

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	8
2.1	No Equality	9
2.1.1	DNF-based	9
2.1.2	NNF-based	12
2.2	With equality	13
3	DLO	15
3.1	Basics	15
3.2	DNF-based quantifier elimination	20
3.3	Examples	24
3.4	Interior Point Method	25
3.5	Quantifier elimination with infinitesimals	30

4	Lists as vectors	34
4.1	+ and -	34
4.2	Inner product	36
5	Linear real arithmetic	37
5.1	Basics	37
5.1.1	Syntax and Semantics	37
5.1.2	Shared constructions	39
5.2	Fourier	44
5.2.1	Tests	47
5.2.2	An optimization	47
5.3	Ferrante-Rackoff	49
5.4	Quantifier elimination with infinitesimals	53
6	Presburger arithmetic	59
6.1	Syntax	59
6.2	LCM and lemmas	61
6.3	Setting coefficients to 1 or -1	62
6.4	DNF-based quantifier elimination	65
6.5	Cooper	74

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 $atoms-map-fm[simp]: atoms(map_{fm}\ f\ \varphi) = f\ `atoms\ \varphi$
by(*induct* φ) *auto*

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} $h\ (list-disj\ fs) = list-disj\ (map\ (amap_{fm}\ h)\ fs)$
by(*induct* fs) (*auto simp:list-disj-def or-def*)

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-and[simp]*: $\llbracket qfree \ \varphi_1; qfree \ \varphi_2 \rrbracket \implies qfree(and \ \varphi_1 \ \varphi_2)$
by (*simp add:and-def*)

lemma *qfree-or[simp]*: $\llbracket qfree \ \varphi_1; qfree \ \varphi_2 \rrbracket \implies qfree(or \ \varphi_1 \ \varphi_2)$
by (*simp add:or-def*)

lemma *qfree-neg[simp]*: $qfree(neg \ \varphi) = qfree \ \varphi$
by (*simp add:neg-def*)

lemma *qfree-foldr-Or[simp]*:
 $qfree(foldr \ Or \ fs \ \varphi) = (qfree \ \varphi \wedge (\forall \varphi \in set \ fs. qfree \ \varphi))$
by (*induct fs*) *auto*

lemma *qfree-list-conj[simp]*:
assumes $\forall \varphi \in set \ fs. qfree \ \varphi$ **shows** $qfree(list-conj \ fs)$
proof –
 { **fix** *fs* φ
 have $\llbracket \forall \varphi \in set \ fs. qfree \ \varphi; qfree \ \varphi \rrbracket \implies qfree(foldr \ and \ fs \ \varphi)$
 by (*induct fs*) *auto*
 } **thus** *?thesis* **using** *assms* **by** (*fastsimp simp add:list-conj-def*)
qed

lemma *qfree-list-disj[simp]*:
assumes $\forall \varphi \in set \ fs. qfree \ \varphi$ **shows** $qfree(list-disj \ fs)$
proof –
 { **fix** *fs* φ
 have $\llbracket \forall \varphi \in set \ fs. qfree \ \varphi; qfree \ \varphi \rrbracket \implies qfree(foldr \ or \ fs \ \varphi)$
 by (*induct fs*) *auto*
 } **thus** *?thesis* **using** *assms* **by** (*fastsimp simp add:list-disj-def*)
qed

lemma *qfree-map-fm*: $qfree \ (map_{fm} \ f \ \varphi) = qfree \ \varphi$
by (*induct* φ) *simp-all*

lemma *atoms-list-disjE*:
 $a : atoms(list-disj \ fs) \implies a : (\bigcup \varphi \in set \ fs. atoms \ \varphi)$
apply (*induct fs*)
 apply (*simp add:list-disj-def*)
apply (*auto simp add:list-disj-def Logic.or-def split:split-if-asm*)
done

lemma *atoms-list-conjE*:
 $a : atoms(list-conj \ fs) \implies a : (\bigcup \varphi \in set \ fs. atoms \ \varphi)$
apply (*induct fs*)

```

apply (simp add:list-conj-def)
apply (auto simp add:list-conj-def Logic.and-def split:split-if-asm)
done

```

```

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

```

```

fun nqfree :: 'a fm  $\Rightarrow$  bool where
  nqfree(Atom a) = True |
  nqfree TrueF = True |
  nqfree FalseF = True |
  nqfree (And  $\varphi_1 \varphi_2$ ) = (nqfree  $\varphi_1$   $\wedge$  nqfree  $\varphi_2$ ) |
  nqfree (Or  $\varphi_1 \varphi_2$ ) = (nqfree  $\varphi_1$   $\wedge$  nqfree  $\varphi_2$ ) |
  nqfree  $\varphi$  = False

```

```

lemma nqfree-qfree[simp]: nqfree  $\varphi \Longrightarrow$  qfree  $\varphi$ 
by (induct  $\varphi$ ) simp-all

```

```

lemma nqfree-map-fm: nqfree (map_fm f  $\varphi$ ) = nqfree  $\varphi$ 
by (induct  $\varphi$ ) simp-all

```

```

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

```

```

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

```

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

begin

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

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

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

lemma $ngfree-nnf: qfree \varphi \Longrightarrow ngfree(nnf \varphi)$

by ($induct \varphi$ rule: $nnf.induct$)
 ($simp-all$ add: $ngfree-aneg$ and-def or-def)

lemma $qfree-nnf[simp]: qfree(nnf \varphi) = qfree \varphi$
by ($induct \varphi$ rule: $nnf.induct$)($simp-all$ add: $ngfree-aneg$)

lemma $I-neg[simp]: I (neg \varphi) xs = I (Neg \varphi) xs$
by ($simp$ add: $neg-def$)

lemma $I-and[simp]: I (and \varphi_1 \varphi_2) xs = I (And \varphi_1 \varphi_2) xs$
by ($simp$ add: $and-def$)

lemma $I-list-conj[simp]:$

$I (list-conj fs) xs = (\forall \varphi \in set fs. I \varphi xs)$

proof –

{ **fix** $fs \varphi$
have $I (foldr and fs \varphi) xs = (I \varphi xs \wedge (\forall \varphi \in set fs. I \varphi xs))$
by ($induct fs$) $auto$

} thus ?thesis by(simp add:list-conj-def)
qed

lemma *I-or[simp]*: $I (or \varphi_1 \varphi_2) xs = I (Or \varphi_1 \varphi_2) xs$
by(simp add:or-def)

lemma *I-list-disj[simp]*:
 $I (list-disj fs) xs = (\exists \varphi \in set fs. I \varphi xs)$
proof –
{ **fix** fs φ
 have $I (foldr or fs \varphi) xs = (I \varphi xs \vee (\exists \varphi \in set fs. I \varphi xs))$
 by (induct fs) auto
} thus ?thesis by(simp add:list-disj-def)
qed

lemma *I-nnf*: $I (nnf \varphi) xs = I \varphi xs$
by(induct rule:nnf.induct)(simp-all add: I_a -aneg)

lemma *I-dnf*:
 $ngfree \varphi \implies (\exists as \in set (dnf \varphi). \forall a \in set as. I_a a xs) = I \varphi xs$
by (induct φ) (fastsimp)+

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

lemma *normal-simps[simp]*:
normal TrueF
normal FalseF
normal (Atom a) \longleftrightarrow anormal a
normal (And $\varphi_1 \varphi_2$) \longleftrightarrow normal $\varphi_1 \wedge$ normal φ_2
normal (Or $\varphi_1 \varphi_2$) \longleftrightarrow normal $\varphi_1 \wedge$ normal φ_2
normal (Neg φ) \longleftrightarrow normal φ
normal (ExQ φ) \longleftrightarrow normal φ
by(auto simp:normal-def)

lemma *normal-aneg[simp]*: *anormal a \implies normal (aneg a)*
by (simp add:anormal-aneg normal-def)

lemma *normal-and[simp]*:
normal $\varphi_1 \implies$ normal $\varphi_2 \implies$ normal (and $\varphi_1 \varphi_2$)
by (simp add:Logic.and-def)

lemma *normal-or[simp]*:
normal $\varphi_1 \implies$ normal $\varphi_2 \implies$ normal (or $\varphi_1 \varphi_2$)
by (simp add:Logic.or-def)

lemma *normal-list-disj[simp]*:
 $\forall \varphi \in set fs. normal \varphi \implies normal (list-disj fs)$
apply(induct fs)
apply (simp add:list-disj-def normal-simps)

```

apply (simp add:list-disj-def normal-simps)
done

```

```

lemma normal-nnf: normal  $\varphi \implies$  normal(nnf  $\varphi$ )
by(induct  $\varphi$  rule:nnf.induct) simp-all

```

```

lemma normal-map-fm:
   $\forall a. \text{anormal}(f\ a) = \text{anormal}(a) \implies \text{normal}(\text{map}_{fm}\ f\ \varphi) = \text{normal}\ \varphi$ 
by(induct  $\varphi$ ) auto

```

```

lemma anormal-nnf:
   $qfree\ \varphi \implies \text{normal}\ \varphi \implies \forall a \in \text{atoms}(\text{nnf}\ \varphi). \text{anormal}\ a$ 
apply(induct  $\varphi$  rule:nnf.induct)
apply(unfold normal-def)
apply(simp-all)
apply (blast dest:anormal-aneg)+
done

```

```

lemma atoms-dnf:  $qfree\ \varphi \implies as : \text{set}(\text{dnf}\ \varphi) \implies a : \text{set}\ as \implies a : \text{atoms}\ \varphi$ 
by(induct  $\varphi$  arbitrary: as a rule:qfree.induct)(auto)

```

```

lemma anormal-dnf-nnf:
   $as : \text{set}(\text{dnf}(\text{nnf}\ \varphi)) \implies qfree\ \varphi \implies \text{normal}\ \varphi \implies a : \text{set}\ as \implies \text{anormal}\ a$ 
apply(induct  $\varphi$  arbitrary: a as rule:nnf.induct)
  apply(simp-all add: normal-def)
  apply clarify
  apply (metis UnE set-append)
  apply metis
  apply metis
  apply fastsimp
apply (metis anormal-aneg atoms-dnf qfree-aneg)
done

```

```

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←as. depends0 a];
   indep = [Atom(decr a). a←as, ¬ depends0 a]
   in and qf (list-conj indep))
```

abbreviation *is-dnf-qe* :: ('a list ⇒ 'a fm) ⇒ 'a list ⇒ bool **where**
is-dnf-qe qe as ≡ ∀ xs. I(qe as) xs = (∃ x. ∀ a ∈ 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: ∧ as. (∀ a ∈ set as. depends₀ a) ⇒ *is-dnf-qe* qe as

shows *is-dnf-qe* (qelim qe) as (**is** ∀ xs. ?P xs)

proof

```
fix xs
let ?as0 = filter depends0 as
let ?as1 = filter (Not o depends0) as
have I (qelim qe as) xs =
  (I (qe ?as0) xs ∧ (∀ a ∈ set (map decr ?as1). Ia a xs))
  (is - = (- ∧ ?B)) by (force simp add: qelim-def)
also have ... = ((∃ x. ∀ a ∈ set ?as0. Ia a (x#xs)) ∧ ?B)
  by (simp add: qe not-dep-decr)
also have ... = (∃ x. (∀ a ∈ set ?as0. Ia a (x#xs)) ∧ ?B) by blast
also have ?B = (∀ a ∈ set ?as1. Ia (decr a) xs) by simp
also have (∃ x. (∀ a ∈ set ?as0. Ia a (x#xs)) ∧ ...) =
  (∃ x. (∀ a ∈ set ?as0. Ia a (x#xs)) ∧
   (∀ a ∈ set ?as1. Ia a (x#xs)))
  by (simp add: not-dep-decr)
also have ... = (∃ x. ∀ a ∈ set (?as0 @ ?as1). Ia a (x#xs))
  by (simp add: ball-Un)
also have ... = (∃ x. ∀ a ∈ set (as). Ia a (x#xs))
  by simp blast
finally show ?P xs .
```

qed

The generic DNF-based quantifier elimination procedure:

```
fun lift-dnf-qe :: ('a list ⇒ 'a fm) ⇒ 'a fm ⇒ 'a fm where
lift-dnf-qe qe (And φ1 φ2) = and (lift-dnf-qe qe φ1) (lift-dnf-qe qe φ2) |
lift-dnf-qe qe (Or φ1 φ2) = or (lift-dnf-qe qe φ1) (lift-dnf-qe qe φ2) |
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 φ = φ

lemma *qfree-lift-dnf-qe*: ($\bigwedge as. (\forall a \in set\ as. depends_0\ a) \implies qfree(qe\ as)$)
 $\implies qfree(lift-dnf-qe\ qe\ \varphi)$
by (*induct* φ) (*simp-all* *add:qelim-def*)

lemma *qfree-lift-dnf-qe2*: $qe : lists\ depends_0 \rightarrow qfree$
 $\implies qfree(lift-dnf-qe\ qe\ \varphi)$
using *in-lists-conv-set*[**where** $?'a = 'a$]
by (*simp* *add:Pi-def* *qfree-lift-dnf-qe* *mem-def* *Collect-def*)

lemma *lem*: $\forall P\ A. (\exists x \in A. \exists y. P\ x\ y) = (\exists y. \exists x \in A. P\ x\ y)$ **by** *blast*

lemma *I-lift-dnf-qe*:
assumes $\bigwedge as. (\forall a \in set\ as. depends_0\ a) \implies qfree(qe\ as)$
and $\bigwedge as. (\forall a \in set\ as. depends_0\ a) \implies is-dnf-qe\ qe\ as$
shows I ($lift-dnf-qe\ qe\ \varphi$) $xs = I\ \varphi\ xs$
proof (*induct* φ *arbitrary:xs*)
case ExQ **thus** $?case$
by (*simp* *add:assms* *I-qelim* *lem* *I-dnf* *qfree-nnf* *qfree-lift-dnf-qe*
I-nnf)
qed *simp-all*

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 ($lift-dnf-qe\ qe\ \varphi$) $xs = I\ \varphi\ xs$
using *assms* *in-lists-conv-set*[**where** $?'a = 'a$]
by (*simp* *add:Pi-def* *I-lift-dnf-qe* *mem-def* *Collect-def*)

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 ($qelim\ qe\ as$) $xs = (\exists x. \forall a \in set\ as. I_a\ a\ (x\#\#xs))$
proof –
let $?as0 = filter\ depends_0\ as$
let $?as1 = filter\ (Not\ o\ depends_0)\ as$
have I ($qelim\ qe\ as$) $xs =$
 $(I\ (qe\ ?as0)\ xs \wedge (\forall a \in set\ (map\ decr\ ?as1). I_a\ a\ xs))$
(is $- = (- \wedge ?B)$ **)** **by** (*force* *simp* *add:qelim-def*)
also **have** $\dots = ((\exists x. \forall a \in set\ ?as0. I_a\ a\ (x\#\#xs)) \wedge ?B)$
by (*simp* *add:qe* *nm* *not-dep-decr*)
also **have** $\dots = (\exists x. (\forall a \in set\ ?as0. I_a\ a\ (x\#\#xs)) \wedge ?B)$ **by** *blast*
also **have** $?B = (\forall a \in set\ ?as1. I_a\ (decr\ a)\ xs)$ **by** *simp*
also **have** $(\exists x. (\forall a \in set\ ?as0. I_a\ a\ (x\#\#xs)) \wedge \dots) =$
 $(\exists x. (\forall a \in set\ ?as0. I_a\ a\ (x\#\#xs)) \wedge$
 $(\forall a \in set\ ?as1. I_a\ a\ (x\#\#xs)))$

```

  by (simp add: not-dep-decr)
  also have ... = ( $\exists x. \forall a \in \text{set}(?as0 @ ?as1). I_a a (x\#xs)$ )
  by (simp add: ball-Un)
  also have ... = ( $\exists x. \forall a \in \text{set}(as). I_a a (x\#xs)$ )
  by simp blast
  finally show ?thesis .
qed

```

```

declare [[simp-depth-limit = 5]]

```

```

lemma anormal-atoms-qelim:

```

```

  ( $\bigwedge as. \forall a \in \text{set } as. \text{depends}_0 a \wedge \text{anormal } a \implies \text{normal}(qe as)$ )  $\implies$ 
   $\forall a \in \text{set } as. \text{anormal } a \implies a : \text{atoms}(qelim qe as) \implies \text{anormal } a$ 
  apply (auto simp add: qelim-def and-def normal-def split: split-if-asm)
  apply (auto simp add: anormal-decr dest!: atoms-list-conjE)
  apply (erule-tac x = filter depends0 as in meta-allE)
  apply (simp)
  apply (erule-tac x = filter depends0 as in meta-allE)
  apply (simp)
done

```

```

lemma normal-lift-dnf-qe:

```

```

  assumes  $\bigwedge as. \forall a \in \text{set } as. \text{depends}_0 a \implies \text{qfree}(qe as)$ 
  and  $\bigwedge as. \forall a \in \text{set } as. \text{depends}_0 a \wedge \text{anormal } a \implies \text{normal}(qe as)$ 
  shows  $\text{normal } \varphi \implies \text{normal}(\text{lift-dnf-qe } qe \varphi)$ 
  proof (simp add: normal-def, induct  $\varphi$ )
  case ExQ thus ?case
  apply (auto dest!: atoms-list-disjE)
  apply (rule anormal-atoms-qelim)
  prefer 3 apply assumption
  apply (simp add: assms)
  apply (simp add: normal-def qfree-lift-dnf-qe anormal-dnf-nnf assms)
  done
qed (simp-all add: and-def or-def neg-def Ball-def)

```

```

declare [[simp-depth-limit = 9]]

```

```

lemma I-lift-dnf-qe-anormal:

```

```

  assumes  $\bigwedge as. \forall a \in \text{set } as. \text{depends}_0 a \implies \text{qfree}(qe as)$ 
  and  $\bigwedge as. \forall a \in \text{set } as. \text{depends}_0 a \wedge \text{anormal } a \implies \text{normal}(qe as)$ 
  and  $\bigwedge xs as. \forall a \in \text{set } as. \text{depends}_0 a \wedge \text{anormal } a \implies \text{is-dnf-qe } qe as$ 
  shows  $\text{normal } f \implies I (\text{lift-dnf-qe } qe f) xs = I f xs$ 
  proof (induct f arbitrary: xs)
  case ExQ thus ?case using normal-lift-dnf-qe[of qe]
  by (simp add: assms[simplified normal-def] anormal-dnf-nnf I-qelim-anormal
  lem I-dnf nqfree-nnf qfree-lift-dnf-qe I-nnf normal-def)
qed (simp-all add: normal-def)
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 $normal\ f \implies I\ (lift-dnf-qe\ qe\ f)\ xs = I\ f\ xs$
using $assms\ in-lists-conv-set[where\ ?'a = 'a]$
by ($simp\ add:Pi-def\ mem-def\ I-lift-dnf-qe-anormal\ Int-def\ Collect-def$)

2.1.2 NNF-based

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

lemma $qfree-lift-nnf-qe: (\bigwedge\varphi. nqfree\ \varphi \implies qfree(qe\ \varphi))$
 $\implies qfree(lift-nnf-qe\ qe\ \varphi)$
by ($induct\ \varphi$) ($simp-all\ add:nqfree-nnf$)

lemma $qfree-lift-nnf-qe2:$
 $qe : nqfree \rightarrow qfree \implies qfree(lift-nnf-qe\ qe\ \varphi)$
by ($simp\ add:Pi-def\ qfree-lift-nnf-qe\ mem-def\ Collect-def$)

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 $I\ (lift-nnf-qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
proof ($induct\ \varphi\ arbitrary:xs$)
case ExQ **thus** $?case$
by ($simp\ add: assms\ nqfree-nnf\ qfree-lift-nnf-qe\ I-nnf$)
qed $simp-all$

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 $I\ (lift-nnf-qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
using $assms\ by(simp\ add:Pi-def\ I-lift-nnf-qe\ mem-def\ Collect-def)$

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 $normal\ \varphi \implies normal(lift-nnf-qe\ qe\ \varphi)$
by ($induct\ \varphi$)
 $(simp-all\ add: assms\ Logic.neg-def\ normal-nnf\ nqfree-nnf\ qfree-lift-nnf-qe)$

lemma $I-lift-nnf-qe-normal:$
assumes $\bigwedge\varphi. nqfree\ \varphi \implies qfree(qe\ \varphi)$

```

and  $\bigwedge \varphi. \text{ngfree } \varphi \implies \text{normal } \varphi \implies \text{normal}(qe \ \varphi)$ 
and  $\bigwedge xs \ \varphi. \text{normal } \varphi \implies \text{ngfree } \varphi \implies I \ (qe \ \varphi) \ xs = (\exists x. I \ \varphi \ (x\#xs))$ 
shows  $\text{normal } \varphi \implies I \ (\text{lift-nnf-qe } qe \ \varphi) \ xs = I \ \varphi \ xs$ 
proof(induct  $\varphi$  arbitrary:xs)
  case ExQ thus ?case
    by (simp add: assms ngfree-nnf qfree-lift-nnf-qe I-nnf
        normal-lift-nnf-qe normal-nnf)
qed auto

lemma I-lift-nnf-qe-normal2:
assumes  $qe : \text{ngfree} \rightarrow \text{qfree}$ 
and  $qe : \text{ngfree} \cap \text{normal} \rightarrow \text{normal}$ 
and  $ALL \ \varphi : \text{normal } Int \ \text{ngfree}. ALL \ xs. I \ (qe \ \varphi) \ xs = (\exists x. I \ \varphi \ (x\#xs))$ 
shows  $\text{normal } \varphi \implies I \ (\text{lift-nnf-qe } qe \ \varphi) \ xs = I \ \varphi \ xs$ 
using assms by (simp add: Pi-def I-lift-nnf-qe-normal mem-def Int-def Collect-def)

end

