

Quantifier Elimination for Linear Arithmetic

Tobias Nipkow

December 12, 2009

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 Weisspfenning 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	8
2.1.1	DNF-based	8
2.1.2	NNF-based	9
2.2	With equality	11
3	DLO	12
3.1	Basics	12
3.2	DNF-based quantifier elimination	16
3.3	Examples	17
3.4	Interior Point Method	18
3.5	Quantifier elimination with infinitesimals	20

4	Lists as vectors	22
4.1	+ and -	22
4.2	Inner product	24
5	Linear real arithmetic	25
5.1	Basics	25
5.1.1	Syntax and Semantics	25
5.1.2	Shared constructions	26
5.2	Fourier	28
5.2.1	Tests	29
5.2.2	An optimization	29
5.3	Ferrante-Rackoff	31
5.4	Quantifier elimination with infinitesimals	32
6	Presburger arithmetic	34
6.1	Syntax	34
6.2	LCM and lemmas	35
6.3	Setting coefficients to 1 or -1	36
6.4	DNF-based quantifier elimination	37
6.5	Cooper	38

1 Logic

```
theory Logic
imports Main 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 'a fm =
  TrueF | FalseF | Atom 'a | And 'a fm 'a fm | Or 'a fm 'a fm |
  Neg 'a fm | ExQ 'a 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 φ_2 else if $\varphi_2=$ FalseF then φ_1 else
if $\varphi_1=$ TrueF $\vee\ \varphi_2=$ 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 $Disj\ is\ f \equiv list-disj\ (map\ f\ is)$

fun $atoms :: 'a\ fm \Rightarrow 'a\ set$ **where**
 $atoms\ TrueF = \{\}$ |
 $atoms\ FalseF = \{\}$ |
 $atoms\ (Atom\ a) = \{a\}$ |
 $atoms\ (And\ \varphi_1\ \varphi_2) = atoms\ \varphi_1 \cup atoms\ \varphi_2$ |
 $atoms\ (Or\ \varphi_1\ \varphi_2) = atoms\ \varphi_1 \cup atoms\ \varphi_2$ |
 $atoms\ (Neg\ \varphi) = atoms\ \varphi$ |
 $atoms\ (ExQ\ \varphi) = atoms\ \varphi$

fun $map-fm :: ('a \Rightarrow 'b) \Rightarrow 'a\ fm \Rightarrow 'b\ fm$ (map_{fm}) **where**
 $map_{fm}\ h\ TrueF = TrueF$ |
 $map_{fm}\ h\ FalseF = FalseF$ |
 $map_{fm}\ h\ (Atom\ a) = Atom(h\ a)$ |
 $map_{fm}\ h\ (And\ \varphi_1\ \varphi_2) = And\ (map_{fm}\ h\ \varphi_1)\ (map_{fm}\ h\ \varphi_2)$ |
 $map_{fm}\ h\ (Or\ \varphi_1\ \varphi_2) = Or\ (map_{fm}\ h\ \varphi_1)\ (map_{fm}\ h\ \varphi_2)$ |
 $map_{fm}\ h\ (Neg\ \varphi) = Neg\ (map_{fm}\ h\ \varphi)$ |
 $map_{fm}\ h\ (ExQ\ \varphi) = ExQ\ (map_{fm}\ h\ \varphi)$

lemma $atoms-map-fm[simp]$: $atoms(map_{fm}\ f\ \varphi) = f\ `atoms\ \varphi$
 <proof>

fun $amap-fm :: ('a \Rightarrow 'b\ fm) \Rightarrow 'a\ fm \Rightarrow 'b\ fm$ ($amap_{fm}$) **where**
 $amap_{fm}\ h\ TrueF = TrueF$ |
 $amap_{fm}\ h\ FalseF = FalseF$ |
 $amap_{fm}\ h\ (Atom\ a) = h\ a$ |
 $amap_{fm}\ h\ (And\ \varphi_1\ \varphi_2) = and\ (amap_{fm}\ h\ \varphi_1)\ (amap_{fm}\ h\ \varphi_2)$ |
 $amap_{fm}\ h\ (Or\ \varphi_1\ \varphi_2) = or\ (amap_{fm}\ h\ \varphi_1)\ (amap_{fm}\ h\ \varphi_2)$ |
 $amap_{fm}\ h\ (Neg\ \varphi) = neg\ (amap_{fm}\ h\ \varphi)$

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**
 $qfree(ExQ\ f) = False$ |

$qfree(And \ \varphi_1 \ \varphi_2) = (qfree \ \varphi_1 \ \wedge \ qfree \ \varphi_2) \mid$
 $qfree(Or \ \varphi_1 \ \varphi_2) = (qfree \ \varphi_1 \ \wedge \ qfree \ \varphi_2) \mid$
 $qfree(Neg \ \varphi) = (qfree \ \varphi) \mid$
 $qfree \ \varphi = True$

lemma $qfree\text{-and}[simp]$: $\llbracket qfree \ \varphi_1; qfree \ \varphi_2 \rrbracket \implies qfree(And \ \varphi_1 \ \varphi_2)$
 $\langle proof \rangle$

lemma $qfree\text{-or}[simp]$: $\llbracket qfree \ \varphi_1; qfree \ \varphi_2 \rrbracket \implies qfree(Or \ \varphi_1 \ \varphi_2)$
 $\langle proof \rangle$

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

lemma $qfree\text{-foldr-Or}[simp]$:
 $qfree(foldr \ Or \ fs \ \varphi) = (qfree \ \varphi \ \wedge \ (\forall \varphi \in set \ fs. qfree \ \varphi))$
 $\langle proof \rangle$

lemma $qfree\text{-list-conj}[simp]$:
assumes $\forall \varphi \in set \ fs. qfree \ \varphi$ **shows** $qfree(list\text{-conj} \ fs)$
 $\langle proof \rangle$

lemma $qfree\text{-list-disj}[simp]$:
assumes $\forall \varphi \in set \ fs. qfree \ \varphi$ **shows** $qfree(list\text{-disj} \ fs)$
 $\langle proof \rangle$

lemma $qfree\text{-map-fm}$: $qfree \ (map_{fm} \ f \ \varphi) = qfree \ \varphi$
 $\langle proof \rangle$

lemma $atoms\text{-list-disj}E$:
 $a : atoms(list\text{-disj} \ fs) \implies a : (\bigcup \varphi \in set \ fs. atoms \ \varphi)$
 $\langle proof \rangle$

lemma $atoms\text{-list-conj}E$:
 $a : atoms(list\text{-conj} \ fs) \implies a : (\bigcup \varphi \in set \ fs. atoms \ \varphi)$
 $\langle proof \rangle$

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

fun $ngfree :: 'a \ fm \Rightarrow bool$ **where**
 $ngfree \ (Atom \ a) = True \mid$
 $ngfree \ TrueF = True \mid$
 $ngfree \ FalseF = True \mid$

```

ngfree (And  $\varphi_1 \varphi_2$ ) = (ngfree  $\varphi_1 \wedge$  ngfree  $\varphi_2$ ) |
ngfree (Or  $\varphi_1 \varphi_2$ ) = (ngfree  $\varphi_1 \wedge$  ngfree  $\varphi_2$ ) |
ngfree  $\varphi$  = False

```

lemma *ngfree-qfree[simp]*: $ngfree \varphi \implies qfree \varphi$
<proof>

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

