

# Quantifier Elimination for Linear Arithmetic

Tobias Nipkow

June 17, 2024

## Abstract

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

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

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

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

## Contents

<b>1</b>	<b>Logic</b>	<b>2</b>
1.1	Atoms . . . . .	5
<b>2</b>	<b>Quantifier elimination</b>	<b>8</b>
2.1	No Equality . . . . .	8
2.1.1	DNF-based . . . . .	8
2.1.2	NNF-based . . . . .	11
2.2	With equality . . . . .	13
<b>3</b>	<b>DLO</b>	<b>14</b>
3.1	Basics . . . . .	14
3.2	DNF-based quantifier elimination . . . . .	20
3.3	Examples . . . . .	23
3.4	Interior Point Method . . . . .	25
3.5	Quantifier elimination with infinitesimals . . . . .	29

<b>4</b>	<b>Linear real arithmetic</b>	<b>34</b>
4.1	Basics . . . . .	34
4.1.1	Syntax and Semantics . . . . .	34
4.1.2	Shared constructions . . . . .	35
4.2	Fourier . . . . .	40
4.2.1	Tests . . . . .	43
4.2.2	An optimization . . . . .	44
4.3	Ferrante-Rackoff . . . . .	45
4.4	Quantifier elimination with infinitesimals . . . . .	49
<b>5</b>	<b>Presburger arithmetic</b>	<b>55</b>
5.1	Syntax . . . . .	56
5.2	LCM and lemmas . . . . .	58
5.3	Setting coefficients to 1 or -1 . . . . .	58
5.4	DNF-based quantifier elimination . . . . .	61
5.5	Cooper . . . . .	70

## 1 Logic

```
theory Logic
imports Main HOL-Library.FuncSet
begin
```

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

```
declare Let-def[simp]
```

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

```
notation map-fm (map_fm)
```

```
abbreviation Imp where Imp  $\varphi_1 \varphi_2 \equiv Or (Neg \varphi_1) \varphi_2$ 
```

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

```
definition neg where
```

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

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

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

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

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

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

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

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

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

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

**fun** *amap-fm* :: ('a  $\Rightarrow$  'b fm)  $\Rightarrow$  'a fm  $\Rightarrow$  'b fm (*amap\_fm*) **where**  
*amap\_fm* h TrueF = TrueF |  
*amap\_fm* h FalseF = FalseF |  
*amap\_fm* h (Atom a) = h a |  
*amap\_fm* h (And  $\varphi_1 \varphi_2$ ) = and (*amap\_fm* h  $\varphi_1$ ) (*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*  $\varphi_2$ ) |  
*qfree*(Or  $\varphi_1 \varphi_2$ ) = (*qfree*  $\varphi_1 \wedge$  *qfree*  $\varphi_2$ ) |  
*qfree*(Neg  $\varphi$ ) = (*qfree*  $\varphi$ ) |  
*qfree*  $\varphi$  = True

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

**lemma** *qfree-or[simp]*:  $\llbracket$  *qfree*  $\varphi_1$ ; *qfree*  $\varphi_2$   $\rrbracket \Longrightarrow$  *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  $\varphi$   
  **have**  $\llbracket \forall \varphi \in$  set fs. *qfree*  $\varphi$ ; *qfree*  $\varphi$   $\rrbracket \Longrightarrow$  *qfree*(foldr and fs  $\varphi$ )

```

    by (induct fs) auto
  } thus ?thesis using assms by (fastforce simp add:list-conj-def)
qed

```

```

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

```

```

lemma qfree-map-fm:  $\text{qfree}(\text{map}_{fm} f \ \varphi) = \text{qfree } \varphi$ 
by (induct  $\varphi$ ) simp-all

```

```

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

```

```

lemma atoms-list-conjE:
   $a \in \text{atoms}(\text{list-conj } fs) \implies a \in (\bigcup \varphi \in \text{set } fs. \text{atoms } \varphi)$ 
apply (induct fs)
apply (simp add:list-conj-def)
apply (auto simp add:list-conj-def Logic.and-def split:if-split-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]:  $\text{nqfree } \varphi \implies \text{qfree } \varphi$ 
by (induct  $\varphi$ ) simp-all

```

**lemma** *ngfree-map-fm*:  $ngfree (map_{fm} f \varphi) = ngfree \varphi$   
**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  $\vee$  interpret h  $\varphi_2$  xs) |  
*interpret* h (Neg  $\varphi$ ) xs = ( $\neg$  interpret h  $\varphi$  xs) |  
*interpret* h (ExQ  $\varphi$ ) xs = ( $\exists x. interpret h \varphi (x\#xs)$ )

## 1.1 Atoms

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

**locale** ATOM =  
**fixes** *aneg* :: 'a  $\Rightarrow$  'a fm  
**fixes** *anormal* :: 'a  $\Rightarrow$  bool  
**assumes** *ngfree-aneg*:  $ngfree(aneg a)$   
**assumes** *anormal-aneg*:  $anormal a \implies \forall b \in atoms(aneg a). anormal b$

