

Quantifier Elimination for Linear Arithmetic

Tobias Nipkow

October 21, 2018

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	2
1.1	Atoms	5
2	Quantifier elimination	7
2.1	No Equality	7
2.1.1	DNF-based	7
2.1.2	NNF-based	9
2.2	With equality	10
3	DLO	12
3.1	Basics	12
3.2	DNF-based quantifier elimination	15
3.3	Examples	17
3.4	Interior Point Method	18
3.5	Quantifier elimination with infinitesimals	20

4	Linear real arithmetic	22
4.1	Basics	22
4.1.1	Syntax and Semantics	22
4.1.2	Shared constructions	23
4.2	Fourier	25
4.2.1	Tests	26
4.2.2	An optimization	27
4.3	Ferrante-Rackoff	28
4.4	Quantifier elimination with infinitesimals	29
5	Presburger arithmetic	31
5.1	Syntax	31
5.2	LCM and lemmas	32
5.3	Setting coefficients to 1 or -1	33
5.4	DNF-based quantifier elimination	34
5.5	Cooper	35

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  $\varphi_1 \varphi_2 \equiv Or (Neg \varphi_1) \varphi_2$ 

```

```

abbreviation AllQ where AllQ  $\varphi \equiv Neg(ExQ(Neg \varphi))$ 

```

```

definition neg where

```

```

neg  $\varphi = (if \varphi = TrueF then FalseF else if \varphi = FalseF then TrueF else Neg \varphi)$ 

```

```

definition and :: 'a fm  $\Rightarrow$  'a fm  $\Rightarrow$  'a fm where

```

```

and  $\varphi_1 \varphi_2 =$ 
  (if  $\varphi_1 = TrueF$  then  $\varphi_2$  else if  $\varphi_2 = TrueF$  then  $\varphi_1$  else
   if  $\varphi_1 = FalseF \vee \varphi_2 = FalseF$  then FalseF else And  $\varphi_1 \varphi_2$ )

```

```

definition or :: 'a fm  $\Rightarrow$  'a fm  $\Rightarrow$  'a fm where

```

```

or  $\varphi_1 \varphi_2 =$ 
  (if  $\varphi_1 = FalseF$  then  $\varphi_2$  else if  $\varphi_2 = FalseF$  then  $\varphi_1$  else

```

if $\varphi_1 = \text{TrueF} \vee \varphi_2 = \text{TrueF}$ then TrueF else Or $\varphi_1 \varphi_2$

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

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

abbreviation *Disj is f* \equiv *list-disj (map f is)*

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

fun *amap_fm* :: ('a \Rightarrow 'b fm) \Rightarrow 'a fm \Rightarrow '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 $\varphi_1 \varphi_2$) = and (*amap_fm* h φ_1) (*amap_fm* h φ_2) |
amap_fm h (Or $\varphi_1 \varphi_2$) = 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 \Rightarrow bool **where**
qfree(ExQ f) = False |
qfree(And $\varphi_1 \varphi_2$) = (*qfree* $\varphi_1 \wedge$ *qfree* φ_2) |
qfree(Or $\varphi_1 \varphi_2$) = (*qfree* $\varphi_1 \wedge$ *qfree* φ_2) |
qfree(Neg φ) = (*qfree* φ) |
qfree φ = True

lemma *qfree-and[simp]*: \llbracket *qfree* φ_1 ; *qfree* φ_2 $\rrbracket \Longrightarrow$ *qfree*(and $\varphi_1 \varphi_2$)
<proof>

lemma *qfree-or[simp]*: \llbracket *qfree* φ_1 ; *qfree* φ_2 $\rrbracket \Longrightarrow$ *qfree*(or $\varphi_1 \varphi_2$)
<proof>

lemma *qfree-neg[simp]*: *qfree*(neg φ) = *qfree* φ
<proof>

lemma *qfree-foldr-Or[simp]*:
qfree(foldr Or fs φ) = (*qfree* $\varphi \wedge (\forall \varphi \in$ set fs. *qfree* $\varphi)$)
<proof>

lemma *qfree-list-conj[simp]*:
assumes $\forall \varphi \in$ set fs. *qfree* φ **shows** *qfree*(*list-conj* fs)
<proof>

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

assumes $\forall \varphi \in \text{set fs. } \text{qfree } \varphi$ **shows** $\text{qfree}(\text{list-disj fs})$
 <proof>

lemma $\text{qfree-map-fm: } \text{qfree}(\text{map}_{fm} f \varphi) = \text{qfree } \varphi$
 <proof>

lemma atoms-list-disjE:
 $a \in \text{atoms}(\text{list-disj fs}) \implies a \in (\bigcup \varphi \in \text{set fs. } \text{atoms } \varphi)$
 <proof>

lemma atoms-list-conjE:
 $a \in \text{atoms}(\text{list-conj fs}) \implies a \in (\bigcup \varphi \in \text{set fs. } \text{atoms } \varphi)$
 <proof>

fun $\text{dnf} :: 'a \text{ fm} \Rightarrow 'a \text{ list list}$ **where**
 $\text{dnf TrueF} = [[]] \mid$
 $\text{dnf FalseF} = [] \mid$
 $\text{dnf (Atom } \varphi) = [[\varphi]] \mid$
 $\text{dnf (And } \varphi_1 \varphi_2) = [d1 @ d2. d1 \leftarrow \text{dnf } \varphi_1, d2 \leftarrow \text{dnf } \varphi_2] \mid$
 $\text{dnf (Or } \varphi_1 \varphi_2) = \text{dnf } \varphi_1 @ \text{dnf } \varphi_2$

fun $\text{ngfree} :: 'a \text{ fm} \Rightarrow \text{bool}$ **where**
 $\text{ngfree (Atom } a) = \text{True} \mid$
 $\text{ngfree TrueF} = \text{True} \mid$
 $\text{ngfree FalseF} = \text{True} \mid$
 $\text{ngfree (And } \varphi_1 \varphi_2) = (\text{ngfree } \varphi_1 \wedge \text{ngfree } \varphi_2) \mid$
 $\text{ngfree (Or } \varphi_1 \varphi_2) = (\text{ngfree } \varphi_1 \wedge \text{ngfree } \varphi_2) \mid$
 $\text{ngfree } \varphi = \text{False}$

lemma $\text{ngfree-qfree[simp]: } \text{ngfree } \varphi \implies \text{qfree } \varphi$
 <proof>

lemma $\text{ngfree-map-fm: } \text{ngfree}(\text{map}_{fm} f \varphi) = \text{ngfree } \varphi$
 <proof>

fun $\text{interpret} :: ('a \Rightarrow 'b \text{ list} \Rightarrow \text{bool}) \Rightarrow 'a \text{ fm} \Rightarrow 'b \text{ list} \Rightarrow \text{bool}$ **where**
 $\text{interpret } h \text{ TrueF } xs = \text{True} \mid$
 $\text{interpret } h \text{ FalseF } xs = \text{False} \mid$
 $\text{interpret } h \text{ (Atom } a) xs = h a xs \mid$
 $\text{interpret } h \text{ (And } \varphi_1 \varphi_2) xs = (\text{interpret } h \varphi_1 xs \wedge \text{interpret } h \varphi_2 xs) \mid$
 $\text{interpret } h \text{ (Or } \varphi_1 \varphi_2) xs = (\text{interpret } h \varphi_1 xs \vee \text{interpret } h \varphi_2 xs) \mid$
 $\text{interpret } h \text{ (Neg } \varphi) xs = (\neg \text{interpret } h \varphi xs) \mid$
 $\text{interpret } h \text{ (ExQ } \varphi) xs = (\exists x. \text{interpret } h \varphi (x\#xs))$

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 ngfree-aneg: ngfree(aneg a)
assumes anormal-aneg: anormal a  $\Longrightarrow$   $\forall b \in \text{atoms}(\text{aneg } a). \text{anormal } b$ 

fixes Ia :: 'a  $\Rightarrow$  'b list  $\Rightarrow$  bool
assumes Ia-aneg: interpret Ia (aneg a) xs = ( $\neg$  Ia a xs)

fixes depends0 :: 'a  $\Rightarrow$  bool
and decr :: 'a  $\Rightarrow$  'a
assumes not-dep-decr:  $\neg \text{depends}_0$  a  $\Longrightarrow$  Ia a (x#xs) = Ia (decr a) xs
assumes anormal-decr:  $\neg \text{depends}_0$  a  $\Longrightarrow$  anormal a  $\Longrightarrow$  anormal(decr a)

begin

```

```

fun atoms0 :: 'a fm  $\Rightarrow$  'a list where
atoms0 TrueF = [] |
atoms0 FalseF = [] |
atoms0 (Atom a) = (if depends0 a then [a] else []) |
atoms0 (And  $\varphi_1$   $\varphi_2$ ) = atoms0  $\varphi_1$  @ atoms0  $\varphi_2$  |
atoms0 (Or  $\varphi_1$   $\varphi_2$ ) = atoms0  $\varphi_1$  @ atoms0  $\varphi_2$  |
atoms0 (Neg  $\varphi$ ) = atoms0  $\varphi$ 

```

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 *ngfree-nnf*: *qfree* φ \Longrightarrow *ngfree*(*nnf* φ)
<proof>

lemma *qfree-nnf[simp]*: *qfree*(*nnf* φ) = *qfree* φ
<proof>

lemma *I-neg[simp]*: $I (\text{neg } \varphi) xs = I (\text{Neg } \varphi) xs$
<proof>

lemma *I-and[simp]*: $I (\text{and } \varphi_1 \varphi_2) xs = I (\text{And } \varphi_1 \varphi_2) xs$
<proof>

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

lemma *I-or[simp]*: $I (\text{or } \varphi_1 \varphi_2) xs = I (\text{Or } \varphi_1 \varphi_2) xs$
<proof>

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

lemma *I-nnf*: $I (\text{nnf } \varphi) xs = I \varphi xs$
<proof>

lemma *I-dnf*:
 $\text{ngfree } \varphi \implies (\exists as \in \text{set } (\text{dnf } \varphi). \forall a \in \text{set } as. I_a a xs) = I \varphi xs$
<proof>

definition *normal* $\varphi = (\forall a \in \text{atoms } \varphi. \text{anormal } a)$