```

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

```

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  $\implies \forall b \in atoms(aneg a). anormal b$ 

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:  $\neg depends_0 a \implies I_a a (x\#xs) = I_a (decr a) xs$ 
assumes anormal-decr:  $\neg depends_0 a \implies anormal a \implies anormal(decr a)$ 

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 $I \equiv \text{interpret } I_a$

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

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

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

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

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

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

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

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

lemma I-nnf : $I (\text{nnf } \varphi) \text{ xs} = I \varphi \text{ xs}$
 $\langle \text{proof} \rangle$

lemma I-dnf :
 $\text{ngfree } \varphi \Longrightarrow (\exists \text{as} \in \text{set } (\text{dnf } \varphi). \forall a \in \text{set } \text{as}. I_a a \text{ xs}) = I \varphi \text{ xs}$
 $\langle \text{proof} \rangle$

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

lemma $\text{normal-simps[simp]}$:
 normal TrueF
 normal FalseF

$normal (Atom a) \longleftrightarrow anormal a$
 $normal (And \varphi_1 \varphi_2) \longleftrightarrow normal \varphi_1 \wedge normal \varphi_2$
 $normal (Or \varphi_1 \varphi_2) \longleftrightarrow normal \varphi_1 \vee normal \varphi_2$
 $normal (Neg \varphi) \longleftrightarrow normal \varphi$
 $normal (ExQ \varphi) \longleftrightarrow normal \varphi$
 <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]*:
 $\forall \varphi \in set fs. normal \varphi \implies normal (list-disj fs)$
 <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*: $nqfree \varphi \implies as : set(dnf \varphi) \implies a : set as \implies a : atoms \varphi$
 <proof>

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

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 $qf = qe\ [a \leftarrow as.\ depends_0\ a]$;
 $indep = [Atom(decr\ a).\ a \leftarrow as,\ \neg\ depends_0\ a]$
 in and $qf\ (list-conj\ indep)$)

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

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

lemma *I-qelim*:

assumes $qe: \bigwedge as.\ (\forall a \in set\ as.\ depends_0\ a) \Longrightarrow is-dnf-qe\ qe\ as$

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**

$lift-dnf-qe\ qe\ (And\ \varphi_1\ \varphi_2) = and\ (lift-dnf-qe\ qe\ \varphi_1)\ (lift-dnf-qe\ qe\ \varphi_2) \mid$
 $lift-dnf-qe\ qe\ (Or\ \varphi_1\ \varphi_2) = or\ (lift-dnf-qe\ qe\ \varphi_1)\ (lift-dnf-qe\ qe\ \varphi_2) \mid$
 $lift-dnf-qe\ qe\ (Neg\ \varphi) = neg\ (lift-dnf-qe\ qe\ \varphi) \mid$
 $lift-dnf-qe\ qe\ (ExQ\ \varphi) = Disj\ (dnf\ (nnf\ (lift-dnf-qe\ qe\ \varphi)))\ (qelim\ qe) \mid$
 $lift-dnf-qe\ qe\ \varphi = \varphi$

lemma *qfree-lift-dnf-qe*: $(\bigwedge as.\ (\forall a \in set\ as.\ depends_0\ a) \Longrightarrow qfree(qe\ as))$

$\Longrightarrow qfree(lift-dnf-qe\ qe\ \varphi)$

<proof>

lemma *qfree-lift-dnf-qe2*: $qe : lists\ depends_0 \rightarrow qfree$

$\Longrightarrow qfree(lift-dnf-qe\ qe\ \varphi)$

<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 $\bigwedge as.\ (\forall a \in set\ as.\ depends_0\ a) \Longrightarrow qfree(qe\ as)$

and $\bigwedge as. (\forall a \in set\ as. depends_0\ a) \implies is_dnf_qe\ qe\ as$
shows $I\ (lift_dnf_qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
 $\langle proof \rangle$

lemma *I-lift-dnf-qe2*:
assumes $qe : lists\ depends_0 \rightarrow qfree$
and $\forall as \in lists\ depends_0. is_dnf_qe\ qe\ as$
shows $I\ (lift_dnf_qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
 $\langle proof \rangle$

Quantifier elimination with invariant (needed for Presburger):

lemma *I-qelim-anormal*:
assumes $qe: \bigwedge xs\ as. \forall a \in set\ as. depends_0\ a \wedge anormal\ a \implies is_dnf_qe\ qe\ as$
and $nm: \forall a \in set\ as. anormal\ a$
shows $I\ (qelim\ qe\ as)\ xs = (\exists x. \forall a \in set\ as. I_a\ a\ (x \# xs))$
 $\langle proof \rangle$

declare $[[simp_depth_limit = 5]]$

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

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 $normal\ \varphi \implies normal(lift_dnf_qe\ qe\ \varphi)$
 $\langle proof \rangle$

declare $[[simp_depth_limit = 9]]$

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 $\bigwedge xs\ as. \forall a \in set\ as. depends_0\ a \wedge anormal\ a \implies is_dnf_qe\ qe\ as$
shows $normal\ f \implies I\ (lift_dnf_qe\ qe\ f)\ xs = I\ f\ xs$
 $\langle proof \rangle$
declare $[[simp_depth_limit = 50]]$

lemma *I-lift-dnf-qe-anormal2*:
assumes $qe : lists\ depends_0 \rightarrow qfree$
and $qe : lists(depends_0 \cap anormal) \rightarrow normal$
and $\forall as \in lists(depends_0 \cap anormal). is_dnf_qe\ qe\ as$
shows $normal\ f \implies I\ (lift_dnf_qe\ qe\ f)\ xs = I\ f\ xs$
 $\langle proof \rangle$

2.1.2 NNF-based

fun *lift-nnf-qe* :: $('a\ fm \Rightarrow 'a\ fm) \Rightarrow 'a\ fm \Rightarrow 'a\ fm$ **where**
 $lift_nnf_qe\ qe\ (And\ \varphi_1\ \varphi_2) = and\ (lift_nnf_qe\ qe\ \varphi_1)\ (lift_nnf_qe\ qe\ \varphi_2) \mid$

$lift\text{-}nnf\text{-}qe\ qe\ (Or\ \varphi_1\ \varphi_2) = or\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi_1)\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi_2) \mid$
 $lift\text{-}nnf\text{-}qe\ qe\ (Neg\ \varphi) = neg\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi) \mid$
 $lift\text{-}nnf\text{-}qe\ qe\ (ExQ\ \varphi) = qe\ (nnf\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)) \mid$
 $lift\text{-}nnf\text{-}qe\ qe\ \varphi = \varphi$

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

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

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

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

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

lemma *I-lift-nnf-qe-normal2*:
assumes $qe : nqfree \rightarrow qfree$
and $qe : nqfree \cap normal \rightarrow normal$
and $ALL\ \varphi : normal\ Int\ nqfree. ALL\ xs. I\ (qe\ \varphi)\ xs = (\exists x. I\ \varphi\ (x\#xs))$
shows $normal\ \varphi \implies I\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
 $\langle proof \rangle$

end

2.2 With equality

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

```

locale ATOM-EQ = ATOM +
fixes solvable0 :: 'a ⇒ bool
and trivial :: 'a ⇒ bool
and subst0 :: 'a ⇒ 'a ⇒ 'a
assumes subst0:
  [ solvable0 eq; ¬trivial eq; Ia eq (x#xs); depends0 a ]
  ⇒ Ia (subst0 eq a) xs = Ia a (x#xs)
and trivial: trivial eq ⇒ Ia eq xs
and solvable: solvable0 eq ⇒ ∃x. Ia eq (x#xs)
and is-triv-self-subst: solvable0 eq ⇒ trivial (subst0 eq eq)

```

begin