**fixes** *I<sub>a</sub>* :: 'a  $\Rightarrow$  'b list  $\Rightarrow$  bool  
**assumes** *I<sub>a</sub>-aneg*:  $interpret I_a (aneg a) xs = (\neg I_a a xs)$

**fixes** *depends<sub>0</sub>* :: 'a  $\Rightarrow$  bool  
**and** *decr* :: 'a  $\Rightarrow$  'a  
**assumes** *not-dep-decr*:  $\neg depends_0 a \implies I_a a (x\#xs) = I_a (decr a) xs$   
**assumes** *anormal-decr*:  $\neg depends_0 a \implies anormal a \implies anormal(decr a)$

**begin**

**fun** *atoms<sub>0</sub>* :: 'a fm  $\Rightarrow$  'a list **where**  
*atoms<sub>0</sub>* TrueF = [] |  
*atoms<sub>0</sub>* FalseF = [] |  
*atoms<sub>0</sub>* (Atom a) = (if depends<sub>0</sub> a then [a] else []) |  
*atoms<sub>0</sub>* (And  $\varphi_1 \varphi_2$ ) = atoms<sub>0</sub>  $\varphi_1$  @ atoms<sub>0</sub>  $\varphi_2$  |  
*atoms<sub>0</sub>* (Or  $\varphi_1 \varphi_2$ ) = atoms<sub>0</sub>  $\varphi_1$  @ atoms<sub>0</sub>  $\varphi_2$  |  
*atoms<sub>0</sub>* (Neg  $\varphi$ ) = atoms<sub>0</sub>  $\varphi$

**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 |

$nnf (Neg (Neg \varphi)) = (nnf \varphi) \mid$   
 $nnf (Neg (And \varphi_1 \varphi_2)) = (Or (nnf (Neg \varphi_1)) (nnf (Neg \varphi_2))) \mid$   
 $nnf (Neg (Or \varphi_1 \varphi_2)) = (And (nnf (Neg \varphi_1)) (nnf (Neg \varphi_2))) \mid$   
 $nnf (Neg (Atom a)) = aneg a \mid$   
 $nnf \varphi = \varphi$

**lemma** *ngfree- $nnf$* :  $qfree \varphi \implies ngfree(nnf \varphi)$   
**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**  $fs \varphi$   
**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<sub>a</sub>-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*  $\varphi$ ) (*fastforce*)+

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

**lemma** *normal-simps*[simp]:

*normal TrueF*

*normal FalseF*

*normal (Atom a)  $\longleftrightarrow$  anormal a*

*normal (And  $\varphi_1 \varphi_2$ )  $\longleftrightarrow$  normal  $\varphi_1 \wedge$  normal  $\varphi_2$*

*normal (Or  $\varphi_1 \varphi_2$ )  $\longleftrightarrow$  normal  $\varphi_1 \vee$  normal  $\varphi_2$*

*normal (Neg  $\varphi$ )  $\longleftrightarrow$  normal  $\neg \varphi$*

*normal (ExQ  $\varphi$ )  $\longleftrightarrow$  normal  $\exists x. \varphi$*

**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]:

*$\forall \varphi \in \text{set } fs. \text{normal } \varphi \implies \text{normal (list-disj } fs)$*

**apply**(*induct fs*)

**apply** (*simp add:list-disj-def*)

**apply** (*simp add:list-disj-def*)

**done**

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

**by**(*induct  $\varphi$  rule:nnf.induct simp-all*)

**lemma** *normal-map-fm*:

*$\forall a. \text{anormal}(f a) = \text{anormal}(a) \implies \text{normal (map}_{fm} f \varphi) = \text{normal } \varphi$*

**by**(*induct  $\varphi$  auto*)

**lemma** *anormal-nnf*:

*qfree  $\varphi \implies$  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*: *ngfree  $\varphi \implies as \in \text{set}(\text{dnf } \varphi) \implies a \in \text{set } as \implies a \in \text{atoms } \varphi$*

**by**(*induct  $\varphi$  arbitrary: as a rule:ngfree.induct*)(*auto*)

**lemma** *anormal-dnf-nnf*:

```

    as ∈ set(dnf(nnf φ)) ⇒ qfree φ ⇒ normal φ ⇒ a ∈ set as ⇒ anormal a
apply(induct φ arbitrary: a as rule:nnf.induct)
    apply(simp-all add: normal-def)
    apply clarify
    apply (metis UnE set-append)
    apply metis
    apply metis
    apply fastforce
apply (metis anormal-aneg atoms-dnf nqfree-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.