```

2.2 With equality

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

```

locale ATOM-EQ = ATOM +
fixes  $\text{solvable}_0 :: 'a \Rightarrow \text{bool}$ 
and  $\text{trivial} :: 'a \Rightarrow \text{bool}$ 
and  $\text{subst}_0 :: 'a \Rightarrow 'a \Rightarrow 'a$ 
assumes subst_0:
  [  $\text{solvable}_0 \ eq; \neg \text{trivial } eq; I_a \ eq \ (x\#xs); \text{depends}_0 \ a$  ]
   $\implies I_a \ (\text{subst}_0 \ eq \ a) \ xs = I_a \ a \ (x\#xs)$ 
and trivial:  $\text{trivial } eq \implies I_a \ eq \ xs$ 
and solvable:  $\text{solvable}_0 \ eq \implies \exists x. I_a \ eq \ (x\#xs)$ 
and is-triv-self-subst:  $\text{solvable}_0 \ eq \implies \text{trivial} \ (\text{subst}_0 \ eq \ eq)$ 

```

begin

```

definition lift-eq-qe :: ('a list  $\Rightarrow$  'a fm)  $\Rightarrow$  'a list  $\Rightarrow$  'a fm where
lift-eq-qe  $qe \ as =$ 
  (let  $as = [a \leftarrow as. \neg \text{trivial } a]$ 
   in case  $[a \leftarrow as. \text{solvable}_0 \ a]$  of
    []  $\Rightarrow qe \ as$ 
  |  $eq \ \# \ eqs \ \Rightarrow$ 
    (let  $ineqs = [a \leftarrow as. \neg \text{solvable}_0 \ a]$ 
     in list-conj (map (Atom  $\circ$  (subst_0  $eq$ )) (eqs @ ineqs))))

```

```

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

```

```

(is ?L = ?R)
proof -
let ?as = [a←as. ¬ trivial a]
show ?thesis
proof (cases [a←?as. solvable0 a])
case Nil
hence ∀ a∈set as. ¬ trivial a ⟶ ¬ solvable0 a
by(auto simp: filter-empty-conv)
thus ?L = ?R
by(simp add:lift-eq-qe-def dep qe cong:conj-cong) (metis trivial)
next
case (Cons eq -)
then have eq : set as solvable0 eq ¬ trivial eq
by(auto simp: filter-eq-Cons-iff)
then obtain e where Ia eq (e#xs) by(metis solvable)
have ∀ a ∈ set as. Ia a (e # xs) = Ia (subst0 eq a) xs
by(simp add: subst0[OF ‹solvable0 eq› ‹¬ trivial eq› ‹Ia eq (e#xs)›] dep)
thus ?thesis using Cons dep
apply(simp add: lift-eq-qe-def,
clarsimp simp: filter-eq-Cons-iff ball-Un)
apply(rule iffI)
apply(fastsimp intro!:exI[of - e] simp: trivial is-triv-self-subst)
apply (metis subst0)
done
qed
qed

```

definition $lift-dnfeq-qe = lift-dnf-qe \circ lift-eq-qe$

lemma $qfree-lift-eq-qe$:

$(\bigwedge as. \forall a \in set\ as. depends_0\ a \implies qfree\ (qe\ as)) \implies$
 $\forall a \in set\ as. depends_0\ a \implies qfree(lift-eq-qe\ qe\ as)$

by(simp add:lift-eq-qe-def ball-Un split:list.split)

lemma $qfree-lift-dnfeq-qe$: $(\bigwedge as. (\forall a \in set\ as. depends_0\ a) \implies qfree(qe\ as))$
 $\implies qfree(lift-dnfeq-qe\ qe\ \varphi)$

by(simp add: lift-dnfeq-qe-def qfree-lift-dnf-qe qfree-lift-eq-qe)

lemma $I-lift-dnfeq-qe$:

$(\bigwedge as. (\forall a \in set\ as. depends_0\ a) \implies qfree(qe\ as)) \implies$
 $(\bigwedge as. (\forall a \in set\ as. depends_0\ a \wedge \neg solvable_0\ a) \implies is-dnf-qe\ qe\ as) \implies$
 $I\ (lift-dnfeq-qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$

by(simp add:lift-dnfeq-qe-def I-lift-dnf-qe qfree-lift-eq-qe I-lift-eq-qe)

lemma $I-lift-dnfeq-qe2$:

$qe : lists\ depends_0 \rightarrow qfree \implies$
 $(\forall as \in lists\ (depends_0 \cap \neg solvable_0). is-dnf-qe\ qe\ as) \implies$
 $I\ (lift-dnfeq-qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$

using $in-lists-conv-set$ [**where** ?'a = 'a]

by(*simp add:Pi-def I-lift-dnfeq-qe mem-def Int-def Compl-eq Collect-def*)

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

fix *r s* :: *real*
let $?v = (r + s) / 2$
assume $r < s$
hence $r < ?v \wedge ?v < s$ by *simp*
thus $\exists v. r < v \wedge v < s$..

next

fix *r* :: *real*
have $r < r + 1$ by *arith*
thus $\exists s. r < s$..

next

fix *r* :: *real*
have $r - 1 < r$ by *arith*
thus $\exists s. s < r$..

qed

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* = $(\exists i j. a = \text{Less } i j)$

by(*cases a*) *auto*

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

```

by(cases a) auto
lemma not-is-Eq-iff:  $(\forall i j. a \neq \text{Eq } i j) = (\exists i j. a = \text{Less } i j)$ 
by(cases a) auto

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]: lift-nnf-qe = ATOM.lift-nnf-qe negdlo

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 negdlo ( $\lambda a. \text{True}$ ) Idlo dependsdlo decrdlo
apply(unfold-locales)
apply(case-tac a)
apply simp-all
apply(case-tac a)
apply (simp-all add:linorder-class.not-less-iff-gr-or-eq
  linorder-not-less linorder-neq-iff)
apply(case-tac a)
apply(simp-all add:nth-Cons)
done

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

setup  $\ll$  Sign.revert-abbrev @{const-name DLO.I}  $\gg$ 

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

by(*auto simp: lbounds-def split:nat.splits atom.splits*)

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

by(*auto simp: ubounds-def split:nat.splits atom.splits*)

lemma *set-ebounds*:

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

by(*auto simp: ebounds-def split: atom.splits nat.splits*)

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

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

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

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

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

by(*auto simp:EQ-def*)

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

by(*auto simp:EQ-def*)

lemma *EQ-conv-set-ebounds*:

$x \in EQ\ f\ xs = (\exists e \in set(ebounds(DLO.atoms_0\ f)). x = xs!e)$

by(*auto simp: EQ-def set-ebounds*)

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*:

$qfree\ f \Longrightarrow DLO.I\ (subst\ f\ k)\ xs = DLO.I\ f\ (xs!k\ \# xs)$

apply(*induct f*)

apply(*simp-all*)

apply(*case-tac a*)

apply(*simp-all add:nth.simps split:nat.splits*)

done

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) |
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
aplug-inf (Less 0 -) = FalseF |
aplug-inf (Less - 0) = TrueF |
aplug-inf (Less (Suc i) (Suc j)) = Atom(Less i j) |
aplug-inf (Eq 0 0) = TrueF |
aplug-inf (Eq 0 -) = FalseF |
aplug-inf (Eq - 0) = FalseF |
aplug-inf (Eq (Suc i) (Suc j)) = Atom(Eq i j)

```

abbreviation *plus-inf* :: atom fm \Rightarrow atom fm (*inf*₊) **where**
inf₊ \equiv amap_{fm} *aplug-inf*

lemma *min-inf*:

```

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

```

proof(*induct f*)

case (*Atom a*)

show *?case*

proof (*cases a rule: amin-inf.cases*)

case 1 thus *?thesis* **by**(*auto simp add:nth-Cons' linorder-not-less*)

next

case 2 thus *?thesis*

by (*simp*) (*metis no-lb linorder-not-less order-less-le-trans*)

next

case 5 thus *?thesis*

by(*simp add:nth-Cons'*) (*metis no-lb linorder-not-less*)

next

case 6 thus *?thesis* **by** *simp* (*metis no-lb linorder-not-less*)

qed *simp-all*

next

case (*And f1 f2*)

then obtain *x1 x2* **where** *?P f1 x1 ?P f2 x2* **by** *fastsimp+*

hence *?P (And f1 f2) (min x1 x2)* **by**(*force simp:and-def*)

thus *?case ..*

next

case (*Or f1 f2*)

then obtain *x1 x2* **where** *?P f1 x1 ?P f2 x2* **by** *fastsimp+*

hence *?P (Or f1 f2) (min x1 x2)* **by**(*force simp:or-def*)

thus *?case ..*

qed *simp-all*

```

lemma plus-inf:
  nqfree f  $\implies \exists x. \forall y \geq x. DLO.I (inf_+ f) xs = DLO.I f (y \# xs)$ 
  (is -  $\implies \exists x. ?P f x$ )
proof(induct f)
  case (Atom a)
  show ?case
  proof (cases a rule: aplus-inf.cases)
    case 1 thus ?thesis
    by(simp add:nth-Cons')
      (metis no-ub linorder-not-less antisym order-less-trans)
  next
    case 2 thus ?thesis
    by (simp) (metis no-ub linorder-not-less order-less-le-trans)
  next
    case 5 thus ?thesis
    by(simp add:nth-Cons') (metis no-ub linorder-not-less)
  next
    case 6 thus ?thesis by simp (metis no-ub linorder-not-less)
  qed simp-all
next
  case (And f1 f2)
  then obtain x1 x2 where ?P f1 x1 ?P f2 x2 by fastsimp+
  hence ?P (And f1 f2) (max x1 x2) by(force simp:and-def)
  thus ?case ..
next
  case (Or f1 f2)
  then obtain x1 x2 where ?P f1 x1 ?P f2 x2 by fastsimp+
  hence ?P (Or f1 f2) (max x1 x2) by(force simp:or-def)
  thus ?case ..
qed simp-all

```

```

declare[[simp-depth-limit=2]]
lemma LBex:
  [[ nqfree 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(induct f)
  case (Atom a) thus ?case
  by (cases a rule: amin-inf.cases)
    (simp-all add: nth.simps EQ-def split: nat.splits)
qed auto

```

```

lemma UBex:
  [[ nqfree f; DLO.I f (x#xs);  $\neg DLO.I (inf_+ f) xs$ ;  $x \notin EQ f xs$  ]
   $\implies \exists u \in UB f xs. x < u$ 
proof(induct f)
  case (Atom a) thus ?case

```

```

    by (cases a rule: aplus-inf.cases)
      (simp-all add: nth.simps EQ-def split: nat.splits)
qed auto
declare[[simp-depth-limit=50]]

```

```

lemma finite-LB: finite(LB f xs)
proof -
  have LB f xs = (λk. xs!k) ‘ set(lbounds(DLO.atoms0 f))
    by (auto simp:set-lbounds image-def)
  thus ?thesis by simp
qed

```

```

lemma finite-UB: finite(UB f xs)
proof -
  have UB f xs = (λk. xs!k) ‘ set(ubounds(DLO.atoms0 f))
    by (auto simp:set-ubounds image-def)
  thus ?thesis by simp
qed

```

```

lemma qfree-amin-inf: qfree (amin-inf a)
by(cases a rule:amin-inf.cases) simp-all

```

```

lemma qfree-min-inf: nqfree φ ⇒ qfree(inf- φ)
by(induct φ)(simp-all add:qfree-amin-inf)

```

```

lemma qfree-aplus-inf: qfree (aplus-inf a)
by(cases a rule:aplus-inf.cases) simp-all

```

```

lemma qfree-plus-inf: nqfree φ ⇒ qfree(inf+ φ)
by(induct φ)(simp-all add:qfree-aplus-inf)

```

end

```

theory QEdlo
imports DLO ~~/src/HOL/ex/Reflection
begin

```

3.2 DNF-based quantifier elimination

```

definition qe-dlo1 :: atom list ⇒ atom fm where
qe-dlo1 as =
  (if Less 0 0 ∈ set as then FalseF else
   let lbs = [i. Less (Suc i) 0 ← as]; ubs = [j. Less 0 (Suc j) ← as];
       pairs = [Atom(Less i j). i ← lbs, j ← ubs]
   in list-conj pairs)

```

theorem *I-qe-dlo₁*:

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

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

(*is ?L = ?R*)

proof

let *?lbs* = [*i. Less (Suc i) 0* \leftarrow *as*]

let *?ubs* = [*j. Less 0 (Suc j)* \leftarrow *as*]

let *?Ls* = *set ?lbs* **let** *?Us* = *set ?ubs*

let *?lb* = *Max* ($\bigcup x \in ?Ls. \{xs!x\}$)

let *?ub* = *Min* ($\bigcup x \in ?Us. \{xs!x\}$)

have *?*: *Less 0 0* \notin *set as* $\implies \forall a \in \text{set } as.$

($\exists i \in ?Ls. a = \text{Less (Suc } i) 0$) \vee ($\exists i \in ?Us. a = \text{Less } 0 (\text{Suc } i)$)

proof

fix *a* **assume** *Less 0 0* \notin *set as* $a \in \text{set } as$

then obtain *i j* **where** [*simp*]: $a = \text{Less } i j$

using *less* **by** (*force simp:is-Less-iff*)

with dep **obtain** *k* **where** $i = 0 \wedge j = \text{Suc } k \vee i = \text{Suc } k \wedge j = 0$

using (*Less 0 0* \notin *set as*) $\langle a \in \text{set } as \rangle$

by auto (*metis Nat.nat.nchotomy depends_{dlo}.simps(2)*)

moreover **hence** $i=0 \wedge k \in ?Us \vee j=0 \wedge k \in ?Ls$

using $\langle a \in \text{set } as \rangle$ **by force**

ultimately show ($\exists i \in ?Ls. a = \text{Less (Suc } i) 0$) \vee ($\exists i \in ?Us. a = \text{Less } 0 (\text{Suc } i)$)

by force

qed

assume *qe1*: *?L*

hence *0*: *Less 0 0* \notin *set as* **by** (*auto simp:qe-dlo₁-def*)

with qe1 **have** *1*: $\forall x \in ?Ls. \forall y \in ?Us. xs ! x < xs ! y$

by (*fastsimp simp:qe-dlo₁-def*)

{ **fix** *i x*

assume $\text{Less } i 0 \in \text{set } as \mid \text{Less } 0 i \in \text{set } as$

moreover **hence** $i \neq 0$ **using** *0* **by iprover**

ultimately have $(x \# xs) ! i = xs!(i - 1)$ **by** (*simp add: nth-Cons'*)

} **note** *this*[*simp*]

{ **assume** *nonempty*: $?Ls \neq \{\} \wedge ?Us \neq \{\}$

hence *Max* ($\bigcup x \in ?Ls. \{xs!x\}$) $<$ *Min* ($\bigcup x \in ?Us. \{xs!x\}$)

using *1* **by**(*simp*)

then obtain *m* **where** $?lb < m \wedge m < ?ub$ **using** *dense* **by** *blast*

hence $\forall i \in ?Ls. xs!i < m$ **and** $\forall j \in ?Us. m < xs!j$

using *nonempty* **by auto**

hence $\forall a \in \text{set } as. I_{dlo} a (m \# xs)$ **using** *?*[*OF 0*] **by**(*auto simp:less*)

hence *?R ..* }

moreover

{ **assume** *asm*: $?Ls \neq \{\} \wedge ?Us = \{\}$

then obtain *m* **where** $?lb < m$ **using** *no-ub* **by** *blast*

hence $\forall a \in \text{set } as. I_{dlo} a (m \# xs)$ **using** *?*[*OF 0*] *asm* **by auto**

hence *?R ..* }

moreover

{ **assume** *asm*: $?Ls = \{\} \wedge ?Us \neq \{\}$

then obtain *m* **where** $m < ?ub$ **using** *no-lb* **by** *blast*

```

    hence  $\forall a \in \text{set as. } I_{dlo} a (m \# xs)$  using  $\mathcal{Q}[OF\ 0]$  asm by auto
    hence ?R .. }
  moreover
  { assume ?Ls = {}  $\wedge$  ?Us = {}
    hence ?R using  $\mathcal{Q}[OF\ 0]$  by (auto simp add:less)
  }
  ultimately show ?R by blast
next
assume ?R
then obtain x where 1:  $\forall a \in \text{set as. } I_{dlo} a (x \# xs)$  ..
hence 0: Less 0 0  $\notin$  set as by auto
{ fix i j
  assume asm: Less i 0  $\in$  set as Less 0 j  $\in$  set as
  hence  $(x \# xs)!i < x < (x \# xs)!j$  using 1 by auto+
  hence  $(x \# xs)!i < (x \# xs)!j$  by(rule order-less-trans)
  moreover have  $\neg(i = 0 \mid j = 0)$  using 0 asm by blast
  ultimately have  $xs ! (i - 1) < xs ! (j - 1)$  by (simp add: nth-Cons')
}
thus ?L using 0 less
by (fastsimp simp: qe-dlo1-def is-Less-iff split:atom.splits nat.splits)
qed

```

lemma *I-qe-dlo₁-pretty*:

```

 $\forall a \in \text{set as. is-Less } a \wedge \text{depends}_{dlo} a \implies DLO.\text{is-dnf-qe} - \text{qe-dlo}_1$  as
by(metis I-qe-dlo1)

```

definition *subst* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**

```

subst i j k = (if k=0 then if i=0 then j else i else k) - 1

```

fun *subst₀* :: $\text{atom} \Rightarrow \text{atom} \Rightarrow \text{atom}$ **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)
by auto

```

interpretation *DLO_e!*:

```

ATOM-EQ negdlo ( $\lambda a. \text{True}$ ) Idlo dependsdlo decrdlo
  ( $\lambda \text{Eq } i j \Rightarrow i=0 \vee j=0 \mid a \Rightarrow \text{False}$ )
  ( $\lambda \text{Eq } i j \Rightarrow i=j \mid a \Rightarrow \text{False}$ ) subst0
apply(unfold-locales)
apply(fastsimp simp:subst-def nth-Cons' split:atom.splits split-if-asm)
apply(simp add:subst-def split:atom.splits)
apply(fastsimp simp:subst-def nth-Cons' split:atom.splits)
apply(fastsimp simp add:subst-def split:atom.splits)
done

```

setup \ll *Sign.revert-abbrev* @{const-name *DLO_e.lift-dnfeq-qe*} \gg

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

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

by(*auto simp:qe-dlo₁-def intro!:qfree-list-conj*)

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

unfolding *qe-dlo-def*

by(*fastsimp intro!: I-qe-dlo₁ qfree-qe-dlo₁ DLO_e.I-lift-dnfeq-qe*

simp: is-Less-iff not-is-Eq-iff split:atom.splits cong:conj-cong)

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

by(*simp add:qe-dlo-def DLO_e.qfree-lift-dnfeq-qe qfree-qe-dlo₁*)

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*)

by(*simp-all add:interpret-def*)

lemma *interpret-others*:

interpret (*Neg*(*ExQ* (*Neg* *f*))) *xs* = (\forall *x*. *interpret* *f* (*x*#*xs*))

interpret (*Or* (*Neg* *f1*) *f2*) *xs* = (*interpret* *f1* *xs* \longrightarrow *interpret* *f2* *xs*)

by(*simp-all add:interpret-def*)

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*

by(*simp add:I-qe-dlo interpret-def*)

method-setup *reify* = \ll

Attrib.thms --

Scan.option (*Scan.lift* (*Args. \$\$\$* ()) |-- *Args.term* --| *Scan.lift* (*Args. \$\$\$*)))

\gg

```

(fn (eqs, to) => fn ctxt =>
  Method.SIMPLE-METHOD' (Reflection.genreify-tac ctxt (eqs @ (fst (Reify-Data.get
  ctxt)))) to
  THEN' simp-tac (HOL-basic-ss addsimps [@{thminterpret-def}])))
>> dlo reification

```

```

declare Idlo.simps(2)[code del]
declare Logic.interpret.simps[code del]
declare Logic.interpret.simps(1-2)[code]

```

3.3 Examples

```

lemma  $\forall x::real. \neg x < x$ 
apply reify
apply (subst I-qe-dlo[symmetric])
by eval

```

```

lemma  $\forall x y::real. \exists z. x < y \longrightarrow x < z \ \& \ z < y$ 
apply reify
apply (subst I-qe-dlo[symmetric])
by eval

```

```

lemma  $\forall x::real. \neg x < x$ 
apply reify
apply (subst I-qe-dlo[symmetric])
by eval

```

```

lemma  $\forall x y::real. \exists z. x < y \longrightarrow x < z \ \& \ z < y$ 
apply reify
apply (subst I-qe-dlo[symmetric])
by eval

```