```

definition lift-eq-qe :: ('a list ⇒ 'a fm) ⇒ 'a list ⇒ 'a fm where
lift-eq-qe qe as =
  (let as = [a←as. ¬ trivial a]
   in case [a←as. solvable0 a] of
     [] ⇒ qe as
   | eq # eqs ⇒
     (let ineqs = [a←as. ¬ solvable0 a]
      in list-conj (map (Atom ◦ (subst0 eq)) (eqs @ ineqs))))

```

theorem *I-lift-eq-qe*:

assumes *dep*: ∀ a∈set *as*. *depends₀ a*

assumes *qe*: ∧ *as*. (∀ a ∈ set *as*. *depends₀ a* ∧ ¬ *solvable₀ a*) ⇒
*I (qe as) xs = (∃ x. ∀ a ∈ set *as*. *I_a a (x#xs)*)*

shows *I (lift-eq-qe qe as) xs = (∃ x. ∀ a ∈ set *as*. *I_a a (x#xs)*)*
 (is ?L = ?R)

⟨*proof*⟩

definition *lift-dnf-qe* = *lift-dnf-qe* ◦ *lift-eq-qe*

lemma *qfree-lift-eq-qe*:

(∧ *as*. ∀ a∈set *as*. *depends₀ a* ⇒ *qfree (qe as)*) ⇒
 ∀ a∈set *as*. *depends₀ a* ⇒ *qfree (lift-eq-qe qe as)*

⟨*proof*⟩

lemma *qfree-lift-dnf-qe*: (∧ *as*. (∀ a∈set *as*. *depends₀ a*) ⇒ *qfree (qe as)*)

⇒ *qfree (lift-dnf-qe qe φ)*

⟨*proof*⟩

lemma *I-lift-dnf-qe*:

(∧ *as*. (∀ a ∈ set *as*. *depends₀ a*) ⇒ *qfree (qe as)*) ⇒

(∧ *as*. (∀ a ∈ set *as*. *depends₀ a* ∧ ¬ *solvable₀ a*) ⇒ *is-dnf-qe qe as*) ⇒
I (lift-dnf-qe qe φ) xs = I φ xs

<proof>

lemma *I-lift-dnfeq-qe2*:

$qe : lists\ depends_0 \rightarrow qfree \implies$

$(\forall as \in lists(dependends_0 \cap -solvable_0). is-dnf-qe\ qe\ as) \implies$

$I\ (lift-dnfeq-qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$

<proof>

end

end

3 DLO

theory *DLO*

imports *QE Complex-Main*

begin

3.1 Basics

consts-code *undefined* ((*raise Match*))

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 o is-Less*

lemma *is-Less-iff*: $is-Less\ a = (\exists i\ j. a = Less\ i\ j)$

<proof>

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

<proof>

lemma *not-is-Eq-iff*: $(\forall i\ j. a \neq Eq\ i\ j) = (\exists i\ j. a = Less\ i\ j)$

<proof>

fun *neg_dlo* :: *atom* \Rightarrow *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 $I_{dlo} :: atom \Rightarrow 'a::dlo\ list \Rightarrow bool$ **where**
 $I_{dlo} (Eq\ i\ j)\ xs = (xs!i = xs!j) \mid$
 $I_{dlo} (Less\ i\ j)\ xs = (xs!i < xs!j)$

fun $depends_{dlo} :: atom \Rightarrow bool$ **where**
 $depends_{dlo} (Eq\ i\ j) = (i=0 \mid j=0) \mid$
 $depends_{dlo} (Less\ i\ j) = (i=0 \mid j=0)$

fun $decr_{dlo} :: atom \Rightarrow atom$ **where**
 $decr_{dlo} (Less\ i\ j) = Less\ (i - 1)\ (j - 1) \mid$
 $decr_{dlo} (Eq\ i\ j) = Eq\ (i - 1)\ (j - 1)$

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 $DLO!$:

$ATOM\ neg_{dlo}\ (\lambda a. True)\ I_{dlo}\ depends_{dlo}\ decr_{dlo}$
 $\langle proof \rangle$

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

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

$\langle ML \rangle$

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

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

definition $ebounds$ **where**

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

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

$\langle proof \rangle$

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

$\langle proof \rangle$

lemma $set-ebounds:$

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

$\langle proof \rangle$

abbreviation $LB\ f\ xs \equiv \{xs!i \mid i. Less\ (Suc\ i)\ 0 : set(DLO.atoms_0\ f)\}$

abbreviation $UB\ f\ xs \equiv \{xs!i \mid i. Less\ 0\ (Suc\ i) : set(DLO.atoms_0\ f)\}$

definition $EQ\ f\ xs = \{xs!k \mid k.$

$Eq\ (Suc\ k)\ 0 : set(DLO.atoms_0\ f) \vee Eq\ 0\ (Suc\ k) : set(DLO.atoms_0\ f)\}$

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

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

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

fun *isubst where* $isubst k 0 = k \mid isubst k (Suc i) = i$

fun *asubst :: nat \Rightarrow atom \Rightarrow atom where*
 $asubst k (Less i j) = Less (isubst k i) (isubst k j) \mid$
 $asubst k (Eq i j) = Eq (isubst k i) (isubst k j)$

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

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

fun *amin-inf :: atom \Rightarrow atom fm where*
 $amin-inf (Less - 0) = FalseF \mid$
 $amin-inf (Less 0 -) = TrueF \mid$
 $amin-inf (Less (Suc i) (Suc j)) = Atom(Less i j) \mid$
 $amin-inf (Eq 0 0) = TrueF \mid$
 $amin-inf (Eq 0 -) = FalseF \mid$
 $amin-inf (Eq - 0) = FalseF \mid$
 $amin-inf (Eq (Suc i) (Suc j)) = Atom(Eq i j)$

abbreviation *min-inf :: atom fm \Rightarrow atom fm (inf₋) where*
 $inf_- \equiv amap_{fm} amin-inf$

fun *aplus-inf :: atom \Rightarrow atom fm where*
 $aplus-inf (Less 0 -) = FalseF \mid$
 $aplus-inf (Less - 0) = TrueF \mid$
 $aplus-inf (Less (Suc i) (Suc j)) = Atom(Less i j) \mid$
 $aplus-inf (Eq 0 0) = TrueF \mid$
 $aplus-inf (Eq 0 -) = FalseF \mid$
 $aplus-inf (Eq - 0) = FalseF \mid$
 $aplus-inf (Eq (Suc i) (Suc j)) = Atom(Eq i j)$

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 - $\implies \exists x. ?P\ f\ x$)
 $\langle proof \rangle$

declare[[*simp-depth-limit=2*]]
lemma *LBex*:
 $\llbracket ngfree\ f; DLO.I\ f\ (x\ \# \ xs); \neg DLO.I\ (inf_- \ f)\ xs; x \notin EQ\ f\ xs \rrbracket$
 $\implies \exists l \in LB\ f\ xs. l < x$
 $\langle proof \rangle$

lemma *UBex*:
 $\llbracket ngfree\ f; DLO.I\ f\ (x\ \# \ xs); \neg DLO.I\ (inf_+ \ f)\ xs; x \notin EQ\ f\ xs \rrbracket$
 $\implies \exists u \in UB\ f\ xs. x < u$
 $\langle proof \rangle$
declare[[*simp-depth-limit=50*]]

lemma *finite-LB*: $finite(LB\ f\ xs)$
 $\langle proof \rangle$

lemma *finite-UB*: $finite(UB\ f\ xs)$