```

notation (input) Collect (|-)

```

### 2.1 No Equality

```

context ATOM
begin

```

#### 2.1.1 DNF-based

Taking care of atoms independent of variable 0:

**definition**

```

qelim qe as =
  (let qf = qe [a ← 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*<sub>*a*</sub> *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 ∘ 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: (∧ as. (∀ a ∈ set as. depends0 a) ⇒ qfree (qe as))
  ⇒ qfree (lift-dnf-qe qe φ)
by (induct φ) (simp-all add:qelim-def)

```

```

lemma qfree-lift-dnf-qe2: qe ∈ lists |depends0| → |qfree|
  ⇒ qfree (lift-dnf-qe qe φ)
using in-lists-conv-set [where ?'a = 'a]
by (simp add:Pi-def qfree-lift-dnf-qe)

```

```

lemma lem: ∀ P A. (∃ x ∈ A. ∃ y. P x y) = (∃ y. ∃ x ∈ A. P x y) by blast

```

```

lemma I-lift-dnf-qe:
assumes ∧ as. (∀ a ∈ set as. depends0 a) ⇒ qfree (qe as)
and ∧ as. (∀ a ∈ set as. depends0 a) ⇒ is-dnf-qe qe as
shows I (lift-dnf-qe qe φ) xs = I φ xs
proof (induct φ arbitrary:xs)
  case ExQ thus ?case
    by (simp add: assms I-qelim lem I-dnf nqfree-nnf qfree-lift-dnf-qe
      I-nnf)
qed simp-all

```

**lemma** *I-lift-dnf-qe2*:  
**assumes**  $qe \in \text{lists } |\text{depends}_0| \rightarrow |\text{qfree}|$   
**and**  $\forall as \in \text{lists } |\text{depends}_0|. \text{is-dnf-qe } qe \text{ } as$   
**shows**  $I (\text{lift-dnf-qe } qe \ \varphi) \ xs = I \ \varphi \ xs$   
**using** *assms in-lists-conv-set*[**where**  $?'a = 'a$ ]  
**by**(*simp add:Pi-def I-lift-dnf-qe*)

Quantifier elimination with invariant (needed for Presburger):

**lemma** *I-qelim-anormal*:  
**assumes**  $qe: \bigwedge xs \ as. \forall a \in \text{set } as. \text{depends}_0 \ a \wedge \text{anormal } a \implies \text{is-dnf-qe } qe \ as$   
**and**  $nm: \forall a \in \text{set } as. \text{anormal } a$   
**shows**  $I (\text{qelim } qe \ as) \ xs = (\exists x. \forall a \in \text{set } as. I_a \ a \ (x\#xs))$   
**proof** –  
**let**  $?as0 = \text{filter } \text{depends}_0 \ as$   
**let**  $?as1 = \text{filter } (\text{Not } \circ \text{depends}_0) \ as$   
**have**  $I (\text{qelim } qe \ as) \ xs =$   
 $(I (qe \ ?as0) \ xs \wedge (\forall a \in \text{set}(\text{map } \text{decr } ?as1). I_a \ a \ xs))$   
 $(\text{is } - = (- \wedge ?B))$  **by**(*force simp add:qelim-def*)  
**also have**  $\dots = ((\exists x. \forall a \in \text{set } ?as0. I_a \ a \ (x\#xs)) \wedge ?B)$   
**by**(*simp add:qe nm not-dep-decr*)  
**also have**  $\dots = (\exists x. (\forall a \in \text{set } ?as0. I_a \ a \ (x\#xs)) \wedge ?B)$  **by** *blast*  
**also have**  $?B = (\forall a \in \text{set } ?as1. I_a \ (\text{decr } a) \ xs)$  **by** *simp*  
**also have**  $(\exists x. (\forall a \in \text{set } ?as0. I_a \ a \ (x\#xs)) \wedge \dots) =$   
 $(\exists x. (\forall a \in \text{set } ?as0. I_a \ a \ (x\#xs)) \wedge$   
 $(\forall a \in \text{set } ?as1. I_a \ a \ (x\#xs)))$   
**by**(*simp add: not-dep-decr*)  
**also have**  $\dots = (\exists x. \forall a \in \text{set}(\ ?as0 \ @ \ ?as1). I_a \ a \ (x\#xs))$   
**by** (*simp add:ball-Un*)  
**also have**  $\dots = (\exists x. \forall a \in \text{set}(as). I_a \ a \ (x\#xs))$   
**by** *simp blast*  
**finally show**  $?thesis$  .  
**qed**

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

**lemma** *anormal-atoms-qelim*:  
 $(\bigwedge as. \forall a \in \text{set } as. \text{depends}_0 \ a \wedge \text{anormal } a \implies \text{normal}(qe \ as)) \implies$   
 $\forall a \in \text{set } as. \text{anormal } a \implies a \in \text{atoms}(\text{qelim } qe \ as) \implies \text{anormal } a$   
**apply**(*auto simp add:qelim-def and-def normal-def split:if-split-asm*)  
**apply**(*auto simp add:anormal-decr dest!: atoms-list-conjE*)  
**apply**(*erule-tac x = filter depends<sub>0</sub> as in meta-alle*)  
**apply**(*simp*)  
**apply**(*erule-tac x = filter depends<sub>0</sub> as in meta-alle*)  
**apply**(*simp*)  
**done**

**lemma** *normal-lift-dnf-qe*:

```

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

```

**end**

```

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

```

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

```

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

```

**end**

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

```

assumes  $qe \in lists\ |depends_0| \rightarrow |qfree|$ 
and  $qe \in lists\ (|depends_0| \cap |anormal|) \rightarrow |normal|$ 
and  $\forall as \in lists\ (|depends_0| \cap |anormal|). is-dnf-qe\ qe\ as$ 
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 I-lift-dnf-qe-anormal Int-def)

```

## 2.1.2 NNF-based

```

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

```

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

$\implies$   $qfree(\text{lift-nnf-ge } qe \ \varphi)$   
**by** (*induct*  $\varphi$ ) (*simp-all add:ngfree-nnf*)

**lemma** *qfree-lift-nnf-ge2*:  
 $qe \in |ngfree| \rightarrow |qfree| \implies qfree(\text{lift-nnf-ge } qe \ \varphi)$   
**by**(*simp add:Pi-def qfree-lift-nnf-ge*)

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

**lemma** *I-lift-nnf-ge2*:  
**assumes**  $qe \in |ngfree| \rightarrow |qfree|$   
**and**  $\forall \varphi \in |ngfree|. \forall xs. I(qe \ \varphi) \ xs = (\exists x. I \ \varphi(x\#xs))$   
**shows**  $I(\text{lift-nnf-ge } qe \ \varphi) \ xs = I \ \varphi \ xs$   
**using** *assms* **by**(*simp add:Pi-def I-lift-nnf-ge*)

**lemma** *normal-lift-nnf-ge*:  
**assumes**  $\bigwedge \varphi. ngfree \ \varphi \implies qfree(qe \ \varphi)$   
**and**  $\bigwedge \varphi. ngfree \ \varphi \implies normal \ \varphi \implies normal(qe \ \varphi)$   
**shows**  $normal \ \varphi \implies normal(\text{lift-nnf-ge } qe \ \varphi)$   
**by** (*induct*  $\varphi$ )  
  (*simp-all add: assms Logic.neg-def normal-nnf ngfree-nnf qfree-lift-nnf-ge*)

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

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

**end**

## 2.2 With equality

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

```

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

```

**begin**

```

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

```

**theorem** *I-lift-eq-qe*:

**assumes** *dep*: ∀ *a* ∈ *set as*. *depends<sub>0</sub> a*

**assumes** *qe*: ∧ *as*. (∀ *a* ∈ *set as*. *depends<sub>0</sub> a* ∧ ¬ *solvable<sub>0</sub> a*) ⇒  
*I (qe as) xs = (∃x. ∀ *a* ∈ *set as*. *I<sub>a</sub> a (x#xs)*)*

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

**proof** –

let ?*as* = [*a←as*. ¬ *trivial a*]

show ?*thesis*

**proof** (*cases* [*a←?as*. *solvable<sub>0</sub> a*])

case *Nil*

hence ∀ *a* ∈ *set as*. ¬ *trivial a* → ¬ *solvable<sub>0</sub> 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* *solvable<sub>0</sub> eq* ¬ *trivial eq*

by (*auto simp: filter-eq-Cons-iff*)

then obtain *e* where *I<sub>a</sub> eq (e#xs)* by (*metis solvable*)

have ∀ *a* ∈ *set as*. *I<sub>a</sub> a (e # xs) = I<sub>a</sub> (subst<sub>0</sub> eq a) xs*

by (*simp add: subst<sub>0</sub>[OF ‹solvable<sub>0</sub> eq› ‹¬ trivial eq› ‹I<sub>a</sub> 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(fastforce intro!: exI[of - e] simp: trivial is-triv-self-subst)
apply (metis subst0)
done
qed
qed

```

**definition** lift-dnf-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-dnf-qe:  $(\bigwedge as. (\forall a \in set\ as. depends_0\ a) \implies qfree\ (qe\ as))$   
 $\implies qfree\ (lift-dnf-qe\ qe\ \varphi)$   
**by**(simp add: lift-dnf-qe-def qfree-lift-dnf-qe qfree-lift-eq-qe)

**lemma** I-lift-dnf-qe:  
 $(\bigwedge as. (\forall a \in set\ as. depends_0\ a) \implies qfree\ (qe\ as)) \implies$   
 $(\bigwedge as. (\forall a \in set\ as. depends_0\ a \wedge \neg solvable_0\ a) \implies is-dnf-qe\ qe\ as) \implies$   
 $I\ (lift-dnf-qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$   
**by**(simp add: lift-dnf-qe-def I-lift-dnf-qe qfree-lift-eq-qe I-lift-eq-qe)