```

lemma  $\neg(\exists x y z. \forall u::real. x < x \mid \neg x < u \mid x < y \ \& \ y < z \ \& \ \neg x < z)$ 
apply reify
apply (subst I-qe-dlo[symmetric])
by eval

```

```

lemma qe-dlo(AllQ (Imp (Atom(Less 0 1)) (Atom(Less 1 0)))) = FalseF
by eval

```

```

lemma qe-dlo(AllQ(AllQ (Imp (Atom(Less 0 1)) (Atom(Less 0 1)))) = TrueF
by eval

```

```

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

```

by *eval*

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

by *eval*

lemma

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

by *eval*

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

by *eval*

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 *asubst₂* :: *nat* ⇒ *nat* ⇒ *atom* ⇒ *atom fm* **where**

asubst₂ *l u* (*Less* 0 0) = *FalseF* |

asubst₂ *l u* (*Less* 0 (*Suc* *j*)) = *Or* (*Atom*(*Less* *u j*)) (*Atom*(*Eq* *u j*)) |

asubst₂ *l u* (*Less* (*Suc* *i*) 0) = *Or* (*Atom*(*Less* *i l*)) (*Atom*(*Eq* *i l*)) |

asubst₂ *l u* (*Less* (*Suc* *i*) (*Suc* *j*)) = *Atom*(*Less* *i j*) |

asubst₂ *l u* (*Eq* 0 0) = *TrueF* |

asubst₂ *l u* (*Eq* 0 -) = *FalseF* |

asubst₂ *l u* (*Eq* - 0) = *FalseF* |

asubst₂ *l u* (*Eq* (*Suc* *i*) (*Suc* *j*)) = *Atom*(*Eq* *i j*)

abbreviation *subst₂* *l u* ≡ *amap_{fm}* (*asubst₂* *l u*)

lemma *I-subst₂1*:

nqfree *f* ⇒ *xs!l* < *xs!u* ⇒ *DLO.I* (*subst₂* *l u f*) *xs*

⇒ *xs!l* < *x* ⇒ *x* < *xs!u* ⇒ *DLO.I f* (*x#xs*)

proof(*induct* *f* *arbitrary*: *x*)

case (*Atom* *a*) **thus** ?*case*

by (cases (l,u,a) rule: asubst2.cases) auto
qed auto

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]:

$nolub\ (And\ f\ g)\ xs\ l\ x\ u = (nolub\ f\ xs\ l\ x\ u \wedge nolub\ g\ xs\ l\ x\ u)$
by(auto simp:nolub-def)

lemma nolub-Or[simp]:

$nolub\ (Or\ f\ g)\ xs\ l\ x\ u = (nolub\ f\ xs\ l\ x\ u \wedge nolub\ g\ xs\ l\ x\ u)$
by(auto simp:nolub-def)

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

lemma innermost-intvl:

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

proof(induct f)

case (Atom a)

show ?thesis

proof (cases a)

case (Less i j)

then show ?thesis using Atom

unfolding nolub-def

by (clarsimp simp: nth.simps Ball-def split:split-if-asm nat.splits)
(metis not-leE order-antisym order-less-trans)+

next

case (Eq i j)[simp]

show ?thesis

proof (cases i)

case 0[simp]

show ?thesis

proof (cases j)

case 0 thus ?thesis using Atom by simp

next

case Suc thus ?thesis using Atom by(simp add:EQ-def)

qed

next

case Suc[simp]

show ?thesis

proof (cases j)

case 0 thus ?thesis using Atom by(simp add:EQ-def)

next

case Suc thus ?thesis using Atom by simp

qed

qed

qed

```

next
  case (And f1 f2) thus ?case by (fastsimp)
next
  case (Or f1 f2) thus ?case by (fastsimp)
qed simp+

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 (subst2 l u f) xs
proof (induct f)
  case (Atom a) show ?case
    apply (cases (l,u,a) rule: asubst2.cases)
    apply (insert Atom, auto simp: EQ-def nolub-def split:split-if-asm)
    done
next
  case Or thus ?case by (simp add: Ball-def)(metis innermost-intvl)
qed auto
declare[[simp-depth-limit=50]]

```

definition

```

qe-interior1  $\varphi$  =
(let as = DLO.atoms0  $\varphi$ ; lbs = lbounds as; ub = ubounds as; ebs = ebounds as;
  intrs = [And (Atom(Less l u)) (subst2 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 : 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)

```

proof –

```

let ?L = {l:L. l < x} let ?U = {u:U. x < u}
let ?ll = Max ?L let ?uu = Min ?U
have ?L  $\neq$  {} using (l : L) (l < x) by (blast intro:order-less-imp-le)
moreover have ?U  $\neq$  {} using (u:U) (x < u) by (blast intro:order-less-imp-le)
ultimately have  $\forall y$ . ?ll < y  $\wedge$  y < x  $\longrightarrow$  y  $\notin$  L  $\forall y$ . x < y  $\wedge$  y < ?uu  $\longrightarrow$  y  $\notin$  U
  using (finite L) (finite U) by force+
moreover have ?ll : L

```

proof

```

  show ?ll : ?L using (finite L) Max-in[OF - (?L  $\neq$  {})] by simp
  show ?L  $\subseteq$  L by blast

```

qed

```

moreover have ?uu : U

```

proof

```

  show ?uu : ?U using (finite U) Min-in[OF - (?U  $\neq$  {})] by simp
  show ?U  $\subseteq$  U by blast

```

qed

moreover have $?ll < x$ **using** $\langle \text{finite } L \rangle \langle ?L \neq \{\} \rangle$ **by** *simp*
moreover have $x < ?uu$ **using** $\langle \text{finite } U \rangle \langle ?U \neq \{\} \rangle$ **by** *simp*
moreover have $?ll < ?uu$ **using** $\langle ?ll < x \rangle \langle x < ?uu \rangle$ **by** *simp*
ultimately show $?thesis$ **using** $\langle l < x \rangle \langle x < u \rangle \langle ?L \neq \{\} \rangle \langle ?U \neq \{\} \rangle$
by(*blast intro! : dense greaterThanLessThan-iff [THEN iffD1]*)
qed

theorem *I-interior1*:

assumes $ngfree \varphi$ **shows** $DLO.I (qe\text{-interior}_1 \varphi) xs = (EX x. DLO.I \varphi (x\#xs))$
(is $?QE = ?EX$ **)**

proof

assume $?QE$

{ **assume** $DLO.I (inf_- \varphi) xs$

hence $?EX$ **using** $\langle ?QE \rangle \text{min-inf}[of \varphi xs] \langle ngfree \varphi \rangle$

by(*auto simp add:qe-interior1-def amap-fm-list-disj*)

} **moreover**

{ **assume** $DLO.I (inf_+ \varphi) xs$

hence $?EX$ **using** $\langle ?QE \rangle \text{plus-inf}[of \varphi xs] \langle ngfree \varphi \rangle$

by(*auto simp add:qe-interior1-def amap-fm-list-disj*)

} **moreover**

{ **assume** $\neg DLO.I (inf_- \varphi) xs \wedge \neg DLO.I (inf_+ \varphi) xs \wedge$

$(\forall x \in EQ \varphi xs. \neg DLO.I \varphi (x\#xs))$

with $\langle ?QE \rangle \langle ngfree \varphi \rangle$ **obtain** $l u$

where $DLO.I (subst_2 l u \varphi) xs$ **and** $xs!l < xs!u$

by(*fastsimp simp: qe-interior1-def set-lbounds set-ubounds I-subst EQ-conv-set-ebounds*)

moreover then obtain x **where** $xs!l < x \wedge x < xs!u$ **by**(*metis dense*)

ultimately have $DLO.I \varphi (x \# xs)$

using $\langle ngfree \varphi \rangle I\text{-subst}_2[OF \langle ngfree \varphi \rangle \langle xs!l < xs!u \rangle]$ **by** *simp*

hence $?EX ..$ }

ultimately show $?EX$ **by** *blast*

next

let $?as = DLO.atoms_0 \varphi$ **let** $?E = \text{set}(\text{ebounds } ?as)$

assume $?EX$

then obtain x **where** $x: DLO.I \varphi (x\#xs) ..$

{ **assume** $DLO.I (inf_- \varphi) xs \vee DLO.I (inf_+ \varphi) xs$

hence $?QE$ **using** $\langle ngfree \varphi \rangle$ **by**(*auto simp:qe-interior1-def*)

} **moreover**

{ **assume** $EX k : ?E. DLO.I (subst \varphi k) xs$

hence $?QE$ **by**(*force simp:qe-interior1-def*) } **moreover**

{ **assume** $\neg DLO.I (inf_- \varphi) xs$ **and** $\neg DLO.I (inf_+ \varphi) xs$

and $\forall k \in ?E. \neg DLO.I (subst \varphi k) xs$

hence $noE: \forall e \in EQ \varphi xs. \neg DLO.I \varphi (e\#xs)$

using $\langle ngfree \varphi \rangle$ **by** (*force simp:set-ebounds EQ-def I-subst*)

hence $x \notin EQ \varphi xs$ **using** x **by** *fastsimp*

obtain l **where** $l : LB \varphi xs \ l < x$

using $LBex[OF \langle ngfree \varphi \rangle x \langle \neg DLO.I (inf_- \varphi) xs \rangle \langle x \notin EQ \varphi xs \rangle] ..$

obtain u **where** $u : UB \varphi xs \ x < u$

using $UBex[OF \langle ngfree \varphi \rangle x \langle \neg DLO.I (inf_+ \varphi) xs \rangle \langle x \notin EQ \varphi xs \rangle] ..$

```

have  $\exists l \in LB \ \varphi \ xs. \ \exists u \in UB \ \varphi \ xs. \ l < x \wedge x < u \wedge \text{nolub } \varphi \ xs \ l \ x \ u \wedge (\forall y. \ l < y \wedge y < u \longrightarrow DLO.I \ \varphi \ (y \# xs))$ 
using dense-interval[where  $P = \lambda x. DLO.I \ \varphi \ (x \# xs)$ , OF finite-LB finite-UB
 $\langle l:LB \ \varphi \ xs \rangle \langle u:UB \ \varphi \ xs \rangle \langle l < x \rangle \langle x < u \rangle x$ ] x innermost-intvl[OF  $\langle \text{nqfree } \varphi \rangle \dots \langle x \notin EQ \ \varphi \ xs \rangle$ ]
by (simp add:nolub-def) fastsimp
then obtain  $m \ n$  where
   $Less \ (Suc \ m) \ 0 : \text{set } ?as \ Less \ 0 \ (Suc \ n) : \text{set } ?as$ 
   $xs!m < x \wedge x < xs!n$ 
   $\text{nolub } \varphi \ xs \ (xs!m) \ x \ (xs!n)$ 
   $\forall y. \ xs!m < y \wedge y < xs!n \longrightarrow DLO.I \ \varphi \ (y \# xs)$ 
by blast
moreover
hence  $DLO.I \ (subst_2 \ m \ n \ \varphi) \ xs$  using noE
by(force intro!: I-subst_2[OF  $\langle \text{nqfree } \varphi \rangle$ ])
ultimately have  $?QE$ 
by(fastsimp simp add:qe-interior_1-def beX-Un set-lbounds set-ubounds)
} ultimately show  $?QE$  by blast
qed

```

```

lemma qfree-astubst_2:  $qfree \ (astubst_2 \ l \ u \ a)$ 
by(cases (l,u,a) rule:astubst_2.cases) simp-all

```

```

lemma qfree-subst_2:  $\text{nqfree } \varphi \Longrightarrow qfree \ (subst_2 \ l \ u \ \varphi)$ 
by(induct  $\varphi$ ) (simp-all add:qfree-astubst_2)

```

```

lemma qfree-interior_1:  $\text{nqfree } \varphi \Longrightarrow qfree(qe-interior_1 \ \varphi)$ 
apply(simp add:qe-interior_1-def)
apply(rule qfree-list-disj)
apply (auto simp:qfree-min-inf qfree-plus-inf qfree-subst_2 qfree-map-fm)
done

```

```

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

```

```

lemma qfree-qe-interior:  $qfree(qe-interior \ \varphi)$ 
by(simp add: qe-interior-def DLO.qfree-lift-nnf-qe qfree-interior_1)

```

```

lemma I-qe-interior:  $DLO.I \ (qe-interior \ \varphi) \ xs = DLO.I \ \varphi \ xs$ 
by(simp add:qe-interior-def DLO.I-lift-nnf-qe qfree-interior_1 I-interior_1)

```

```

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₊ φ *k* \equiv *amap_fm* (*asubst*₊ *k*) φ

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

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

nolb (*And* φ_1 φ_2) *xs l x* = (*nolb* φ_1 *xs l x* \wedge *nolb* φ_2 *xs l x*)

apply(*clarsimp simp:nolb-def*)

apply *blast*

done

lemma *nolb-Or*[*simp*]:

nolb (*Or* φ_1 φ_2) *xs l x* = (*nolb* φ_1 *xs l x* \wedge *nolb* φ_2 *xs l x*)

apply(*clarsimp simp:nolb-def*)

apply *blast*

done

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

lemma *innermost-intvl*:

\llbracket *nqfree* φ ; *nolb* φ *xs l x*; *l* < *x*; *x* \notin *EQ* φ *xs*; *DLO.I* φ (*x#xs*); *l* < *y*; *y* \leq *x*
 \implies *DLO.I* φ (*y#xs*)

proof(*induct* φ)

case (*Atom* *a*)

show *?case*

proof (*cases* *a*)

case (*Less* *i* *j*)

then show *?thesis* **using** *Atom*

unfolding *nolb-def*

by (*clarsimp simp: nth.simps Ball-def split:split-if-asm nat.splits*)

(*metis not-leE order-antisym order-less-trans*)+

next

case (*Eq* *i* *j*) **thus** *?thesis* **using** *Atom*

```

    apply (clarsimp simp:EQ-def nolb-def nth-Cons')
    apply (case-tac i=0 ∧ j=0) apply simp
    apply (case-tac i≠0 ∧ j≠0) apply simp
    apply (case-tac i=0 ∧ j≠0) apply (fastsimp split:split-if-asm)
    apply (case-tac i≠0 ∧ j=0) apply (fastsimp split:split-if-asm)
    apply arith
  done
qed
next
  case And thus ?case by (fastsimp)
next
  case Or thus ?case by (fastsimp)
qed simp+

lemma I-subst-peps2:
  nqfree φ ⇒ xs!l < x ⇒ nolb φ xs (xs!l) x ⇒ x ∉ EQ φ xs
  ⇒ ∀ y ∈ {xs!l <.. x}. DLO.I φ (y#xs)
  ⇒ DLO.I (subst+ φ l) xs
proof (induct φ)
  case FalseF thus ?case
    by simp (metis linorder-antisym-conv1 linorder-neq-iff)
next
  case (Atom a)
  show ?case
  proof (cases (l,a) rule:asubst-peps.cases)
    case 3 thus ?thesis using Atom
      by (auto simp: nolb-def EQ-def Ball-def)
      (metis One-nat-def linorder-antisym-conv1 not-less-iff-gr-or-eq)
  qed (insert Atom, auto simp: nolb-def EQ-def Ball-def)
next
  case Or thus ?case by (simp add: Ball-def) (metis order-refl innermost-intvl)
qed simp-all
declare[[simp-depth-limit=50]]

lemma dense-interval:
  assumes finite L l ∈ L l < x P(x::'a::dlo)
  and dense: ∧ y l. [∀ y ∈ {l <.. <x}. y ∉ L; l < x; l < y; y ≤ x] ⇒ P y
  shows ∃ l ∈ L. l < x ∧ (∀ y ∈ {l <.. <x}. y ∉ L) ∧ (∀ y. l < y ∧ y ≤ x → P y)
proof -
  let ?L = {l ∈ L. l < x}
  let ?ll = Max ?L
  have ?L ≠ {} using ⟨l ∈ L⟩ ⟨l < x⟩ by (blast intro:order-less-imp-le)
  hence ∀ y. ?ll < y ∧ y < x → y ∉ L using ⟨finite L⟩ by force
  moreover have ?ll ∈ L
  proof
    show ?ll ∈ ?L using ⟨finite L⟩ Max-in[OF - ⟨?L ≠ {}⟩] by simp
    show ?L ⊆ L by blast
  qed
qed

```

moreover have $?ll < x$ **using** $\langle \text{finite } L \rangle \langle ?L \neq \{\} \rangle$ **by** *simp*
ultimately show $?thesis$ **using** $\langle l < x \rangle \langle ?L \neq \{\} \rangle$
by(*blast intro! : dense greaterThanLessThan-iff [THEN iffD1]*)
qed

lemma *I-subst-peps:*

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

proof(*induct* φ)

case *TrueF* **thus** $?case$ **by** *simp* (*metis no-ub*)

next

case (*Atom a*)

show $?case$

proof (*cases* (*l, a*) *rule: asubst-peps.cases*)

case 2 **thus** $?thesis$ **using** *Atom*

apply(*auto*)

apply(*drule dense*)

apply(*metis One-nat-def xt1* (γ))

done

next

case 3 **thus** $?thesis$ **using** *Atom*

apply(*auto*)

apply (*metis no-ub*)

apply (*metis no-ub less-trans*)

apply (*metis no-ub*)

done

next

case 4 **thus** $?thesis$ **using** *Atom* **by**(*auto*)(*metis no-ub*)

next

case 5 **thus** $?thesis$ **using** *Atom* **by**(*auto*)(*metis no-ub*)

next

case 8 **thus** $?thesis$ **using** *Atom* **by**(*auto*)(*metis no-ub*)

qed (*insert Atom, auto*)

next

case *And* **thus** $?case$

apply *clarsimp*

apply(*rule-tac x=min leps lepsa in exI*)

apply *simp*

done

next

case *Or* **thus** $?case$ **by** *force*

qed *simp-all*

definition

$qe-eps_1(\varphi) =$

(*let* $as = DLO.atoms_0 \varphi$; $lbs = lbounds\ as$; $ebs = ebounds\ as$

in $list-disj\ (inf_ \varphi \# map\ (subst_+ \varphi)\ lbs\ @\ map\ (subst\ \varphi)\ ebs)$)

```

theorem I-qe-eps1:
assumes ngfree  $\varphi$  shows  $DLO.I (qe-eps_1 \varphi) xs = (\exists x. DLO.I \varphi (x\#xs))$ 
  (is  $?QE = ?EX$ )
proof
  let  $?as = DLO.atoms_0 \varphi$  let  $?ebs = ebounds ?as$ 
  assume  $?QE$ 
  { assume  $DLO.I (inf_ - \varphi) xs$ 
    hence  $?EX$  using  $\langle ?QE \rangle min-inf[of \varphi xs] \langle ngfree \varphi \rangle$ 
    by (auto simp add:qe-eps1-def amap-fm-list-disj)
  } moreover
  { assume  $\forall i \in set ?ebs. \neg DLO.I \varphi (xs!i \# xs)$ 
     $\neg DLO.I (inf_ - \varphi) xs$ 
    with  $\langle ?QE \rangle \langle ngfree \varphi \rangle$  obtain  $l$  where  $DLO.I (subst_+ \varphi l) xs$ 
    by (fastsimp simp: I-subst qe-eps1-def set-ebounds set-lbounds)
    then obtain  $leps$  where  $DLO.I \varphi (leps\#xs)$ 
    using I-subst-peps[OF  $\langle ngfree \varphi \rangle$ ] by fastsimp
    hence  $?EX ..$  }
  ultimately show  $?EX$  by blast
next
  let  $?as = DLO.atoms_0 \varphi$  let  $?ebs = ebounds ?as$ 
  assume  $?EX$ 
  then obtain  $x$  where  $x: DLO.I \varphi (x\#xs) ..$ 
  { assume  $DLO.I (inf_ - \varphi) xs$ 
    hence  $?QE$  using  $\langle ngfree \varphi \rangle$  by (auto simp:qe-eps1-def)
  } moreover
  { assume  $\exists k \in set ?ebs. DLO.I (subst \varphi k) xs$ 
    hence  $?QE$  by (auto simp:qe-eps1-def) } moreover
  { assume  $\neg DLO.I (inf_ - \varphi) xs$ 
    and  $\forall k \in set ?ebs. \neg DLO.I (subst \varphi k) xs$ 
    hence  $noE: \forall e \in EQ \varphi xs. \neg DLO.I \varphi (e\#xs)$  using  $\langle ngfree \varphi \rangle$ 
    by (auto simp:set-ebounds EQ-def I-subst nth-Cons' split:split-if-asm)
    hence  $x \notin EQ \varphi xs$  using  $x$  by fastsimp
    obtain  $l$  where  $l \in LB \varphi xs$   $l < x$ 
    using LBex[OF  $\langle ngfree \varphi \rangle$ ]  $x$   $\langle \neg DLO.I (inf_ - \varphi) xs \rangle$   $\langle x \notin EQ \varphi xs \rangle ..$ 
    have  $\exists l \in LB \varphi xs. l < x \wedge nolib \varphi xs$   $l x \wedge$ 
       $(\forall y. l < y \wedge y \leq x \longrightarrow DLO.I \varphi (y\#xs))$ 
    using dense-interval [where  $P = \lambda x. DLO.I \varphi (x\#xs)$ , OF finite-LB  $\langle l \in LB \varphi xs \rangle$   $\langle l < x \rangle$   $x$  innermost-intvl [OF  $\langle ngfree \varphi \rangle$ ]]  $\langle x \notin EQ \varphi xs \rangle$ 
    by (simp add:nolib-def)
    then obtain  $m$ 
    where Less (Suc  $m$ )  $0 \in set ?as \wedge xs!m < x \wedge nolib \varphi xs$   $(xs!m) x$ 
       $\wedge (\forall y. xs!m < y \wedge y \leq x \longrightarrow DLO.I \varphi (y\#xs))$ 
    by blast
    then moreover have  $DLO.I (subst_+ \varphi m) xs$ 
    using  $noE$  by (auto intro!: I-subst-peps2 [OF  $\langle ngfree \varphi \rangle$ ])
    ultimately have  $?QE$ 
    by (simp add:qe-eps1-def bex-Un set-lbounds set-ebounds) metis
  } ultimately show  $?QE$  by blast

