))

fun *hd-coeff* :: *atom* ⇒ *real* **where**
hd-coeff (*Less r cs*) = (*case cs of* [] ⇒ *0* | *c*#- ⇒ *c*) |
hd-coeff (*Eq r cs*) = (*case cs of* [] ⇒ *0* | *c*#- ⇒ *c*)

definition $depends_R$ *a* = (*hd-coeff a* ≠ *0*)

fun $decr_R$ :: *atom* ⇒ *atom* **where**
$decr_R$ (*Less r rs*) = *Less r* (*tl rs*) |

$decr_R\ (Eq\ r\ rs) = Eq\ r\ (tl\ rs)$

fun I_R :: *atom* ⇒ *real list* ⇒ *bool* **where**
$I_R\ (Less\ r\ cs)\ xs = (r < \langle cs,xs\rangle)$ |
$I_R\ (Eq\ r\ cs)\ xs = (r = \langle cs,xs\rangle)$

definition $atoms_0 = ATOM.atoms_0\ depends_R$

interpretation R: $ATOM\ neg_R\ (\lambda a.\ True)\ I_R\ depends_R\ decr_R$
  rewrites $ATOM.atoms_0\ depends_R = atoms_0$
⟨*proof*⟩

⟨*ML*⟩

4.1.2 Shared constructions

fun *combine* :: (*real* ∗ *real list*) ⇒ (*real* ∗ *real list*) ⇒ *atom* **where**
$combine\ (r_1,cs_1)\ (r_2,cs_2) = Less\ (r_1 - r_2)\ (cs_2 - cs_1)$

definition $lbounds\ as = [(r/c,\ (-1/c) *_s\ cs).\ Less\ r\ (c\#cs) \leftarrow as,\ c>0]$
definition $ubounds\ as = [(r/c,\ (-1/c) *_s\ cs).\ Less\ r\ (c\#cs) \leftarrow as,\ c<0]$
definition $ebounds\ as = [(r/c,\ (-1/c) *_s\ cs).\ Eq\ r\ (c\#cs) \leftarrow as,\ c\neq 0]$

lemma *set-lbounds*:
$set(lbounds\ as) = \{(r/c,\ (-1/c) *_s\ cs)|r\ c\ cs.\ Less\ r\ (c\#cs) \in set\ as \wedge c>0\}$
⟨*proof*⟩
lemma *set-ubounds*:
$set(ubounds\ as) = \{(r/c,\ (-1/c) *_s\ cs)|r\ c\ cs.\ Less\ r\ (c\#cs) \in set\ as \wedge c<0\}$
⟨*proof*⟩
lemma *set-ebounds*:
$set(ebounds\ as) = \{(r/c,\ (-1/c) *_s\ cs)|r\ c\ cs.\ Eq\ r\ (c\#cs) \in set\ as \wedge c\neq 0\}$
⟨*proof*⟩

abbreviation *EQ* **where**
$EQ\ f\ xs \equiv \{(r - \langle cs,xs\rangle)/c|r\ c\ cs.\ Eq\ r\ (c\#cs) \in set(R.atoms_0\ f) \wedge c\neq 0\}$
abbreviation *LB* **where**
$LB\ f\ xs \equiv \{(r - \langle cs,xs\rangle)/c|r\ c\ cs.\ Less\ r\ (c\#cs) \in set(R.atoms_0\ f) \wedge c>0\}$
abbreviation *UB* **where**
$UB\ f\ xs \equiv \{(r - \langle cs,xs\rangle)/c|r\ c\ cs.\ Less\ r\ (c\#cs) \in set(R.atoms_0\ f) \wedge c<0\}$

fun *asubst* :: *real* ∗ *real list* ⇒ *atom* ⇒ *atom* **where**
$asubst\ (r,cs)\ (Less\ s\ (d\#ds)) = Less\ (s - d*r)\ (d *_s\ cs + ds)$ |
$asubst\ (r,cs)\ (Eq\ s\ (d\#ds)) = Eq\ (s - d*r)\ (d *_s\ cs + ds)$ |
$asubst\ (r,cs)\ (Less\ s\ []) = Less\ s\ []$ |

asubst (r,cs) $(Eq\ s\ [])$ = $Eq\ s\ []$

abbreviation *subst* φ *rcs* $\equiv$ map_{fm} (*asubst rcs*) φ

definition *eval* :: *real* $*$ *real list* $\Rightarrow$ *real list* $\Rightarrow$ *real* **where**
eval rcs xs = *fst rcs* + $\langle$*snd rcs*,*xs*$\rangle$

lemma *I-asubst*:
I_R (*asubst t a*) *xs* = I_R *a* (*eval t xs* # *xs*)
$\langle$*proof*$\rangle$

lemma *I-subst*:
qfree $\varphi \Longrightarrow$ *R.I* (*subst* φ *t*) *xs* = *R.I* φ (*eval t xs* # *xs*)
$\langle$*proof*$\rangle$
lemma *I-subst-pretty*:
qfree $\varphi \Longrightarrow$ *R.I* (*subst* φ (*r*,*cs*)) *xs* = *R.I* φ ((*r* + $\langle$*cs*,*xs*$\rangle$) # *xs*)
$\langle$*proof*$\rangle$

fun *min-inf* :: *atom fm* $\Rightarrow$ *atom fm* ($\langle inf_-\rangle$) **where**
inf_- (*And* φ_1 φ_2) = *and* (inf_- φ_1) (inf_- φ_2) |
inf_- (*Or* φ_1 φ_2) = *or* (inf_- φ_1) (inf_- φ_2) |
inf_- (*Atom*(*Less r* (*c*#*cs*))) =
(*if c*<*0 then TrueF else if c*>*0 then FalseF else Atom*(*Less r cs*)) |
inf_- (*Atom*(*Eq r* (*c*#*cs*))) = (*if c*=*0 then Atom*(*Eq r cs*) *else FalseF*) |
inf_- φ = φ