 $\langle proof \rangle$

lemma *qfree-amin-inf*: $qfree\ (amin-inf\ a)$
 $\langle proof \rangle$

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

lemma *qfree-aplus-inf*: $qfree\ (aplus-inf\ a)$
 $\langle proof \rangle$

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

end

theory *QEdlo*

imports *DLO* $\sim\sim$ /src/HOL/ex/Reflection
begin

3.2 DNF-based quantifier elimination

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

qe-dlo₁ 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-dlo₁*:

assumes less: $\forall a \in \text{set as. is-Less } a$ **and** dep: $\forall a \in \text{set as. depends}_{dlo} a$

shows *DLO.I* (*qe-dlo₁* as) xs = $(\exists x. \forall a \in \text{set as. } I_{dlo} a (x\#xs))$

(is ?L = ?R)

\langle proof \rangle

lemma *I-qe-dlo₁-pretty*:

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

\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 *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)

\langle proof \rangle

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*

\langle proof \rangle

\langle ML \rangle

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

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

\langle proof \rangle

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

\langle proof \rangle

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

\langle proof \rangle

end

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

definition *interpret* :: *atom fn* \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* = ($\forall x. \text{interpret } f (x\#xs)$)
interpret (*Or* (*Neg* *f1*) *f2*) *xs* = (*interpret f1 xs* \longrightarrow *interpret f2 xs*)
<*proof*>

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

corollary [*reflection*]: *interpret* (*qe-dlo* *f*) *xs* = *interpret f xs*
<*proof*>

<*ML*>

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

3.3 Examples

lemma $\forall x::\text{real}. \neg x < x$
<*proof*>

lemma $\forall x y::\text{real}. \exists z. x < y \longrightarrow x < z \ \& \ z < y$
<*proof*>

lemma $\forall x::real. \neg x < x$
 $\langle proof \rangle$

lemma $\forall x y::real. \exists z. x < y \longrightarrow x < z \ \& \ z < y$
 $\langle proof \rangle$

lemma $\neg(\exists x y z. \forall u::real. x < x \mid \neg x < u \mid x < y \ \& \ y < z \ \& \ \neg x < z)$
 $\langle proof \rangle$

lemma $qe-dlo(AllQ (Imp (Atom(Less 0 1)) (Atom(Less 1 0)))) = FalseF$
 $\langle proof \rangle$

lemma $qe-dlo(AllQ(AllQ (Imp (Atom(Less 0 1)) (Atom(Less 0 1)))) = TrueF$
 $\langle proof \rangle$

lemma
 $qe-dlo(AllQ(ExQ(AllQ (And (Atom(Less 2 1)) (Atom(Less 1 0)))))) = FalseF$
 $\langle proof \rangle$

lemma $qe-dlo(AllQ(ExQ(ExQ (And (Atom(Less 1 2)) (Atom(Less 2 0)))))) = TrueF$
 $\langle proof \rangle$

lemma
 $qe-dlo(AllQ(AllQ(ExQ (And (Atom(Less 1 0)) (Atom(Less 0 2)))))) = FalseF$
 $\langle proof \rangle$

lemma $qe-dlo(AllQ(AllQ(ExQ (Imp (Atom(Less 1 2)) (And (Atom(Less 1 0)) (Atom(Less 0 2)))))) = TrueF$
 $\langle proof \rangle$

normal-form $qe-dlo(AllQ (Imp (Atom(Less 0 1)) (Atom(Less 0 2))))$

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 §5.3) 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 $asubst_2 :: nat \Rightarrow nat \Rightarrow atom \Rightarrow atom\ fm$ **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 $subst_2\ l\ u \equiv amap_{fm}\ (asubst_2\ l\ u)$

lemma $I\text{-}subst_2\ 1$:

$nqfree\ f \implies xs!l < xs!u \implies DLO.I\ (subst_2\ l\ u\ f)\ xs$
 $\implies xs!l < x \implies x < xs!u \implies DLO.I\ f\ (x\#\ xs)$
 $\langle proof \rangle$

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 $nolub\text{-}And[simp]$:

$nolub\ (And\ f\ g)\ xs\ l\ x\ u = (nolub\ f\ xs\ l\ x\ u \wedge nolub\ g\ xs\ l\ x\ u)$
 $\langle proof \rangle$

lemma $nolub\text{-}Or[simp]$:

$nolub\ (Or\ f\ g)\ xs\ l\ x\ u = (nolub\ f\ xs\ l\ x\ u \wedge nolub\ g\ xs\ l\ x\ u)$
 $\langle proof \rangle$

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

lemma $innermost\text{-}intl$:

$\llbracket nqfree\ f; nolub\ f\ xs\ l\ x\ u; l < x; x < u; x \notin EQ\ f\ xs;$
 $DLO.I\ f\ (x\#\ xs); l < y; y < u \rrbracket$
 $\implies DLO.I\ f\ (y\#\ xs)$
 $\langle proof \rangle$

lemma $I\text{-}subst_2\ 2$:

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

declare $[[simp\text{-}depth\text{-}limit=50]]$

definition

$qe\text{-}interior_1\ \varphi =$

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

lemma $dense\text{-}interval$:

assumes *finite L finite U* $l : L \ u : U \ l < x \ x < u \ P(x::'a::dlo)$
and *dense*: $\bigwedge y \ l \ u. \llbracket \forall y \in \{l <..<x\}. y \notin L; \forall y \in \{x <..<u\}. y \notin U;$
 $l < x; x < u; l < y; y < u \rrbracket \implies P \ y$
shows $\exists l \in L. \exists u \in U. l < x \wedge x < u \wedge (\forall y \in \{l <..<x\}. y \notin L) \wedge (\forall y \in \{x <..<u\}. y \notin U)$
 $\wedge (\forall y. l < y \wedge y < u \longrightarrow P \ y)$
 $\langle proof \rangle$

theorem *I-interior1*:
assumes *nqfree* φ **shows** $DLO.I \ (qe\text{-interior}_1 \ \varphi) \ xs = (EX \ x. DLO.I \ \varphi \ (x\#xs))$
(is $?QE = ?EX$
 $\langle proof \rangle$

lemma *qfree-asubst2*: $qfree \ (asubst_2 \ l \ u \ a)$
 $\langle proof \rangle$

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

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

definition *qe-interior* = $DLO.lift\text{-nnf}\text{-qe} \ qe\text{-interior}_1$