```

qed

lemma *qfree-astubst-peps*: $qfree (astubst_+ k a)$
by (*cases* (k,a) *rule:astubst-peps.cases*) *simp-all*

lemma *qfree-subst-peps*: $nqfree \varphi \implies qfree (subst_+ \varphi k)$
by (*induct* φ) (*simp-all add:qfree-astubst-peps*)

lemma *qfree-qe-eps1*: $nqfree \varphi \implies qfree(qe-eps_1 \varphi)$
apply (*simp add:qe-eps1-def*)
apply (*rule qfree-list-disj*)
apply (*auto simp:qfree-min-inf qfree-subst-peps qfree-map-fm*)
done

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

lemma *qfree-qe-eps*: $qfree(qe-eps \varphi)$
by (*simp add: qe-eps-def DLO.qfree-lift-nnf-qe qfree-qe-eps1*)

lemma *I-qe-eps*: $DLO.I (qe-eps \varphi) xs = DLO.I \varphi xs$
by (*simp add:qe-eps-def DLO.I-lift-nnf-qe qfree-qe-eps1 I-qe-eps1*)

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*) \Rightarrow *'a list* \Rightarrow *'a list* (**infix** $*_s$ 70)
where $x *_s xs \equiv map (op * x) xs$

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

4.1 + and -

fun *zipwith0* :: (*'a::zero* \Rightarrow *'b::zero* \Rightarrow *'c*) \Rightarrow *'a list* \Rightarrow *'b list* \Rightarrow *'c list*
where
 $zipwith0 f [] [] = []$ |
 $zipwith0 f (x\#xs) (y\#ys) = f x y \# zipwith0 f xs ys$ |
 $zipwith0 f (x\#xs) [] = f x 0 \# zipwith0 f xs []$ |
 $zipwith0 f [] (y\#ys) = f 0 y \# zipwith0 f [] ys$

```

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

definition
  list-add-def: op + = zipwith0 (op +)

instance ..

end

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

definition
  list-uminus-def: uminus = map uminus

instance ..

end

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

definition
  list-diff-def: op - = zipwith0 (op -)

instance ..

end

lemma zipwith0-Nil[simp]: zipwith0 f [] ys = map (f 0) ys
by(induct ys) simp-all

lemma list-add-Nil[simp]: [] + xs = (xs::'a::monoid-add list)
by (induct xs) (auto simp:list-add-def)

lemma list-add-Nil2[simp]: xs + [] = (xs::'a::monoid-add list)
by (induct xs) (auto simp:list-add-def)

lemma list-add-Cons[simp]: (x#xs) + (y#ys) = (x+y)#(xs+ys)
by(auto simp:list-add-def)

lemma list-diff-Nil[simp]: [] - xs = -(xs::'a::group-add list)
by (induct xs) (auto simp:list-diff-def list-uminus-def)

lemma list-diff-Nil2[simp]: xs - [] = (xs::'a::group-add list)
by (induct xs) (auto simp:list-diff-def)

```

lemma *list-diff-Cons-Cons*[simp]: $(x\#xs) - (y\#ys) = (x-y)\#(xs-ys)$
by (*induct xs*) (*auto simp:list-diff-def*)

lemma *list-uminus-Cons*[simp]: $-(x\#xs) = (-x)\#(-xs)$
by (*induct xs*) (*auto simp:list-uminus-def*)

lemma *self-list-diff*:
 $xs - xs = \text{replicate } (\text{length}(xs)::'a::\text{group-add list}) 0$
by(*induct xs*) *simp-all*

lemma *list-add-assoc*: **fixes** $xs :: 'a::\text{monoid-add list}$
shows $(xs+ys)+zs = xs+(ys+zs)$
apply(*induct xs arbitrary: ys zs*)
apply *simp*
apply(*case-tac ys*)
apply(*simp*)
apply(*simp*)
apply(*case-tac zs*)
apply(*simp*)
apply(*simp add:add-assoc*)
done

4.2 Inner product

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

lemma *iprod-Nil*[simp]: $\langle [], ys \rangle = 0$
by(*simp add:iprod-def*)

lemma *iprod-Nil2*[simp]: $\langle xs, [] \rangle = 0$
by(*simp add:iprod-def*)

lemma *iprod-Cons*[simp]: $\langle x\#xs, y\#ys \rangle = x*y + \langle xs, ys \rangle$
by(*simp add:iprod-def*)

lemma *iprod0-if-coeffs0*: $\forall c \in \text{set } cs. c = 0 \implies \langle cs, xs \rangle = 0$
apply(*induct cs arbitrary:xs*)
apply *simp*
apply(*case-tac xs*) **apply** *simp*
apply *auto*
done

lemma *iprod-uminus*[simp]: $\langle -xs, ys \rangle = -\langle xs, ys \rangle$
by(*simp add: iprod-def uminus-listsum-map o-def split-def map-zip-map list-uminus-def*)

lemma *iprod-left-add-distrib*: $\langle xs + ys, zs \rangle = \langle xs, zs \rangle + \langle ys, zs \rangle$
apply(*induct xs arbitrary: ys zs*)
apply (*simp add: o-def split-def*)

```

apply(case-tac ys)
apply simp
apply(case-tac zs)
apply (simp)
apply(simp add:left-distrib)
done

```

```

lemma iprod-left-diff-distrib:  $\langle xs - ys, zs \rangle = \langle xs, zs \rangle - \langle ys, zs \rangle$ 
apply(induct xs arbitrary: ys zs)
apply (simp add: o-def split-def)
apply(case-tac ys)
apply simp
apply(case-tac zs)
apply (simp)
apply(simp add:left-diff-distrib)
done

```

```

lemma iprod-assoc:  $\langle x *_s xs, ys \rangle = x * \langle xs, ys \rangle$ 
apply(induct xs arbitrary: ys)
apply simp
apply(case-tac ys)
apply (simp)
apply (simp add:right-distrib mult-assoc)
done

```

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 =  $(\exists r rs. f = \text{Less } r \text{ } rs)$ 
by(induct f) auto

```

lemma *is-Eq-iff*: $(\forall i j. a \neq \text{Less } i j) = (\exists i j. a = \text{Eq } i j)$
by(*cases a*) *auto*

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*) |
decr_R (*Eq r rs*) = *Eq r* (*tl rs*)

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

definition *atoms₀* = *ATOM.atoms₀* *depends_R*

interpretation *R!*: *ATOM neg_R* ($\lambda a. \text{True}$) *I_R* *depends_R* *decr_R*
where *ATOM.atoms₀* *depends_R* = *atoms₀*

proof –
case *goal1*
thus ?*case*
 apply(*unfold-locales*)
 apply(*case-tac a*)
 apply *simp-all*
 apply(*case-tac a*)
 apply *simp-all*
 apply *arith*
 apply *arith*
 apply(*case-tac a*)
 apply(*simp-all add:depends_R-def split:list.splits*)
 done
next
 case *goal2*
 thus ?*case* **by**(*simp add:atoms₀-def*)
qed

setup \ll *Sign.revert-abbrev* @{*const-name R.I*} \gg
setup \ll *Sign.revert-abbrev* @{*const-name R.lift-nnf-qe*} \gg

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}

by(force simp: lbounds-def split:list.splits atom.splits if-splits)

lemma *set-ubounds*:

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

by(force simp: ubounds-def split:list.splits atom.splits if-splits)

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}

by(force simp: ebounds-def split:list.splits atom.splits if-splits)

abbreviation *EQ where*

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

abbreviation *LB where*

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

abbreviation *UB where*

UB f xs \equiv {(r - <cs,xs>)/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 + <snd rcs,xs>

lemma *I-asubst*:

I_R (*asubst* t a) xs = *I_R* a (*eval* t xs # xs)

proof(cases a)

case (*Less* r cs)

thus ?thesis **by**(cases t, cases cs,

simp-all add:eval-def right-distrib iprod-left-add-distrib)

arith

next

case (*Eq* r cs)

thus *?thesis*
by(*cases t, cases cs, simp-all add:eval-def right-distrib iprod-left-add-distrib*)
arith
qed

lemma *I-subst*:

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

by(*induct* φ)(*simp-all add:I-asubst*)

lemma *I-subst-pretty*:

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

by(*simp add:I-subst eval-def*)

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*₋ φ)

by(*induct* φ *rule:min-inf.induct*) *simp-all*

lemma *qfree-plus-inf*: *qfree* $\varphi \implies$ *qfree*(*inf*₊ φ)

by(*induct* φ *rule:plus-inf.induct*) *simp-all*

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(*induct* f)

case (*Atom* a)

show *?case*

proof (*cases* a)

case (*Less* $r cs$)

show *?thesis*

proof(*cases* cs)

case *Nil* **thus** *?thesis* **using** *Less* **by** *simp*

next

case (*Cons* $c cs$)

{ **assume** $c = 0$ **hence** *?thesis* **using** *Less Cons* **by** *simp* }

```

moreover
{ assume  $c < 0$ 
  hence  $?P (Atom\ a) ((r - \langle cs, xs \rangle + 1)/c)$  using Less Cons
  by(auto simp add: field-simps)
  hence ?thesis .. }
moreover
{ assume  $c > 0$ 
  hence  $?P (Atom\ a) ((r - \langle cs, xs \rangle - 1)/c)$  using Less Cons
  by(auto simp add: field-simps)
  hence ?thesis .. }
ultimately show ?thesis by force
qed
next
case (Eq r cs)
show ?thesis
proof(cases cs)
  case Nil thus ?thesis using Eq by simp
next
  case (Cons c cs)
  { assume  $c = 0$  hence ?thesis using Eq Cons by simp }
  moreover
  { assume  $c < 0$ 
    hence  $?P (Atom\ a) ((r - \langle cs, xs \rangle + 1)/c)$  using Eq Cons
    by(auto simp add: field-simps)
    hence ?thesis .. }
  moreover
  { assume  $c > 0$ 
    hence  $?P (Atom\ a) ((r - \langle cs, xs \rangle - 1)/c)$  using Eq Cons
    by(auto simp add: field-simps)
    hence ?thesis .. }
  ultimately show ?thesis by force
  qed
qed
next
case (And f1 f2)
then obtain  $x1\ x2$  where  $?P\ f1\ x1\ ?P\ f2\ x2$  by fastsimp+
hence  $?P (And\ f1\ f2) (min\ x1\ x2)$  by(force simp:and-def)
thus ?case ..
next
case (Or f1 f2)
then obtain  $x1\ x2$  where  $?P\ f1\ x1\ ?P\ f2\ x2$  by fastsimp+
hence  $?P (Or\ f1\ f2) (min\ x1\ x2)$  by(force simp:or-def)
thus ?case ..
qed simp-all

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(induct f)

```

```

case (Atom a)
show ?case
proof (cases a)
  case (Less r cs)
  show ?thesis
proof (cases cs)
  case Nil thus ?thesis using Less by simp
next
  case (Cons c cs)
  { assume c=0 hence ?thesis using Less Cons by simp }
  moreover
  { assume c<0
    hence ?P (Atom a) ((r - ⟨cs,xs⟩ - 1)/c) using Less Cons
      by(auto simp add: field-simps)
    hence ?thesis .. }
  moreover
  { assume c>0
    hence ?P (Atom a) ((r - ⟨cs,xs⟩ + 1)/c) using Less Cons
      by(auto simp add: field-simps)
    hence ?thesis .. }
  ultimately show ?thesis by force
qed
next
case (Eq r cs)
show ?thesis
proof (cases cs)
  case Nil thus ?thesis using Eq by simp
next
  case (Cons c cs)
  { assume c=0 hence ?thesis using Eq Cons by simp }
  moreover
  { assume c<0
    hence ?P (Atom a) ((r - ⟨cs,xs⟩ - 1)/c) using Eq Cons
      by(auto simp add: field-simps)
    hence ?thesis .. }
  moreover
  { assume c>0
    hence ?P (Atom a) ((r - ⟨cs,xs⟩ + 1)/c) using Eq Cons
      by(auto simp add: field-simps)
    hence ?thesis .. }
  ultimately show ?thesis by force
qed
qed
next
case (And f1 f2)
then obtain x1 x2 where ?P f1 x1 ?P f2 x2 by fastsimp+
hence ?P (And f1 f2) (max x1 x2) by(force simp:and-def)
thus ?case ..
next

```

```

    case (Or f1 f2)
    then obtain x1 x2 where ?P f1 x1 ?P f2 x2 by fastsimp+
    hence ?P (Or f1 f2) (max x1 x2) by (force simp: or-def)
    thus ?case ..
qed simp-all

```

```

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

```

```

lemma LBeX:
  [[ nqfree f; R.I f (x#xs); ¬R.I (inf_ f) xs; x ∉ EQ f xs ]
  ⇒ ∃ l ∈ LB f xs. l < x
apply (induct f)
apply simp
apply simp
apply (case-tac a)
  apply (simp add: depends_R-def split:if-splits list.splits)
  apply (simp add: field-simps)
apply (fastsimp simp add: depends_R-def split:if-splits list.splits)
apply fastsimp
apply fastsimp
apply simp
apply simp
done

```

```

lemma UBeX:
  [[ nqfree f; R.I f (x#xs); ¬R.I (inf_+ f) xs; x ∉ EQ f xs ]
  ⇒ ∃ u ∈ UB f xs. x < u
apply (induct f)
apply simp
apply simp
apply (case-tac a)
  apply (simp add: depends_R-def split:if-splits list.splits)
  apply (simp add: field-simps)
apply (fastsimp simp add: depends_R-def split:if-splits list.splits)
apply fastsimp
apply fastsimp
apply simp
apply simp
done

```

```

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

```

```

lemma finite-LB: finite (LB f xs)
proof -
  have LB f xs = (λ(r,cs). r + ⟨cs,xs⟩) ‘ set (lbounds (R.atoms_0 f))
  by (force simp: set-lbounds image-def field-simps)
  thus ?thesis by simp

```

qed

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

proof –

have $UB\ f\ xs = (\lambda(r,cs). r + \langle cs, xs \rangle) \text{ ` } \text{set}(\text{ubounds}(R.\text{atoms}_0\ f))$

by (*force simp:set-ubounds image-def field-simps*)

thus *?thesis* **by** *simp*

qed

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*: $\forall a \in \text{set } as. \text{is-Less } a$ **and** *dep*: $\forall a \in \text{set } as. \text{depends}_R\ a$

shows *R.I* (*qe-FM₁* *as*) $x_s = (\exists x. \forall a \in \text{set } as. I_R\ a\ (x \# xs))$ (**is** *?L* = *?R*)

proof

let *?Ls* = *set(lbounds as)* **let** *?Us* = *set(ubounds as)*

let *?lbs* = *UN* (*r,cs*):*?Ls*. $\{r + \langle cs, xs \rangle\}$

let *?ubs* = *UN* (*r,cs*):*?Us*. $\{r + \langle cs, xs \rangle\}$

have *fins*: *finite ?lbs* \wedge *finite ?ubs* **by** *auto*

have *2*: $\forall f \in \text{set } as. \exists r\ c\ cs. f = \text{Less } r\ (c \# cs) \wedge$

$(c > 0 \wedge (r/c, (-1/c) *_s\ cs) \in ?Ls \vee c < 0 \wedge (r/c, (-1/c) *_s\ cs) \in ?Us)$

using *dep less*

by(*fastsimp simp:set-lbounds set-ubounds is-Less-iff depends_R-def*
 split:list.splits)

assume *?L*

have *1*: $\forall x \in ?lbs. \forall y \in ?ubs. x < y$

proof (*rule ballI*)+

fix *x y* **assume** $x \in ?lbs\ y \in ?ubs$

then obtain *r cs*

where $(r,cs) \in ?Ls \wedge x = r + \langle cs, xs \rangle$ **by** *fastsimp*

moreover from $\langle y \in ?ubs \rangle$ **obtain** *s ds*

where $(s,ds) \in ?Us \wedge y = s + \langle ds, xs \rangle$ **by** *fastsimp*

ultimately show $x < y$ **using** *?L*

by(*fastsimp simp:qe-FM₁-def algebra-simps iprod-left-diff-distrib*)

qed

{ **assume** *nonempty*: $?lbs \neq \{\}$ \wedge $?ubs \neq \{\}$

hence *Max ?lbs* < *Min ?ubs* **using** *fins 1*

```

    by(blast intro: Max-less-iff[THEN iffD2] Min-gr-iff[THEN iffD2])
  then obtain m where Max ?lbs < m  $\wedge$  m < Min ?ubs
    using dense[where 'a = real] by blast
  hence  $\forall a \in \text{set } as. I_R a (m \# xs)$  using 2 nonempty
    by (auto simp:Ball-def Bex-def)(fastsimp simp:field-simps)
  hence ?R .. }
moreover
{ assume asm: ?lbs  $\neq$  {}  $\wedge$  ?ubs = {}
  have  $\forall a \in \text{set } as. I_R a ((Max ?lbs + 1) \# xs)$ 
  proof
    fix a assume a  $\in$  set as
    then obtain r c cs
      where a = Less r (c#cs) c > 0 (r/c, (-1/c)*s cs)  $\in$  ?Ls
      using asm 2 by fastsimp
    moreover hence (r - <cs,xs>)/c  $\leq$  Max ?lbs
      using asm fins
    by(auto intro!: Max-ge-iff[THEN iffD2])
      (force simp add:field-simps)
    ultimately show I_R a ((Max ?lbs + 1) # xs) by (simp add: field-simps)
  qed
  hence ?R .. }
moreover
{ assume asm: ?lbs = {}  $\wedge$  ?ubs  $\neq$  {}
  have  $\forall a \in \text{set } as. I_R a ((Min ?ubs - 1) \# xs)$ 
  proof
    fix a assume a  $\in$  set as
    then obtain r c cs
      where a = Less r (c#cs) c < 0 (r/c, (-1/c)*s cs)  $\in$  ?Us
      using asm 2 by fastsimp
    moreover hence Min ?ubs  $\leq$  (r - <cs,xs>)/c
      using asm fins
    by(auto intro!: Min-le-iff[THEN iffD2])
      (force simp add:field-simps)
    ultimately show I_R a ((Min ?ubs - 1) # xs) by (simp add: field-simps)
  qed
  hence ?R .. }
moreover
{ assume ?lbs = {}  $\wedge$  ?ubs = {}
  hence ?R using 2 less by auto (rule, fast)
}
ultimately show ?R by blast
next
let ?Ls = set(lbounds as) let ?Us = set(ubounds as)
assume ?R
then obtain x where 1:  $\forall a \in \text{set } as. I_R a (x \# xs)$  ..
{ fix r c cs s d ds
  assume Less r (c#cs)  $\in$  set as 0 < c Less s (d#ds)  $\in$  set as d < 0
  hence r < c*x + <cs,xs> s < d*x + <ds,xs> c > 0 d < 0
  using 1 by auto

```

hence $(r - \langle cs, xs \rangle) / c < x \ x < (s - \langle ds, xs \rangle) / d$ **by** (*simp add: field-simps*) +
hence $(r - \langle cs, xs \rangle) / c < (s - \langle ds, xs \rangle) / d$ **by** *arith*
}
thus ?L **by** (*auto simp: qe-FM₁-def iprod-left-diff-distrib less field-simps set-lbounds set-ubounds*)
qed

corollary *I-qe-FM₁-pretty*:

$\forall a \in \text{set } as. \text{is-Less } a \wedge \text{depends}_R a \implies R.\text{is-dnf-qe } qe\text{-FM}_1 \text{ as}$
by (*metis I-qe-FM₁*)

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*:

subst₀ (*Eq* *r* (*c* # *cs*)) (*Less* *s* (*d* # *ds*)) = *Less* (*s* - (*r* * *d*) / *c*) (*ds* - (*d* / *c*) *_s *cs*)
subst₀ (*Eq* *r* (*c* # *cs*)) (*Eq* *s* (*d* # *ds*)) = *Eq* (*s* - (*r* * *d*) / *c*) (*ds* - (*d* / *c*) *_s *cs*)
by *auto*

lemma *I-subst₀*: $\text{depends}_R a \implies c \neq 0 \implies$

$I_R (\text{subst}_0 (\text{Eq } r (c \# cs)) a) \text{ xs} = I_R a ((r - \langle cs, xs \rangle) / c \# xs)$

apply (*cases* *a*)

by (*auto simp add: depends_R-def iprod-left-diff-distrib algebra-simps diff-divide-distrib split:list.splits*)

interpretation *R_e!*:

ATOM-EQ *neg_R* ($\lambda a. \text{True}$) *I_R* $\text{depends}_R \text{decr}_R$

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

$(\lambda \text{Eq } r \text{ cs} \Rightarrow r = 0 \wedge (\forall c \in \text{set } cs. c = 0) \mid - \Rightarrow \text{False}) \text{subst}_0$

apply (*unfold-locales*)

apply (*simp del:subst₀.simps add:I-subst₀ split:atom.splits list.splits*)

apply (*simp add: iprod0-if-coeffs0 split:atom.splits*)

apply (*simp split:atom.splits list.splits*)

apply (*rename-tac* *r* *ds* *c* *cs*)

apply (*rule-tac* $x = (r - \langle cs, xs \rangle) / c$ **in** *exI*)

apply (*simp add: algebra-simps diff-divide-distrib*)

apply (*simp add: self-list-diff set-replicate-conv-if split:atom.splits list.splits*)

done

definition *qe-FM* = *R_e.lift-dnf-qe* *qe-FM₁*

lemma *qfree-qe-FM₁*: *qfree* (*qe-FM₁* *as*)

by (*auto simp: qe-FM₁-def intro!: qfree-list-conj*)