fun *plus-inf* :: *atom fm* $\Rightarrow$ *atom fm* ($\langle inf_+\rangle$) **where**
inf_+ (*And* φ_1 φ_2) = *and* (inf_+ φ_1) (inf_+ φ_2) |
inf_+ (*Or* φ_1 φ_2) = *or* (inf_+ φ_1) (inf_+ φ_2) |
inf_+ (*Atom*(*Less r* (*c*#*cs*))) =
(*if c*>*0 then TrueF else if c*<*0 then FalseF else Atom*(*Less r cs*)) |
inf_+ (*Atom*(*Eq r* (*c*#*cs*))) = (*if c*=*0 then Atom*(*Eq r cs*) *else FalseF*) |
inf_+ φ = φ

lemma *qfree-min-inf*: *qfree* $\varphi \Longrightarrow$ *qfree*(inf_- φ)
$\langle$*proof*$\rangle$

lemma *qfree-plus-inf*: *qfree* $\varphi \Longrightarrow$ *qfree*(inf_+ φ)
$\langle$*proof*$\rangle$

lemma *min-inf*:
nqfree f $\Longrightarrow \exists x.\ \forall y \le x.$ *R.I* (inf_- *f*) *xs* = *R.I f* (*y* # *xs*)
(**is** - $\Longrightarrow \exists x.$ *?P f x*)
$\langle$*proof*$\rangle$

lemma *plus-inf*:
nqfree f $\Longrightarrow \exists x.\ \forall y \ge x.$ *R.I* (inf_+ *f*) *xs* = *R.I f* (*y* # *xs*)
(**is** - $\Longrightarrow \exists x.$ *?P f x*)

⟨*proof*⟩

context notes [[*simp-depth-limit* = *4*]]
begin

lemma *LBex*:
⟦ *nqfree f*; *R.I f* $(x\#xs)$; $\neg$*R.I* $(inf_{-}\ f)$ *xs*; $x \notin EQ\ f\ xs$ ⟧
$\Longrightarrow \exists\, l \in LB\ f\ xs.\ l < x$
⟨*proof*⟩

lemma *UBex*:
⟦ *nqfree f*; *R.I f* $(x\#xs)$; $\neg$*R.I* $(inf_{+}\ f)$ *xs*; $x \notin EQ\ f\ xs$ ⟧
$\Longrightarrow \exists\, u \in UB\ f\ xs.\ x < u$
⟨*proof*⟩

end

lemma *finite-LB*: *finite*(*LB f xs*)
⟨*proof*⟩

lemma *finite-UB*: *finite*(*UB f xs*)
⟨*proof*⟩

end

theory *QElin*
imports *LinArith*
begin

4.2 Fourier

definition $qe\text{-}FM_1$ :: *atom list* ⇒ *atom fm* **where**
$qe\text{-}FM_1$ *as* = *list-conj* [*Atom*(*combine p q*). *p*←*lbounds as*, *q*←*ubounds as*]

theorem $I\text{-}qe\text{-}FM_1$:
assumes *less*: $\forall\, a \in$ *set as. is-Less a* **and** *dep*: $\forall\, a \in$ *set as.* $depends_R\ a$
shows *R.I* $(qe\text{-}FM_1\ as)$ *xs* = $(\exists\, x.\ \forall\, a \in set\ as.\ I_R\ a\ (x\#xs))$ (**is** *?L* = *?R*)
⟨*proof*⟩

corollary $I\text{-}qe\text{-}FM_1\text{-}pretty$:
$\forall\, a \in$ *set as. is-Less a* $\wedge$ $depends_R\ a \Longrightarrow$ *R.is-dnf-qe* $qe\text{-}FM_1$ *as*
⟨*proof*⟩

fun $subst_0$ :: *atom* ⇒ *atom* ⇒ *atom* **where**
$subst_0$ (*Eq r* $(c\#cs)$) *a* = (*case a of*
  Less s $(d\#ds) \Rightarrow$ *Less* $(s - (r*d)/c)$ $(ds - (d/c) *_s cs)$

$| \ Eq\ s\ (d\#ds) \Rightarrow Eq\ (s - (r{*}d)/c)\ (ds - (d/c) *_s cs))$

lemma $subst_0$*-pretty*:
$subst_0\ (Eq\ r\ (c\#cs))\ (Less\ s\ (d\#ds)) = Less\ (s - (r{*}d)/c)\ (ds - (d/c) *_s cs)$
$subst_0\ (Eq\ r\ (c\#cs))\ (Eq\ s\ (d\#ds)) = Eq\ (s - (r{*}d)/c)\ (ds - (d/c) *_s cs)$
⟨*proof*⟩

lemma *I-subst*$_0$: $depends_R\ a \Longrightarrow c \neq 0 \Longrightarrow$
$I_R\ (subst_0\ (Eq\ r\ (c\#cs))\ a)\ xs = I_R\ a\ ((r - \langle cs,xs\rangle)/c\ \#\ xs)$
⟨*proof*⟩

interpretation R_e:
ATOM-EQ $neg_R\ (\lambda a.\ True)\ I_R\ depends_R\ decr_R$
$(\lambda Eq\ \text{-}\ (c\#\text{-}) \Rightarrow c \neq 0 \mid \text{-} \Rightarrow False)$
$(\lambda Eq\ r\ cs \Rightarrow r{=}0 \wedge (\forall c{\in}\ set\ cs.\ c{=}0) \mid \text{-} \Rightarrow False)\ subst_0$
⟨*proof*⟩