**lemma** I-lift-dnf-qe2:  
 $qe \in lists\ |depends_0| \rightarrow |qfree| \implies$   
 $(\forall as \in lists\ (|depends_0| \cap -\ |solvable_0|). is-dnf-qe\ qe\ as) \implies$   
 $I\ (lift-dnf-qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$   
**using** in-lists-conv-set[**where** ?'a = 'a]  
**by**(simp add: Pi-def I-lift-dnf-qe Int-def Compl-eq)

**end**

**end**

### 3 DLO

```

theory DLO
imports QE Complex-Main
begin

```

#### 3.1 Basics

```

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

```

```

instance real :: dlo
proof
  fix r s :: real
  let ?v = (r + s) / 2
  assume r < s
  hence r < ?v ∧ ?v < s by simp
  thus ∃ v. r < v ∧ v < s ..
next
  fix r :: real
  have r < r + 1 by arith
  thus ∃ s. r < s ..
next
  fix r :: real
  have r - 1 < r by arith
  thus ∃ s. s < r ..
qed

datatype atom = Less nat nat | Eq nat nat

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

abbreviation is-Eq ≡ Not ∘ is-Less

lemma is-Less-iff: is-Less a = (∃ i j. a = Less i j)
by(cases a) auto
lemma is-Eq-iff: (∀ i j. a ≠ Less i j) = (∃ i j. a = Eq i j)
by(cases a) auto
lemma not-is-Eq-iff: (∀ i j. a ≠ Eq i j) = (∃ i j. a = Less i j)
by(cases a) auto

fun negdlo :: atom ⇒ 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 ⇒ 'a::dlo list ⇒ bool where
Idlo (Eq i j) xs = (xs!i = xs!j) |
Idlo (Less i j) xs = (xs!i < xs!j)

fun dependsdlo :: atom ⇒ bool where
dependsdlo (Eq i j) = (i=0 | j=0) |
dependsdlo (Less i j) = (i=0 | j=0)

fun decrdlo :: atom ⇒ 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 (λa. 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** `⟨Sign.revert-abbrev @{const-abbrev DLO.I}⟩`

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

**lemma** `set-lbounds: set(lbounds as) = {i. Less (Suc i) 0 ∈ set as}`  
**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 ∨ Eq 0 (Suc k) ∈ set as}`  
**by** `(auto simp: ebounds-def split: atom.splits nat.splits)`

**abbreviation** `LB f xs ≡ {xs!i|i. Less (Suc i) 0 ∈ set(DLO.atoms0 f)}`  
**abbreviation** `UB f xs ≡ {xs!i|i. Less 0 (Suc i) ∈ set(DLO.atoms0 f)}`  
**definition** `EQ f xs = {xs!k|k.  
Eq (Suc k) 0 ∈ set(DLO.atoms0 f) ∨ Eq 0 (Suc k) ∈ set(DLO.atoms0 f)}`

**lemma** `EQ-And[simp]: EQ (And f g) xs = (EQ f xs ∪ EQ g xs)`  
**by** `(auto simp:EQ-def)`



**lemma** *EQ-Or*[simp]:  $EQ (Or f g) xs = (EQ f xs \cup EQ g xs)$   
**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**(*rename-tac a*)  
**apply**(*case-tac a*)  
**apply**(*simp-all add:nth.simps split:nat.splits*)  
**done**

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

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

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

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

```

lemma min-inf:
  ngfree f  $\implies \exists x. \forall y \leq x. DLO.I (inf_- f) xs = DLO.I f (y \# xs)$ 
  (is -  $\implies \exists x. ?P f x$ )
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 fastforce+
  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 fastforce+
  hence ?P (Or f1 f2) (min x1 x2) by (force simp:or-def)
  thus ?case ..
qed simp-all

lemma plus-inf:
  ngfree f  $\implies \exists x. \forall y \geq x. DLO.I (inf_+ f) xs = DLO.I f (y \# xs)$ 
  (is -  $\implies \exists x. ?P f x$ )
proof (induct f)
  have dlo-bound:  $\bigwedge z :: 'a. \exists x. \forall y \geq x. y > z$ 
  proof -
    fix z
    from no-ub obtain w :: 'a where w > z ..
    then have  $\forall y \geq w. y > z$  by auto
    then show ?thesis z ..
  qed
  case (Atom a)
  show ?case
  proof (cases a rule: aplus-inf.cases)
    case 1 thus ?thesis
    by (simp add: nth-Cons') (metis linorder-not-less)
  next
    case 2 thus ?thesis by (auto intro: dlo-bound)
  next

```