```

corollary I-qe-FM:  $R.I (qe-FM \varphi) xs = R.I \varphi xs$ 
unfolding qe-FM-def
apply(rule Re.I-lift-dnfeq-qe)
  apply(rule qfree-qe-FM1)
apply(rule allI)
apply(rule I-qe-FM1)
  prefer 2 apply blast
apply(clarify)
apply(drule-tac x=a in bspec) apply simp
apply(simp add: dependsR-def split:atom.splits list.splits)
done

```

```

theorem qfree-qe-FM:  $qfree (qe-FM f)$ 
by(simp add:qe-FM-def Re.qfree-lift-dnfeq-qe qfree-qe-FM1)

```

5.2.1 Tests

```

lemmas qesimps = qe-FM-def Re.lift-dnfeq-qe-def Re.lift-eq-qe-def R.qelim-def qe-FM1-def
lbounds-def ubounds-def list-conj-def list-disj-def and-def or-def dependsR-def

```

```

lemma qe-FM(TrueF) = TrueF
by(simp add:qesimps)

```

```

lemma
  qe-FM(ExQ (And (Atom(Less 0 [1])) (Atom(Less 0 [-1]))) = Atom(Less 0 [])
by(simp add:qesimps)

```

```

lemma
  qe-FM(ExQ (And (Atom(Less 0 [1])) (Atom(Less -1 [-1]))) = Atom(Less -1
  [])
by(simp add:qesimps)

```

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  $\Rightarrow$  (if  $\forall c \in \text{set } cs. c = 0$ 
      then if  $r < 0$  then TrueF else FalseF
      else Atom a) |

```

$Eq\ r\ cs \Rightarrow (if\ \forall c \in\ set\ cs.\ c = 0$
 $then\ if\ r=0\ then\ TrueF\ else\ FalseF\ else\ Atom\ a))$

lemma *asimp-pretty*:
asimp (Less r cs) =
 (*if* $\forall c \in\ set\ cs.\ c = 0$
 then *if* $r < 0$ *then* *TrueF* *else* *FalseF*
 else *Atom(Less r cs)*)
asimp (Eq r cs) =
 (*if* $\forall c \in\ set\ cs.\ c = 0$
 then *if* $r = 0$ *then* *TrueF* *else* *FalseF*
 else *Atom(Eq r cs)*)
by(*auto simp:asimp-def*)

definition *qe-FMo₁* :: *atom list* \Rightarrow *atom fm* **where**
qe-FMo₁ as = list-conj [asimp(combine p q). p \leftarrow lbounds as, q \leftarrow ubounds as]

lemma *I-asimp*: *R.I (asimp a) xs = I_R a xs*
by(*simp add:asimp-def iprod0-if-coeffs0 split:atom.split*)

lemma *I-qe-FMo₁*: *R.I (qe-FMo₁ as) xs = R.I (qe-FM₁ as) xs*
by(*simp add:qe-FM₁-def qe-FMo₁-def I-asimp*)

definition *qe-FMo* = *R_e.lift-dnf_{eq}-qe qe-FMo₁*

lemma *qfree-qe-FMo₁*: *qfree (qe-FMo₁ as)*
by(*auto simp:qe-FM₁-def qe-FMo₁-def asimp-def intro!:qfree-list-conj*
 split:atom.split)

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

unfolding *qe-FMo-def*
apply(*rule R_e.I-lift-dnf_{eq}-qe*)
 apply(*rule qfree-qe-FMo₁*)
apply(*rule allI*)
apply(*subst I-qe-FMo₁*)
apply(*rule I-qe-FM₁*)
 prefer 2 **apply** *blast*
apply(*clarify*)
apply(*drule-tac x=a in bspec*) **apply** *simp*
apply(*simp add: depends_R-def split:atom.splits list.splits*)
done

theorem *qfree-qe-FMo*: *qfree (qe-FMo f)*
by(*simp add:qe-FMo-def R_e.qfree-lift-dnf_{eq}-qe qfree-qe-FMo₁*)

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 FR₁ :: atom fm \Rightarrow atom fm **where**

```

FR1  $\varphi$  =
(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 : L u : U l < x x < u P(x::real)

and dense: $\bigwedge y \ l \ u. \llbracket \forall y \in \{l \dots x\}. y \notin L; \forall y \in \{x \dots u\}. y \notin U;$

$l < x; x < u; l < y; y < u \rrbracket \Longrightarrow P \ y$

shows $\exists l \in L. \exists u \in U. l < u \wedge (\forall y. l < y \wedge y < u \longrightarrow P \ y)$

proof –

let ?L = {l:L. l < x} **let** ?U = {u:U. x < u}

let ?ll = Max ?L **let** ?uu = Min ?U

have ?L \neq {} **using** ⟨l : L⟩ ⟨l < x⟩ **by** (blast intro:order-less-imp-le)

moreover have ?U \neq {} **using** ⟨u:U⟩ ⟨x < u⟩ **by** (blast intro:order-less-imp-le)

ultimately have $\forall y. ?ll < y \wedge y < x \longrightarrow y \notin L \ \forall y. x < y \wedge y < ?uu \longrightarrow y \notin U$

using ⟨finite L⟩ ⟨finite U⟩ **by** force+

moreover have ?ll : L

proof

show ?ll : ?L **using** ⟨finite L⟩ Max-in[OF - ⟨?L \neq {}⟩] **by** simp

show ?L \subseteq L **by** blast

qed

moreover have ?uu : U

proof

show ?uu : ?U **using** ⟨finite U⟩ Min-in[OF - ⟨?U \neq {}⟩] **by** simp

show ?U \subseteq U **by** blast

qed

moreover have ?ll < x **using** ⟨finite L⟩ ⟨?L \neq {}⟩ **by** simp

moreover have x < ?uu **using** ⟨finite U⟩ ⟨?U \neq {}⟩ **by** simp

moreover have ?ll < ?uu **using** ⟨?ll < x⟩ ⟨x < ?uu⟩ **by** arith

ultimately show ?thesis **using** ⟨l < x⟩ ⟨x < u⟩ ⟨?L \neq {}⟩ ⟨?U \neq {}⟩

by(blast intro!:dense greaterThanLessThan-iff[THEN iffD1])

qed

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

```

lemma dense:
  [[ 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);  $l < y$ ;  $y < u$  ] ]
   $\implies R.I\ f\ (y\#\!xs)$ 
proof(induct f)
  case (Atom a)
  show ?case
  proof (cases a)
    case (Less r cs)
    show ?thesis
    proof (cases cs)
      case Nil thus ?thesis using Atom Less by (simp add:dependsR-def)
    next
      case (Cons c cs)
      hence  $r < c*x + \langle cs,xs \rangle$  using Atom Less by simp
      { assume  $c=0$  hence ?thesis using Atom Less Cons by simp }
      moreover
      { assume  $c < 0$ 
        hence  $x < (r - \langle cs,xs \rangle)/c$  (is  $- < ?u$ ) using  $\langle r < c*x + \langle cs,xs \rangle \rangle$ 
          by (simp add:field-simps)
          have ?thesis
          proof (rule ccontr)
            assume  $\neg R.I\ (Atom\ a)\ (y\#\!xs)$ 
            hence  $?u \leq y$  using Atom Less Cons  $\langle c < 0 \rangle$ 
              by (auto simp add:field-simps)
            hence  $?u < u$  using  $\langle y < u \rangle$  by simp
            with  $\langle x < ?u \rangle$  show False using Atom Less Cons  $\langle c < 0 \rangle$ 
              by (auto simp:dependsR-def)
          qed } moreover
      { assume  $c > 0$ 
        hence  $x > (r - \langle cs,xs \rangle)/c$  (is  $- > ?l$ ) using  $\langle r < c*x + \langle cs,xs \rangle \rangle$ 
          by (simp add:field-simps)
          have ?thesis
          proof (rule ccontr)
            assume  $\neg R.I\ (Atom\ a)\ (y\#\!xs)$ 
            hence  $?l \geq y$  using Atom Less Cons  $\langle c > 0 \rangle$ 
              by (auto simp add:field-simps)
            hence  $?l > l$  using  $\langle y > l \rangle$  by simp
            with  $\langle ?l < x \rangle$  show False using Atom Less Cons  $\langle c > 0 \rangle$ 
              by (auto simp:dependsR-def)
          qed }
      ultimately show ?thesis by force
    }
  qed
next
  case (Eq r cs)
  show ?thesis
  proof (cases cs)
    case Nil thus ?thesis using Atom Eq by (simp add:dependsR-def)
  next

```

```

    case (Cons c cs)
    hence  $r = c*x + \langle cs, xs \rangle$  using Atom Eq by simp
    { assume  $c=0$  hence ?thesis using Atom Eq Cons by simp }
    moreover
    { assume  $c \neq 0$ 
      hence ?thesis using  $\langle r = c*x + \langle cs, xs \rangle$  Atom Eq Cons  $\langle l < y \rangle \langle y < u \rangle$ 
        by(auto simp: mult-ac dependsR-def split:if-splits) }
    ultimately show ?thesis by force
  qed
next
case (And f1 f2) thus ?case by (fastsimp simp:Ball-def)
next
case (Or f1 f2) thus ?case by (fastsimp simp:Ball-def)
qed fastsimp+

```

theorem *I-FR₁*:

assumes *ngfree* φ shows $R.I (FR_1 \varphi) xs = (\exists x. R.I \varphi (x\#xs))$
(is ?FR = ?EX)

proof

```

  assume ?FR
  { assume  $R.I (inf_- \varphi) xs$ 
    hence ?EX using  $\langle ?FR \rangle$  min-inf[OF  $\langle ngfree \varphi \rangle$ , where  $xs=xs$ ]
      by(auto simp add:FR1-def)
  } moreover
  { assume  $R.I (inf_+ \varphi) xs$ 
    hence ?EX using  $\langle ?FR \rangle$  plus-inf[OF  $\langle ngfree \varphi \rangle$ , where  $xs=xs$ ]
      by(auto simp add:FR1-def)
  } moreover
  { assume  $\exists x \in EQ \varphi xs. R.I \varphi (x\#xs)$ 
    hence ?EX using  $\langle ?FR \rangle$  by(auto simp add:FR1-def)
  } moreover
  { assume  $\neg R.I (inf_- \varphi) xs \wedge \neg R.I (inf_+ \varphi) xs \wedge$ 
    ( $\forall x \in EQ \varphi xs. \neg R.I \varphi (x\#xs)$ )
    with  $\langle ?FR \rangle$  obtain  $r cs s ds$ 
      where  $R.I (subst \varphi (between (r,cs) (s,ds))) xs$ 
      by(auto simp: FR1-def eval-def diff-minus[symmetric]
        diff-divide-distrib set-ebounds I-subst  $\langle ngfree \varphi \rangle$ ) blast
    hence  $R.I \varphi (eval (between (r,cs) (s,ds)) xs \# xs)$ 
      by(simp add:I-subst  $\langle ngfree \varphi \rangle$ )
    hence ?EX .. }
  ultimately show ?EX by blast
next
  assume ?EX
  then obtain  $x$  where  $x: R.I \varphi (x\#xs)$  ..
  { assume  $R.I (inf_- \varphi) xs \vee R.I (inf_+ \varphi) xs$ 
    hence ?FR by(auto simp:FR1-def)
  } moreover
  { assume  $x : EQ \varphi xs$ 

```

then obtain $r\ cs$
where $(r,cs) : \text{set}(\text{ebounds}(R.\text{atoms}_0\ \varphi)) \wedge x = r + \langle cs, xs \rangle$
by $(\text{force simp:set-ebounds field-simps})$
moreover hence $R.I\ (\text{subst}\ \varphi\ (r,cs))\ xs$ **using** x
by $(\text{auto simp:I-subst}\ \langle \text{ngfree}\ \varphi\ \text{eval-def} \rangle)$
ultimately have $?FR$ **by** $(\text{force simp:FR}_1\text{-def})$ **} moreover**
{ assume $\neg R.I\ (\text{inf}_-\ \varphi)\ xs$ **and** $\neg R.I\ (\text{inf}_+\ \varphi)\ xs$ **and** $x \notin EQ\ \varphi\ xs$
obtain l **where** $l : LB\ \varphi\ xs\ l < x$
using $LBex[OF\ \langle \text{ngfree}\ \varphi\ x\ \langle \neg R.I\ (\text{inf}_-\ \varphi)\ xs \rangle\ \langle x \notin EQ\ \varphi\ xs \rangle] ..$
obtain u **where** $u : UB\ \varphi\ xs\ x < u$
using $UBex[OF\ \langle \text{ngfree}\ \varphi\ x\ \langle \neg R.I\ (\text{inf}_+\ \varphi)\ xs \rangle\ \langle x \notin EQ\ \varphi\ xs \rangle] ..$
have $\exists l \in LB\ \varphi\ xs. \exists u \in UB\ \varphi\ xs. l < u \wedge (\forall y. l < y \wedge y < u \longrightarrow R.I\ \varphi\ (y\#xs))$
using $\text{dense-interval}[\text{where } P = \lambda x. R.I\ \varphi\ (x\#xs), OF\ \text{finite-LB}\ \text{finite-UB}$
 $\langle l:LB\ \varphi\ xs \rangle\ \langle u:UB\ \varphi\ xs \rangle\ \langle l < x \rangle\ \langle x < u \rangle\ x\ \text{dense}[OF\ \langle \text{ngfree}\ \varphi \rangle\ \text{---}\ \langle x \notin EQ\ \varphi$
 $xs \rangle]$ **by** simp
then obtain $r\ c\ cs\ s\ d\ ds$
where $\text{Less}\ r\ (c\#cs) : \text{set}(R.\text{atoms}_0\ \varphi) \wedge c > 0 \wedge$
 $\text{Less}\ s\ (d\#ds) : \text{set}(R.\text{atoms}_0\ \varphi) \wedge d < 0 \wedge$
 $(r - \langle cs, xs \rangle) / c < (s - \langle ds, xs \rangle) / d \wedge$
 $(\forall y. (r - \langle cs, xs \rangle) / c < y \wedge y < (s - \langle ds, xs \rangle) / d \longrightarrow R.I\ \varphi\ (y\#xs))$
by blast
moreover hence $(r - \langle cs, xs \rangle) / c < \text{eval}\ (\text{between}\ (r/c, (-1/c)\ *_s\ cs)$
 $(s/d, (-1/d)\ *_s\ ds))\ xs \wedge \text{eval}\ (\text{between}\ (r/c, (-1/c)\ *_s\ cs)\ (s/d, (-1/d)\ *_s\ ds))$
 $xs < (s - \langle ds, xs \rangle) / d$
apply $(\text{auto simp add: field-simps eval-def iprod-left-add-distrib})$
apply $(\text{rule real-mult-less-iff1}[of\ 2, THEN\ iffD1])$ **apply** simp
apply simp
apply $(\text{rule real-mult-less-iff1}[of\ -d, THEN\ iffD1])$ **apply** simp
apply $(\text{simp add: ring-distrib add-divide-distrib mult-ac})$
apply $(\text{rule real-mult-less-iff1}[of\ 2, THEN\ iffD1])$ **apply** simp
apply simp
apply $(\text{rule real-mult-less-iff1}[of\ c, THEN\ iffD1])$ **apply** simp
apply $(\text{simp add: algebra-simps add-divide-distrib diff-divide-distrib})$
done
ultimately have $?FR$
by $(\text{fastsimp simp:FR}_1\text{-def}\ \text{bex-Un}\ \text{set-lbounds}\ \text{set-ubounds}\ \text{set-ebounds}\ \text{I-subst}$
 $\langle \text{ngfree}\ \varphi \rangle)$
} ultimately show $?FR$ **by** blast
qed

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

lemma $\text{qfree-FR}_1: \text{ngfree}\ \varphi \implies \text{qfree}\ (FR_1\ \varphi)$
apply $(\text{simp add:FR}_1\text{-def})$
apply $(\text{rule qfree-list-disj})$
apply $(\text{auto simp:qfree-min-inf qfree-plus-inf set-ubounds set-lbounds set-ebounds}$
 $\text{image-def qfree-map-fm})$

done

theorem *I-FR*: $R.I (FR \varphi) xs = R.I \varphi xs$
by(*simp add:I-FR₁ FR-def R.I-lift-nnf-qe qfree-FR₁*)

theorem *qfree-FR*: $qfree (FR \varphi)$
by(*simp add:FR-def R.qfree-lift-nnf-qe qfree-FR₁*)

end

theory *QElim-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*
 $let\ u = s - d*r; v = d *_{\delta} cs + ds; less = Atom(Less\ u\ v)$
 in *if* $d < 0$ *then* $less$ *else* $Or\ less\ (Atom(Eq\ u\ v))$) |
asubst-peps $r\ cs\ (Eq\ r\ (d\#ds)) = (if\ d=0\ then\ Atom(Eq\ r\ ds)\ else\ FalseF)$ |
asubst-peps $r\ cs\ a = Atom\ a$

abbreviation *subst-peps* :: $atom\ fm \Rightarrow real * real\ list \Rightarrow atom\ fm (subst_+)$
where $subst_+ \varphi\ r\ cs \equiv amap_{fm} (asubst_+ r\ cs) \varphi$

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

lemma *nolb-And[simp]*:
 $nolb (And\ f\ g)\ xs\ l\ x = (nolb\ f\ xs\ l\ x \wedge nolb\ g\ xs\ l\ x)$
apply(*clarsimp simp:nolb-def*)
apply *blast*
done

lemma *nolb-Or[simp]*:
 $nolb (Or\ f\ g)\ xs\ l\ x = (nolb\ f\ xs\ l\ x \wedge nolb\ g\ xs\ l\ x)$
apply(*clarsimp simp:nolb-def*)
apply *blast*
done

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(induct f)
  case (Atom a)
  show ?case
  proof (cases a)
    case (Less r cs)[simp]
    show ?thesis
    proof (cases cs)
      case Nil thus ?thesis using Atom by (simp add:dependsR-def)
    next
    case (Cons c cs)[simp]
    hence  $r < c*x + \langle cs, xs \rangle$  using Atom by simp
    { assume  $c=0$  hence ?thesis using Atom by simp }
    moreover
    { assume  $c < 0$ 
      hence  $x < (r - \langle cs, xs \rangle)/c$  (is - < ?u) using  $\langle r < c*x + \langle cs, xs \rangle$ 
        by (simp add: field-simps)
      have ?thesis
      proof (rule ccontr)
        assume  $\neg R.I$  (Atom a) ( $y \# xs$ )
        hence  $?u \leq y$  using Atom  $\langle c < 0 \rangle$ 
          by (auto simp add: field-simps)
        with  $\langle x < ?u \rangle$  show False using Atom  $\langle c < 0 \rangle$ 
          by(auto simp:dependsR-def)
      qed } moreover
    { assume  $c > 0$ 
      hence  $x > (r - \langle cs, xs \rangle)/c$  (is - > ?l) using  $\langle r < c*x + \langle cs, xs \rangle$ 
        by (simp add: field-simps)
      then have  $?l < y$  using Atom  $\langle c > 0 \rangle$ 
        by (auto simp:dependsR-def Ball-def nolb-def)
          (metis linorder-not-le antisym order-less-trans)
      hence ?thesis using  $\langle c > 0 \rangle$  by (simp add: field-simps)
    } ultimately show ?thesis by force
  qed
next
case (Eq r cs)[simp]
show ?thesis
proof (cases cs)
  case Nil thus ?thesis using Atom by (simp add:dependsR-def)
next
case (Cons c cs)[simp]
hence  $r = c*x + \langle cs, xs \rangle$  using Atom by simp
{ assume  $c=0$  hence ?thesis using Atom by simp }
moreover
{ assume  $c \neq 0$ 
  hence ?thesis using  $\langle r = c*x + \langle cs, xs \rangle$  Atom
    by(auto simp: mult-ac dependsR-def split:if-splits) }
ultimately show ?thesis by force
qed
qed