definition *qe-FM* = R_e*.lift-dnfeq-qe* *qe-FM*$_1$

lemma *qfree-qe-FM*$_1$: *qfree* (*qe-FM*$_1$ *as*)
⟨*proof*⟩

corollary *I-qe-FM*: *R.I* (*qe-FM* φ) *xs* = *R.I* φ *xs*
⟨*proof*⟩

theorem *qfree-qe-FM*: *qfree* (*qe-FM f*)
⟨*proof*⟩

4.2.1 Tests

lemmas *qesimps* = *qe-FM-def* R_e*.lift-dnfeq-qe-def* R_e*.lift-eq-qe-def* *R.qelim-def* *qe-FM*$_1$*-def* *lbounds-def* *ubounds-def* *list-conj-def* *list-disj-def* *and-def* *or-def* $depends_R$*-def*

lemma *qe-FM*(*TrueF*) = *TrueF*
⟨*proof*⟩

lemma
qe-FM(*ExQ* (*And* (*Atom*(*Less* *0* [*1*])) (*Atom*(*Less* *0* [−*1*])))) = *Atom*(*Less* *0* [])
⟨*proof*⟩

lemma
qe-FM(*ExQ* (*And* (*Atom*(*Less* *0* [*1*])) (*Atom*(*Less* (− *1*) [−*1*])))) = *Atom*(*Less* (− *1*) [])
⟨*proof*⟩

end

theory *QElin-opt*
imports *QElin*
begin

4.2.2 An optimization

Atoms are simplified asap.

definition
$asimp\ a = (case\ a\ of$
$\ Less\ r\ cs \Rightarrow (if\ \forall\, c\in\ set\ cs.\ c = 0$
$\qquad then\ if\ r<0\ then\ TrueF\ else\ FalseF$
$\qquad else\ Atom\ a)\ |$
$\ Eq\ r\ cs \Rightarrow (if\ \forall\, c\in\ set\ cs.\ c = 0$
$\qquad then\ if\ r=0\ then\ TrueF\ else\ FalseF\ else\ Atom\ a))$

lemma *asimp-pretty*:
$asimp\ (Less\ r\ cs) =$
$\ (if\ \forall\, c\in\ set\ cs.\ c = 0$
$\ \ then\ if\ r<0\ then\ TrueF\ else\ FalseF$
$\ \ else\ Atom(Less\ r\ cs))$
$asimp\ (Eq\ r\ cs) =$
$\ (if\ \forall\, c\in\ set\ cs.\ c = 0$
$\ \ then\ if\ r=0\ then\ TrueF\ else\ FalseF$
$\ \ else\ Atom(Eq\ r\ cs))$
$\langle proof\rangle$

definition $qe\text{-}FMo_1 :: atom\ list \Rightarrow atom\ fm$ **where**
$qe\text{-}FMo_1\ as = list\text{-}conj\ [asimp(combine\ p\ q).\ p\leftarrow lbounds\ as,\ q\leftarrow ubounds\ as]$

lemma *I-asimp*: $R.I\ (asimp\ a)\ xs = I_R\ a\ xs$
$\langle proof\rangle$

lemma $I\text{-}qe\text{-}FMo_1$: $R.I\ (qe\text{-}FMo_1\ as)\ xs = R.I\ (qe\text{-}FM_1\ as)\ xs$
$\langle proof\rangle$

definition $qe\text{-}FMo = R_e.lift\text{-}dnfeq\text{-}qe\ qe\text{-}FMo_1$

lemma $qfree\text{-}qe\text{-}FMo_1$: $qfree\ (qe\text{-}FMo_1\ as)$
$\langle proof\rangle$

corollary *I-qe-FMo*: $R.I\ (qe\text{-}FMo\ \varphi)\ xs = R.I\ \varphi\ xs$
$\langle proof\rangle$

theorem *qfree-qe-FMo*: $qfree\ (qe\text{-}FMo\ f)$
$\langle proof\rangle$

end

theory *FRE*
imports *LinArith*
begin

4.3 Ferrante-Rackoff

This section formalizes a slight variant of Ferrante and Rackoff's algorithm [2]. We consider equalities separately, which improves performance.

fun $\mathit{between} :: \mathit{real} * \mathit{real\ list} \Rightarrow \mathit{real} * \mathit{real\ list} \Rightarrow \mathit{real} * \mathit{real\ list}$
where $\mathit{between}\ (r,cs)\ (s,ds) = ((r+s)/2,\ (1/2) *_s\ (cs+ds))$

definition $FR_1 :: \mathit{atom\ fm} \Rightarrow \mathit{atom\ fm}$ **where**
$FR_1\ \varphi =$
$(\mathit{let\ as} = R.\mathit{atoms}_0\ \varphi;\ \mathit{lbs} = \mathit{lbounds\ as};\ \mathit{ubs} = \mathit{ubounds\ as};\ \mathit{ebs} = \mathit{ebounds\ as};$
$\quad \mathit{intrs} = [\mathit{subst}\ \varphi\ (\mathit{between}\ l\ u)\ .\ l \leftarrow \mathit{lbs},\ u \leftarrow \mathit{ubs}]$
$\mathit{in\ list\text{-}disj}\ (\mathit{inf}_-\ \varphi\ \#\ \mathit{inf}_+\ \varphi\ \#\ \mathit{intrs}\ @\ \mathit{map}\ (\mathit{subst}\ \varphi)\ \mathit{ebs}))$