```

    case 5 thus ?thesis
      by simp (metis dlo-bound less-imp-neq)
  next
    case 6 thus ?thesis
      by simp (metis dlo-bound less-imp-neq)
  qed simp-all
next
  case (And f1 f2)
  then obtain x1 x2 where ?P f1 x1 ?P f2 x2 by fastforce+
  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 fastforce+
  hence ?P (Or f1 f2) (max x1 x2) by (force simp:or-def)
  thus ?case ..
qed simp-all

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

lemma LBex:
  [[ nqfree f; DLO.I f (x#xs); ¬DLO.I (inf_ f) xs; x ∉ EQ f xs ]]
  ⇒ ∃ l ∈ 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); ¬DLO.I (inf_+ f) xs; x ∉ EQ f xs ]]
  ⇒ ∃ u ∈ 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

end

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**  $UB\ f\ xs = (\lambda k. xs!k) \text{ ` } set(ubounds(DLO.atoms_0\ f))$   
**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*:  $nqfree\ \varphi \implies qfree(inf\_-\ \varphi)$   
**by**(*induct \varphi*)(*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*:  $nqfree\ \varphi \implies qfree(inf\_+\ \varphi)$   
**by**(*induct \varphi*)(*simp-all add:qfree-aplus-inf*)

**end**

**theory** *QEdlo*  
**imports** *DLO*  
**begin**

### 3.2 DNF-based quantifier elimination

**definition** *qe-dlo<sub>1</sub>* :: *atom list*  $\Rightarrow$  *atom fm* **where**  
*qe-dlo<sub>1</sub> as* =  
 (*if Less 0 0*  $\in$  *set as* *then FalseF* *else*  
*let lbs* = [*i. Less (Suc i) 0*  $\leftarrow$  *as*]; *ubs* = [*j. Less 0 (Suc j)*  $\leftarrow$  *as*];  
     *pairs* = [*Atom(Less i j). i*  $\leftarrow$  *lbs, j*  $\leftarrow$  *ubs*]  
*in list-conj pairs*)

**theorem** *I-qe-dlo<sub>1</sub>*:  
**assumes** *less*:  $\forall a \in set\ as. is\_Less\ a$  **and** *dep*:  $\forall a \in set\ as. depends_{dlo}\ a$   
**shows**  $DLO.I\ (qe\_dlo_1\ as)\ xs = (\exists x. \forall a \in set\ as. I_{dlo}\ a\ (x\#\ xs))$   
 (**is**  $?L = ?R$ )

**proof**  
**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** *?*:  $Less\ 0\ 0 \notin set\ as \implies \forall a \in set\ as.$   
 ( $\exists i \in ?Ls. a = Less\ (Suc\ i)\ 0$ )  $\vee$  ( $\exists i \in ?Us. a = Less\ 0\ (Suc\ i)$ )  
**proof**  
**fix** *a* **assume**  $Less\ 0\ 0 \notin set\ as\ a \in set\ as$

**then obtain**  $i\ j$  **where**  $[simp]: a = Less\ i\ j$   
**using**  $less$  **by**  $(force\ simp:is-Less-iff)$   
**with dep obtain**  $k$  **where**  $i = 0 \wedge j = Suc\ k \vee i = Suc\ k \wedge j = 0$   
**using**  $\langle Less\ 0\ 0 \notin set\ as \rangle \langle a \in set\ as \rangle$   
**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**  $\langle a \in set\ as \rangle$  **by**  $force$   
**ultimately show**  $(\exists i \in ?Ls. a = Less\ (Suc\ i)\ 0) \vee (\exists i \in ?Us. a = Less\ 0\ (Suc\ i))$   
**by**  $force$   
**qed**  
**assume**  $qe1: ?L$   
**hence**  $0: Less\ 0\ 0 \notin set\ as$  **by**  $(auto\ simp:qe-dlo_1-def)$   
**with**  $qe1$  **have**  $1: \forall x \in ?Ls. \forall y \in ?Us. xs\ !\ x < xs\ !\ y$   
**by**  $(fastforce\ simp:qe-dlo_1-def)$   
**have**  $finite: finite\ ?Ls\ finite\ ?Us$  **by**  $(rule\ finite-set)+$   
**{ fix**  $i\ x$   
**assume**  $Less\ i\ 0 \in set\ as \mid Less\ 0\ i \in set\ as$   
**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**  $1\ finite$  **by**  $auto$   
**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**  $nonempty\ finite$  **by**  $auto$   
**hence**  $\forall a \in set\ as. I_{dlo}\ a\ (m\ \#\ xs)$  **using**  $2[OF\ 0]$  **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**  $\forall a \in set\ as. I_{dlo}\ a\ (m\ \#\ xs)$  **using**  $2[OF\ 0]$   $asm\ finite$  **by**  $auto$   
**hence**  $?R\ ..\ }$   
**moreover**  
**{ assume**  $asm: ?Ls = \{\} \wedge ?Us \neq \{\}$   
**then obtain**  $m$  **where**  $m < ?ub$  **using**  $no-lb$  **by**  $blast$   
**hence**  $\forall a \in set\ as. I_{dlo}\ a\ (m\ \#\ xs)$  **using**  $2[OF\ 0]$   $asm\ finite$  **by**  $auto$   
**hence**  $?R\ ..\ }$   
**moreover**  
**{ assume**  $?Ls = \{\} \wedge ?Us = \{\}$   
**hence**  $?R$  **using**  $2[OF\ 0]$  **by**  $(auto\ simp\ add:less)$   
**}**  
**ultimately show**  $?R$  **by**  $blast$   
**next**  
**assume**  $?R$   
**then obtain**  $x$  **where**  $1: \forall a \in set\ as. I_{dlo}\ a\ (x\ \#\ xs)\ ..$   
**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 (fastforce simp: qe-dlo<sub>1</sub>-def is-Less-iff split:atom.splits nat.splits)  
 qed