```

```

next
  case (And f1 f2) thus ?case by (fastsimp)
next
  case (Or f1 f2) thus ?case by (fastsimp)
qed simp+

definition EQ2 = EQ

lemma EQ2-Or[simp]: EQ2 (Or f g) xs = (EQ2 f xs Un EQ2 g xs)
by(auto simp:EQ2-def)

lemma EQ2-And[simp]: EQ2 (And f g) xs = (EQ2 f xs Un EQ2 g xs)
by(auto simp:EQ2-def)

lemma innermost-intvl2:
  [| nqfree f; nolb f xs l x; l < x; x ∉ EQ2 f xs; R.I f (x#xs); l < y; y ≤ x |]
  ⇒ R.I f (y#xs)
unfolding EQ2-def by(blast intro:innermost-intvl)

lemma I-subst-peps2:
  nqfree f ⇒ r+⟨cs,xs⟩ < x ⇒ nolb f xs (r+⟨cs,xs⟩) x
  ⇒ ∀ y ∈ {r+⟨cs,xs⟩ <.. x}. R.I f (y#xs) ∧ y ∉ EQ2 f xs
  ⇒ R.I (subst+ f (r,cs)) xs
proof(induct f)
  case FalseF thus ?case
  by simp (metis linorder-antisym-conv1 linorder-neq-iff)
next
  case (Atom a)
  show ?case
  proof(cases ((r,cs),a) rule:asubst-peps.cases)
    case (1 r cs s d ds)
    { assume d=0 hence ?thesis using Atom 1 by auto }
    moreover
    { assume d<0
      have s < d*x + ⟨ds,xs⟩ using Atom 1 by simp
      moreover have d*x < d*(r + ⟨cs,xs⟩) using ⟨d<0⟩ Atom 1
        by (simp add: mult-strict-left-mono-neg)
      ultimately have s < d * (r + ⟨cs,xs⟩) + ⟨ds,xs⟩ by (simp add: algebra-simps)
      hence ?thesis using 1
        by (auto simp add: iprod-left-add-distrib algebra-simps)
    } moreover
    { let ?L = (s - ⟨ds,xs⟩) / d let ?U = r + ⟨cs,xs⟩
      assume d>0
      hence ?U < x and ∀ y. ?U < y ∧ y < x ⟶ y ≠ ?L
        and ∀ y. ?U < y ∧ y ≤ x ⟶ ?L < y using Atom 1
        by (simp-all add: nolb-def depends_R-def Ball-def field-simps)
      hence ?L < ?U ∨ ?L = ?U
        by (metis linorder-neqE-ordered-idom real-le-refl)
      hence ?thesis using Atom 1 ⟨d>0⟩
    }
  }
end

```

```

    by (simp add: iprod-left-add-distrib field-simps)
  } ultimately show ?thesis by force
next
  case 2 thus ?thesis using Atom
    by(fastsimp simp:nolb-def EQ2-def dependsR-def field-simps split:split-if-asm)
qed (insert Atom, auto)
next
  case Or thus ?case by(simp add:Ball-def)(metis order-refl innermost-intvl2)
qed simp-all
declare[[simp-depth-limit=50]]

lemma I-subst-peps:
  ngfree f  $\implies$  R.I (subst+ f (r,cs)) xs  $\implies$ 
  ( $\exists$  leps>r+⟨cs,xs⟩.  $\forall$  x. r+⟨cs,xs⟩ < x  $\wedge$  x  $\leq$  leps  $\longrightarrow$  R.I f (x#xs))
proof(induct f)
  case TrueF thus ?case by simp (metis ordered-semidom-class.less-add-one)
next
  case (Atom a)
  show ?case
  proof (cases ((r,cs),a) rule: asubst-peps.cases)
    case (1 r cs s d ds)
    { assume d=0 hence ?thesis using Atom 1 by auto(metis ordered-semidom-class.less-add-one)
    }
    moreover
    { assume d<0
      with Atom 1 have r + ⟨cs,xs⟩ < (s - ⟨ds,xs⟩)/d (is ?a < ?b)
      by(simp add:field-simps iprod-left-add-distrib)
      then obtain x where ?a < x x < ?b by(metis dense)
      hence  $\forall$  y. ?a < y  $\wedge$  y  $\leq$  x  $\longrightarrow$  s < d*y + ⟨ds,xs⟩
      using ⟨d<0⟩ by (simp add:field-simps)
      (metis add-le-cancel-right mult-le-cancel-left real-le-antisym real-le-linear real-mult-commute
      xt1(8))
      hence ?thesis using 1 ⟨?a<x⟩ by auto
    } moreover
    { let ?a = s - d * r let ?b = ⟨d *s cs + ds,xs⟩
      assume d>0
      with Atom 1 have ?a < ?b  $\vee$  ?a = ?b by auto
      hence ?thesis
      proof
        assume ?a = ?b
        thus ?thesis using ⟨d>0⟩ Atom 1
          by(simp add:field-simps iprod-left-add-distrib)
          (metis add-0-left add-less-cancel-right right-distrib real-mult-commute
          real-mult-less-mono2)
      }
    }
  }
  next
  assume ?a < ?b
  { fix x assume r+⟨cs,xs⟩ < x  $\wedge$  x  $\leq$  r+⟨cs,xs⟩ + 1
    hence d*(r + ⟨cs,xs⟩) < d*x
    using ⟨d>0⟩ by(metis real-mult-less-mono2)
  }

```

```

    hence  $s < d*x + \langle ds, xs \rangle$  using  $\langle d > 0 \rangle \langle ?a < ?b \rangle$ 
    by (simp add: algebra-simps iprod-left-add-distrib)
  }
  thus ?thesis using 1  $\langle d > 0 \rangle$ 
  by (force simp: iprod-left-add-distrib)
qed
} ultimately show ?thesis by (metis less-linear)
qed (insert Atom, auto split: split-if-asm intro: ordered-semidom-class.less-add-one)
next
case And thus ?case
  apply clarsimp
  apply (rule-tac  $x = \min \text{leps lepsa}$  in exI)
  apply simp
  done
next
case Or thus ?case by force
qed simp-all

```

lemma dense-interval:
assumes *finite* L $l \in L$ $l < x$ $P(x::\text{real})$
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 –
 let $?L = \{l \in L. l < x\}$
 let $?ll = \text{Max } ?L$
 have $?L \neq \{\}$ using $\langle l \in L \rangle \langle l < x \rangle$ by (blast intro: order-less-imp-le)
 hence $\forall y. ?ll < y \wedge y < x \longrightarrow y \notin L$ using *finite* L by force
 moreover have $?ll \in L$
proof
 show $?ll \in ?L$ using *finite* L $\text{Max-in}[OF - \langle ?L \neq \{\} \rangle]$ by simp
 show $?L \subseteq L$ by blast
qed
 moreover have $?ll < x$ using *finite* L $\langle ?L \neq \{\} \rangle$ by simp
 ultimately show ?thesis using $\langle l < x \rangle \langle ?L \neq \{\} \rangle$
 by (blast intro!: dense greaterThanLessThan-iff[THEN iffD1])
qed

definition
 $qe\text{-eps}_1(f) =$
 (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** $R.I (qe\text{-eps}_1 f) xs = (\exists x. R.I f (x \# xs))$
 (is $?QE = ?EX$)
proof
 let $?as = R.\text{atoms}_0 f$ let $?ebs = \text{ebounds } ?as$
 assume $?QE$
 { assume $R.I (\text{inf}_- f) xs$

```

    hence ?EX using ⟨?QE⟩ min-inf[of f xs] ⟨nqfree f⟩
      by(auto simp add:qe-eps1-def amap-fm-list-disj)
  } moreover
  { assume ∀ x ∈ EQ f xs. ¬R.I f (x#xs)
    ¬ R.I (inf_ f) xs
    with ⟨?QE⟩ ⟨nqfree f⟩ obtain r cs where R.I (subst+ f (r,cs)) xs
      by(fastsimp simp:qe-eps1-def set-ebounds diff-divide-distrib eval-def
        diff-minus[symmetric] I-subst ⟨nqfree f⟩)
    then obtain leps where R.I f (leps#xs)
      using I-subst-peps[OF ⟨nqfree f⟩] by fastsimp
    hence ?EX .. }
  ultimately show ?EX by blast
next
let ?as = R.atoms0 f let ?ebs = ebounds ?as
assume ?EX
then obtain x where x: R.I f (x#xs) ..
{ assume R.I (inf_ f) xs
  hence ?QE using ⟨nqfree f⟩ by(auto simp:qe-eps1-def)
} moreover
{ assume ∃ rcs ∈ set ?ebs. R.I (subst f rcs) xs
  hence ?QE by(auto simp:qe-eps1-def) } moreover
{ assume ¬ R.I (inf_ f) xs
  and ∀ rcs ∈ set ?ebs. ¬ R.I (subst f rcs) xs
  hence noE: ∀ e ∈ EQ f xs. ¬ R.I f (e#xs) using ⟨nqfree f⟩
    by (force simp:set-ebounds I-subst diff-divide-distrib eval-def
      diff-minus[symmetric] split:split-if-asm)
  hence x ∉ EQ f xs using x by fastsimp
  obtain l where l ∈ LB f xs l < x
    using LBex[OF ⟨nqfree f⟩] x ⟨¬ R.I (inf_ f) xs⟩ ⟨x ∉ EQ f xs⟩ ..
  have ∃ l ∈ LB f xs. l < x ∧ nolb f xs l x ∧
    (∀ y. l < y ∧ y ≤ x ⟶ R.I f (y#xs))
    using dense-interval[where P = λx. R.I f (x#xs), OF finite-LB ⟨l ∈ LB f xs⟩
      ⟨l < x⟩ x] innermost-intvl[OF ⟨nqfree f⟩] - - ⟨x ∉ EQ f xs⟩]
    by (simp add:nolb-def)
  then obtain r c cs
    where Less r (c#cs) ∈ set(R.atoms0 f) ∧ c > 0 ∧
      (r - ⟨cs,xs⟩)/c < x ∧ nolb f xs ((r - ⟨cs,xs⟩)/c) x
      ∧ (∀ y. (r - ⟨cs,xs⟩)/c < y ∧ y ≤ x ⟶ R.I f (y#xs))
    by blast
  then moreover have R.I (subst+ f (r/c, (-1/c) *s cs)) xs using noE
    by(auto intro!: I-subst-peps2[OF ⟨nqfree f⟩]
      simp:EQ2-def diff-divide-distrib algebra-simps)
  ultimately have ?QE
    by(simp add:qe-eps1-def bex-Un set-lbounds)metis
  } ultimately show ?QE by blast
qed

```

lemma qfree-astubst-peps: qfree (astubst₊ rcs a)
 by(cases (rcs,a) rule:astubst-peps.cases) simp-all

```

lemma qfree-subst-peps:  $nqfree\ \varphi \implies qfree\ (subst_+\ \varphi\ rcs)$ 
by(induct  $\varphi$ ) (simp-all add:qfree-asubst-peps)

lemma qfree-qe-eps1:  $nqfree\ \varphi \implies qfree(qe-eps_1\ \varphi)$ 
apply(simp add:qe-eps1-def)
apply(rule qfree-list-disj)
apply (auto simp:qfree-min-inf qfree-subst-peps qfree-map-fm)
done

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

lemma qfree-qe-eps:  $qfree(qe-eps\ \varphi)$ 
by(simp add: qe-eps-def R.qfree-lift-nnf-qe qfree-qe-eps1)

lemma I-qe-eps:  $R.I\ (qe-eps\ \varphi)\ xs = R.I\ \varphi\ xs$ 
by(simp add:qe-eps-def R.I-lift-nnf-qe qfree-qe-eps1 I-eps1)

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 $decr_Z :: atom \Rightarrow atom$ **where**
 $decr_Z (Le\ i\ ks) = Le\ i\ (tl\ ks) \mid$
 $decr_Z (Dvd\ d\ i\ ks) = Dvd\ d\ i\ (tl\ ks) \mid$
 $decr_Z (NDvd\ d\ i\ ks) = NDvd\ d\ i\ (tl\ ks)$

fun $I_Z :: atom \Rightarrow int\ list \Rightarrow bool$ **where**
 $I_Z (Le\ i\ ks)\ xs = (i \leq \langle ks, xs \rangle) \mid$
 $I_Z (Dvd\ d\ i\ ks)\ xs = (d\ dvd\ i + \langle ks, xs \rangle) \mid$
 $I_Z (NDvd\ d\ i\ ks)\ xs = (\neg\ d\ dvd\ i + \langle ks, xs \rangle)$

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

interpretation $Z!$:

$ATOM\ neg_Z (\lambda a. divisor\ a \neq 0)\ I_Z (\lambda a. hd-coeff\ a \neq 0)\ decr_Z$
where $ATOM.atoms_0 (\lambda a. hd-coeff\ a \neq 0) = atoms_0$

proof –

case $goal1$
thus $?case$
apply $(unfold-locales)$
apply $(case-tac\ a)$
apply $simp-all$
apply $(case-tac\ a)$
apply $simp-all$
apply $(case-tac\ a)$
apply $(simp-all)$
apply $arith$
apply $(case-tac\ a)$
apply $(simp-all\ add: split: list.splits)$
apply $(case-tac\ a)$
apply $simp-all$
done
next
case $goal2$ **thus** $?case$ **by** $(simp\ add: atoms_0-def)$
qed

setup $\ll Sign.revert-abbrev\ @\{const-name\ Z.I\} \gg$

setup $\ll Sign.revert-abbrev\ @\{const-name\ Z.lift-dnf-qe\} \gg$

abbreviation

$hd-coeff-is1\ a \equiv$

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

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 $IZ-asubst: I_Z\ (asubst\ i\ ks\ a)\ xs = I_Z\ a\ ((i + \langle ks, xs \rangle) \#\ xs)$
apply $(cases\ a)$
apply $(case-tac\ list)$
apply $(simp-all\ add:algebra-simps\ iprod-left-add-distrib)$
apply $(case-tac\ list)$
apply $(simp-all\ add:algebra-simps\ iprod-left-add-distrib)$
apply $(case-tac\ list)$
apply $(simp-all\ add:algebra-simps\ iprod-left-add-distrib)$
done

lemma $I-subst:$
 $qfree\ \varphi \Longrightarrow Z.I\ \varphi\ ((i + \langle ks, xs \rangle) \#\ xs) = Z.I\ (subst\ i\ ks\ \varphi)\ xs$
by $(induct\ \varphi)\ (simp-all\ add:IZ-asubst)$

lemma $divisor-asubst[simp]: divisor\ (asubst\ i\ ks\ a) = divisor\ a$
by $(induct\ i\ ks\ a\ rule:asubst.induct)\ auto$

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 $set-lbounds:$

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

by $(auto\ simp: lbounds-def\ split:list.splits\ atom.splits\ if-splits)$

lemma $set-ubounds:$

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

by $(auto\ simp: ubounds-def\ split:list.splits\ atom.splits\ if-splits)$

lemma $lbounds-append[simp]: lbounds(as\ @\ bs) = lbounds\ as\ @\ lbounds\ bs$
by $(simp\ add:lbounds-def)$

6.2 LCM and lemmas

lemma $zdiv-eq-0-iff:$

$(i :: int)\ div\ k = 0 \iff k = 0 \vee 0 \leq i \wedge i < k \vee i \leq 0 \wedge k < i$

apply $(auto\ simp: div-pos-pos-trivial\ div-neg-neg-trivial)$

apply $(metis\ int-0-neq-1\ linorder-not-less\ neg-imp-zdiv-nonneg-iff\ zdiv-mono1\ zdiv-self\ zero-le-one\ zle-antisym\ zle-refl\ zless-linear)$

apply $(metis\ int-0-neq-1\ linorder-not-less\ pos-imp-zdiv-nonneg-iff\ zdiv-mono1-neg\ zdiv-self\ zero-le-one\ zle-antisym\ zle-refl\ zless-linear)$

apply $(metis\ int-0-neq-1\ zdiv-self\ zless-linear)$

done

lemma $pos-imp-zdiv-pos-iff:$

$0 < k \implies 0 < (i :: \text{int}) \text{ div } k \iff k \leq i$
using *pos-imp-zdiv-nonneg-iff*[of k i] *zdiv-eq-0-iff*[of i k]
by *arith*

lemma *zmod-le-nonneg-dividend*: $(m :: \text{int}) \geq 0 \implies m \bmod k \leq m$
apply(*cases k=0*) **apply** *simp*
by(*metis linorder-not-le mod-pos-pos-trivial neg-mod-conj pos-mod-conj zle-refl zle-trans zless-le*)

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

lemma *dvd-zlcms*: $i : \text{set is} \implies i \text{ dvd } \text{zlcms is}$
by(*induct is*) *auto*

lemma *zlcms-pos*: $\forall i \in \text{set is. } i \neq 0 \implies \text{zlcms is} > 0$
by(*induct is*)(*auto simp:lcm-pos-int*)

lemma *zlcms0-iff*[*simp*]: $(\text{zlcms is} = 0) = (0 : \text{set is})$
by (*metis DIVISION-BY-ZERO dvd-eq-mod-eq-0 dvd-zlcms zlcms-pos zless-le*)

lemma *elem-le-zlcms*: $\forall i \in \text{set is. } i \neq 0 \implies i : \text{set is} \implies i \leq \text{zlcms is}$
by (*metis dvd-zlcms zdvd-imp-le zlcms-pos*)

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 [$k \neq 0; k \text{ dvd } m$] \implies
 $I_Z (\text{hd-coeff1 } m (\text{Dvd } d \ i \ (k\#ks))) (x\#e) = (\text{let } m' = m \text{ div } (\text{abs } k) \text{ in}$
 $I_Z (\text{Dvd } (m'*d) (m'*i) (\text{sgn } k \ \# \ (m' *_s \ ks))) (x\#e))$
apply(*auto simp:algebra-simps abs-if sgn-if*)
apply(*simp add: zdiv-zminus2-eq-if dvd-eq-mod-eq-0[THEN iffD1] algebra-simps*)
apply (*metis diff-minus comm-monoid-add.mult-left-commute dvd-minus-iff minus-add-distrib*)
apply(*simp add: zdiv-zminus2-eq-if dvd-eq-mod-eq-0[THEN iffD1] algebra-simps*)

apply (*metis diff-minus comm-monoid-add.mult-left-commute dvd-minus-iff minus-add-distrib*)
done

lemma *I-hd-coeff1-mult-a*: **assumes** $m > 0$
shows $hd\text{-coeff } a \text{ dvd } m \mid hd\text{-coeff } a = 0 \implies I_Z (hd\text{-coeff1 } m \ a) (m*x\#xs) = I_Z$
 $a \ (x\#xs)$
proof (*induct a*)
 case (*Le i ks*) [*simp*]
 show ?*case*
 proof (*cases ks*)
 case *Nil* **thus** ?*thesis* **by** *simp*
 next
 case (*Cons k ks'*) [*simp*]
 show ?*thesis*
 proof *cases*
 assume $k=0$ **thus** ?*thesis* **by** *simp*
 next
 assume $k \neq 0$
 with *Le* **have** $|k| \text{ dvd } m$ **by** *simp*
 let $?m' = m \text{ div } |k|$
 have $?m' > 0$ **using** $\langle |k| \text{ dvd } m \rangle$ *pos-imp-zdiv-pos-iff* $\langle m > 0 \rangle$ $\langle k \neq 0 \rangle$
 by (*simp add: zdiv-imp-le*)
 have $1: k*(x*?m') = \text{sgn } k * x * m$
 proof –
 have $k*(x*?m') = (\text{sgn } k * \text{abs } k) * (x * ?m')$
 by (*simp only: mult-sgn-abs*)
 also have $\dots = \text{sgn } k * x * (\text{abs } k * ?m')$ **by** *simp*
 also have $\dots = \text{sgn } k * x * m$
 using *zdiv-mult-div-cancel[OF* $\langle |k| \text{ dvd } m \rangle$ **by** (*simp add: algebra-simps*)
 finally show ?*thesis* .
 qed
 have $I_Z (hd\text{-coeff1 } m \ (Le \ i \ ks)) (m*x\#xs) \longleftrightarrow$
 $(i*?m' \leq \text{sgn } k * m*x + ?m' * \langle ks',xs \rangle)$
 using $\langle k \neq 0 \rangle$ **by** (*simp add: algebra-simps*)
 also have $\dots \longleftrightarrow ?m'*i \leq ?m' * (k*x + \langle ks',xs \rangle)$ **using** 1
 by (*simp (no-asm-simp) add: algebra-simps*)
 also have $\dots \longleftrightarrow i \leq k*x + \langle ks',xs \rangle$ **using** $\langle ?m' > 0 \rangle$
 by (*simp add: mult-compare-simps*)
 finally show ?*thesis* **by** (*simp*)
 qed
qed
next
 case (*Dvd d i ks*) [*simp*]
 show ?*case*
 proof (*cases ks*)
 case *Nil* **thus** ?*thesis* **by** *simp*
 next
 case (*Cons k ks'*) [*simp*]