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* φ_2
normal (Or $\varphi_1 \varphi_2$) \longleftrightarrow *normal* $\varphi_1 \vee$ *normal* φ_2
normal (Neg φ) \longleftrightarrow *normal* φ
normal (ExQ φ) \longleftrightarrow *normal* φ
<proof>

lemma *normal-aneg[simp]*: *anormal* a \implies *normal* (aneg a)
<proof>

lemma *normal-and[simp]*:
normal $\varphi_1 \implies$ *normal* $\varphi_2 \implies$ *normal* (and $\varphi_1 \varphi_2$)
<proof>

lemma *normal-or[simp]*:
normal $\varphi_1 \implies$ *normal* $\varphi_2 \implies$ *normal* (or $\varphi_1 \varphi_2$)
<proof>

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

<proof>

lemma *normal-nnf*: $normal\ \varphi \implies normal(nnf\ \varphi)$
<proof>

lemma *normal-map-fm*:
 $\forall a. anormal(f\ a) = anormal(a) \implies normal\ (map_{fm}\ f\ \varphi) = normal\ \varphi$
<proof>

lemma *anormal-nnf*:
 $qfree\ \varphi \implies normal\ \varphi \implies \forall a \in atoms(nnf\ \varphi). anormal\ a$
<proof>

lemma *atoms-dnf*: $nqfree\ \varphi \implies as \in set(dnf\ \varphi) \implies a \in set\ as \implies a \in atoms\ \varphi$
<proof>

lemma *anormal-dnf-nnf*:
 $as \in set(dnf(nnf\ \varphi)) \implies qfree\ \varphi \implies normal\ \varphi \implies a \in set\ as \implies anormal\ a$
<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-conj indep))

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

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

lemma *I-qelim*:

assumes qe: $\bigwedge as. (\forall a \in \text{set } as. \text{depends}_0 a) \Longrightarrow \text{is-dnf-qe } qe \text{ as}$

shows *is-dnf-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-dnf-qe qe (And $\varphi_1 \varphi_2$) = and (*lift-dnf-qe* qe φ_1) (*lift-dnf-qe* qe φ_2) |

lift-dnf-qe qe (Or $\varphi_1 \varphi_2$) = or (*lift-dnf-qe* qe φ_1) (*lift-dnf-qe* qe φ_2) |

lift-dnf-qe qe (Neg φ) = neg(*lift-dnf-qe* qe φ) |

lift-dnf-qe qe (ExQ φ) = Disj (dnf(nnf(*lift-dnf-qe* qe φ))) (qelim qe) |

lift-dnf-qe qe $\varphi = \varphi$

lemma *qfree-lift-dnf-qe*: $(\bigwedge as. (\forall a \in \text{set } as. \text{depends}_0 a) \Longrightarrow \text{qfree}(qe as))$

$\Longrightarrow \text{qfree}(\text{lift-dnf-qe } qe \varphi)$

<proof>

lemma *qfree-lift-dnf-qe2*: $qe \in \text{lists } |\text{depends}_0| \rightarrow |\text{qfree}|$

$\Longrightarrow \text{qfree}(\text{lift-dnf-qe } qe \varphi)$

<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 $\bigwedge as. (\forall a \in \text{set } as. \text{depends}_0 a) \Longrightarrow \text{qfree}(qe as)$

and $\bigwedge as. (\forall a \in \text{set } as. \text{depends}_0 a) \Longrightarrow \text{is-dnf-qe } qe \text{ as}$

shows *I* (*lift-dnf-qe* qe φ) xs = *I* φ xs

<proof>

lemma *I-lift-dnf-qe2*:

assumes $qe \in \text{lists } |\text{depends}_0| \rightarrow |\text{qfree}|$

and $\forall as \in \text{lists } |\text{depends}_0|. \text{is-dnf-qe } qe \text{ as}$

shows *I* (*lift-dnf-qe* qe φ) xs = *I* φ xs

<proof>

Quantifier elimination with invariant (needed for Presburger):

lemma *I-qelim-anormal*:

assumes qe: $\bigwedge xs \text{ as}. \forall a \in \text{set } as. \text{depends}_0 a \wedge \text{anormal } a \Longrightarrow \text{is-dnf-qe } qe \text{ as}$

and nm: $\forall a \in \text{set } as. \text{anormal } a$

shows *I* (qelim qe as) xs = $(\exists x. \forall a \in \text{set } as. I_a a (x \# xs))$

<proof>

context notes $[[simp\text{-}depth\text{-}limit = 5]]$
begin

lemma *anormal-atoms-qelim*:

$(\bigwedge as. \forall a \in set\ as. depends_0\ a \wedge anormal\ a \implies normal(qe\ as)) \implies$
 $\forall a \in set\ as. anormal\ a \implies a \in atoms(qelim\ qe\ as) \implies anormal\ a$
 $\langle proof \rangle$

lemma *normal-lift-dnf-qe*:

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

end

context notes $[[simp\text{-}depth\text{-}limit = 9]]$
begin

lemma *I-lift-dnf-qe-anormal*:

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

end

lemma *I-lift-dnf-qe-anormal2*:

assumes $qe \in lists\ |depends_0| \rightarrow |qfree|$
and $qe \in lists\ (|depends_0| \cap |anormal|) \rightarrow |normal|$
and $\forall as \in lists\ (|depends_0| \cap |anormal|). is\text{-}dnf\text{-}qe\ qe\ as$
shows $normal\ f \implies I\ (lift\text{-}dnf\text{-}qe\ qe\ f)\ xs = I\ f\ xs$
 $\langle proof \rangle$

2.1.2 NNF-based

fun *lift-nnf-qe* :: $('a\ fm \Rightarrow 'a\ fm) \Rightarrow 'a\ fm \Rightarrow 'a\ fm$ **where**

lift-nnf-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-nnf-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-nnf-qe $qe\ (Neg\ \varphi) = neg\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)$ |
lift-nnf-qe $qe\ (ExQ\ \varphi) = qe\ (nnf\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi))$ |
lift-nnf-qe $qe\ \varphi = \varphi$

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

lemma *qfree-lift-nnf-qe2*:

$qe \in |ngfree| \rightarrow |qfree| \implies qfree(\text{lift-nnf-qe } qe \ \varphi)$
 <proof>

lemma *I-lift-nnf-qe*:

assumes $\bigwedge \varphi. ngfree \ \varphi \implies qfree(qe \ \varphi)$
and $\bigwedge xs \ \varphi. ngfree \ \varphi \implies I \ (qe \ \varphi) \ xs = (\exists x. I \ \varphi \ (x\#xs))$
shows $I \ (\text{lift-nnf-qe } qe \ \varphi) \ xs = I \ \varphi \ xs$
 <proof>

lemma *I-lift-nnf-qe2*:

assumes $qe \in |ngfree| \rightarrow |qfree|$
and $\forall \varphi \in |ngfree|. \forall xs. I \ (qe \ \varphi) \ xs = (\exists x. I \ \varphi \ (x\#xs))$
shows $I \ (\text{lift-nnf-qe } qe \ \varphi) \ xs = I \ \varphi \ xs$
 <proof>

lemma *normal-lift-nnf-qe*:

assumes $\bigwedge \varphi. ngfree \ \varphi \implies qfree(qe \ \varphi)$
and $\bigwedge \varphi. ngfree \ \varphi \implies normal \ \varphi \implies normal(qe \ \varphi)$
shows $normal \ \varphi \implies normal(\text{lift-nnf-qe } qe \ \varphi)$
 <proof>

lemma *I-lift-nnf-qe-normal*:

assumes $\bigwedge \varphi. ngfree \ \varphi \implies qfree(qe \ \varphi)$
and $\bigwedge \varphi. ngfree \ \varphi \implies normal \ \varphi \implies normal(qe \ \varphi)$
and $\bigwedge xs \ \varphi. normal \ \varphi \implies ngfree \ \varphi \implies I \ (qe \ \varphi) \ xs = (\exists x. I \ \varphi \ (x\#xs))$
shows $normal \ \varphi \implies I \ (\text{lift-nnf-qe } qe \ \varphi) \ xs = I \ \varphi \ xs$
 <proof>

lemma *I-lift-nnf-qe-normal2*:

assumes $qe \in |ngfree| \rightarrow |qfree|$
and $qe \in |ngfree| \cap |normal| \rightarrow |normal|$
and $\forall \varphi \in |normal| \cap |ngfree|. \forall xs. I \ (qe \ \varphi) \ xs = (\exists x. I \ \varphi \ (x\#xs))$
shows $normal \ \varphi \implies I \ (\text{lift-nnf-qe } qe \ \varphi) \ xs = I \ \varphi \ xs$
 <proof>

end

2.2 With equality

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

locale *ATOM-EQ* = *ATOM* +

fixes *solvable*₀ :: 'a ⇒ bool

and *trivial* :: 'a ⇒ bool

and *subst*₀ :: 'a ⇒ 'a ⇒ 'a

assumes *subst*₀:

[*solvable*₀ eq; ¬*trivial* eq; *I*_a eq (x#xs); *depends*₀ a]
 ⇒ *I*_a (*subst*₀ eq a) xs = *I*_a a (x#xs)

and *trivial*: *trivial* eq ⇒ *I*_a eq xs

and solvable: $\text{solvable}_0 \text{ eq} \implies \exists x. I_a \text{ eq } (x\#xs)$
and is-triv-self-subst: $\text{solvable}_0 \text{ eq} \implies \text{trivial } (\text{subst}_0 \text{ eq } \text{eq})$

begin

definition $\text{lift-eq-qe} :: ('a \text{ list} \implies 'a \text{ fm}) \implies 'a \text{ list} \implies 'a \text{ fm}$ **where**
 $\text{lift-eq-qe } qe \text{ as} =$
 $(\text{let } \text{as} = [a \leftarrow \text{as}. \neg \text{trivial } a]$
 $\text{in case } [a \leftarrow \text{as}. \text{solvable}_0 a] \text{ of}$
 $\quad [] \implies qe \text{ as}$
 $\quad | \text{eq} \# \text{eqs} \implies$
 $\quad \quad (\text{let } \text{ineqs} = [a \leftarrow \text{as}. \neg \text{solvable}_0 a]$
 $\quad \quad \text{in list-conj } (\text{map } (\text{Atom} \circ (\text{subst}_0 \text{ eq})) (\text{eqs} @ \text{ineqs}))))$

theorem $I\text{-lift-eq-qe}$:

assumes $\text{dep}: \forall a \in \text{set } \text{as}. \text{depends}_0 a$

assumes $\text{qe}: \bigwedge \text{as}. (\forall a \in \text{set } \text{as}. \text{depends}_0 a \wedge \neg \text{solvable}_0 a) \implies$
 $I (qe \text{ as}) \text{ xs} = (\exists x. \forall a \in \text{set } \text{as}. I_a a (x\#xs))$

shows $I (\text{lift-eq-qe } qe \text{ as}) \text{ xs} = (\exists x. \forall a \in \text{set } \text{as}. I_a a (x\#xs))$

(is ?L = ?R)

$\langle \text{proof} \rangle$

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

lemma qfree-lift-eq-qe :

$(\bigwedge \text{as}. \forall a \in \text{set } \text{as}. \text{depends}_0 a \implies \text{qfree } (qe \text{ as})) \implies$

$\forall a \in \text{set } \text{as}. \text{depends}_0 a \implies \text{qfree}(\text{lift-eq-qe } qe \text{ as})$

$\langle \text{proof} \rangle$

lemma qfree-lift-dnf-qe : $(\bigwedge \text{as}. (\forall a \in \text{set } \text{as}. \text{depends}_0 a) \implies \text{qfree}(qe \text{ as}))$

$\implies \text{qfree}(\text{lift-dnf-qe } qe \text{ as})$

$\langle \text{proof} \rangle$

lemma $I\text{-lift-dnf-qe}$:

$(\bigwedge \text{as}. (\forall a \in \text{set } \text{as}. \text{depends}_0 a) \implies \text{qfree}(qe \text{ as})) \implies$

$(\bigwedge \text{as}. (\forall a \in \text{set } \text{as}. \text{depends}_0 a \wedge \neg \text{solvable}_0 a) \implies \text{is-dnf-qe } qe \text{ as}) \implies$

$I (\text{lift-dnf-qe } qe \text{ as}) \text{ xs} = I \text{ as } \text{xs}$

$\langle \text{proof} \rangle$

lemma $I\text{-lift-dnf-qe2}$:

$qe \in \text{lists } |\text{depends}_0| \rightarrow |\text{qfree}| \implies$

$(\forall \text{as} \in \text{lists } (|\text{depends}_0| \cap \neg |\text{solvable}_0|). \text{is-dnf-qe } qe \text{ as}) \implies$

$I (\text{lift-dnf-qe } qe \text{ as}) \text{ xs} = I \text{ as } \text{xs}$

$\langle \text{proof} \rangle$

end

end

3 DLO

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

3.1 Basics

```
class dlo = linorder +
assumes dense:  $x < z \implies \exists y. x < y \wedge y < z$ 
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  $\Rightarrow$  bool where
is-Less (Less i j) = True |
is-Less f = False
```

```
abbreviation is-Eq  $\equiv$  Not  $\circ$  is-Less
```

```
lemma is-Less-iff: is-Less a =  $(\exists i j. a = \text{Less } i j)$ 
⟨proof⟩
```

```
lemma is-Eq-iff:  $(\forall i j. a \neq \text{Less } i j) = (\exists i j. a = \text{Eq } i j)$ 
⟨proof⟩
```

```
lemma not-is-Eq-iff:  $(\forall i j. a \neq \text{Eq } i j) = (\exists i j. a = \text{Less } i j)$ 
⟨proof⟩
```

```
fun negdlo :: atom  $\Rightarrow$  atom fm where
negdlo (Less i j) = Or (Atom(Less j i)) (Atom(Eq i j)) |
negdlo (Eq i j) = Or (Atom(Less i j)) (Atom(Less j i))
```

```
fun Idlo :: atom  $\Rightarrow$  'a::dlo list  $\Rightarrow$  bool where
Idlo (Eq i j) xs = (xs!i = xs!j) |
Idlo (Less i j) xs = (xs!i < xs!j)
```

```
fun dependsdlo :: atom  $\Rightarrow$  bool where
dependsdlo (Eq i j) = (i=0 | j=0) |
dependsdlo (Less i j) = (i=0 | j=0)
```

```
fun decrdlo :: atom  $\Rightarrow$  atom where
decrdlo (Less i j) = Less (i - 1) (j - 1) |
decrdlo (Eq i j) = Eq (i - 1) (j - 1)
```

```
definition [code del]: nnf = ATOM.nnf negdlo
```

```
definition [code del]: qelim = ATOM.qelim dependsdlo decrdlo
```

```
definition [code del]: lift-dnf-qe = ATOM.lift-dnf-qe negdlo dependsdlo decrdlo
```

definition [code del]: $\text{lift-nnf-qe} = \text{ATOM.lift-nnf-qe } \text{neg}_{dlo}$

hide-const $\text{nnf qelim lift-dnf-qe lift-nnf-qe}$

lemmas $\text{DLO-code-lemmas} = \text{nnf-def qelim-def lift-dnf-qe-def lift-nnf-qe-def}$

interpretation DLO :

$\text{ATOM } \text{neg}_{dlo} (\lambda a. \text{True}) \text{I}_{dlo} \text{depends}_{dlo} \text{decr}_{dlo}$
 $\langle \text{proof} \rangle$

lemmas [folded DLO-code-lemmas , code] =

$\text{DLO.nnf.simps DLO.qelim-def DLO.lift-dnf-qe.simps DLO.lift-dnf-qe.simps}$

$\langle \text{ML} \rangle$

definition $\text{lbounds where lbounds } as = [i. \text{Less } (\text{Suc } i) 0 \leftarrow as]$

definition $\text{ubounds where ubounds } as = [i. \text{Less } 0 (\text{Suc } i) \leftarrow as]$

definition ebounds where

$\text{ebounds } as = [i. \text{Eq } (\text{Suc } i) 0 \leftarrow as] @ [i. \text{Eq } 0 (\text{Suc } i) \leftarrow as]$

lemma $\text{set-lbounds: set(lbounds } as) = \{i. \text{Less } (\text{Suc } i) 0 \in \text{set } as\}$

$\langle \text{proof} \rangle$

lemma $\text{set-ubounds: set(ubounds } as) = \{i. \text{Less } 0 (\text{Suc } i) \in \text{set } as\}$

$\langle \text{proof} \rangle$

lemma set-ebounds:

$\text{set(ebounds } as) = \{k. \text{Eq } (\text{Suc } k) 0 \in \text{set } as \vee \text{Eq } 0 (\text{Suc } k) \in \text{set } as\}$

$\langle \text{proof} \rangle$

abbreviation $\text{LB } f \text{ } xs \equiv \{xs!i | i. \text{Less } (\text{Suc } i) 0 \in \text{set}(\text{DLO.atoms}_0 f)\}$

abbreviation $\text{UB } f \text{ } xs \equiv \{xs!i | i. \text{Less } 0 (\text{Suc } i) \in \text{set}(\text{DLO.atoms}_0 f)\}$

definition $\text{EQ } f \text{ } xs = \{xs!k | k.$

$\text{Eq } (\text{Suc } k) 0 \in \text{set}(\text{DLO.atoms}_0 f) \vee \text{Eq } 0 (\text{Suc } k) \in \text{set}(\text{DLO.atoms}_0 f)\}$

lemma $\text{EQ-And[simp]: EQ } (\text{And } f g) \text{ } xs = (\text{EQ } f \text{ } xs \cup \text{EQ } g \text{ } xs)$

$\langle \text{proof} \rangle$

lemma $\text{EQ-Or[simp]: EQ } (\text{Or } f g) \text{ } xs = (\text{EQ } f \text{ } xs \cup \text{EQ } g \text{ } xs)$

$\langle \text{proof} \rangle$

lemma $\text{EQ-conv-set-ebounds:}$

$x \in \text{EQ } f \text{ } xs = (\exists e \in \text{set}(\text{ebounds}(\text{DLO.atoms}_0 f)). x = xs!e)$

$\langle \text{proof} \rangle$

fun $\text{isubst where isubst } k 0 = k \mid \text{isubst } k (\text{Suc } i) = i$

fun $\text{asubst} :: \text{nat} \Rightarrow \text{atom} \Rightarrow \text{atom} \text{ where}$

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

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

lemma *I-subst*:

$ngfree\ f \implies DLO.I\ (subst\ f\ k)\ xs = DLO.I\ f\ (xs!k\ \# \ xs)$
 $\langle proof \rangle$

fun *amin-inf* :: $atom \Rightarrow atom\ fm$ **where**

$amin-inf\ (Less\ -\ 0) = FalseF$ |
 $amin-inf\ (Less\ 0\ -) = TrueF$ |
 $amin-inf\ (Less\ (Suc\ i)\ (Suc\ j)) = Atom(Less\ i\ j)$ |
 $amin-inf\ (Eq\ 0\ 0) = TrueF$ |
 $amin-inf\ (Eq\ 0\ -) = FalseF$ |
 $amin-inf\ (Eq\ -\ 0) = FalseF$ |
 $amin-inf\ (Eq\ (Suc\ i)\ (Suc\ j)) = Atom(Eq\ i\ j)$

abbreviation *min-inf* :: $atom\ fm \Rightarrow atom\ fm\ (inf_-)$ **where**

$inf_- \equiv amap_{fm}\ amin-inf$

fun *aplus-inf* :: $atom \Rightarrow atom\ fm$ **where**

$aplus-inf\ (Less\ 0\ -) = FalseF$ |
 $aplus-inf\ (Less\ -\ 0) = TrueF$ |
 $aplus-inf\ (Less\ (Suc\ i)\ (Suc\ j)) = Atom(Less\ i\ j)$ |
 $aplus-inf\ (Eq\ 0\ 0) = TrueF$ |
 $aplus-inf\ (Eq\ 0\ -) = FalseF$ |
 $aplus-inf\ (Eq\ -\ 0) = FalseF$ |
 $aplus-inf\ (Eq\ (Suc\ i)\ (Suc\ j)) = Atom(Eq\ i\ j)$

abbreviation *plus-inf* :: $atom\ fm \Rightarrow atom\ fm\ (inf_+)$ **where**

$inf_+ \equiv amap_{fm}\ aplus-inf$

lemma *min-inf*:

$ngfree\ f \implies \exists x. \forall y \leq x. DLO.I\ (inf_-)\ f\ xs = DLO.I\ f\ (y\ \# \ xs)$
 $(is\ - \implies \exists x. ?P\ f\ x)$
 $\langle proof \rangle$

lemma *plus-inf*:

$ngfree\ f \implies \exists x. \forall y \geq x. DLO.I\ (inf_+)\ f\ xs = DLO.I\ f\ (y\ \# \ xs)$
 $(is\ - \implies \exists x. ?P\ f\ x)$
 $\langle proof \rangle$

context notes $[[simp-depth-limit=2]]$

begin

lemma *LBeX*:

$[[\ ngfree\ f;\ DLO.I\ f\ (x\ \# \ xs);\ \neg DLO.I\ (inf_-)\ f\ xs;\ x \notin EQ\ f\ xs \]]$

$\implies \exists l \in LB\ f\ xs. l < x$
 <proof>

lemma *UBex*:

$\llbracket nqfree\ f; DLO.I\ f\ (x\#\!xs); \neg DLO.I\ (inf_+\ f)\ xs; x \notin EQ\ f\ xs \rrbracket$
 $\implies \exists u \in UB\ f\ xs. x < u$
 <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*: $nqfree\ \varphi \implies qfree(inf_-\ \varphi)$
 <proof>

lemma *qfree-aplus-inf*: $qfree\ (aplug-inf\ a)$
 <proof>

lemma *qfree-plus-inf*: $nqfree\ \varphi \implies qfree(inf_+\ \varphi)$
 <proof>

end

theory *QEdlo*
imports *DLO*
begin

3.2 DNF-based quantifier elimination

definition *qe-dlo₁* :: $atom\ list \Rightarrow atom\ fm$ **where**

$qe-dlo_1\ as =$
 (if $Less\ 0\ 0 \in set\ as$ then $FalseF$ else
 let $lbs = [i. Less\ (Suc\ i)\ 0 \leftarrow as]$; $ubs = [j. Less\ 0\ (Suc\ j) \leftarrow as]$;
 $pairs = [Atom(Less\ i\ j). i \leftarrow lbs, j \leftarrow ub]$
 in $list-conj\ pairs$)

theorem *I-qe-dlo₁*:

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 $?L = ?R$)

<proof>

lemma *I-qe-dlo₁-pretty*:

$\forall a \in \text{set as. is-Less } a \wedge \text{depends}_{dlo} a \implies \text{DLO.is-dnf-qe} - \text{qe-dlo}_1 \text{ as}$
<proof>

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 *subst₀* :: *atom* \Rightarrow *atom* \Rightarrow *atom* **where**

subst₀ (*Eq* *i j*) *a* = (case *a* of
 Less *m n* \Rightarrow *Less* (*subst* *i j m*) (*subst* *i j n*)
 | *Eq* *m n* \Rightarrow *Eq* (*subst* *i j m*) (*subst* *i j n*))

lemma *subst₀-pretty*:

subst₀ (*Eq* *i j*) (*Less* *m n*) = *Less* (*subst* *i j m*) (*subst* *i j n*)

subst₀ (*Eq* *i j*) (*Eq* *m n*) = *Eq* (*subst* *i j m*) (*subst* *i j n*)

<proof>

interpretation *DLO_e*:

ATOM-EQ *neg_{dlo}* ($\lambda a. \text{True}$) *I_{dlo}* *depends_{dlo}* *decr_{dlo}*
 ($\lambda \text{Eq } i j \Rightarrow i=0 \vee j=0 \mid a \Rightarrow \text{False}$)
 ($\lambda \text{Eq } i j \Rightarrow i=j \mid a \Rightarrow \text{False}$) *subst₀*

<proof>

<ML>

definition *qe-dlo* = *DLO_e.lift-dnfeq-qe* *qe-dlo₁*

lemma *qfree-qe-dlo₁*: *qfree* (*qe-dlo₁* *as*)

<proof>

theorem *I-qe-dlo*: *DLO.I* (*qe-dlo* φ) *xs* = *DLO.I* φ *xs*

<proof>

theorem *qfree-qe-dlo*: *qfree* (*qe-dlo* φ)

<proof>

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*)

<proof>

lemma *interpret-others*:

interpret (Neg (ExQ (Neg f))) xs = (∀ x. interpret f (x#xs))
interpret (Or (Neg f1) f2) xs = (interpret f1 xs → 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 $\forall x y::real. \exists z. x < y \rightarrow x < z \wedge z < y$

⟨*proof*⟩

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

⟨*proof*⟩

lemma $\forall x::real. \neg x < x$

⟨*proof*⟩

lemma $\forall x y::real. \exists z. x < y \rightarrow x < z \wedge z < y$

⟨*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

$qe\text{-}dlo(\text{AllQ}(\text{AllQ}(\text{ExQ}(\text{And}(\text{Atom}(\text{Less } 1\ 0))(\text{Atom}(\text{Less } 0\ 2)))))) = \text{FalseF}$
 ⟨*proof*⟩

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

value $qe\text{-}dlo(\text{AllQ}(\text{Imp}(\text{Atom}(\text{Less } 0\ 1))(\text{Atom}(\text{Less } 0\ 2))))$

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 $asubst_2 :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{atom} \Rightarrow \text{atom } fm$ **where**
 $asubst_2\ l\ u\ (\text{Less } 0\ 0) = \text{FalseF} \mid$
 $asubst_2\ l\ u\ (\text{Less } 0\ (\text{Suc } j)) = \text{Or}(\text{Atom}(\text{Less } u\ j))(\text{Atom}(\text{Eq } u\ j)) \mid$
 $asubst_2\ l\ u\ (\text{Less } (\text{Suc } i)\ 0) = \text{Or}(\text{Atom}(\text{Less } i\ l))(\text{Atom}(\text{Eq } i\ l)) \mid$
 $asubst_2\ l\ u\ (\text{Less } (\text{Suc } i)\ (\text{Suc } j)) = \text{Atom}(\text{Less } i\ j) \mid$
 $asubst_2\ l\ u\ (\text{Eq } 0\ 0) = \text{TrueF} \mid$
 $asubst_2\ l\ u\ (\text{Eq } 0\ -) = \text{FalseF} \mid$
 $asubst_2\ l\ u\ (\text{Eq } -\ 0) = \text{FalseF} \mid$
 $asubst_2\ l\ u\ (\text{Eq } (\text{Suc } i)\ (\text{Suc } j)) = \text{Atom}(\text{Eq } i\ j)$

abbreviation $subst_2\ l\ u \equiv \text{amap}_{fm}(asubst_2\ l\ u)$

lemma *I-subst₂1*:

$nqfree\ f \Longrightarrow xs!l < xs!u \Longrightarrow DLO.I(subst_2\ l\ u\ f)\ xs$
 $\Longrightarrow xs!l < x \Longrightarrow x < xs!u \Longrightarrow DLO.I\ f\ (x\#\ xs)$
 ⟨*proof*⟩

definition

$nolub\ f\ xs\ l\ x\ u \longleftrightarrow (\forall y \in \{l <..<x\}. y \notin LB\ f\ xs) \wedge (\forall y \in \{x <..<u\}. y \notin UB\ f\ xs)$

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*:

[[$nqfree f$; $nolub f xs l x u$; $l < x$; $x < u$; $x \notin EQ f xs$;
 $DLO.I f (x\#xs)$; $l < y$; $y < u$]]
 $\implies DLO.I f (y\#xs)$
 ⟨proof⟩

lemma *I-subst₂2*:

$nqfree f \implies xs!l < x \wedge x < xs!u \implies nolub f xs (xs!l) x (xs!u)$
 $\implies \forall x \in \{xs!l <..< xs!u\}. DLO.I f (x\#xs) \wedge x \notin EQ f xs$
 $\implies DLO.I (subst_2 l u f) xs$
 ⟨proof⟩

end

definition

qe-interior₁ $\varphi =$
 (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 $list-disj (inf_- \varphi \# inf_+ \varphi \# intrs @ map (subst \varphi) ebs)$)

lemma *dense-interval*:

assumes *finite L finite U* $l \in L u \in U l < x x < u P(x::'a::dlo)$
and *dense*: $\bigwedge y l u. [\forall y \in \{l <..< x\}. y \notin L; \forall y \in \{x <..< u\}. y \notin U;$
 $l < x; x < u; l < y; y < u] \implies P y$
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* φ **shows** $DLO.I (qe-interior_1 \varphi) xs = (\exists x. DLO.I \varphi (x\#xs))$
 (is ?QE = ?EX)
 ⟨proof⟩

lemma *qfree-asubst₂*: *qfree* (*asubst₂* $l u a$)
 ⟨proof⟩

lemma *qfree-subst₂*: *nqfree* $\varphi \implies qfree (subst_2 l u \varphi)$
 ⟨proof⟩

lemma *qfree-interior1*: $nqfree\ \varphi \implies qfree(qe\text{-interior}_1\ \varphi)$
 ⟨proof⟩

definition *qe-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-peps* :: $nat \Rightarrow atom \Rightarrow atom\ fm\ (asubst_+)$ **where**
asubst-peps *k* (*Less* 0 0) = *FalseF* |
asubst-peps *k* (*Less* 0 (*Suc* *j*)) = *Atom*(*Less* *k* *j*) |
asubst-peps *k* (*Less* (*Suc* *i*) 0) = (if *i=k* then *TrueF*
 else *Or* (*Atom*(*Less* *i* *k*)) (*Atom*(*Eq* *i* *k*))) |
asubst-peps *k* (*Less* (*Suc* *i*) (*Suc* *j*)) = *Atom*(*Less* *i* *j*) |
asubst-peps *k* (*Eq* 0 0) = *TrueF* |
asubst-peps *k* (*Eq* 0 -) = *FalseF* |
asubst-peps *k* (*Eq* - 0) = *FalseF* |
asubst-peps *k* (*Eq* (*Suc* *i*) (*Suc* *j*)) = *Atom*(*Eq* *i* *j*)

abbreviation *subst-peps* :: $atom\ fm \Rightarrow nat \Rightarrow atom\ fm\ (subst_+)$ **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]*:
nolb (*And* $\varphi_1\ \varphi_2$) *xs* *l* *x* = (*nolb* φ_1 *xs* *l* *x* \wedge *nolb* φ_2 *xs* *l* *x*)
 ⟨proof⟩

lemma *nolb-Or[simp]*:
nolb (*Or* $\varphi_1\ \varphi_2$) *xs* *l* *x* = (*nolb* φ_1 *xs* *l* *x* \wedge *nolb* φ_2 *xs* *l* *x*)
 ⟨proof⟩

context notes $[[simp\text{-}depth\text{-}limit=3]]$

begin

lemma *innermost-intvl*:

$[[\text{ngfree } \varphi; \text{ nolb } \varphi \text{ xs } l \ x; l < x; x \notin EQ \ \varphi \ \text{xs}; DLO.I \ \varphi \ (x\#\text{xs}); l < y; y \leq x]]$
 $\implies DLO.I \ \varphi \ (y\#\text{xs})$
<proof>

lemma *I-subst-peps2*:

$\text{ngfree } \varphi \implies \text{xs}!l < x \implies \text{nolb } \varphi \ \text{xs} \ (\text{xs}!l) \ x \implies x \notin EQ \ \varphi \ \text{xs}$
 $\implies \forall y \in \{\text{xs}!l <.. x\}. DLO.I \ \varphi \ (y\#\text{xs})$
 $\implies DLO.I \ (\text{subst}_+ \ \varphi \ l) \ \text{xs}$
<proof>

end

lemma *dense-interval*:

assumes *finite* $L \ l \in L \ l < x \ P(x::'a::dlo)$
and *dense*: $\bigwedge y \ l. [[\forall y \in \{l <.. x\}. y \notin L; l < x; l < y; y \leq x]] \implies P \ y$
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*:

$\text{ngfree } \varphi \implies DLO.I \ (\text{subst}_+ \ \varphi \ l) \ \text{xs} \longrightarrow$
 $(\exists \text{leps} > \text{xs}!l. \forall x. \text{xs}!l < x \wedge x \leq \text{leps} \longrightarrow DLO.I \ \varphi \ (x\#\text{xs}))$
<proof>

definition

$qe\text{-eps}_1(\varphi) =$
(let $as = DLO.atoms_0 \ \varphi; lbs = l\text{bounds } as; ebs = e\text{bounds } as$
in $list\text{-disj} \ (\text{inf}_- \ \varphi \ \# \ \text{map} \ (\text{subst}_+ \ \varphi) \ lbs \ @ \ \text{map} \ (\text{subst} \ \varphi) \ ebs)$

theorem *I-qe-eps1*:

assumes *ngfree* φ **shows** $DLO.I \ (qe\text{-eps}_1 \ \varphi) \ \text{xs} = (\exists x. DLO.I \ \varphi \ (x\#\text{xs}))$
(is $?QE = ?EX$ **)**
<proof>

lemma *qfree-astubst-peps*: $q\text{free} \ (\text{astubst}_+ \ k \ a)$

<proof>

lemma *qfree-subst-peps*: $q\text{free} \ \varphi \implies q\text{free} \ (\text{subst}_+ \ \varphi \ k)$

<proof>

lemma *qfree-qe-eps1*: $q\text{free} \ \varphi \implies q\text{free} \ (qe\text{-eps}_1 \ \varphi)$

<proof>

definition $qe-eps = DLO.lift-nnf-qe\ qe-eps_1$

lemma $qfree-qe-eps: qfree(qe-eps\ \varphi)$
 $\langle proof \rangle$

lemma $I-qe-eps: DLO.I\ (qe-eps\ \varphi)\ xs = DLO.I\ \varphi\ xs$
 $\langle proof \rangle$

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 $is-Less :: atom \Rightarrow bool$ **where**
 $is-Less\ (Less\ r\ rs) = True\ |\$
 $is-Less\ f = False$

abbreviation $is-Eq \equiv Not\ \circ\ is-Less$

lemma $is-Less-iff: is-Less\ f = (\exists\ r\ rs.\ f = Less\ r\ rs)$
 $\langle proof \rangle$

lemma $is-Eq-iff: (\forall\ i\ j.\ a \neq Less\ i\ j) = (\exists\ i\ j.\ a = Eq\ i\ j)$
 $\langle proof \rangle$

fun $neg_R :: atom \Rightarrow atom\ fm$ **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 $hd-coeff :: atom \Rightarrow real$ **where**
 $hd-coeff\ (Less\ r\ cs) = (case\ cs\ of\ [] \Rightarrow 0\ |\\ c\#_ \Rightarrow c)\ |\$
 $hd-coeff\ (Eq\ r\ cs) = (case\ cs\ of\ [] \Rightarrow 0\ |\\ c\#_ \Rightarrow c)$

definition $depends_R\ a = (hd-coeff\ a \neq 0)$

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

$$\text{decr}_R (Eq\ r\ rs) = Eq\ r\ (tl\ rs)$$

fun $I_R :: \text{atom} \Rightarrow \text{real list} \Rightarrow \text{bool}$ **where**

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

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

definition $\text{atoms}_0 = \text{ATOM}.\text{atoms}_0\ \text{depends}_R$

interpretation $R: \text{ATOM}\ \text{neg}_R\ (\lambda a. \text{True})\ I_R\ \text{depends}_R\ \text{decr}_R$

rewrites $\text{ATOM}.\text{atoms}_0\ \text{depends}_R = \text{atoms}_0$

$\langle \text{proof} \rangle$

$\langle \text{ML} \rangle$

4.1.2 Shared constructions

fun $\text{combine} :: (\text{real} * \text{real list}) \Rightarrow (\text{real} * \text{real list}) \Rightarrow \text{atom}$ **where**

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

definition $\text{lbounds}\ as = [(r/c, (-1/c) *_{\text{s}}\ cs). Less\ r\ (c\#\ cs) \leftarrow as, c > 0]$

definition $\text{ubounds}\ as = [(r/c, (-1/c) *_{\text{s}}\ cs). Less\ r\ (c\#\ cs) \leftarrow as, c < 0]$

definition $\text{ebounds}\ as = [(r/c, (-1/c) *_{\text{s}}\ cs). Eq\ r\ (c\#\ cs) \leftarrow as, c \neq 0]$

lemma set-lbounds :

$$\text{set}(\text{lbounds}\ as) = \{(r/c, (-1/c) *_{\text{s}}\ cs) \mid r\ c\ cs. Less\ r\ (c\#\ cs) \in \text{set}\ as \wedge c > 0\}$$

$\langle \text{proof} \rangle$

lemma set-ubounds :

$$\text{set}(\text{ubounds}\ as) = \{(r/c, (-1/c) *_{\text{s}}\ cs) \mid r\ c\ cs. Less\ r\ (c\#\ cs) \in \text{set}\ as \wedge c < 0\}$$

$\langle \text{proof} \rangle$

lemma set-ebounds :

$$\text{set}(\text{ebounds}\ as) = \{(r/c, (-1/c) *_{\text{s}}\ cs) \mid r\ c\ cs. Eq\ r\ (c\#\ cs) \in \text{set}\ as \wedge c \neq 0\}$$

$\langle \text{proof} \rangle$

abbreviation EQ **where**

$$EQ\ f\ xs \equiv \{(r - \langle cs, xs \rangle) / c \mid r\ c\ cs. Eq\ r\ (c\#\ cs) \in \text{set}(R.\text{atoms}_0\ f) \wedge c \neq 0\}$$

abbreviation LB **where**

$$LB\ f\ xs \equiv \{(r - \langle cs, xs \rangle) / c \mid r\ c\ cs. Less\ r\ (c\#\ cs) \in \text{set}(R.\text{atoms}_0\ f) \wedge c > 0\}$$

abbreviation UB **where**

$$UB\ f\ xs \equiv \{(r - \langle cs, xs \rangle) / c \mid r\ c\ cs. Less\ r\ (c\#\ cs) \in \text{set}(R.\text{atoms}_0\ f) \wedge c < 0\}$$

fun $\text{asubst} :: \text{real} * \text{real list} \Rightarrow \text{atom} \Rightarrow \text{atom}$ **where**

$$\text{asubst}\ (r, cs)\ (Less\ s\ (d\#\ ds)) = Less\ (s - d*r)\ (d *_{\text{s}}\ cs + ds) \mid$$

$$\text{asubst}\ (r, cs)\ (Eq\ s\ (d\#\ ds)) = Eq\ (s - d*r)\ (d *_{\text{s}}\ cs + ds) \mid$$

$$\text{asubst}\ (r, cs)\ (Less\ s\ []) = Less\ s\ [] \mid$$

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

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

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*:

$qfree\ \varphi \Longrightarrow R.I\ (subst\ \varphi\ t)\ xs = R.I\ \varphi\ (eval\ t\ xs\ \# xs)$
 $\langle proof \rangle$

lemma *I-subst-pretty*:

$qfree\ \varphi \Longrightarrow R.I\ (subst\ \varphi\ (r,cs))\ xs = R.I\ \varphi\ ((r + \langle cs, xs \rangle)\ \# xs)$
 $\langle proof \rangle$

fun *min-inf* :: $atom\ fm \Rightarrow atom\ fm\ (inf_-)$ **where**

$inf_- (And\ \varphi_1\ \varphi_2) = and\ (inf_-\ \varphi_1)\ (inf_-\ \varphi_2) \mid$
 $inf_- (Or\ \varphi_1\ \varphi_2) = or\ (inf_-\ \varphi_1)\ (inf_-\ \varphi_2) \mid$
 $inf_- (Atom(Less\ r\ (c\#\ cs))) =$
 $(if\ c < 0\ then\ TrueF\ else\ if\ c > 0\ then\ FalseF\ else\ Atom(Less\ r\ cs)) \mid$
 $inf_- (Atom(Eq\ r\ (c\#\ cs))) = (if\ c = 0\ then\ Atom(Eq\ r\ cs)\ else\ FalseF) \mid$
 $inf_-\ \varphi = \varphi$

fun *plus-inf* :: $atom\ fm \Rightarrow atom\ fm\ (inf_+)$ **where**

$inf_+ (And\ \varphi_1\ \varphi_2) = and\ (inf_+\ \varphi_1)\ (inf_+\ \varphi_2) \mid$
 $inf_+ (Or\ \varphi_1\ \varphi_2) = or\ (inf_+\ \varphi_1)\ (inf_+\ \varphi_2) \mid$
 $inf_+ (Atom(Less\ r\ (c\#\ cs))) =$
 $(if\ c > 0\ then\ TrueF\ else\ if\ c < 0\ then\ FalseF\ else\ Atom(Less\ r\ cs)) \mid$
 $inf_+ (Atom(Eq\ r\ (c\#\ cs))) = (if\ c = 0\ then\ Atom(Eq\ r\ cs)\ else\ FalseF) \mid$
 $inf_+\ \varphi = \varphi$

lemma *qfree-min-inf*: $qfree\ \varphi \Longrightarrow qfree(inf_-\ \varphi)$

$\langle proof \rangle$

lemma *qfree-plus-inf*: $qfree\ \varphi \Longrightarrow qfree(inf_+\ \varphi)$

$\langle proof \rangle$

lemma *min-inf*:

$ngfree\ f \Longrightarrow \exists x. \forall y \leq x. R.I\ (inf_-\ f)\ xs = R.I\ f\ (y\ \# xs)$
 $(is\ - \Longrightarrow \exists x. ?P\ f\ x)$
 $\langle proof \rangle$

lemma *plus-inf*:

$ngfree\ f \Longrightarrow \exists x. \forall y \geq x. R.I\ (inf_+\ f)\ xs = R.I\ f\ (y\ \# xs)$
 $(is\ - \Longrightarrow \exists x. ?P\ f\ x)$

<proof>

context notes $[[simp\text{-}depth\text{-}limit = 4]]$
begin

lemma *LBex*:

$[[\text{ngfree } f; R.I\ f\ (x\#\!xs); \neg R.I\ (\text{inf}_-\ f)\ xs; x \notin EQ\ f\ xs]]$
 $\implies \exists l \in LB\ f\ xs. l < x$
<proof>

lemma *UBex*:

$[[\text{ngfree } f; R.I\ f\ (x\#\!xs); \neg R.I\ (\text{inf}_+\ f)\ xs; x \notin EQ\ f\ xs]]$
 $\implies \exists u \in UB\ f\ xs. x < u$
<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-FM₁* :: $atom\ list \implies atom\ fm$ **where**
 $qe\text{-}FM_1\ as = list\text{-}conj\ [Atom(\text{combine } p\ q). p\leftarrow l\text{bounds } as, q\leftarrow u\text{bounds } as]$

theorem *I-qe-FM₁*:

assumes *less*: $\forall a \in set\ as. is\text{-}Less\ a$ **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-qe-FM₁-pretty*:

$\forall a \in set\ as. is\text{-}Less\ a \wedge depends_R\ a \implies R.is\text{-}dnf\text{-}qe\ qe\text{-}FM_1\ as$
<proof>

fun *subst₀* :: $atom \implies atom \implies atom$ **where**

$subst_0\ (Eq\ r\ (c\#\!cs))\ a = (\text{case } a\ \text{of}$
 $Less\ s\ (d\#\!ds) \implies Less\ (s - (r*d)/c)\ (ds - (d/c) * _s\ cs)$

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

lemma *subst₀-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-subst₀*: $depends_R\ a \Longrightarrow c \neq 0 \Longrightarrow$

$I_R\ (subst_0\ (Eq\ r\ (c\#cs))\ a)\ xs = I_R\ a\ ((r - \langle cs, xs \rangle)/c \# xs)$

<proof>

interpretation *R_e*:

ATOM-EQ $neg_R\ (\lambda a. True)\ I_R\ depends_R\ decr_R$

$(\lambda Eq\ -\ (c\#-) \Rightarrow c \neq 0 \mid - \Rightarrow False)$

$(\lambda Eq\ r\ cs \Rightarrow r=0 \wedge (\forall c \in set\ cs. c=0) \mid - \Rightarrow False)\ subst_0$

<proof>

definition *qe-FM* = $R_e.lift\ dnfeq\ qe\ qe-FM_1$

lemma *qfree-qe-FM₁*: $qfree\ (qe-FM_1\ as)$

<proof>

corollary *I-qe-FM*: $R.I\ (qe-FM\ \varphi)\ xs = R.I\ \varphi\ xs$

<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 ⇒ (if ∀ c ∈ set cs. c = 0
                then if r < 0 then TrueF else FalseF
                else Atom a) |
  Eq r cs ⇒ (if ∀ c ∈ set cs. c = 0
              then if r = 0 then TrueF else FalseF else Atom a))

```

lemma *asimp-pretty*:

```

asimp (Less r cs) =
  (if ∀ c ∈ set cs. c = 0
   then if r < 0 then TrueF else FalseF
   else Atom(Less r cs))
asimp (Eq r cs) =
  (if ∀ c ∈ set cs. c = 0
   then if r = 0 then TrueF else FalseF
   else Atom(Eq r cs))
⟨proof⟩

```

definition *qe-FMo₁* :: atom list ⇒ atom fm **where**

```

qe-FMo1 as = list-conj [asimp(combine p q). p ← lbounds as, q ← ubounds as]

```

lemma *I-asimp*: $R.I$ (*asimp* a) *xs* = I_R a *xs*

⟨proof⟩

lemma *I-qe-FMo₁*: $R.I$ (*qe-FMo₁* as) *xs* = $R.I$ (*qe-FM₁* as) *xs*

⟨proof⟩

definition *qe-FMo* = $R_e.lift-dnfeq-qe$ *qe-FMo₁*

lemma *qfree-qe-FMo₁*: *qfree* (*qe-FMo₁* as)

⟨proof⟩

corollary *I-qe-FMo*: $R.I$ (*qe-FMo* φ) *xs* = $R.I$ φ *xs*

⟨proof⟩

theorem *qfree-qe-FMo*: *qfree* (*qe-FMo* f)

⟨proof⟩

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

```

definition $FR_1 :: atom\ fm \Rightarrow atom\ fm$ **where**

```

 $FR_1\ \varphi =$ 
  (let as = R.atoms0  $\varphi$ ; lbs = lbounds as; ubs = ubounds as; ebs = ebounds as;
      intrs = [subst  $\varphi$  (between l u) . l  $\leftarrow$  lbs, u  $\leftarrow$  ubs]
      in list-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::real)
and dense:  $\bigwedge y\ l\ u. \llbracket \forall y \in \{l <..<x\}. y \notin L; \forall y \in \{x <..<u\}. y \notin U;$ 
              $l < x; x < u; l < y; y < u \rrbracket \Longrightarrow P\ y$ 
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:

```

 $\llbracket nqfree\ f; \forall y \in \{l <..<x\}. y \notin LB\ f\ xs; \forall y \in \{x <..<u\}. y \notin UB\ f\ xs;$ 
   $l < x; x < u; x \notin EQ\ f\ xs; R.I\ f\ (x\#\!xs); l < y; y < u \rrbracket$ 
 $\Longrightarrow R.I\ f\ (y\#\!xs)$ 
<proof>

```

theorem I-FR₁:

```

assumes nqfree  $\varphi$  shows R.I (FR1  $\varphi$ ) xs =  $(\exists x. R.I\ \varphi\ (x\#\!xs))$ 
  (is ?FR = ?EX)
<proof>

```

definition FR = R.lift-nnf-qe FR₁

lemma qfree-FR₁: nqfree $\varphi \Longrightarrow$ qfree (FR₁ φ)
<proof>

theorem I-FR: R.I (FR φ) xs = R.I φ xs
<proof>

theorem qfree-FR: qfree (FR φ)
<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* \Rightarrow *atom* \Rightarrow *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* \Rightarrow *real * real list* \Rightarrow *atom fm* (*subst+*)
where *subst+* φ *rcs* \equiv *amap_{fm}* (*asubst+* *rcs*) φ

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

lemma *nolb-And[simp]*:
nolb (And f g) xs l x = $(nolb f xs l x \wedge nolb g xs l x)$
 $\langle proof \rangle$

lemma *nolb-Or[simp]*:
nolb (Or f g) xs l x = $(nolb f xs l x \wedge nolb g xs l x)$
 $\langle proof \rangle$

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

lemma *innermost-intvl*:
 $[[\text{nfree } f; \text{nolb } f \text{ xs } l \text{ x}; l < x; x \notin EQ f \text{ xs}; R.If (x\#xs); l < y; y \leq x]]$
 $\implies R.If (y\#xs)$
 $\langle proof \rangle$

definition *EQ2* = *EQ*

lemma *EQ2-Or[simp]*: *EQ2 (Or f g) xs* = $(EQ2 f xs \cup EQ2 g xs)$
 $\langle proof \rangle$

lemma *EQ2-And[simp]*: *EQ2 (And f g) xs* = $(EQ2 f xs \cup EQ2 g xs)$
 $\langle proof \rangle$

lemma *innermost-intvl2*:

$\llbracket \text{ngfree } f; \text{nolb } f \text{ xs } l \text{ x}; l < x; x \notin \text{EQ2 } f \text{ xs}; R.I \text{ f } (x\#xs); l < y; y \leq x \rrbracket$
 $\implies R.I \text{ f } (y\#xs)$
 <proof>

lemma *I-subst-peps2*:

$\text{ngfree } f \implies r+\langle cs, xs \rangle < x \implies \text{nolb } f \text{ xs } (r+\langle cs, xs \rangle) \text{ x}$
 $\implies \forall y \in \{r+\langle cs, xs \rangle <.. x\}. R.I \text{ f } (y\#xs) \wedge y \notin \text{EQ2 } f \text{ xs}$
 $\implies R.I (\text{subst}_+ f (r, cs)) \text{ xs}$
 <proof>

end

lemma *I-subst-peps*:

$\text{ngfree } f \implies R.I (\text{subst}_+ f (r, cs)) \text{ xs} \implies$
 $(\exists \text{leps} > r+\langle cs, xs \rangle. \forall x. r+\langle cs, xs \rangle < x \wedge x \leq \text{leps} \longrightarrow R.I \text{ f } (x\#xs))$
 <proof>

lemma *dense-interval*:

assumes *finite* $L \ l \in L \ l < x \ P(x::\text{real})$
and *dense*: $\bigwedge y \ l. \llbracket \forall y \in \{l <.. x\}. y \notin L; l < x; l < y; y \leq x \rrbracket \implies P \ y$
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>

definition

$\text{qe-eps}_1(f) =$
 (let $as = R.\text{atoms}_0 \ f$; $lbs = \text{lbounds } as$; $ebs = \text{ebounds } as$
 in $\text{list-disj } (\text{inf- } f \ \# \ \text{map } (\text{subst}_+ f) \ lbs \ @ \ \text{map } (\text{subst } f) \ ebs)$)

theorem *I-eps1*:

assumes *ngfree* f **shows** $R.I (\text{qe-eps}_1 f) \text{ xs} = (\exists x. R.I \text{ f } (x\#xs))$
 (is ?QE = ?EX)
 <proof>

lemma *qfree-astubst-peps*: $\text{qfree } (astubst_+ \text{ rcs } a)$

<proof>

lemma *qfree-subst-peps*: $\text{ngfree } \varphi \implies \text{qfree } (\text{subst}_+ \varphi \text{ rcs})$

<proof>

lemma *qfree-qe-eps1*: $\text{ngfree } \varphi \implies \text{qfree}(\text{qe-eps}_1 \varphi)$

<proof>

definition $\text{qe-eps} = R.\text{lift-nnf-qe } \text{qe-eps}_1$

lemma *qfree-qe-eps*: $\text{qfree}(\text{qe-eps } \varphi)$

<proof>

lemma *I-qe-eps*: $R.I (\text{qe-eps } \varphi) \text{ xs} = R.I \varphi \text{ xs}$

<proof>

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* \Rightarrow *int* **where**

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

fun *neg_Z* :: *atom* \Rightarrow *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* \Rightarrow *int* **where**

hd-coeff (*Le i ks*) = (case *ks* of [] \Rightarrow 0 | *k#-* \Rightarrow *k*) |
hd-coeff (*Dvd d i ks*) = (case *ks* of [] \Rightarrow 0 | *k#-* \Rightarrow *k*) |
hd-coeff (*NDvd d i ks*) = (case *ks* of [] \Rightarrow 0 | *k#-* \Rightarrow *k*)

fun *decr_Z* :: *atom* \Rightarrow *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* \Rightarrow *int list* \Rightarrow *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* = (\neg *d dvd i* + \langle *ks*,*xs* \rangle)

definition *atoms₀* = *ATOM.atoms₀* ($\lambda a. \text{hd-coeff } a \neq 0$)

interpretation *Z*:

ATOM neg_Z ($\lambda a. \text{divisor } a \neq 0$) *I_Z* ($\lambda a. \text{hd-coeff } a \neq 0$) *decr_Z*
rewrites *ATOM.atoms₀* ($\lambda a. \text{hd-coeff } a \neq 0$) = *atoms₀*
<proof>

$\langle ML \rangle$

abbreviation

$hd\text{-coeff}\text{-is1 } a \equiv$

$(\text{case } a \text{ of } Le \text{ - } - \Rightarrow hd\text{-coeff } a \in \{1, -1\} \mid - \Rightarrow hd\text{-coeff } a = 1)$

fun $asubst :: int \Rightarrow int \text{ list} \Rightarrow atom \Rightarrow atom$ **where**

$asubst \ i' \ ks' \ (Le \ i \ (k\#ks)) = Le \ (i - k*i') \ (k *_s \ ks' + ks) \mid$
 $asubst \ i' \ ks' \ (Dvd \ d \ i \ (k\#ks)) = Dvd \ d \ (i + k*i') \ (k *_s \ ks' + ks) \mid$
 $asubst \ i' \ ks' \ (NDvd \ d \ i \ (k\#ks)) = NDvd \ d \ (i + k*i') \ (k *_s \ ks' + ks) \mid$
 $asubst \ i' \ ks' \ a = a$

abbreviation $subst :: int \Rightarrow int \text{ list} \Rightarrow atom \text{ fm} \Rightarrow atom \text{ fm}$

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

lemma $IZ\text{-asubst}$: $I_Z \ (asubst \ i \ ks \ a) \ xs = I_Z \ a \ ((i + \langle ks, xs \rangle) \# xs)$

$\langle proof \rangle$

lemma $I\text{-subst}$:

$gfree \ \varphi \Longrightarrow Z.I \ \varphi \ ((i + \langle ks, xs \rangle) \# xs) = Z.I \ (subst \ i \ ks \ \varphi) \ xs$

$\langle proof \rangle$

lemma $divisor\text{-asubst}[simp]$: $divisor \ (asubst \ i \ ks \ a) = divisor \ a$

$\langle proof \rangle$

definition $lbounds \ as = [(i, ks). Le \ i \ (k\#ks) \leftarrow as, k > 0]$

definition $ubounds \ as = [(i, ks). Le \ i \ (k\#ks) \leftarrow as, k < 0]$

lemma $set\text{-lbounds}$:

$set(lbounds \ as) = \{(i, ks) \mid i \ k \ ks. Le \ i \ (k\#ks) \in set \ as \wedge k > 0\}$

$\langle proof \rangle$

lemma $set\text{-ubounds}$:

$set(ubounds \ as) = \{(i, ks) \mid i \ k \ ks. Le \ i \ (k\#ks) \in set \ as \wedge k < 0\}$

$\langle proof \rangle$

lemma $lbounds\text{-append}[simp]$: $lbounds(as \ @ \ bs) = lbounds \ as \ @ \ lbounds \ bs$

$\langle proof \rangle$

5.2 LCM and lemmas

fun $zlcms :: int \text{ list} \Rightarrow int$ **where**

$zlcms \ [] = 1 \mid$

$zlcms \ (i\#is) = lcm \ i \ (zlcms \ is)$

lemma $dvd\text{-zlcms}$: $i \in set \ is \Longrightarrow i \ dvd \ zlcms \ is$

<proof>

lemma *zlcms-pos*: $\forall i \in \text{set } is. i \neq 0 \implies \text{zlcms } is > 0$
<proof>

lemma *zlcms0-iff[simp]*: $(\text{zlcms } is = 0) = (0 \in \text{set } is)$
<proof>

lemma *elem-le-zlcms*: $\forall i \in \text{set } is. i \neq 0 \implies i \in \text{set } is \implies i \leq \text{zlcms } is$
<proof>

5.3 Setting coefficients to 1 or -1

fun *hd-coeff1* :: *int* \Rightarrow *atom* \Rightarrow *atom* **where**

hd-coeff1 *m* (*Le* *i* (*k*#*ks*)) =
 (*if* *k*=0 *then* *Le* *i* (*k*#*ks*)
 else *let* *m'* = *m* *div* (*abs* *k*) *in* *Le* (*m'***i*) (*sgn* *k* # (*m'***s* *ks*))) |
hd-coeff1 *m* (*Dvd* *d* *i* (*k*#*ks*)) =
 (*if* *k*=0 *then* *Dvd* *d* *i* (*k*#*ks*)
 else *let* *m'* = *m* *div* *k* *in* *Dvd* (*m'***d*) (*m'***i*) (*1* # (*m'***s* *ks*))) |
hd-coeff1 *m* (*NDvd* *d* *i* (*k*#*ks*)) =
 (*if* *k*=0 *then* *NDvd* *d* *i* (*k*#*ks*)
 else *let* *m'* = *m* *div* *k* *in* *NDvd* (*m'***d*) (*m'***i*) (*1* # (*m'***s* *ks*))) |
hd-coeff1 - *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 $\llbracket k \neq 0; k \text{ dvd } m \rrbracket \implies$
 $I_Z (\text{hd-coeff1 } m (\text{Dvd } d \ i \ (k\#ks))) (x\#e) = (\text{let } m' = m \text{ div } (\text{abs } k) \text{ in}$
 $I_Z (\text{Dvd } (m'*d) (m'*i) (\text{sgn } k \# (m'*_s \ ks))) (x\#e))$
<proof>

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

lemma *I-hd-coeff1-mult*: **assumes** $m > 0$
shows $\text{qfree } \varphi \implies \forall a \in \text{set}(Z.\text{atoms}_0 \ \varphi). \text{hd-coeff } a \text{ dvd } m \implies$
 $Z.I (\text{map}_{f \ m} (\text{hd-coeff1 } m) \ \varphi) (m*x\#xs) = Z.I \ \varphi (x\#xs)$
<proof>

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:  $\forall a \in \text{set } as. \text{hd-coeff } a \neq 0$  shows
  ( $\exists x. \forall a \in \text{set}(hd-coeffs1 \text{ as}). I_Z a (x \# xs)$ ) =
  ( $\exists x. \forall a \in \text{set } as. I_Z a (x \# xs)$ ) (is ?B = ?A)
<proof>

```

abbreviation *is-dvd* $a \equiv \text{case } a \text{ of } Le \ - \Rightarrow \text{False} \mid \ - \Rightarrow \text{True}$

definition

```

qe-pres1 as =
  (let ds = filter is-dvd as; (d::int) = zlcms(map divisor ds); ls = lbounds as
   in if ls = []
     then Disj [0..d - 1] ( $\lambda n. \text{list-conj}(\text{map } (Atom \circ \text{asubst } n \ [])) \text{ ds}$ )
     else
       Disj ls ( $\lambda(li, lks). \text{Disj } [0..d - 1] (\lambda n. \text{list-conj}(\text{map } (Atom \circ \text{asubst } (li + n) (-lks)) \text{ as})))$ )