lemma *dense-interval*:
assumes $\mathit{finite}\ L\ \mathit{finite}\ U\ l \in L\ u \in U\ l < x\ x < u\ P(x::\mathit{real})$
and dense: $\bigwedge y\ l\ u.\ [\![\ \forall y \in \{l<..<x\}.\ y \notin L;\ \ \forall y \in \{x<..<u\}.\ y \notin U;$
$\quad l<x;x<u;\ l<y;y<u\ ]\!] \Longrightarrow P\ y$
shows $\exists l \in L. \exists u \in U.\ l<u \wedge (\forall y.\ l<y \wedge y<u \longrightarrow P\ y)$
⟨*proof*⟩

lemma *dense*:
$[\![\ \mathit{nqfree}\ f;\ \forall y \in \{l<..<x\}.\ y \notin LB\ f\ xs;\ \forall y \in \{x<..<u\}.\ y \notin UB\ f\ xs;$
$\quad l < x;\ x < u;\ x \notin EQ\ f\ xs;\ \ R.I\ f\ (x\#xs);\ l < y;\ y < u]\!]$
$\Longrightarrow R.I\ f\ (y\#xs)$
⟨*proof*⟩

theorem *I-FR*$_1$:
assumes $\mathit{nqfree}\ \varphi$ **shows** $R.I\ (FR_1\ \varphi)\ xs = (\exists x.\ R.I\ \varphi\ (x\#xs))$
(**is** $?FR = ?EX$)
⟨*proof*⟩

definition $FR = R.\mathit{lift\text{-}nnf\text{-}qe}\ FR_1$

lemma *qfree-FR*$_1$: $\mathit{nqfree}\ \varphi \Longrightarrow \mathit{qfree}\ (FR_1\ \varphi)$
⟨*proof*⟩

theorem *I-FR*: $R.I\ (FR\ \varphi)\ xs = R.I\ \varphi\ xs$
⟨*proof*⟩

theorem *qfree-FR*: $\mathit{qfree}\ (FR\ \varphi)$
⟨*proof*⟩

end

theory *QElin-inf*
imports *LinArith*
begin

4.4 Quantifier elimination with infinitesimals

This section formalizes Loos and Weispfenning's quantifier elimination procedure based on (the simulation of) infinitesimals [3].

fun *asubst-peps* :: *real* ∗ *real list* ⇒ *atom* ⇒ *atom fm* (‹*asubst*$_+$›) **where**
asubst-peps (*r*,*cs*) (*Less s* (*d*#*ds*)) =
 (*if d*=*0 then Atom*(*Less s ds*) *else*
  let u = *s* − *d*∗*r*; *v* = *d* ∗$_s$ *cs* + *ds*; *less* = *Atom*(*Less u v*)
  in if d<*0 then less else Or less* (*Atom*(*Eq u v*))) |
asubst-peps rcs (*Eq r* (*d*#*ds*)) = (*if d*=*0 then Atom*(*Eq r ds*) *else FalseF*) |
asubst-peps rcs a = *Atom a*

abbreviation *subst-peps* :: *atom fm* ⇒ *real* ∗ *real list* ⇒ *atom fm* (‹*subst*$_+$›)
where *subst*$_+$ *φ rcs* ≡ *amap*$_{fm}$ (*asubst*$_+$ *rcs*) *φ*

definition *nolb f xs l x* = (∀ *y*∈{*l*<..<*x*}. *y* ∉ *LB f xs*)

lemma *nolb-And*[*simp*]:
 nolb (*And f g*) *xs l x* = (*nolb f xs l x* ∧ *nolb g xs l x*)
⟨*proof*⟩

lemma *nolb-Or*[*simp*]:
 nolb (*Or f g*) *xs l x* = (*nolb f xs l x* ∧ *nolb g xs l x*)
⟨*proof*⟩

context notes [[*simp-depth-limit*=*4*]]
begin

lemma *innermost-intvl*:
 ⟦ *nqfree f*; *nolb f xs l x*; *l* < *x*; *x* ∉ *EQ f xs*; *R.I f* (*x*#*xs*); *l* < *y*; *y* ≤ *x*⟧
 ⟹ *R.I f* (*y*#*xs*)
⟨*proof*⟩

definition *EQ2* = *EQ*

lemma *EQ2-Or*[*simp*]: *EQ2* (*Or f g*) *xs* = (*EQ2 f xs* ∪ *EQ2 g xs*)
⟨*proof*⟩

lemma *EQ2-And*[*simp*]: *EQ2* (*And f g*) *xs* = (*EQ2 f xs* ∪ *EQ2 g xs*)
⟨*proof*⟩

lemma *innermost-intvl2*:
$[\![$ *nqfree f*; *nolb f xs l x*; $l < x$; $x \notin$ *EQ2 f xs*; *R.I f* $(x\#xs)$; $l < y$; $y \leq x]\!]$
$\Longrightarrow$ *R.I f* $(y\#xs)$
⟨*proof*⟩

lemma *I-subst-peps2*:
nqfree f $\Longrightarrow r+\langle cs,xs\rangle < x \Longrightarrow$ *nolb f xs* $(r+\langle cs,xs\rangle)$ x
$\Longrightarrow \forall\, y \in \{r+\langle cs,xs\rangle <..\ x\}$. *R.I f* $(y\#xs) \wedge y \notin$ *EQ2 f xs*
$\Longrightarrow$ *R.I* $(subst_+\ f\ (r,cs))$ *xs*
⟨*proof*⟩

end

lemma *I-subst-peps*:
nqfree f $\Longrightarrow$ *R.I* $(subst_+\ f\ (r,cs))$ *xs* $\Longrightarrow$
$(\exists\, leps>r+\langle cs,xs\rangle.\ \forall\, x.\ r+\langle cs,xs\rangle < x \wedge x \leq leps \longrightarrow$ *R.I f* $(x\#xs))$
⟨*proof*⟩