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

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

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 §5.4.

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 $nolb-And[simp]$:

$nolb\ (And\ \varphi_1\ \varphi_2)\ xs\ l\ x = (nolb\ \varphi_1\ xs\ l\ x \wedge nolb\ \varphi_2\ xs\ l\ x)$
 $\langle proof \rangle$

lemma $nolb-Or[simp]$:

$nolb\ (Or\ \varphi_1\ \varphi_2)\ xs\ l\ x = (nolb\ \varphi_1\ xs\ l\ x \wedge nolb\ \varphi_2\ xs\ l\ x)$
 $\langle proof \rangle$

declare $[simp-depth-limit=3]$

lemma $innermost-intvl$:

$\llbracket\ ngfree\ \varphi; nolb\ \varphi\ xs\ l\ x; l < x; x \notin EQ\ \varphi\ xs; DLO.I\ \varphi\ (x\#\!xs); l < y; y \leq x \rrbracket$
 $\implies DLO.I\ \varphi\ (y\#\!xs)$
 $\langle proof \rangle$

lemma $I-subst-peps2$:

$ngfree\ \varphi \implies xs!\!l < x \implies nolb\ \varphi\ xs\ (xs!\!l)\ x \implies x \notin EQ\ \varphi\ xs$
 $\implies \forall y \in \{xs!\!l <.. x\}. DLO.I\ \varphi\ (y\#\!xs)$
 $\implies DLO.I\ (subst_+\ \varphi\ l)\ xs$
 $\langle proof \rangle$

declare $[simp-depth-limit=50]$

lemma $dense-interval$:

assumes $finite\ L\ l \in L\ l < x\ P(x::'a::dlo)$
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 $\exists l \in L. l < x \wedge (\forall y \in \{l <..<x\}. y \notin L) \wedge (\forall y. l < y \wedge y \leq x \longrightarrow P\ y)$
 $\langle proof \rangle$

lemma $I-subst-peps$:

$ngfree\ \varphi \implies DLO.I\ (subst_+\ \varphi\ l)\ xs \longrightarrow$
 $(\exists leps > xs!\!l. \forall x. xs!\!l < x \wedge x \leq leps \longrightarrow DLO.I\ \varphi\ (x\#\!xs))$
 $\langle proof \rangle$

definition

$qe-eps_1(\varphi) =$

(let as = DLO.atoms₀ φ; lbs = lbounds as; ebs = ebounds as
in list-disj (inf₋ φ # map (subst₊ φ) lbs @ map (subst φ) ebs))

theorem *I-qe-eps1*:

assumes nqfree φ **shows** DLO.I (qe-eps₁ φ) xs = (∃ x. DLO.I φ (x#xs))
(is ?QE = ?EX)

⟨proof⟩

lemma qfree-astubst-peps: qfree (astubst₊ k a)

⟨proof⟩

lemma qfree-subst-peps: nqfree φ ⇒ qfree (subst₊ φ k)

⟨proof⟩

lemma qfree-qe-eps₁: nqfree φ ⇒ qfree(qe-eps₁ φ)

⟨proof⟩

definition qe-eps = DLO.lift-nnf-qe qe-eps₁

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 Lists as vectors

theory ListVector

imports List Main

begin

A vector-space like structure of lists and arithmetic operations on them. Is only a vector space if restricted to lists of the same length.

Multiplication with a scalar:

abbreviation scale :: ('a::times) ⇒ 'a list ⇒ 'a list (**infix** *_s 70)

where x *_s xs ≡ map (op * x) xs

lemma scaleI[simp]: (1::'a::monoid-mult) *_s xs = xs

⟨proof⟩

4.1 + and −

fun zipwith0 :: ('a::zero ⇒ 'b::zero ⇒ 'c) ⇒ 'a list ⇒ 'b list ⇒ 'c list

where

zipwith0 f [] [] = [] |

$zipwith0\ f\ (x\#xs)\ (y\#ys) = f\ x\ y\ \# \ zipwith0\ f\ xs\ ys \mid$
 $zipwith0\ f\ (x\#xs)\ [] = f\ x\ 0\ \# \ zipwith0\ f\ xs\ [] \mid$
 $zipwith0\ f\ []\ (y\#ys) = f\ 0\ y\ \# \ zipwith0\ f\ []\ ys$

instantiation $list :: (\{zero, plus\})\ plus$
begin

definition

$list-add-def: op\ + = zipwith0\ (op\ +)$

instance $\langle proof \rangle$

end

instantiation $list :: (\{zero, uminus\})\ uminus$
begin

definition

$list-uminus-def: uminus = map\ uminus$

instance $\langle proof \rangle$

end

instantiation $list :: (\{zero, minus\})\ minus$
begin

definition

$list-diff-def: op\ - = zipwith0\ (op\ -)$

instance $\langle proof \rangle$

end

lemma $zipwith0-Nil[simp]: zipwith0\ f\ []\ ys = map\ (f\ 0)\ ys$
 $\langle proof \rangle$

lemma $list-add-Nil[simp]: []\ +\ xs = (xs::'a::monoid-add\ list)$
 $\langle proof \rangle$

lemma $list-add-Nil2[simp]: xs\ +\ [] = (xs::'a::monoid-add\ list)$
 $\langle proof \rangle$

lemma $list-add-Cons[simp]: (x\#xs)\ +\ (y\#ys) = (x+y)\#(xs+ys)$
 $\langle proof \rangle$

lemma $list-diff-Nil[simp]: []\ -\ xs = -(xs::'a::group-add\ list)$
 $\langle proof \rangle$

lemma *list-diff-Nil2[simp]*: $xs - [] = (xs :: 'a :: \text{group-add list})$
 ⟨proof⟩

lemma *list-diff-Cons-Cons[simp]*: $(x \# xs) - (y \# ys) = (x - y) \# (xs - ys)$
 ⟨proof⟩

lemma *list-uminus-Cons[simp]*: $-(x \# xs) = (-x) \# (-xs)$
 ⟨proof⟩

lemma *self-list-diff*:
 $xs - xs = \text{replicate } (\text{length}(xs :: 'a :: \text{group-add list})) \ 0$
 ⟨proof⟩

lemma *list-add-assoc*: **fixes** $xs :: 'a :: \text{monoid-add list}$
shows $(xs + ys) + zs = xs + (ys + zs)$
 ⟨proof⟩

4.2 Inner product

definition *iproduct* :: $'a :: \text{ring list} \Rightarrow 'a \text{ list} \Rightarrow 'a \langle \langle -, - \rangle \rangle$ **where**
 $\langle xs, ys \rangle = (\sum (x, y) \leftarrow \text{zip } xs \ ys. \ x * y)$

lemma *iproduct-Nil[simp]*: $\langle [], ys \rangle = 0$
 ⟨proof⟩

lemma *iproduct-Nil2[simp]*: $\langle xs, [] \rangle = 0$
 ⟨proof⟩

lemma *iproduct-Cons[simp]*: $\langle x \# xs, y \# ys \rangle = x * y + \langle xs, ys \rangle$
 ⟨proof⟩

lemma *iproduct-if-coeffs0*: $\forall c \in \text{set } cs. \ c = 0 \implies \langle cs, xs \rangle = 0$
 ⟨proof⟩

lemma *iproduct-uminus[simp]*: $\langle -xs, ys \rangle = -\langle xs, ys \rangle$
 ⟨proof⟩

lemma *iproduct-left-add-distrib*: $\langle xs + ys, zs \rangle = \langle xs, zs \rangle + \langle ys, zs \rangle$
 ⟨proof⟩

lemma *iproduct-left-diff-distrib*: $\langle xs - ys, zs \rangle = \langle xs, zs \rangle - \langle ys, zs \rangle$
 ⟨proof⟩

lemma *iproduct-assoc*: $\langle x *_s xs, ys \rangle = x * \langle xs, ys \rangle$
 ⟨proof⟩

end

5 Linear real arithmetic