**lemma** *I-qe-dlo<sub>1</sub>-pretty*:

$\forall a \in \text{set as. is-Less } a \wedge \text{depends}_{dlo} a \implies \text{DLO.is-dnf-qe - qe-dlo}_1 \text{ as}$   
 by (metis I-qe-dlo<sub>1</sub>)

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

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

**fun** *subst<sub>0</sub>* ::  $\text{atom} \Rightarrow \text{atom} \Rightarrow \text{atom}$  **where**

*subst<sub>0</sub>* (Eq  $i j$ )  $a = (\text{case } a \text{ of}$   
 $\text{Less } m n \Rightarrow \text{Less } (\text{subst } i j m) (\text{subst } i j n)$   
 $\mid \text{Eq } m n \Rightarrow \text{Eq } (\text{subst } i j m) (\text{subst } i j n)$ )

**lemma** *subst<sub>0</sub>-pretty*:

*subst<sub>0</sub>* (Eq  $i j$ ) (Less  $m n$ ) = Less (subst  $i j m$ ) (subst  $i j n$ )

*subst<sub>0</sub>* (Eq  $i j$ ) (Eq  $m n$ ) = Eq (subst  $i j m$ ) (subst  $i j n$ )

by auto

**interpretation** *DLO<sub>e</sub>*:

$\text{ATOM-EQ } \text{neg}_{dlo} (\lambda a. \text{True}) \text{I}_{dlo} \text{depends}_{dlo} \text{decr}_{dlo}$   
 $(\lambda \text{Eq } i j \Rightarrow i=0 \vee j=0 \mid a \Rightarrow \text{False})$   
 $(\lambda \text{Eq } i j \Rightarrow i=j \mid a \Rightarrow \text{False}) \text{subst}_0$

apply (unfold-locales)

apply (fastforce simp: subst-def nth-Cons' split:atom.splits if-split-asm)

apply (simp add: subst-def split:atom.splits)

apply (fastforce simp: subst-def nth-Cons' split:atom.splits)

apply (fastforce simp add: subst-def split:atom.splits)

done

**setup**  $\langle \text{Sign.revert-abbrev } @\{\text{const-abbrev } \text{DLO}_e.\text{lift-dnf-qe}\} \rangle$

**definition** *qe-dlo* =  $\text{DLO}_e.\text{lift-dnf-qe } \text{qe-dlo}_1$

**lemma** *qfree-qe-dlo<sub>1</sub>*: qfree (qe-dlo<sub>1</sub> as)

by (auto simp: qe-dlo<sub>1</sub>-def intro!: qfree-list-conj)

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

**unfolding** *qe-dlo-def*

by (fastforce intro!: I-qe-dlo<sub>1</sub> qfree-qe-dlo<sub>1</sub> DLO<sub>e</sub>.I-lift-dnf-qe

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

**theorem** *qfree-qe-dlo*: qfree (qe-dlo  $\varphi$ )

by (simp add: qe-dlo-def DLO<sub>e</sub>.qfree-lift-dnf-qe qfree-qe-dlo<sub>1</sub>)

**end**

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

**definition** *interpret* :: *atom fn*  $\Rightarrow$  *'a::dlo list*  $\Rightarrow$  *bool* **where**  
*interpret* = *Logic.interpret I<sub>dlo</sub>*

**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* =  
*Logic.interpret.simps(1,2,4-7)[of I<sub>dlo</sub>, folded interpret-def]*  
*interpret-others interpret-Atoms*

**method-setup** *dlo-reify* =  $\langle$   
  *Scan.succeed*  
  (*fn ctxt* =>  
    *Method.SIMPLE-METHOD' (Reification.tac ctxt @{\thms reify-eqs} NONE*  
      *THEN' simp-tac (put-simpset HOL-basic-ss ctxt addsimps [@{\thminterpret-def}])))*  
   $\rangle$  *dlo reification*