lemma *dense-interval*:
assumes *finite L* $l \in L$ $l < x$ $P(x{::}real)$
and *dense*: $\bigwedge y\ l.\ [\![\ \forall\, y{\in}\{l{<}..{<}x\}.\ y \notin L;\ l{<}x;\ l{<}y;\ y{\le}x\ ]\!] \Longrightarrow P\ y$
shows $\exists\, l{\in}L.\ l{<}x \wedge (\forall\, y{\in}\{l{<}..{<}x\}.\ y \notin L) \wedge (\forall\, y.\ l{<}y \wedge y{\le}x \longrightarrow P\ y)$
⟨*proof*⟩

definition
$qe\text{-}eps_1(f) =$
$(let\ as = R.atoms_0\ f;\ lbs = lbounds\ as;\ ebs = ebounds\ as$
$\ in\ list\text{-}disj\ (inf_-\ f\ \#\ map\ (subst_+\ f)\ lbs\ @\ map\ (subst\ f)\ ebs))$

theorem *I-eps1*:
assumes *nqfree f* **shows** *R.I* $(qe\text{-}eps_1\ f)$ *xs* $= (\exists\, x.$ *R.I f* $(x\#xs))$
(**is** *?QE* = *?EX*)
⟨*proof*⟩

lemma *qfree-asubst-peps*: *qfree* $(asubst_+\ rcs\ a)$
⟨*proof*⟩

lemma *qfree-subst-peps*: *nqfree* $\varphi \Longrightarrow$ *qfree* $(subst_+\ \varphi\ rcs)$
⟨*proof*⟩

lemma $qfree\text{-}qe\text{-}eps_1$: *nqfree* $\varphi \Longrightarrow qfree(qe\text{-}eps_1\ \varphi)$
⟨*proof*⟩

definition *qe-eps* = *R.lift-nnf-qe* $qe\text{-}eps_1$

lemma *qfree-qe-eps*: *qfree*(*qe-eps* φ)
⟨*proof*⟩

lemma *I-qe-eps*: *R.I* (*qe-eps* φ) *xs* = *R.I* φ *xs*

$\langle proof \rangle$

end

5 Presburger arithmetic

theory *PresArith*
imports *QE HOL−Library.ListVector*
begin

declare *iprod-assoc*[*simp*]

5.1 Syntax

datatype *atom* =
  Le int int list | *Dvd int int int list* | *NDvd int int int list*

fun *divisor* :: *atom* ⇒ *int* **where**
divisor (*Le i ks*) = *1* |
divisor (*Dvd d i ks*) = *d* |
divisor (*NDvd d i ks*) = *d*

fun neg_Z :: *atom* ⇒ *atom fm* **where**
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*)

fun *hd-coeff* :: *atom* ⇒ *int* **where**
hd-coeff (*Le i ks*) = (*case ks of* [] ⇒ *0* | *k*#- ⇒ *k*) |
hd-coeff (*Dvd d i ks*) = (*case ks of* [] ⇒ *0* | *k*#- ⇒ *k*) |
hd-coeff (*NDvd d i ks*) = (*case ks of* [] ⇒ *0* | *k*#- ⇒ *k*)

fun $decr_Z$ :: *atom* ⇒ *atom* **where**
$decr_Z$ (*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*)

fun I_Z :: *atom* ⇒ *int list* ⇒ *bool* **where**
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* = (¬ *d dvd i*+$\langle ks,xs \rangle$)

definition $atoms_0$ = $ATOM.atoms_0$ ($\lambda a.$ *hd-coeff a* ≠ *0*)

interpretation *Z*:
  ATOM neg_Z ($\lambda a.$ *divisor a* ≠ *0*) I_Z ($\lambda a.$ *hd-coeff a* ≠ *0*) $decr_Z$
  rewrites $ATOM.atoms_0$ ($\lambda a.$ *hd-coeff a* ≠ *0*) = $atoms_0$
$\langle proof \rangle$

⟨*ML*⟩

abbreviation
hd-coeff-is1 a ≡
  (*case a of Le - -* ⇒ *hd-coeff a* ∈ {*1*,−*1*} | *-* ⇒ *hd-coeff a* = *1*)

fun *asubst* :: *int* ⇒ *int list* ⇒ *atom* ⇒ *atom* **where**
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*) |
asubst i' ks' a = *a*

abbreviation *subst* :: *int* ⇒ *int list* ⇒ *atom fm* ⇒ *atom fm*
where *subst i ks* ≡ map_{fm} (*asubst i ks*)

lemma *IZ-asubst*: I_Z (*asubst i ks a*) *xs* = I_Z *a* ((*i* + ⟨*ks*,*xs*⟩) # *xs*)
⟨*proof*⟩

lemma *I-subst*:
  qfree φ ⟹ *Z.I* φ ((*i* + ⟨*ks*,*xs*⟩) # *xs*) = *Z.I* (*subst i ks* φ) *xs*
⟨*proof*⟩

lemma *divisor-asubst*[*simp*]: *divisor* (*asubst i ks a*) = *divisor a*
⟨*proof*⟩

definition *lbounds as* = [(*i*,*ks*). *Le i* (*k#ks*) ← *as*, *k*>*0*]
definition *ubounds as* = [(*i*,*ks*). *Le i* (*k#ks*) ← *as*, *k*<*0*]
lemma *set-lbounds*:
  set(*lbounds as*) = {(*i*,*ks*)|*i k ks*. *Le i* (*k#ks*) ∈ *set as* ∧ *k*>*0*}
⟨*proof*⟩
lemma *set-ubounds*:
  set(*ubounds as*) = {(*i*,*ks*)|*i k ks*. *Le i* (*k#ks*) ∈ *set as* ∧ *k*<*0*}
⟨*proof*⟩

lemma *lbounds-append*[*simp*]: *lbounds*(*as* @ *bs*) = *lbounds as* @ *lbounds bs*
⟨*proof*⟩