```

show ?thesis
proof cases
  assume k=0 thus ?thesis by simp
next
  assume k≠0
  with Dvd have k dvd m by simp
  let ?m' = m div k
  have ?m' ≠ 0 using ⟨k dvd m⟩ zdiv-eq-0-iff ⟨m>0⟩ ⟨k≠0⟩
    by(simp add:linorder-not-less zdvd-imp-le)
  have 1: k*(x*?m') = x * m
  proof -
    have k*(x*?m') = x*(k*?m') by(simp add:algebra-simps)
    also have ... = x*m using zdvd-mult-div-cancel[OF ⟨k dvd m⟩]
      by(simp add:algebra-simps)
    finally show ?thesis .
  qed
  have I_Z (hd-coeff1 m (Dvd d i ks)) (m*x#xs) ⟷
    (?m'*d dvd ?m'*i + m*x + ?m' * ⟨ks',xs⟩)
    using ⟨k≠0⟩ by(simp add: algebra-simps)
  also have ... ⟷ ?m'*d dvd ?m' * (i + k*x + ⟨ks',xs⟩) using 1
    by(simp (no-asm-simp) add:algebra-simps)
  also have ... ⟷ d dvd i + k*x + ⟨ks',xs⟩ using (?m'≠0) by(simp)
  finally show ?thesis by(simp add:algebra-simps)
qed
qed
next
case (NDvd d i ks)[simp]
show ?case
proof(cases ks)
  case Nil thus ?thesis by simp
next
case (Cons k ks')[simp]
show ?thesis
proof cases
  assume k=0 thus ?thesis by simp
next
  assume k≠0
  with NDvd have k dvd m by simp
  let ?m' = m div k
  have ?m' ≠ 0 using ⟨k dvd m⟩ zdiv-eq-0-iff ⟨m>0⟩ ⟨k≠0⟩
    by(simp add:linorder-not-less zdvd-imp-le)
  have 1: k*(x*?m') = x * m
  proof -
    have k*(x*?m') = x*(k*?m') by(simp add:algebra-simps)
    also have ... = x*m using zdvd-mult-div-cancel[OF ⟨k dvd m⟩]
      by(simp add:algebra-simps)
    finally show ?thesis .
  qed
  have I_Z (hd-coeff1 m (NDvd d i ks)) (m*x#xs) ⟷

```

```

       $\neg(?m'*d \text{ dvd } ?m'*i + m*x + ?m' * \langle ks',xs \rangle)$ 
    using  $\langle k \neq 0 \rangle$  by(simp add: algebra-simps)
  also have ...  $\longleftrightarrow \neg ?m'*d \text{ dvd } ?m' * (i + k*x + \langle ks',xs \rangle)$  using 1
    by(simp (no-asm-simp) add: algebra-simps)
  also have ...  $\longleftrightarrow \neg d \text{ dvd } i + k*x + \langle ks',xs \rangle$  using  $\langle ?m' \neq 0 \rangle$  by(simp)
  finally show ?thesis by(simp add: algebra-simps)
qed
qed
qed

```

```

lemma I-hd-coeff1-mult: assumes  $m > 0$ 
shows  $qfree \varphi \implies \forall a \in \text{set}(Z.\text{atoms}_0 \varphi). \text{hd-coeff } a \text{ dvd } m \implies$ 
   $Z.I (\text{map}_{f_m} (\text{hd-coeff1 } m) \varphi) (m*x\#xs) = Z.I \varphi (x\#xs)$ 
proof(induct  $\varphi$ )
  case (Atom a)
  thus ?case using I-hd-coeff1-mult-a[OF  $\langle m > 0 \rangle$ ] by(simp split: split-if-asm)
qed simp-all
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
   $(\exists x. \forall a \in \text{set}(hd-coeffs1 \text{ as}). I_Z a (x\#xs)) =$ 
   $(\exists x. \forall a \in \text{set } as. I_Z a (x\#xs))$  (is ?B = ?A)
proof -
  let ?m = zlcms(map hd-coeff as)
  have ?m > 0 using 0 by(simp add: zlcms-pos)
  have ?A =  $(\exists x. \forall a \in \text{set } as. I_Z (hd-coeff1 ?m a) (?m*x\#xs))$ 
    by (simp add: I-hd-coeff1-mult-a[OF  $\langle ?m > 0 \rangle$ ] dvd-zlcms 0)
  also have ... =  $(\exists x. ?m \text{ dvd } x+0 \wedge (\forall a \in \text{set } as. I_Z (hd-coeff1 ?m a) (x\#xs)))$ 
    by(rule unity-coeff-ex[THEN meta-eq-to-obj-eq])
  finally show ?thesis by(simp add: hd-coeffs1-def)
qed

```

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] (λn. list-conj(map (Atom ∘ asubst n []) ds))
     else
       Disj ls (λ(li,lks).
         Disj [0..d - 1] (λn.
           list-conj(map (Atom ∘ asubst (li + n) (-lks)) as))))

```

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

lemma *I-cyclic*:

assumes $is-dvd\ a$ **and** $hd-coeff\ a = 1$ **and** $i\ mod\ divisor\ a = j\ mod\ divisor\ a$

shows $I_Z\ a\ (i\ \#e) = I_Z\ a\ (j\ \#e)$

proof (cases a)

case (Dvd d l ks)

with $\langle hd-coeff\ a = 1 \rangle$ **obtain** ks' **where** [simp]: $ks = 1\ \#ks'$

by(simp split:list.splits)

have $(l + (i + \langle ks',e \rangle))\ mod\ d = (l + (j + \langle ks',e \rangle))\ mod\ d$ (**is** $?l=?r$)

proof -

have $?l = (l\ mod\ d + (i + \langle ks',e \rangle)\ mod\ d)\ mod\ d$

by(rule mod-add-eq)

also have $(i + \langle ks',e \rangle)\ mod\ d = (i\ mod\ d + \langle ks',e \rangle\ mod\ d)\ mod\ d$

by(rule mod-add-eq)

also have $i\ mod\ d = j\ mod\ d$

using $\langle i\ mod\ divisor\ a = j\ mod\ divisor\ a \rangle$ Dvd **by** simp

also have $(j\ mod\ d + \langle ks',e \rangle\ mod\ d)\ mod\ d = (j + \langle ks',e \rangle)\ mod\ d$

by(rule mod-add-eq[symmetric])

also have $(l\ mod\ d + (j + \langle ks',e \rangle)\ mod\ d)\ mod\ d = ?r$

by(rule mod-add-eq[symmetric])

finally show $?thesis$.

qed

thus $?thesis$ **using** Dvd **by** (simp add:dvd-eq-mod-eq-0)

next

case (NDvd d l ks)

with $\langle hd-coeff\ a = 1 \rangle$ **obtain** ks' **where** [simp]: $ks = 1\ \#ks'$

by(simp split:list.splits)

have $(l + (i + \langle ks',e \rangle))\ mod\ d = (l + (j + \langle ks',e \rangle))\ mod\ d$ (**is** $?l=?r$)

proof -

have $?l = (l\ mod\ d + (i + \langle ks',e \rangle)\ mod\ d)\ mod\ d$

by(rule mod-add-eq)

also have $(i + \langle ks',e \rangle)\ mod\ d = (i\ mod\ d + \langle ks',e \rangle\ mod\ d)\ mod\ d$

by(rule mod-add-eq)

also have $i\ mod\ d = j\ mod\ d$

using $\langle i\ mod\ divisor\ a = j\ mod\ divisor\ a \rangle$ NDvd **by** simp

also have $(j\ mod\ d + \langle ks',e \rangle\ mod\ d)\ mod\ d = (j + \langle ks',e \rangle)\ mod\ d$

```

    by(rule mod-add-eq[symmetric])
  also have (l mod d + (j + ⟨ks',e⟩) mod d) mod d = ?r
    by(rule mod-add-eq[symmetric])
  finally show ?thesis .
qed
thus ?thesis using NDvd by (simp add:dvd-eq-mod-eq-0)
next
case Le thus ?thesis using ⟨is-dvd a⟩ by simp
qed

```

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

proof –

let ?lbs = lbounds as

let ?ds = filter is-dvd as

let ?lcm = zlcms(map divisor ?ds)

let ?Ds = {a ∈ set as. is-dvd a}

let ?Us = {a ∈ set as. case a of Le - (k#-) ⇒ k < 0 | - ⇒ False}

let ?Ls = {a ∈ set as. case a of Le - (k#-) ⇒ k > 0 | - ⇒ False}

have as: set as = ?Ds ∪ ?Ls ∪ ?Us (is - = ?S)

proof –

{ fix x assume x ∈ set as

hence x ∈ ?S using hd by (cases x)(auto split:list.splits) }

moreover

{ fix x assume x ∈ ?S

hence x ∈ set as by auto }

ultimately show ?thesis by blast

qed

have 1: $\forall a \in ?Ds. \text{hd-coeff } a = 1$ using hd by (fastsimp split:atom.splits)

show ?thesis (is ?QE = $(\exists x. ?P x)$)

proof

assume ?QE

{ assume ?lbs = []

with ⟨?QE⟩ obtain n where n < ?lcm and

A: $\forall a \in ?Ds. I_Z a (n \# xs)$ using 1

by (auto simp:IZ-asubst qe-pres₁-def)

have ?Ls = {} using ⟨?lbs = []⟩ set-lbounds[of as]

by (auto simp add:filter-empty-conv split:atom.split list.split)

have $\exists x. ?P x$

proof cases

assume ?Us = {}

with ⟨?Ls = {}⟩ have set as = ?Ds using as by (simp (no-asm-use))blast

hence ?P n using A by auto

thus ?thesis ..

next

assume ?Us ≠ {}

let ?M = {⟨tl ks, xs⟩ - i | ks i. Le i ks ∈ ?Us} let ?m = Min ?M

```

have finite ?M
proof -
  have finite ( (λLe i ks ⇒ ⟨tl ks, xs⟩ - i) ‘
    {a∈set as. ∃ i k ks. k<0 ∧ a = Le i (k#ks)} )
    (is finite ?B)
  by simp
  also have ?B = ?M using hd
  by(fastsimp simp:image-def neq-Nil-conv split:atom.splits list.splits)
  finally show ?thesis by auto
qed
have ?M ≠ {}
proof -
  from ⟨?Us ≠ {}⟩ obtain i k ks where Le i (k#ks) ∈ ?Us ∧ k<0
  by (fastsimp split:atom.splits list.splits)
  thus ?thesis by auto
qed
let ?k = (n - ?m) div ?lcm + 1 let ?x = n - ?k * ?lcm
have ∀ a ∈ ?Ds. I_Z a (?x # xs)
proof (intro allI ballI)
  fix a assume a ∈ ?Ds
  let ?d = divisor a
  have ?l: ?d dvd ?lcm using ⟨a ∈ ?Ds⟩ by(simp add:dvd-zlcms)
  have ?x mod ?d = n mod ?d (is ?l = ?r)
  proof -
    have ?l = (?r - ((?k * ?lcm) mod ?d)) mod ?d
    by(rule mod-diff-eq)
    also have (?k * ?lcm) mod ?d = 0
    by(simp add: dvd-eq-mod-eq-0[symmetric] dvd-mult[OF ?l])
    finally show ?thesis by simp
  qed
  thus I_Z a (?x#xs) using A I-cyclic[of a n ?x] ⟨a ∈ ?Ds⟩ 1 by auto
qed
moreover
have ∀ a ∈ ?Us. I_Z a (?x#xs)
proof
  fix a assume a ∈ ?Us
  then obtain l ks where [simp]: a = Le l (-1#ks) using hd
  by(fastsimp split:atom.splits list.splits)
  have ?m ≤ ⟨ks,xs⟩ - l
  using Min-le-iff[OF ⟨finite ?M⟩ ⟨?M ≠ {}⟩] ⟨a ∈ ?Us⟩ by fastsimp
  moreover have (n - ?m) mod ?lcm < ?lcm
  by(simp add: pos-mod-bound[OF zlcms-pos] norm)
  ultimately show I_Z a (?x#xs)
  by(simp add:zmult-div-cancel algebra-simps)
qed
moreover
have set as = ?Ds ∪ ?Us using as ⟨?Ls = {}⟩
by (simp (no-asm-use)) blast
ultimately have ?P(?x) by auto

```

```

    thus ?thesis ..
  qed }
moreover
{ assume ?lbs ≠ []
  with ⟨?QE⟩ obtain il ksl m
    where ∀ a ∈ set as. I_Z (asubst (il + m) ksl a) xs
    by(auto simp:qe-pres1-def)
  hence ?P(il + m + ⟨ksl,xs⟩) by(simp add:IZ-asubst)
  hence ∃ x. ?P x .. }
ultimately show ∃ x. ?P x by blast
next
assume ∃ x. ?P x then obtain x where x: ?P x ..
show ?QE
proof cases
  assume ?lbs = []
  moreover
  have ∃ x. 0 ≤ x ∧ x < ?lcm ∧ (∀ a ∈ ?Ds. I_Z a (x # xs))
    (is ∃ x. ?P x)
  proof
    { fix a assume a ∈ ?Ds
      hence I_Z a ((x mod ?lcm) # xs) = I_Z a (x # xs) using 1
      by (fastsimp del:iffI intro: I-cyclic
        simp: mod-mod-cancel dvd-zlcms) }
    thus ?P(x mod ?lcm) using x norm by(simp add: zlcms-pos)
  }
  qed
ultimately show ?thesis by (auto simp:qe-pres1-def IZ-asubst)
next
assume ?lbs ≠ []
let ?L = {i - ⟨ks,xs⟩ | ks i. (i,ks) ∈ set(lbounds as)}
let ?lm = Max ?L
let ?n = (x - ?lm) mod ?lcm
have finite ?L
proof -
  have finite((λ(i,ks). i - ⟨ks,xs⟩) ` set(lbounds as) ) (is finite ?B)
  by simp
  also have ?B = ?L by auto
  finally show ?thesis by auto
qed
moreover have ?L ≠ {} using ⟨?lbs ≠ []⟩
  by(fastsimp simp:neq-Nil-conv)
ultimately have ?lm ∈ ?L by(rule Max-in)
then obtain li lks where (li,lks) ∈ set ?lbs and lm: ?lm = li - ⟨lks,xs⟩
  by blast
moreover
have n: 0 ≤ ?n ∧ ?n < ?lcm using norm by(simp add:zlcms-pos)
moreover
{ fix a assume a ∈ set as
  with x have I_Z a (x # xs) by blast
  have I_Z a ((li + ?n - ⟨lks,xs⟩) # xs)

```

```

proof -
  { assume a ∈ ?Ls
    then obtain i ks where [simp]: a = Le i (1#ks) using hd
      by(fastsimp split:atom.splits list.splits)
    from ⟨a ∈ ?Ls⟩ have i-⟨ks,xs⟩ ∈ ?L by(fastsimp simp:set-lbounds)
    hence i-⟨ks,xs⟩ ≤ li - ⟨lks,xs⟩
      using lm[symmetric] ⟨finite ?L⟩ ⟨?L ≠ {}⟩ by auto
    hence ?thesis using n by simp }
  moreover
  { assume a ∈ ?Us
    then obtain i ks where [simp]: a = Le i (-1#ks) using hd
      by(fastsimp split:atom.splits list.splits)
    have Le li (1#lks) ∈ set as using ⟨(li,lks) ∈ set ?lbs⟩ hd
      by(auto simp:set-lbounds)
    hence li - ⟨lks,xs⟩ ≤ x using x by auto
    hence (x - ?lm) mod ?lcm ≤ x - ?lm
      using lm by(simp add: zmod-le-nonneg-dividend)
    hence ?thesis using ⟨I_Z a (x # xs)⟩ lm by auto }
  moreover
  { assume a ∈ ?Ds
    have ?thesis
      proof(rule I-cyclic[THEN iffD2, OF - - - ⟨I_Z a (x # xs)⟩])
        show is-dvd a using ⟨a ∈ ?Ds⟩ by simp
        show hd-coeff a = 1 using ⟨a ∈ ?Ds⟩ hd
          by(fastsimp split:atom.splits list.splits)
        have li + (x-?lm) mod ?lcm - ⟨lks,xs⟩ = ?lm + (x-?lm) mod ?lcm
          using lm by arith
        hence (li + (x-?lm) mod ?lcm - ⟨lks,xs⟩) mod divisor a =
          (?lm + (x-?lm) mod ?lcm) mod divisor a by (simp only:)
        also have ... =
          (?lm mod divisor a + (x-?lm) mod ?lcm mod divisor a) mod divisor a
          by(rule mod-add-eq)
        also have
          ... = (?lm mod divisor a + (x-?lm) mod divisor a) mod divisor a
          using ⟨is-dvd a⟩ ⟨a ∈ set as⟩
          by(simp add: mod-mod-cancel dvd-zlcms)
        also have ... = (?lm + (x-?lm)) mod divisor a
          by(rule mod-add-eq[symmetric])
        also have ... = x mod divisor a by simp
        finally
        show (li + ?n - ⟨lks,xs⟩) mod divisor a = x mod divisor a
          using norm by(auto simp:pos-mod-sign zlcms-pos)
        qed }
    ultimately show ?thesis using ⟨a ∈ set as⟩ as by blast
  qed
}
ultimately show ?thesis using ⟨?lbs ≠ []⟩
  by (simp (no-asm-simp) add:qe-pres1-def IZ-asubst split-def)
    (force simp del:int-nat-eq)

```

qed
 qed
 qed

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 –

let *?m* = *zlcms*(*map* *hd-coeff* *as*)

from *a* **have** $a = \text{Dvd } ?m \ 0 \ [1] \vee (\exists b \in \text{set } as. a = \text{hd-coeff1 } ?m \ b)$

(**is** *?A* \vee *?B*)

by(*auto simp:hd-coeffs1-def*)

thus *?thesis*

proof

assume *?A* **thus** *?thesis* **using** *hd0* **by**(*auto*)

next

assume *?B*

then obtain *b* **where** $b \in \text{set } as$ **and** [*simp*]: $a = \text{hd-coeff1 } ?m \ b \ ..$

hence *b*: $\text{hd-coeff } b \neq 0$ *divisor* $b \neq 0$ **using** *div0* *hd0* **by** *auto*

show *?thesis*

proof (*cases* *b*)

case (*Le* *i* *ks*) **thus** *?thesis* **using** *b* **by**(*auto split:list.splits*)

next

case (*Dvd* *d* *i* *ks*)[*simp*]

then obtain *k* *ks'* **where** [*simp*]: $ks = k \# ks'$ **using** *b*

by(*auto split:list.splits*)

have *k*: $k \in \text{set}(\text{map } \text{hd-coeff } as)$ **using** $\langle b \in \text{set } as \rangle$ **by** *force*

have *zlcms* (*map* *hd-coeff* *as*) *div* *k* $\neq 0$

using *b* *hd0* *dvd-zlcms[OF* *k*]

by(*auto simp add:dvd-def*)

thus *?thesis* **using** *b* **by** (*simp*)

next

case (*NDvd* *d* *i* *ks*)[*simp*]

then obtain *k* *ks'* **where** [*simp*]: $ks = k \# ks'$ **using** *b*

by(*auto split:list.splits*)

have *k*: $k \in \text{set}(\text{map } \text{hd-coeff } as)$ **using** $\langle b \in \text{set } as \rangle$ **by** *force*

have *zlcms* (*map* *hd-coeff* *as*) *div* *k* $\neq 0$

using *b* *hd0* *dvd-zlcms[OF* *k*]

by(*auto simp add:dvd-def*)

thus *?thesis* **using** *b* **by** (*simp*)

qed

qed

qed

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 –

let *?m* = *zlcms*(*map* *hd-coeff* *as*)

```

from  $a$  have  $a = \text{Dvd } ?m \ 0 \ [1] \vee (\exists b \in \text{set } as. a = \text{hd-coeff1 } ?m \ b)$ 
  (is  $?A \vee ?B$ )
  by(auto simp:hd-coeffs1-def)
thus  $?thesis$ 
proof
  assume  $?A$  thus  $?thesis$  using hd0 by simp
next
  assume  $?B$ 
  then obtain  $b$  where  $b \in \text{set } as$  and [simp]:  $a = \text{hd-coeff1 } ?m \ b ..$ 
  hence  $b: \text{hd-coeff } b \neq 0$  using hd0 by auto
  show  $?thesis$  using  $b$ 
    by (cases b) (auto simp:sgn-if split:list.splits)
qed
qed

```

```

lemma I-qe-pres1-o:
   $\llbracket \forall a \in \text{set } as. \text{divisor } a \neq 0; \forall a \in \text{set } as. \text{hd-coeff } a \neq 0 \rrbracket \implies$ 
   $Z.I ((qe-pres_1 \circ \text{hd-coeffs1}) \text{ as}) \ e = (\exists x. \forall a \in \text{set } as. I_Z \ a \ (x\#e))$ 
apply(simp)
apply(subst I-qe-pres1)
  apply(simp add:divisors-hd-coeffs1)
  apply(simp add:hd-coeff-is1-hd-coeffs1)
using I-hd-coeffs1 apply(simp)
done

```

```

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

```

```

lemma qfree-qe-pres-o:  $qfree ((qe-pres_1 \circ \text{hd-coeffs1}) \text{ as})$ 
by(auto simp:qe-pres1-def intro!:qfree-list-disj)

```

```

lemma normal-qe-pres1-o:
   $\forall a \in \text{set } as. \text{hd-coeff } a \neq 0 \wedge \text{divisor } a \neq 0 \implies$ 
   $Z.\text{normal } ((qe-pres_1 \circ \text{hd-coeffs1}) \text{ as})$ 
apply(auto simp:qe-pres1-def Z.normal-def)
  dest!:atoms-list-disjE atoms-list-conjE)