```
theory LinArith
imports QE ListVector Complex-Main
begin
```

```
declare iprod-assoc[simp]
```

5.1 Basics

5.1.1 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 o is-Less
```

```
lemma is-Less-iff: is-Less f = ( $\exists$  r rs. f = Less r rs)
<proof>
```

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

```
fun negR :: atom  $\Rightarrow$  atom fm where
negR (Less r t) = Or (Atom(Less (-r) (-t))) (Atom(Eq r t)) |
negR (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 dependsR a = (hd-coeff a  $\neq$  0)
```

```
fun decrR :: atom  $\Rightarrow$  atom where
decrR (Less r rs) = Less r (tl rs) |
decrR (Eq r rs) = Eq r (tl rs)
```

```
fun IR :: atom  $\Rightarrow$  real list  $\Rightarrow$  bool where
IR (Less r cs) xs = (r < <cs,xs>) |
IR (Eq r cs) xs = (r = <cs,xs>)
```

```
definition atoms0 = ATOM.atoms0 dependsR
```

```
interpretation R!: ATOM negR ( $\lambda$ a. True) IR dependsR decrR
  where ATOM.atoms0 dependsR = atoms0
<proof>
```

$\langle ML \rangle$

5.1.2 Shared constructions

fun *combine* :: (real * real list) \Rightarrow (real * real list) \Rightarrow atom **where**
combine (r₁,cs₁) (r₂,cs₂) = Less (r₁-r₂) (cs₂ - cs₁)

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) : set as \wedge c>0}
 \langle proof \rangle

lemma *set-ubounds*:

set(*ubounds* as) = {(r/c, (-1/c) *_s cs)|r c cs. Less r (c#cs) : set as \wedge c<0}
 \langle proof \rangle

lemma *set-ebounds*:

set(*ebounds* as) = {(r/c, (-1/c) *_s cs)|r c cs. Eq r (c#cs) : set as \wedge c \neq 0}
 \langle proof \rangle

abbreviation *EQ* **where**

EQ f xs \equiv {(r - \langle cs,xs \rangle)/c|r c cs. Eq r (c#cs) : set(R.atoms₀ 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) : set(R.atoms₀ f) \wedge c>0}

abbreviation *UB* **where**

UB f xs \equiv {(r - \langle cs,xs \rangle)/c|r c cs. Less r (c#cs) : set(R.atoms₀ f) \wedge c<0}

fun *asubst* :: real * real list \Rightarrow atom \Rightarrow 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*:

$qfree\ \varphi \implies R.I\ (subst\ \varphi\ t)\ xs = R.I\ \varphi\ (eval\ t\ xs\ \# xs)$
 <proof>

lemma *I-subst-pretty*:

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

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

inf₋ (*And* $\varphi_1\ \varphi_2$) = *and* (*inf*₋ φ_1) (*inf*₋ φ_2) |
inf₋ (*Or* $\varphi_1\ \varphi_2$) = *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₋ $\varphi = \varphi$

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

inf₊ (*And* $\varphi_1\ \varphi_2$) = *and* (*inf*₊ φ_1) (*inf*₊ φ_2) |
inf₊ (*Or* $\varphi_1\ \varphi_2$) = *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₊ $\varphi = \varphi$

lemma *qfree-min-inf*: $qfree\ \varphi \implies qfree(inf_{-}\ \varphi)$
 <proof>

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

lemma *min-inf*:

$ngfree\ f \implies \exists x. \forall y \leq x. R.I\ (inf_{-}\ f)\ xs = R.I\ f\ (y\ \#\ xs)$
 (**is** - $\implies \exists x. ?P\ f\ x$)
 <proof>

lemma *plus-inf*:

$ngfree\ f \implies \exists x. \forall y \geq x. R.I\ (inf_{+}\ f)\ xs = R.I\ f\ (y\ \#\ xs)$
 (**is** - $\implies \exists x. ?P\ f\ x$)
 <proof>

declare [[*simp-depth-limit* = 2]]

lemma *LBex*:

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

lemma *UBex*:

$$\llbracket \text{ngfree } f; R.I \text{ } f \text{ } (x \# xs); \neg R.I \text{ } (inf_+ f) \text{ } xs; x \notin EQ \text{ } f \text{ } xs \rrbracket$$

$$\implies \exists u \in UB \text{ } f \text{ } xs. x < u$$
 <proof>

declare $[[simp\text{-}depth\text{-}limit = 50]]$

lemma *finite-LB*: $finite(LB \text{ } f \text{ } xs)$
 <proof>

lemma *finite-UB*: $finite(UB \text{ } f \text{ } xs)$
 <proof>

end

theory *QElin*
imports *LinArith*
begin

5.2 Fourier

definition *qe-FM₁* :: *atom list* \Rightarrow *atom fm* **where**
qe-FM₁ *as* = *list-conj* [*Atom*(*combine* *p* *q*). *p*←*lbounds* *as*, *q*←*ubounds* *as*]

theorem *I-qe-FM₁*:
assumes *less*: $\forall a \in set \text{ } as. is\text{-}Less \text{ } a$ **and** *dep*: $\forall a \in set \text{ } as. depends_R \text{ } a$
shows $R.I \text{ } (qe\text{-}FM_1 \text{ } as) \text{ } xs = (\exists x. \forall a \in set \text{ } as. I_R \text{ } a \text{ } (x \# xs))$ (**is** ?*L* = ?*R*)
 <proof>

corollary *I-qe-FM₁-pretty*:
 $\forall a \in set \text{ } as. is\text{-}Less \text{ } a \wedge depends_R \text{ } a \implies R.is\text{-}dnf\text{-}qe \text{ } qe\text{-}FM_1 \text{ } as$
 <proof>

fun *subst₀* :: *atom* \Rightarrow *atom* \Rightarrow *atom* **where**
subst₀ (*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 *subst₀-pretty*:
 $subst_0 \text{ } (Eq \text{ } r \text{ } (c \# cs)) \text{ } (Less \text{ } s \text{ } (d \# ds)) = Less \text{ } (s - (r * d) / c) \text{ } (ds - (d / c) *_{s} cs)$
 $subst_0 \text{ } (Eq \text{ } r \text{ } (c \# cs)) \text{ } (Eq \text{ } s \text{ } (d \# ds)) = Eq \text{ } (s - (r * d) / c) \text{ } (ds - (d / c) *_{s} cs)$
 <proof>

lemma *I-subst₀*: $depends_R \text{ } a \implies c \neq 0 \implies$
 $I_R \text{ } (subst_0 \text{ } (Eq \text{ } r \text{ } (c \# cs)) \text{ } a) \text{ } xs = I_R \text{ } a \text{ } ((r - \langle cs, xs \rangle) / c \# xs)$
 <proof>

interpretation $R_e!$:

$ATOM-EQ\ neg_R (\lambda a. True) I_R\ depends_R\ decr_R$
 $(\lambda Eq\ -\ (c\#\ -) \Rightarrow c \neq 0 \mid - \Rightarrow False)$
 $(\lambda Eq\ r\ cs \Rightarrow r=0 \wedge (\forall c \in\ set\ cs.\ c=0) \mid - \Rightarrow False)\ subst_0$
 $\langle proof \rangle$