```

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  $i \bmod \text{divisor } a = j \bmod \text{divisor } a$ 
shows  $I_Z a (i \# e) = I_Z a (j \# e)$ 
<proof>

```

lemma I-qe-pres1:

```

assumes norm:  $\forall a \in \text{set } as. \text{divisor } a \neq 0$ 
and hd:  $\forall a \in \text{set } as. \text{hd-coeff-is1 } a$ 
shows  $Z.I (qe-pres1 \text{ as}) \text{ xs} = (\exists x. \forall a \in \text{set } as. I_Z a (x \# xs))$ 
<proof>

```

lemma divisors-hd-coeffs1:

```

assumes div0:  $\forall a \in \text{set } as. \text{divisor } a \neq 0$  and hd0:  $\forall a \in \text{set } as. \text{hd-coeff } a \neq 0$ 
and a:  $a \in \text{set } (hd-coeffs1 \text{ as})$  shows  $\text{divisor } a \neq 0$ 
<proof>

```

lemma hd-coeff-is1-hd-coeffs1:

```

assumes hd0:  $\forall a \in \text{set } as. \text{hd-coeff } a \neq 0$ 

```

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

lemma $I\text{-qe-pres}_1\text{-o}$:

$\llbracket \forall a \in \text{set } as. \text{divisor } a \neq 0; \forall a \in \text{set } as. \text{hd-coeff } a \neq 0 \rrbracket \implies$
 $Z.I ((\text{qe-pres}_1 \circ \text{hd-coeffs1}) as) e = (\exists x. \forall a \in \text{set } as. I_Z a (x \# e))$
 $\langle \text{proof} \rangle$

definition $\text{qe-pres} = Z.\text{lift-dnf-qe } (\text{qe-pres}_1 \circ \text{hd-coeffs1})$

lemma qfree-qe-pres-o : $\text{qfree } ((\text{qe-pres}_1 \circ \text{hd-coeffs1}) as)$
 $\langle \text{proof} \rangle$

lemma $\text{normal-qe-pres}_1\text{-o}$:

$\forall a \in \text{set } as. \text{hd-coeff } a \neq 0 \wedge \text{divisor } a \neq 0 \implies$
 $Z.\text{normal } ((\text{qe-pres}_1 \circ \text{hd-coeffs1}) as)$
 $\langle \text{proof} \rangle$

theorem $I\text{-pres-qe}$: $Z.\text{normal } \varphi \implies Z.I (\text{qe-pres } \varphi) xs = Z.I \varphi xs$
 $\langle \text{proof} \rangle$

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

end

theory Cooper
imports PresArith
begin

5.5 Cooper

This section formalizes Cooper's algorithm [1].

lemma set-atoms0-iff :

$\text{qfree } \varphi \implies a \in \text{set}(Z.\text{atoms}_0 \varphi) \longleftrightarrow a \in \text{atoms } \varphi \wedge \text{hd-coeff } a \neq 0$
 $\langle \text{proof} \rangle$

definition

$\text{hd-coeffs1 } \varphi =$
 $(\text{let } m = \text{zlcms}(\text{map } \text{hd-coeff } (Z.\text{atoms}_0 \varphi))$
 $\text{in } \text{And } (\text{Atom}(\text{Dvd } m \ 0 \ [1])) (\text{map}_{f\ m} (\text{hd-coeff1 } m) \varphi))$

lemma $I\text{-hd-coeffs1}$:

assumes $\text{qfree } \varphi$

shows $(\exists x. Z.I (\text{hd-coeffs1 } \varphi) (x \# xs)) = (\exists x. Z.I \varphi (x \# xs))$ (**is** $?L = ?R$)
 $\langle \text{proof} \rangle$

fun *min-inf* :: *atom fm* \Rightarrow *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] (λn . *subst* *n* [] (*inf*_ φ)))
 (*Disj* *ls* ($\lambda(i,ks)$.
Disj [0..*d* - 1] (λn . *subst* (*i* + *n*) (-*ks*) φ))))

lemma *min-inf*:

ngfree *f* $\Longrightarrow \forall a \in \text{set}(Z.\text{atoms}_0 f). \text{hd-coeff-is1 } a$
 $\Longrightarrow \exists x. \forall y < x. Z.I (\text{inf}_- f) (y \# xs) = Z.I f (y \# xs)$
 (**is** - \Longrightarrow - $\Longrightarrow \exists x. ?P f x$)
 <proof>

lemma *min-inf-repeats*:

ngfree $\varphi \Longrightarrow \forall a \in \text{set}(Z.\text{atoms}_0 \varphi). \text{divisor } a \text{ dvd } d \Longrightarrow$
 $Z.I (\text{inf}_- \varphi) ((x - k*d) \# xs) = Z.I (\text{inf}_- \varphi) (x \# xs)$
 <proof>

lemma *atoms-subset*: *qfree* *f* $\Longrightarrow \text{set}(Z.\text{atoms}_0(f::\text{atom fm})) \leq \text{atoms } f$

<proof>

lemma β :

$\llbracket \text{ngfree } \varphi; \forall a \in \text{set}(Z.\text{atoms}_0 \varphi). \text{hd-coeff-is1 } a;$
 $\forall a \in \text{set}(Z.\text{atoms}_0 \varphi). \text{divisor } a \text{ dvd } d; d > 0;$
 $\neg(\exists j \in \{0 \dots d - 1\}. \exists (i, ks) \in \text{set}(\text{lbounds}(Z.\text{atoms}_0 \varphi)).$
 $x = i - \langle ks, xs \rangle + j); Z.I \varphi (x \# xs) \rrbracket$
 $\Longrightarrow Z.I \varphi ((x-d) \# xs)$
 <proof>

lemma *periodic-finite-ex*:

assumes *dpos*: (0::int) < *d* **and** *modd*: $\forall x k. P x = P(x - k*d)$
shows ($\exists x. P x$) = ($\exists j \in \{0 \dots d - 1\}. P j$)
 (**is** ?LHS = ?RHS)
 <proof>

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

theorem *cp-thm*:

assumes *nq*: *nqfree* φ
and *u*: $\forall a \in \text{set}(Z.\text{atoms}_0\ \varphi). \text{hd-coeff-is1}\ a$
and *d*: $\forall a \in \text{set}(Z.\text{atoms}_0\ \varphi). \text{divisor}\ a\ \text{dvd}\ d$
and *dp*: $d > 0$
shows $(\exists x. Z.I\ \varphi\ (x \# xs)) =$
 $(\exists j \in \{0..d-1\}. Z.I\ (\text{inf}_-\ \varphi)\ (j \# xs)) \vee$
 $(\exists (i, ks) \in \text{set}(\text{lbounds}(Z.\text{atoms}_0\ \varphi)). Z.I\ \varphi\ ((i - \langle ks, xs \rangle + j) \# xs))$
(is $(\exists x. ?P\ (x)) = (\exists j \in ?D. ?M\ j \vee (\exists (i, ks) \in ?B. ?P\ (?I\ i\ ks + j)))$
 $\langle proof \rangle$

lemma *qfree-min-inf[simp]*: *qfree* $\varphi \implies \text{qfree}\ (\text{inf}_-\ \varphi)$
 $\langle proof \rangle$

lemma *I-qe-cooper₁*:

assumes *norm*: $\forall a \in \text{atoms}\ \varphi. \text{divisor}\ a \neq 0$
and *hd*: $\forall a \in \text{set}(Z.\text{atoms}_0\ \varphi). \text{hd-coeff-is1}\ a$ **and** *nqfree* φ
shows $Z.I\ (\text{qe-cooper}_1\ \varphi)\ xs = (\exists x. Z.I\ \varphi\ (x \# xs))$
 $\langle proof \rangle$

lemma *divisor-hd-coeff1-neq0*:

qfree $\varphi \implies a \in \text{atoms}\ \varphi \implies \text{divisor}\ a \neq 0 \implies$
 $\text{divisor}\ (\text{hd-coeff1}\ (\text{zlcms}\ (\text{map}\ \text{hd-coeff}\ (Z.\text{atoms}_0\ \varphi)))\ a) \neq 0$
 $\langle proof \rangle$

lemma *hd-coeff-is1-hd-coeff1*:

$\text{hd-coeff}\ (\text{hd-coeff1}\ m\ a) \neq 0 \longrightarrow \text{hd-coeff-is1}\ (\text{hd-coeff1}\ m\ a)$
 $\langle proof \rangle$

lemma *I-cooper1-hd-coeffs1*: *Z.normal* $\varphi \implies \text{nqfree}\ \varphi$

$\implies Z.I\ (\text{qe-cooper}_1(\text{hd-coeffs1}\ \varphi))\ xs = (\exists x. Z.I\ \varphi\ (x \# xs))$
 $\langle proof \rangle$

definition *qe-cooper* = *Z.lift-nnf-qe* (*qe-cooper₁* \circ *hd-coeffs1*)

lemma *qfree-cooper1-hd-coeffs1*: *qfree* $\varphi \implies \text{qfree}\ (\text{qe-cooper}_1\ (\text{hd-coeffs1}\ \varphi))$

$\langle proof \rangle$

lemma *normal-min-inf*: $Z.normal\ \varphi \implies Z.normal(inf_ \varphi)$
<proof>

lemma *normal-cooper1*: $Z.normal\ \varphi \implies Z.normal(qe-cooper_1\ \varphi)$
<proof>

lemma *normal-hd-coeffs1*: $qfree\ \varphi \implies Z.normal\ \varphi \implies Z.normal(hd-coeffs1\ \varphi)$
<proof>

theorem *I-cooper*: $Z.normal\ \varphi \implies Z.I\ (qe-cooper\ \varphi)\ xs = Z.I\ \varphi\ xs$
<proof>

theorem *qfree-cooper*: $qfree\ (qe-cooper\ \varphi)$
<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.