5.2 LCM and lemmas

fun *zlcms* :: *int list* ⇒ *int* **where**
zlcms [] = *1* |
zlcms (*i#is*) = *lcm i* (*zlcms is*)

lemma *dvd-zlcms*: *i* ∈ *set is* ⟹ *i dvd zlcms is*

$\langle proof \rangle$

lemma *zlcms-pos*: $\forall\, i \in set\ is.\ i{\neq}0 \Longrightarrow zlcms\ is > 0$
$\langle proof \rangle$

lemma *zlcms0-iff* [*simp*]: $(zlcms\ is = 0) = (0 \in set\ is)$
$\langle proof \rangle$

lemma *elem-le-zlcms*: $\forall\, i \in set\ is.\ i \neq 0 \Longrightarrow i \in set\ is \Longrightarrow i \leq zlcms\ is$
$\langle proof \rangle$

5.3 Setting coeffiencients to 1 or -1

fun *hd-coeff1* :: $int \Rightarrow atom \Rightarrow atom$ **where**
$hd\text{-}coeff1\ m\ (Le\ i\ (k\#ks)) =$
$\quad (if\ k{=}0\ then\ Le\ i\ (k\#ks)$
$\quad\ else\ let\ m' = m\ div\ (abs\ k)\ in\ Le\ (m'{*}i)\ (sgn\ k\ \#\ (m'\ {*}_s\ ks)))\ |$
$hd\text{-}coeff1\ m\ (Dvd\ d\ i\ (k\#ks)) =$
$\quad (if\ k{=}0\ then\ Dvd\ d\ i\ (k\#ks)$
$\quad\ else\ let\ m' = m\ div\ k\ in\ Dvd\ (m'{*}d)\ (m'{*}i)\ (1\ \#\ (m'\ {*}_s\ ks)))\ |$
$hd\text{-}coeff1\ m\ (NDvd\ d\ i\ (k\#ks)) =$
$\quad (if\ k{=}0\ then\ NDvd\ d\ i\ (k\#ks)$
$\quad\ else\ let\ m' = m\ div\ k\ in\ NDvd\ (m'{*}d)\ (m'{*}i)\ (1\ \#\ (m'\ {*}_s\ ks)))\ |$
$hd\text{-}coeff1\ \text{-}\ a = a$

The def of *hd-coeff1* on *Dvd* and *NDvd* is different from the *Le* because it allows the resulting head coefficient to be 1 rather than 1 or -1. We show that the other version has the same semantics:

lemma $[\![\ k \neq 0;\ k\ dvd\ m\ ]\!] \Longrightarrow$
$\ \ I_Z\ (hd\text{-}coeff1\ m\ (Dvd\ d\ i\ (k\#ks)))\ (x\#e) = (let\ m' = m\ div\ (abs\ k)\ in$
$\ \ I_Z\ (Dvd\ (m'{*}d)\ (m'{*}i)\ (sgn\ k\ \#\ (m'\ {*}_s\ ks)))\ (x\#e))$
$\langle proof \rangle$

lemma *I-hd-coeff1-mult-a*: **assumes** $m{>}0$
shows $hd\text{-}coeff\ a\ dvd\ m\ |\ hd\text{-}coeff\ a = 0 \Longrightarrow I_Z\ (hd\text{-}coeff1\ m\ a)\ (m{*}x\#xs) = I_Z\ a\ (x\#xs)$
$\langle proof \rangle$

lemma *I-hd-coeff1-mult*: **assumes** $m{>}0$
shows $qfree\ \varphi \Longrightarrow \forall\ a \in set(Z.atoms_0\ \varphi).\ hd\text{-}coeff\ a\ dvd\ m \Longrightarrow$
$\ Z.I\ (map_{fm}\ (hd\text{-}coeff1\ m)\ \varphi)\ (m{*}x\#xs) = Z.I\ \varphi\ (x\#xs)$
$\langle proof \rangle$

end

theory *QEpres*

imports *PresArith*
begin

5.4 DNF-based quantifier elimination

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

lemma *I-hd-coeffs1*:
assumes *0*: $\forall\, a\in set\ as.\ hd\text{-}coeff\ a \neq 0$ **shows**
$(\exists\, x.\ \forall\, a \in set(hd\text{-}coeffs1\ as).\ I_Z\ a\ (x\#xs)) =$
$(\exists\, x.\ \forall\, a \in set\ as.\ I_Z\ a\ (x\#xs))$ (**is** *?B* = *?A*)
⟨*proof*⟩

abbreviation *is-dvd a* ≡ *case a of Le - -* ⇒ *False* | *-* ⇒ *True*

definition
$qe\text{-}pres_1$ *as* =
(*let ds* = *filter is-dvd as*; (*d*::*int*) = *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(*li*,*lks*).
Disj [*0*..*d* − *1*] (λ*n*.
list-conj(*map* (*Atom* ∘ *asubst* (*li* + *n*) (−*lks*)) *as*))))

Note the optimization in the case *ls* = []: only the divisibility atoms are tested, not the inequalities. This complicates the proof.