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

lemma $qfree-qe-FM_1: qfree\ (qe-FM_1\ as)$
 $\langle proof \rangle$

corollary $I-qe-FM: R.I\ (qe-FM\ \varphi)\ xs = R.I\ \varphi\ xs$
 $\langle proof \rangle$

theorem $qfree-qe-FM: qfree\ (qe-FM\ f)$
 $\langle proof \rangle$

5.2.1 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 $qe-FM(TrueF) = TrueF$
 $\langle proof \rangle$

lemma
 $qe-FM(ExQ\ (And\ (Atom(Less\ 0\ [1]))\ (Atom(Less\ 0\ [-1])))) = Atom(Less\ 0\ [])$
 $\langle proof \rangle$

lemma
 $qe-FM(ExQ\ (And\ (Atom(Less\ 0\ [1]))\ (Atom(Less\ -1\ [-1])))) = Atom(Less\ -1\ [])$
 $\langle proof \rangle$

end

theory $QElin-opt$
imports $QElin$
begin

5.2.2 An optimization

Atoms are simplified asap.

definition

```

asimp a = (case a of
  Less r cs => (if  $\forall c \in \text{set } cs. c = 0$ 
    then if  $r < 0$  then TrueF else FalseF
    else Atom a) |
  Eq r cs => (if  $\forall c \in \text{set } cs. c = 0$ 
    then if  $r = 0$  then TrueF else FalseF else Atom a))

```

lemma *asimp-pretty*:

```

asimp (Less r cs) =
  (if  $\forall c \in \text{set } cs. c = 0$ 
    then if  $r < 0$  then TrueF else FalseF
    else Atom(Less r cs))
asimp (Eq r cs) =
  (if  $\forall c \in \text{set } cs. c = 0$ 
    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* = $R_e.lift-dnfq-qe$ 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

5.3 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 φ =
 (let *as* = *R.atoms*₀ φ ; *lbs* = *lbounds as*; *ubs* = *ubounds as*; *ebs* = *ebounds as*;
 intrs = [subst φ (*between l u*) . *l* \leftarrow *lbs*, *u* \leftarrow *ubs*]
 in *list-disj* (*inf*₋ φ # *inf*₊ φ # *intrs* @ *map* (*subst* φ) *ebs*))

lemma *dense-interval*:

assumes *finite L finite U l : L u : 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>

declare [[*simp-depth-limit* = 50]]

lemma *dense*:

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

theorem *I-FR*₁:

assumes *ngfree* φ **shows** *R.I* (*FR*₁ φ) *xs* = ($\exists x. R.I \varphi (x \# xs)$)
 (is ?*FR* = ?*EX*)
 <proof>

definition *FR* = *R.lift-nnf-qe FR*₁

lemma *qfree-FR*₁: *ngfree* $\varphi \Longrightarrow$ *qfree* (*FR*₁ 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

```

5.4 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+  $\varphi$  rcs  $\equiv$  amap_fm (asubst+ rcs)  $\varphi$ 

```

```

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)
<proof>

```

```

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

```

```

declare[[simp-depth-limit=3]]
lemma innermost-intvl:
  [[ nqfree f; nolb f xs l x;  $l < x$ ;  $x \notin EQ f xs$ ; R.I f (x#xs);  $l < y$ ;  $y \leq x$  ]]
   $\implies$  R.I f (y#xs)
<proof>

```

```

definition EQ2 = EQ

```

```

lemma EQ2-Or[simp]: EQ2 (Or f g) xs = (EQ2 f xs Un EQ2 g xs)
<proof>

```

```

lemma EQ2-And[simp]: EQ2 (And f g) xs = (EQ2 f xs Un 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);  $l < y$ ;  $y \leq x$  ]]
   $\implies$  R.I f (y#xs)
<proof>

```

lemma *I-subst-peps2*:

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

<proof>

declare[[*simp-depth-limit=50*]]

lemma *I-subst-peps*:

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

lemma *dense-interval*:

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

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

theorem *I-eps1*:

assumes *ngfree* f **shows** $R.I\ (qe-eps_1\ f)\ xs = (\exists x. R.I\ f\ (x\#\#xs))$
(is $?QE = ?EX$ **)**
<proof>

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

<proof>

lemma *qfree-subst-peps*: $ngfree\ \varphi \implies qfree\ (subst_+\ \varphi\ rcs)$

<proof>

lemma *qfree-qe-eps1*: $ngfree\ \varphi \implies qfree\ (qe-eps_1\ \varphi)$

<proof>

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

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

<proof>

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

<proof>

end

6 Presburger arithmetic

```

theory PresArith
imports GCD QE ListVector
begin

```

```

declare iprod-assoc[simp]

```

6.1 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 negZ :: atom  $\Rightarrow$  atom fm where
  negZ (Le i ks) = Atom(Le (1-i) (-ks)) |
  negZ (Dvd d i ks) = Atom(NDvd d i ks) |
  negZ (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 decrZ :: atom  $\Rightarrow$  atom where
  decrZ (Le i ks) = Le i (tl ks) |
  decrZ (Dvd d i ks) = Dvd d i (tl ks) |
  decrZ (NDvd d i ks) = NDvd d i (tl ks)

```

```

fun IZ :: atom  $\Rightarrow$  int list  $\Rightarrow$  bool where
  IZ (Le i ks) xs = (i  $\leq$  <ks,xs>) |
  IZ (Dvd d i ks) xs = (d dvd i + <ks,xs>) |
  IZ (NDvd d i ks) xs = ( $\neg$  d dvd i + <ks,xs>)

```

```

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

```

interpretation Z!:

```

  ATOM negZ ( $\lambda a.$  divisor a  $\neq$  0) IZ ( $\lambda a.$  hd-coeff a  $\neq$  0) decrZ
  where ATOM.atoms0 ( $\lambda a.$  hd-coeff a  $\neq$  0) = atoms0
  <proof>

```

```

  <ML>

```

abbreviation

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

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

fun asubst :: int \Rightarrow int 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 list \Rightarrow atom fm \Rightarrow atom 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-subst:

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

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-lbounds:

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

lemma set-ubounds:

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

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

6.2 LCM and lemmas

lemma zdiv-eq-0-iff:

$(i::int) \ div \ k = 0 \ \longleftrightarrow \ k=0 \ \vee \ 0 \leq i \wedge i < k \ \vee \ i \leq 0 \wedge k < i$
 $\langle proof \rangle$

lemma pos-imp-zdiv-pos-iff:

$0 < k \ \Longrightarrow \ 0 < (i::int) \ div \ k \ \longleftrightarrow \ k \leq i$
 $\langle proof \rangle$

lemma $zmod\text{-le-nonneg-dividend}$: $(m::int) \geq 0 \ \Longrightarrow \ m \ mod \ k \leq m$

$\langle proof \rangle$

fun *zlcms* :: *int list* \Rightarrow *int* **where**

zlcms [] = 1 |
zlcms (i#is) = lcm i (*zlcms* is)

lemma *dvd-zlcms*: $i : \text{set } is \Longrightarrow i \text{ dvd } zlcms \ is$
 <proof>

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

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

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

6.3 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 $\llbracket k \neq 0; k \text{ dvd } m \rrbracket \Longrightarrow$
 $I_Z (\text{hd-coeff1 } m (\text{Dvd } d \ i \ (k\#ks))) (x\#e) = (\text{let } m' = m \ \text{div} \ (\text{abs } k) \ \text{in}$
 $I_Z (\text{Dvd } (m'*d) (m'*i) (\text{sgn } k \ # \ (m' *_s \ ks))) (x\#e))$
 <proof>

lemma *I-hd-coeff1-mult-a*: **assumes** $m > 0$

shows $\text{hd-coeff } a \ \text{dvd } m \mid \text{hd-coeff } a = 0 \Longrightarrow I_Z (\text{hd-coeff1 } m \ a) (m*x\#xs) = I_Z$
 $a \ (x\#xs)$
 <proof>

lemma *I-hd-coeff1-mult*: **assumes** $m > 0$

shows $\text{qfree } \varphi \Longrightarrow \forall a \in \text{set}(Z.\text{atoms}_0 \ \varphi). \text{hd-coeff } a \ \text{dvd } m \Longrightarrow$
 $Z.I (\text{map}_{fm} (\text{hd-coeff1 } m) \ \varphi) (m*x\#xs) = Z.I \ \varphi \ (x\#xs)$