```

```

apply(simp add:hd-coeffs1-def)
apply(erule disjE) apply fastsimp
apply (clarsimp)
apply(case-tac xa)
  apply(case-tac list) apply fastsimp apply (simp split:split-if-asm)
  apply(case-tac list) apply fastsimp
  apply (simp split:split-if-asm) apply fastsimp
apply(erule disjE) prefer 2 apply fastsimp
apply(simp add:zdiv-eq-0-iff)
apply(subgoal-tac a \in set(map hd-coeff as))
  prefer 2 apply force

```

```

apply(subgoal-tac  $\forall i \in \text{set}(\text{map } \text{hd-coeff } \text{as}). i \neq 0$ )
  prefer 2 apply simp
apply (metis elem-le-zlcms linorder-not-le zlcms-pos)
apply(case-tac list) apply fastsimp
apply (simp split:split-if-asm) apply fastsimp
apply(simp add:zdiv-eq-0-iff)
apply(subgoal-tac  $\forall i \in \text{set}(\text{map } \text{hd-coeff } \text{as}). i \neq 0$ )
  prefer 2 apply simp
apply(subgoal-tac  $a \in \text{set}(\text{map } \text{hd-coeff } \text{as})$ )
  prefer 2 apply force
apply(erule disjE)
  apply (metis elem-le-zlcms linorder-not-le)
apply(erule disjE)
  apply (metis linorder-not-le zlcms-pos)
apply fastsimp

apply(simp add: hd-coeffs1-def)
apply(erule disjE) apply fastsimp
apply (clarsimp)
apply(case-tac xa)
  apply(case-tac list) apply fastsimp apply (simp split:split-if-asm)
apply(case-tac list) apply fastsimp
apply (simp split:split-if-asm) apply fastsimp
apply(erule disjE) prefer 2 apply fastsimp
apply(simp add:zdiv-eq-0-iff)
apply(subgoal-tac  $a \in \text{set}(\text{map } \text{hd-coeff } \text{as})$ )
  prefer 2 apply force
apply(subgoal-tac  $\forall i \in \text{set}(\text{map } \text{hd-coeff } \text{as}). i \neq 0$ )
  prefer 2 apply simp
apply (metis elem-le-zlcms linorder-not-le zlcms-pos)
apply(case-tac list) apply fastsimp
apply (simp split:split-if-asm) apply fastsimp
apply(simp add:zdiv-eq-0-iff)
apply(subgoal-tac  $\forall i \in \text{set}(\text{map } \text{hd-coeff } \text{as}). i \neq 0$ )
  prefer 2 apply simp
apply(subgoal-tac  $a \in \text{set}(\text{map } \text{hd-coeff } \text{as})$ )
  prefer 2 apply force
apply(erule disjE)
  apply (metis elem-le-zlcms linorder-not-le)
apply(erule disjE)
  apply (metis linorder-not-le zlcms-pos)
apply fastsimp
done

```

theorem *I-pres-qe*: $Z.\text{normal } \varphi \implies Z.I (\text{qe-pres } \varphi) \text{ xs} = Z.I \varphi \text{ xs}$
by (*simp add:qe-pres-def Z.I-lift-dnf-qe-anormal I-qe-pres₁-o qfree-qe-pres-o normal-qe-pres₁-o del:o-apply*)

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

```

by(simp add:qe-pres-def Z.qfree-lift-dnf-qe qfree-qe-pres-o del:o-apply)

end

```

```

theory Cooper
imports PresArith
begin

```

6.5 Cooper

This section formalizes Cooper's algorithm [1].

```

lemma set-atoms0-iff:
  qfree  $\varphi \implies a : \text{set}(Z.\text{atoms}_0 \varphi) \longleftrightarrow a : \text{atoms } \varphi \wedge \text{hd-coeff } a \neq 0$ 
by(induct  $\varphi$ ) (auto split:split-if-asm)

```

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  $\varphi$ 
shows  $(\exists x. Z.I (\text{hd-coeffs1 } \varphi) (x\#xs)) = (\exists x. Z.I \varphi (x\#xs))$  (is ?L = ?R)
proof -
  let ?l =  $\text{zlcms}(\text{map } \text{hd-coeff } (Z.\text{atoms}_0 \varphi))$ 
  have ?l>0 by(simp add:zlcms-pos set-atoms0-iff[OF qfree  $\varphi$ ])
  have ?L =  $(\exists x. ?l \text{ dvd } x+0 \wedge Z.I (\text{map}_{fm} (\text{hd-coeff1 } ?l) \varphi) (x\#xs))$ 
    by(simp add:hd-coeffs1-def)
  also have ... =  $(\exists x. Z.I (\text{map}_{fm} (\text{hd-coeff1 } ?l) \varphi) (?l*x\#xs))$ 
    by(rule unity-coeff-ex[THEN meta-eq-to-obj-eq,symmetric])
  also have ... = ?R
    by(simp add:I-hd-coeff1-mult[OF ?l>0 qfree  $\varphi$ ] dvd-zlcms)
  finally show ?thesis .
qed

```

```

fun min-inf :: atom fm  $\Rightarrow$  atom fm (inf _) where
inf_ (And  $\varphi_1 \varphi_2$ ) = and (inf_  $\varphi_1$ ) (inf_  $\varphi_2$ ) |
inf_ (Or  $\varphi_1 \varphi_2$ ) = or (inf_  $\varphi_1$ ) (inf_  $\varphi_2$ ) |
inf_ (Atom(Le  $i (k\#ks)$ )) =
  (if  $k<0$  then TrueF else if  $k>0$  then FalseF else Atom(Le  $i (0\#ks)$ )) |
inf_  $\varphi = \varphi$ 

```

definition

```

qe-cooper1  $\varphi =$ 
  (let  $as = Z.\text{atoms}_0 \varphi$ ;  $d = \text{zlcms}(\text{map } \text{divisor } as)$ ;  $ls = \text{lbounds } as$ 

```

$in\ or\ (Disj\ [0..d - 1]\ (\lambda n.\ subst\ n\ []\ (inf_ -\ \varphi)))$
 $(Disj\ ls\ (\lambda(i,ks).\$
 $\quad Disj\ [0..d - 1]\ (\lambda n.\ subst\ (i + n)\ (-ks)\ \varphi)))$

lemma *min-inf*:

$ngfree\ f\ \Longrightarrow\ \forall a \in set(Z.atoms_0\ f).\ hd-coeff-is1\ a$
 $\Longrightarrow\ \exists x.\ \forall y < x.\ Z.I\ (inf_ -\ f)\ (y\ \#\ xs) = Z.I\ f\ (y\ \#\ xs)$
 $(is\ -\ \Longrightarrow\ -\ \Longrightarrow\ \exists x.\ ?P\ f\ x)$

proof(*induct f rule: min-inf.induct*)

case ($\exists\ i\ k\ ks$)

{ **assume** $k=0$ **hence** $?case$ **using** \exists **by** *simp* }

moreover

{ **assume** $k = -1$

hence $?P\ (Atom\ (Le\ i\ (k\ \#\ ks)))\ (-i + \langle ks, xs \rangle - 1)$ **using** \exists **by** *auto*

hence $?case\ ..$ }

moreover

{ **assume** $k=1$

hence $?P\ (Atom\ (Le\ i\ (k\ \#\ ks)))\ (i - \langle ks, xs \rangle - 1)$ **using** \exists **by** *auto*

hence $?case\ ..$ }

ultimately show $?case$ **using** \exists **by** *auto*

next

case ($1\ f1\ f2$)

then obtain $x1\ x2$ **where** $?P\ f1\ x1\ ?P\ f2\ x2$ **by** *fastsimp+*

hence $?P\ (And\ f1\ f2)\ (min\ x1\ x2)$ **by** *simp*

thus $?case\ ..$

next

case ($2\ f1\ f2$)

then obtain $x1\ x2$ **where** $?P\ f1\ x1\ ?P\ f2\ x2$ **by** *fastsimp+*

hence $?P\ (Or\ f1\ f2)\ (min\ x1\ x2)$ **by** *simp*

thus $?case\ ..$

qed *simp-all*

lemma *min-inf-repeats*:

$ngfree\ \varphi\ \Longrightarrow\ \forall a \in set(Z.atoms_0\ \varphi).\ divisor\ a\ dvd\ d\ \Longrightarrow$

$Z.I\ (inf_ -\ \varphi)\ ((x - k*d)\ \#\ xs) = Z.I\ (inf_ -\ \varphi)\ (x\ \#\ xs)$

proof(*induct φ rule: min-inf.induct*)

case ($4-4\ da\ i\ ks$)

show $?case$

proof (*cases ks*)

case *Nil* **thus** $?thesis$ **by** *simp*

next

case (*Cons j js*)

show $?thesis$

proof *cases*

assume $j=0$ **thus** $?thesis$ **using** *Cons* **by** *simp*

next

assume $j \neq 0$

```

hence da dvd d using Cons 4-4 by simp
hence da dvd i + (j * x - j * (k * d) + ⟨js,xs⟩) ⟷
  da dvd i + (j * x + ⟨js,xs⟩)
proof -
  have da dvd i + (j * x - j * (k * d) + ⟨js,xs⟩) ⟷
    da dvd (i + j*x + ⟨js,xs⟩) - (j*k)*d
  by(simp add: algebra-simps)
  also have ... ⟷ da dvd i + j*x + ⟨js,xs⟩ using ⟨da dvd d⟩
  by (metis dvd-diff zdvd-zdiffD dvd-mult zmult-commute)
  also have ... ⟷ da dvd i + (j * x + ⟨js,xs⟩)
  by(simp add: algebra-simps)
  finally show ?thesis .
qed
then show ?thesis using Cons by (simp add:ring-distrib)
qed
qed
next
case (4-5 da i ks)
show ?case
proof (cases ks)
  case Nil thus ?thesis by simp
next
case (Cons j js)
show ?thesis
proof cases
  assume j=0 thus ?thesis using Cons by simp
next
  assume j≠0
  hence da dvd d using Cons 4-5 by simp
  hence da dvd i + (j * x - j * (k * d) + ⟨js,xs⟩) ⟷
    da dvd i + (j * x + ⟨js,xs⟩)
  proof -
    have da dvd i + (j * x - j * (k * d) + ⟨js,xs⟩) ⟷
      da dvd (i + j*x + ⟨js,xs⟩) - (j*k)*d
    by(simp add: algebra-simps)
    also have ... ⟷ da dvd i + j*x + ⟨js,xs⟩ using ⟨da dvd d⟩
    by (metis dvd-diff zdvd-zdiffD dvd-mult zmult-commute)
    also have ... ⟷ da dvd i + (j * x + ⟨js,xs⟩)
    by(simp add: algebra-simps)
    finally show ?thesis .
  qed
  then show ?thesis using Cons by (simp add:ring-distrib)
qed
qed
qed simp-all

```

lemma *atoms-subset*: $qfree\ f \implies set(Z.atoms_0(f::atom\ fm)) \leq atoms\ f$
by (*induct f*) *auto*

lemma β :

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

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

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

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

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

proof(*induct* φ)

case (*Atom* a)

show *?case*

proof (*cases* a)

case (*Le* i js)

show *?thesis*

proof (*cases* js)

case *Nil* **thus** *?thesis* **using** *Le Atom by simp*

next

case (*Cons* k ks) **thus** *?thesis* **using** *Le Atom*

by (*auto simp:lbounds-def Ball-def split:split-if-asm*) *arith*

qed

next

case (*Dvd* m i js)

show *?thesis*

proof (*cases* js)

case *Nil* **thus** *?thesis* **using** *Dvd Atom by simp*

next

case (*Cons* k ks)

show *?thesis*

proof *cases*

assume $k=0$ **thus** *?thesis* **using** *Cons Dvd Atom by simp*

next

assume $k \neq 0$

hence $m \text{ dvd } d$ **using** *Cons Dvd Atom by auto*

have $m \text{ dvd } i + (x + \langle ks, xs \rangle) \implies m \text{ dvd } i + (x - d + \langle ks, xs \rangle)$

(is *?L* \implies *-*)

proof *-*

assume *?L*

hence $m \text{ dvd } i + (x + \langle ks, xs \rangle) - d$

by (*metis* $\langle m \text{ dvd } d \rangle$ *dvd-diff*)

thus *?thesis* **by**(*simp add:algebra-simps*)

qed

thus *?thesis* **using** *Atom Dvd Cons* **by**(*auto split:split-if-asm*)

qed

qed

next

case (*NDvd* m i js)

show *?thesis*

proof (*cases* js)

case *Nil* **thus** *?thesis* **using** *NDvd Atom by simp*

```

next
  case (Cons k ks)
  show ?thesis
  proof cases
    assume k=0 thus ?thesis using Cons NDvd Atom by simp
  next
    assume k≠0
    hence m dvd d using Cons NDvd Atom by auto
    have m dvd i + (x - d + ⟨ks,xs⟩)  $\implies$  m dvd i + (x + ⟨ks,xs⟩)
      (is ?L  $\implies$  -)
    proof -
      assume ?L
      hence m dvd i + (x + ⟨ks,xs⟩) - d by(simp add:algebra-simps)
      thus ?thesis by (metis ⟨m dvd d⟩ z dvd-zdiffD)
    qed
  thus ?thesis using Atom NDvd Cons by(auto split:split-if-asm)
qed
qed
qed
qed force+

```

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)
proof
  assume ?LHS
  then obtain x where P: P x ..
  have x mod d = x - (x div d)*d
    by(simp add:zmod-zdiv-equality mult-ac eq-diff-eq)
  hence Pmod: P x = P(x mod d) using modd by simp
  have P(x mod d) using dpos P Pmod by(simp add:pos-mod-sign pos-mod-bound)
  moreover have x mod d : {0..d - 1} using dpos by(auto simp:pos-mod-sign)
  ultimately show ?RHS ..
qed auto

```

```

lemma cpmi-eq: (0::int) < D  $\implies$   $(\exists z. \forall x. x < z \longrightarrow (P x = P1 x))$ 
 $\implies \forall x. \neg(\exists j \in \{0..D - 1\}. \exists b \in B. P(b+j)) \longrightarrow P(x) \longrightarrow P(x - D)$ 
 $\implies \forall x. \forall k. P1 x = P1(x - k*D)$ 
 $\implies (\exists x. P(x)) = ((\exists j \in \{0..D - 1\}. P1(j)) \vee (\exists j \in \{0..D - 1\}. \exists b \in B. P(b+j)))$ 
apply(rule iffI)
prefer 2
apply(drule minusinfinity)
apply assumption+
apply(fastsimp)
apply clarsimp
apply(subgoal-tac  $\wedge k. 0 \leq k \implies \forall x. P x \longrightarrow P(x - k*D)$ )
apply(frule-tac x = x and z=z in decr-lemma)

```

```

apply(subgoal-tac  $P1(x - (|x - z| + 1) * D)$ )
prefer 2
apply(subgoal-tac  $0 \leq (|x - z| + 1)$ )
prefer 2 apply arith
  apply fastsimp
apply(drule (1) periodic-finite-ex)
apply blast
apply(blast dest:decr-mult-lemma)
done

```

theorem cp-thm:

```

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

```

proof –

```

from min-inf[OF nq u] have th:  $\exists z. \forall x < z. ?P x = ?M x$  by blast
let  $?B' = \{?I i ks \mid i ks. (i, ks) \in ?B\}$ 
have  $BB': (\exists j \in ?D. \exists (i, ks) \in ?B. ?P (?I i ks + j)) = (\exists j \in ?D. \exists b \in ?B'. ?P$ 
 $(b + j))$  by auto
hence th2:  $\forall x. \neg (\exists j \in ?D. \exists b \in ?B'. ?P ((b + j))) \longrightarrow ?P (x) \longrightarrow ?P ((x$ 
 $- d))$ 
using  $\beta[OF nq u d dp, of - xs]$  by(simp add: Bex-def) metis
from min-inf-repeats[OF nq d]
have th3:  $\forall x k. ?M x = ?M (x - k * d)$  by simp
from cpmi-eq[OF dp th th2 th3]  $BB'$  show ?thesis by simp blast
qed

```

lemma qfree-min-inf[simp]: qfree $\varphi \implies$ qfree $(\text{inf}_- \varphi)$
by (induct φ rule:min-inf.induct) simp-all

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  $\varphi$ 
shows  $Z.I (\text{qe-cooper}_1 \varphi) xs = (\exists x. Z.I \varphi (x \# xs))$ 

```

proof –

```

let  $?as = Z.\text{atoms}_0 \varphi$ 
let  $?d = \text{zlcms}(\text{map } \text{divisor } ?as)$ 
have  $?d > 0$  using norm atoms-subset[of  $\varphi$ ]  $\langle$ nqfree  $\varphi$ \rangle
by(fastsimp intro:zlcms-pos)
have alld:  $\forall a \in \text{set}(Z.\text{atoms}_0 \varphi). \text{divisor } a \text{ dvd } ?d$  by(simp add:dvd-zlcms)
from cp-thm[OF  $\langle$ nqfree  $\varphi$  hd alld  $\langle$ ?d>0\rangle]

```

show *?thesis using* $\langle \text{ngfree } \varphi \rangle$
by (*simp add:qe-cooper₁-def I-subst[symmetric] split-def algebra-simps*) *blast*
qed

lemma *divisor-hd-coeff1-neq0:*
 $\text{qfree } \varphi \implies a \in \text{atoms } \varphi \implies \text{divisor } a \neq 0 \implies$
 $\text{divisor } (\text{hd-coeff1 } (\text{zlcms } (\text{map hd-coeff } (Z.\text{atoms}_0 \varphi))) a) \neq 0$
apply (*case-tac a*)

apply *simp*
apply(*case-tac list*) **apply** *simp* **apply**(*simp split:split-if-asm*)

apply *simp*
apply(*case-tac list*) **apply** *simp*
apply(*clarsimp split:split-if-asm*)
apply(*subgoal-tac a : set(map hd-coeff (Z.atoms₀ φ))*)
apply(*subgoal-tac $\forall i \in \text{set}(\text{map hd-coeff } (Z.\text{atoms}_0 \varphi)). i \neq 0$*)
apply (*metis dvd-zlcms mult-eq-0-iff zdvd-mult-div-cancel zlcms0-iff*)
apply (*simp add:set-atoms0-iff*)
apply(*fastsimp simp:image-def set-atoms0-iff Bex-def*)

apply *simp*
apply(*case-tac list*) **apply** *simp*
apply(*clarsimp split:split-if-asm*)
apply(*subgoal-tac a : set(map hd-coeff (Z.atoms₀ φ))*)
apply(*subgoal-tac $\forall i \in \text{set}(\text{map hd-coeff } (Z.\text{atoms}_0 \varphi)). i \neq 0$*)
apply (*metis dvd-zlcms mult-eq-0-iff zdvd-mult-div-cancel zlcms0-iff*)
apply (*simp add:set-atoms0-iff*)
apply(*fastsimp simp:image-def set-atoms0-iff Bex-def*)
done

lemma *hd-coeff-is1-hd-coeff1:*
 $\text{hd-coeff } (\text{hd-coeff1 } m a) \neq 0 \implies \text{hd-coeff-is1 } (\text{hd-coeff1 } m a)$
by (*induct a rule: hd-coeff1.induct*) (*simp-all add:zsgn-def*)

lemma *I-cooper1-hd-coeffs1: Z.normal $\varphi \implies \text{ngfree } \varphi$*
 $\implies Z.I (\text{qe-cooper}_1(\text{hd-coeffs1 } \varphi)) \text{xs} = (\exists x. Z.I \varphi (x \# \text{xs}))$
apply(*simp add:Z.normal-def*)
apply(*subst I-qe-cooper₁*)
apply(*clarsimp simp:hd-coeffs1-def image-def set-atoms0-iff divisor-hd-coeff1-neq0*)
apply (*clarsimp simp:hd-coeffs1-def qfree-map-fm set-atoms0-iff*
 $\text{hd-coeff-is1-hd-coeff1}$)
apply(*simp add:hd-coeffs1-def ngfree-map-fm*)
apply(*simp add: I-hd-coeffs1*)
done

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

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

by(*auto simp:qe-cooper₁-def hd-coeffs1-def qfree-map-fm*
intro!: qfree-or qfree-and qfree-list-disj qfree-min-inf)

lemma *normal-min-inf*: $Z.normal\ \varphi \implies Z.normal(inf_ \varphi)$
by(*induct φ rule:min-inf.induct*) *simp-all*

lemma *normal-cooper1*: $Z.normal\ \varphi \implies Z.normal(qe-cooper_1\ \varphi)$
by(*simp add:qe-cooper₁-def Logic.or-def Z.normal-map-fm normal-min-inf split-def*)

lemma *normal-hd-coeffs1*: $qfree\ \varphi \implies Z.normal\ \varphi \implies Z.normal(hd-coeffs1\ \varphi)$
by(*auto simp: hd-coeffs1-def image-def set-atoms0-iff*
divisor-hd-coeff1-neq0 Z.normal-def)

theorem *I-cooper*: $Z.normal\ \varphi \implies Z.I\ (qe-cooper\ \varphi)\ xs = Z.I\ \varphi\ xs$
by(*simp add:qe-cooper-def Z.I-lift-nnf-qe-normal qfree-cooper1-hd-coeffs1 I-cooper1-hd-coeffs1*
normal-cooper1 normal-hd-coeffs1)

theorem *qfree-cooper*: $qfree\ (qe-cooper\ \varphi)$
by(*simp add:qe-cooper-def Z.qfree-lift-nnf-qe qfree-cooper1-hd-coeffs1*)

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.