lemma *I-cyclic*:
assumes *is-dvd a* **and** *hd-coeff a* = *1* **and** *i mod divisor a* = *j mod divisor a*
shows $I_Z\ a\ (i\#e) = I_Z\ a\ (j\#e)$
⟨*proof*⟩

lemma $I\text{-}qe\text{-}pres_1$:
assumes *norm*: $\forall\, a \in set\ as.\ divisor\ a \neq 0$
and *hd*: $\forall\, a \in set\ as.\ hd\text{-}coeff\text{-}is1\ a$
shows $Z.I\ (qe\text{-}pres_1\ as)\ xs = (\exists\, x.\ \forall\, a\in set\ as.\ I_Z\ a\ (x\#xs))$
⟨*proof*⟩

lemma *divisors-hd-coeffs1*:
assumes *div0*: $\forall\, a\in set\ as.\ divisor\ a \neq 0$ **and** *hd0*: $\forall\, a\in set\ as.\ hd\text{-}coeff\ a \neq 0$
and *a*: $a\in set\ (hd\text{-}coeffs1\ as)$ **shows** $divisor\ a \neq 0$
⟨*proof*⟩

lemma *hd-coeff-is1-hd-coeffs1*:
assumes *hd0*: $\forall\, a\in set\ as.\ hd\text{-}coeff\ a \neq 0$

and *a*: $a{\in}set$ (*hd-coeffs1 as*) **shows** *hd-coeff-is1 a*
⟨*proof*⟩

lemma *I-qe-pres*$_1$*-o*:
$[\![\ \forall\, a \in set\ as.\ divisor\ a \neq 0;\ \forall\, a{\in}set\ as.\ hd\text{-}coeff\ a \neq 0\ ]\!] \Longrightarrow$
$Z.I\ ((qe\text{-}pres_1 \circ hd\text{-}coeffs1)\ as)\ e = (\exists\, x.\ \forall\, a{\in}\ set\ as.\ I_Z\ a\ (x\#e))$
⟨*proof*⟩

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

lemma *qfree-qe-pres-o*: $qfree\ ((qe\text{-}pres_1 \circ hd\text{-}coeffs1)\ as)$
⟨*proof*⟩

lemma *normal-qe-pres*$_1$*-o*:
$\forall\, a \in set\ as.\ hd\text{-}coeff\ a \neq 0 \wedge divisor\ a \neq 0 \Longrightarrow$
$Z.normal\ ((qe\text{-}pres_1 \circ hd\text{-}coeffs1)\ as)$
⟨*proof*⟩

theorem *I-pres-qe*: $Z.normal\ \varphi \Longrightarrow\ Z.I\ (qe\text{-}pres\ \varphi)\ xs = Z.I\ \varphi\ xs$
⟨*proof*⟩

theorem *qfree-pres-qe*: $qfree\ (qe\text{-}pres\ f)$
⟨*proof*⟩

end

theory *Cooper*
imports *PresArith*
begin

5.5 Cooper

This section formalizes Cooper's algorithm [1].

lemma *set-atoms0-iff*:
$qfree\ \varphi \Longrightarrow a \in set(Z.atoms_0\ \varphi) \longleftrightarrow a \in atoms\ \varphi \wedge hd\text{-}coeff\ a \neq 0$
⟨*proof*⟩

definition
$hd\text{-}coeffs1\ \varphi =$
$(let\ m = zlcms(map\ hd\text{-}coeff\ (Z.atoms_0\ \varphi))$
$in\ And\ (Atom(Dvd\ m\ 0\ [1]))\ (map_{fm}\ (hd\text{-}coeff1\ m)\ \varphi))$

lemma *I-hd-coeffs1*:
assumes *qfree* φ
shows $(\exists\, x.\ Z.I\ (hd\text{-}coeffs1\ \varphi)\ (x\#xs)) = (\exists\, x.\ Z.I\ \varphi\ (x\#xs))$ (**is** *?L = ?R*)
⟨*proof*⟩

fun *min-inf* :: *atom fm* ⇒ *atom fm* (‹*inf*$_-$›) **where**
inf$_-$ (*And* φ_1 φ_2) = *and* (*inf*$_-$ φ_1) (*inf*$_-$ φ_2) |
inf$_-$ (*Or* φ_1 φ_2) = *or* (*inf*$_-$ φ_1) (*inf*$_-$ φ_2) |
inf$_-$ (*Atom*(*Le i* (*k*#*ks*))) =
  (*if k<0 then TrueF else if k>0 then FalseF else Atom*(*Le i* (*0*#*ks*))) |
inf$_-$ φ = φ

definition
qe-cooper$_1$ φ =
 (*let as* = *Z.atoms*$_0$ φ; *d* = *zlcms*(*map divisor as*); *ls* = *lbounds as*
  in or (*Disj* [*0*..*d* − *1*] (λ*n. subst n* [] (*inf*$_-$ φ)))
    (*Disj ls* (λ(*i*,*ks*).
      Disj [*0*..*d* − *1*] (λ*n. subst* (*i* + *n*) (−*ks*) φ))))

lemma *min-inf*:
  nqfree f ⟹ ∀ *a*∈*set*(*Z.atoms*$_0$ *f*). *hd-coeff-is1 a*
  ⟹ ∃ *x*.∀ *y*<*x*. *Z.I* (*inf*$_-$ *f*) (*y* # *xs*) = *Z.I f* (*y* # *xs*)
  (**is** - ⟹ - ⟹ ∃ *x*. *?P f x*)
⟨*proof*⟩

lemma *min-inf-repeats*:
  nqfree φ ⟹ ∀ *a*∈*set*(*Z.atoms*$_0$ φ). *divisor a dvd d* ⟹
  Z.I (*inf*$_-$ φ) ((*x* − *k*∗*d*)#*xs*) = *Z.I* (*inf*$_-$ φ) (*x*#*xs*)
⟨*proof*⟩

lemma *atoms-subset*: *qfree f* ⟹ *set*(*Z.atoms*$_0$(*f*::*atom fm*)) ≤ *atoms f*
⟨*proof*⟩