<proof>

end

theory *QEpres*
imports *PresArith*
begin

6.4 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**
($\exists x. \forall a \in \text{set}(hd-coeffs1 as). I_Z a (x\#xs)$) =
($\exists x. \forall a \in \text{set } as. I_Z a (x\#xs)$) (is ?B = ?A)
<proof>

abbreviation *is-dvd a \equiv case a of Le - - \Rightarrow False | - \Rightarrow 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] ($\lambda n. \text{list-conj}(\text{map} (\text{Atom} \circ \text{asubst } n \ []) ds)$)
else
Disj ls ($\lambda(l_i, l_k s). \text{Disj} [0..d - 1] (\lambda n. \text{list-conj}(\text{map} (\text{Atom} \circ \text{asubst} (l_i + n) (-l_k s)) as))$)

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 *$Z.I (qe-pres_1 as) xs = (\exists x. \forall a \in \text{set } as. I_Z a (x\#xs))$*
<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*: $a \in \text{set } (\text{hd-coeffs1 } as)$ **shows** *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** *hd-coeff-is1* a

<proof>

lemma *I-qe-pres1-o*:

$\llbracket \forall a \in \text{set } as. \text{divisor } a \neq 0; \forall a \in \text{set } as. \text{hd-coeff } a \neq 0 \rrbracket \implies$

$Z.I ((\text{qe-pres}_1 \circ \text{hd-coeffs1}) \text{ as}) e = (\exists x. \forall a \in \text{set } as. I_Z a (x \# e))$

<proof>

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

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

<proof>

lemma *normal-qe-pres1-o*:

$\forall a \in \text{set } as. \text{hd-coeff } a \neq 0 \wedge \text{divisor } a \neq 0 \implies$

$Z.\text{normal } ((\text{qe-pres}_1 \circ \text{hd-coeffs1}) \text{ as})$

<proof>

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

<proof>

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

<proof>

end

theory *Cooper*

imports *PresArith*

begin

6.5 Cooper

This section formalizes Cooper's algorithm [1].

lemma *set-atoms0-iff*:

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

<proof>

definition

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

lemma *I-hd-coeffs1*:

assumes *qfree* φ

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 _ ($\text{And } \varphi_1 \varphi_2$) = *and* (*inf* _ φ_1) (*inf* _ φ_2) |

inf _ ($\text{Or } \varphi_1 \varphi_2$) = *or* (*inf* _ φ_1) (*inf* _ φ_2) |

inf _ ($\text{Atom}(\text{Le } i \ (k \# ks))$) =

(if $k < 0$ then *TrueF* else if $k > 0$ then *FalseF* else $\text{Atom}(\text{Le } i \ (0 \# ks))$) |

inf _ $\varphi = \varphi$

definition

*qe-cooper*₁ $\varphi =$

(let $as = Z.\text{atoms}_0 \varphi$; $d = \text{zlcms}(\text{map } \text{divisor } as)$; $ls = \text{lbounds } as$

in *or* ($\text{Disj } [0..d - 1] (\lambda n. \text{subst } n \ [] (\text{inf}_- \varphi))$)

($\text{Disj } ls (\lambda(i, ks).$

$\text{Disj } [0..d - 1] (\lambda n. \text{subst } (i + n) (-ks) \varphi))$)

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** $- \Longrightarrow - \Longrightarrow \exists x. ?P f x$)

$\langle \text{proof} \rangle$

lemma *min-inf-repeats*:

ngfree $\varphi \Longrightarrow \forall a \in \text{set}(Z.\text{atoms}_0 \varphi). \text{divisor } a \ \text{dvd } d \Longrightarrow$

$Z.I (\text{inf}_- \varphi) ((x - k*d) \# xs) = Z.I (\text{inf}_- \varphi) (x \# xs)$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma β :

$\llbracket \text{ngfree } \varphi; \forall a \in \text{set}(Z.\text{atoms}_0 \varphi). \text{hd-coeff-is1 } a;$

$\forall a \in \text{set}(Z.\text{atoms}_0 \varphi). \text{divisor } a \ \text{dvd } d; d > 0;$

$\neg(\exists j \in \{0 .. d - 1\}. \exists(i, ks) \in \text{set}(\text{lbounds}(Z.\text{atoms}_0 \varphi)).$

$x = i - \langle ks, xs \rangle + j); Z.I \varphi (x \# xs) \rrbracket$

$\Longrightarrow Z.I \varphi ((x - d) \# xs)$

$\langle proof \rangle$

lemma *periodic-finite-ex*:

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

$\langle proof \rangle$

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

$\implies (\exists x. P(x)) = ((\exists j \in \{0..D - 1\}. P1(j)) \vee (\exists j \in \{0..D - 1\}. \exists b \in B. P(b+j)))$
 $\langle proof \rangle$

theorem *cp-thm*:

assumes $ng: ngfree \varphi$
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 $(\exists x. ?P(x)) = (\exists j \in ?D. ?M j \vee (\exists (i, ks) \in ?B. ?P(?I i ks + j)))$)

$\langle proof \rangle$

lemma *qfree-min-inf[simp]*: $qfree \varphi \implies qfree (inf_ \varphi)$

$\langle proof \rangle$

lemma *I-qe-cooper₁*:

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** $ngfree \varphi$
shows $Z.I (qe-cooper_1 \varphi) xs = (\exists x. Z.I \varphi (x \# xs))$

$\langle proof \rangle$

lemma *divisor-hd-coeff1-neq0*:

$qfree \varphi \implies a \in atoms \varphi \implies divisor a \neq 0 \implies$
 $divisor (hd-coeff1 (zlcms (map hd-coeff (Z.atoms_0 \varphi))) a) \neq 0$

$\langle proof \rangle$

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

$hd-coeff (hd-coeff1 m a) \neq 0 \longrightarrow hd-coeff-is1 (hd-coeff1 m a)$

$\langle proof \rangle$

lemma *I-cooper1-hd-coeffs1*: $Z.normal \varphi \implies ngfree \varphi$

$\implies Z.I (qe-cooper_1 (hd-coeffs1 \varphi)) xs = (\exists x. Z.I \varphi (x \# xs))$

<proof>

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

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

<proof>

lemma $normal\text{-min}\text{-inf}: Z.normal \varphi \implies Z.normal(inf_ \varphi)$

<proof>

lemma $normal\text{-cooper1}: Z.normal \varphi \implies Z.normal(qe\text{-cooper}_1 \varphi)$

<proof>

lemma $normal\text{-hd}\text{-coeffs1}: qfree \varphi \implies Z.normal \varphi \implies Z.normal(hd\text{-coeffs1} \varphi)$

<proof>

theorem $I\text{-cooper}: Z.normal \varphi \implies Z.I (qe\text{-cooper} \varphi) xs = Z.I \varphi xs$

<proof>

theorem $qfree\text{-cooper}: qfree (qe\text{-cooper} \varphi)$

<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.