lemma β:
  ⟦ *nqfree* φ; ∀ *a*∈*set*(*Z.atoms*$_0$ φ). *hd-coeff-is1 a*;
   ∀ *a*∈*set*(*Z.atoms*$_0$ φ). *divisor a dvd d*; *d* > *0*;
   ¬(∃ *j*∈{*0* .. *d* − *1*}. ∃ (*i*,*ks*) ∈ *set*(*lbounds*(*Z.atoms*$_0$ φ)).
     x = *i* − ⟨*ks*,*xs*⟩ + *j*); *Z.I* φ (*x*#*xs*) ⟧
  ⟹ *Z.I* φ ((*x*−*d*)#*xs*)
⟨*proof*⟩

lemma *periodic-finite-ex*:
  assumes *dpos*: (*0*::*int*) < *d* **and** *modd*: ∀ *x k*. *P x* = *P*(*x* − *k*∗*d*)
  shows (∃ *x*. *P x*) = (∃ *j*∈{*0*..*d* − *1*}. *P j*)
  (**is** *?LHS* = *?RHS*)
⟨*proof*⟩

lemma *cpmi-eq*: $(0::int) < D \Longrightarrow (\exists z. \forall x. x < z \longrightarrow (P\ x = P1\ x))$
$\Longrightarrow \forall x. \neg(\exists j \in \{0..D - 1\}. \exists b \in B. P(b+j)) \longrightarrow P\ (x) \longrightarrow P\ (x - D)$
$\Longrightarrow \forall x. \forall k. P1\ x = P1(x-k*D)$
$\Longrightarrow (\exists x. P(x)) = ((\exists j \in \{0..D - 1\}. P1(j)) \vee (\exists j \in \{0..D - 1\}. \exists b \in B. P(b+j)))$
⟨*proof*⟩

theorem *cp-thm*:
assumes *nq*: *nqfree* φ
and *u*: $\forall a \in set(Z.atoms_0\ \varphi)$. *hd-coeff-is1 a*
and *d*: $\forall a \in set(Z.atoms_0\ \varphi)$. *divisor a dvd d*
and *dp*: $d > 0$
shows $(\exists x. Z.I\ \varphi\ (x\#xs)) =$
$(\exists j \in \{0..d - 1\}. Z.I\ (inf_{-}\ \varphi)\ (j\#xs) \vee$
$(\exists (i,ks) \in set(lbounds(Z.atoms_0\ \varphi)). Z.I\ \varphi\ ((i - \langle ks,xs \rangle + j)\ \#\ xs)))$
(**is** $(\exists x. ?P\ (x)) = (\exists\ j \in\ ?D.\ ?M\ j \vee (\exists (i,ks) \in\ ?B.\ ?P\ (?I\ i\ ks + j))))$
⟨*proof*⟩

lemma *qfree-min-inf*[*simp*]: *qfree* $\varphi \Longrightarrow$ *qfree* $(inf_{-}\ \varphi)$
⟨*proof*⟩

lemma *I-qe-cooper*$_1$:
assumes *norm*: $\forall a \in atoms\ \varphi.\ divisor\ a \neq 0$
and *hd*: $\forall a \in set(Z.atoms_0\ \varphi)$. *hd-coeff-is1 a* **and** *nqfree* φ
shows $Z.I\ (qe\text{-}cooper_1\ \varphi)\ xs = (\exists x. Z.I\ \varphi\ (x\#xs))$
⟨*proof*⟩

lemma *divisor-hd-coeff1-neq0*:
qfree $\varphi \Longrightarrow a \in atoms\ \varphi \Longrightarrow divisor\ a \neq 0 \Longrightarrow$
divisor (*hd-coeff1* (*zlcms* (*map hd-coeff* $(Z.atoms_0\ \varphi)$)) *a*) $\neq 0$
⟨*proof*⟩

lemma *hd-coeff-is1-hd-coeff1*:
hd-coeff (*hd-coeff1 m a*) $\neq 0 \longrightarrow$ *hd-coeff-is1* (*hd-coeff1 m a*)
⟨*proof*⟩

lemma *I-cooper1-hd-coeffs1*: *Z.normal* $\varphi \Longrightarrow$ *nqfree* φ
$\Longrightarrow Z.I\ (qe\text{-}cooper_1(hd\text{-}coeffs1\ \varphi))\ xs = (\exists x. Z.I\ \varphi\ (x\ \#\ xs))$
⟨*proof*⟩

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

lemma *qfree-cooper1-hd-coeffs1*: *qfree* $\varphi \Longrightarrow$ *qfree* $(qe\text{-}cooper_1\ (hd\text{-}coeffs1\ \varphi))$
⟨*proof*⟩

lemma *normal-min-inf*: $Z.normal\ \varphi \Longrightarrow Z.normal(inf_{-}\ \varphi)$
⟨*proof*⟩

lemma *normal-cooper1*: $Z.normal\ \varphi \Longrightarrow Z.normal(qe\text{-}cooper_1\ \varphi)$
⟨*proof*⟩

lemma *normal-hd-coeffs1*: $qfree\ \varphi \Longrightarrow Z.normal\ \varphi \Longrightarrow Z.normal(hd\text{-}coeffs1\ \varphi)$
⟨*proof*⟩

theorem *I-cooper*: $Z.normal\ \varphi \Longrightarrow\ Z.I\ (qe\text{-}cooper\ \varphi)\ xs = Z.I\ \varphi\ xs$
⟨*proof*⟩

theorem *qfree-cooper*: $qfree\ (qe\text{-}cooper\ \varphi)$
⟨*proof*⟩

end

References

[1] 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.

[2] 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.

[3] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36:450–462, 1993.

[4] 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. www.in.tum.de/~nipkow/pubs/ijcar08.html.

[5] 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*. IOS Press, 2008. Proc. Marktoberdorf Summer School 2007, to appear.