**declare** *I<sub>dlo</sub>.simps(1)[code]*  
**declare** *Logic.interpret.simps[code del]*  
**declare** *Logic.interpret.simps(1-2)[code]*

### 3.3 Examples

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

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

```

lemma  $\exists x::real. a+b < x \wedge x < c*d$ 
apply dlo-reify
apply (subst I-qe-dlo [symmetric])
apply normalization
oops

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

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

lemma  $\neg(\exists x y z. \forall u::real. x < x \vee \neg x < u \vee x < y \wedge y < z \wedge \neg x < z)$ 
apply dlo-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

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

end

```



```

theory QEdlo-fr
imports DLO
begin

```

### 3.4 Interior Point Method

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

```

fun 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 ≡ amapfm (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 ⟷ (∀ y ∈ {l <..x}. y ∉ LB f xs) ∧ (∀ y ∈ {x <..u}. y ∉ UB f xs)

```

```

lemma nolub-And[simp]:

```

```

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

```

```

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

```

```

begin

```

```

lemma innermost-intvl:

```

```

  [[ nqfree f; nolub f xs l x u; l < x; x < u; x ∉ EQ f xs;

```

```

    DLO.I f (x#xs); l < y; y < u]]
    ⇒ DLO.I f (y#xs)
proof(induct f)
  case (Atom a)
  show ?case
  proof (cases a)
    case (Less i j)
    then show ?thesis using Atom
      unfolding nolub-def
      by (clarsimp simp: nth.simps Ball-def split:if-split-asm nat.splits)
        (metis not-le-imp-less order-antisym order-less-trans)+
  next
  case [simp]: (Eq i j)
  show ?thesis
  proof (cases i)
    case [simp]: 0
    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 [simp]: Suc
  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 (fastforce)
next
  case (Or f1 f2) thus ?case by (fastforce)
qed simp+

lemma I-subst22:
  nqfree f ⇒ xs!l < x ∧ x < xs!u ⇒ nolub f xs (xs!l) x (xs!u)
  ⇒ ∀ x ∈ {xs!l <..xs!u}. DLO.I f (x#xs) ∧ x ∉ EQ f xs
  ⇒ 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:if-split-asm)
  done
next
  case Or thus ?case by (simp add: Ball-def)(metis innermost-intvl)

```

qed auto

end

**definition**

$qe\text{-interior}_1 \varphi =$   
(let  $as = DLO.atoms_0 \varphi$ ;  $lbs = lbounds\ as$ ;  $ubs = ubounds\ as$ ;  $ebs = ebounds\ as$ ;  
intrs = [And (Atom(Less l u)) (subst<sub>2</sub> l u  $\varphi$ ).  $l \leftarrow lbs$ ,  $u \leftarrow ubs$ ]  
in list-disj (inf<sub>-</sub>  $\varphi$  # inf<sub>+</sub>  $\varphi$  # intrs @ map (subst  $\varphi$ ) ebs))

**lemma** dense-interval:

**assumes** finite L finite U  $l \in L$   $u \in U$   $l < x < u$   $P(x::'a::dlo)$

**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  $\langle l \in L \rangle \langle l < x \rangle$  by (blast intro:order-less-imp-le)

moreover have  $?U \neq \{\}$  using  $\langle u:U \rangle \langle x < u \rangle$  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  $\langle finite\ L \rangle \langle finite\ U \rangle$  by force+

moreover have  $?ll \in L$

**proof**

show  $?ll \in ?L$  using  $\langle finite\ L \rangle$  Max-in[OF -  $\langle ?L \neq \{\} \rangle$ ] by simp

show  $?L \subseteq L$  by blast

qed

moreover have  $?uu \in U$

**proof**

show  $?uu \in ?U$  using  $\langle finite\ U \rangle$  Min-in[OF -  $\langle ?U \neq \{\} \rangle$ ] by simp

show  $?U \subseteq U$  by blast

qed

moreover have  $?ll < x$  using  $\langle finite\ L \rangle \langle ?L \neq \{\} \rangle$  by simp

moreover have  $x < ?uu$  using  $\langle finite\ U \rangle \langle ?U \neq \{\} \rangle$  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** nqfree  $\varphi$  **shows**  $DLO.I\ (qe\text{-interior}_1\ \varphi)\ xs = (\exists x. DLO.I\ \varphi\ (x\#\ xs))$

(is  $?QE = ?EX$ )

**proof**

assume  $?QE$

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

hence  $?EX$  using  $\langle ?QE \rangle$  min-inf[of  $\varphi\ xs$ ]  $\langle nqfree\ \varphi \rangle$

by (auto simp add:qe-interior<sub>1</sub>-def amap-fm-list-disj)

```

} moreover
{ assume  $DLO.I (inf_+ \varphi) xs$ 
  hence  $?EX$  using  $\langle ?QE \rangle$   $plus-inf[of \varphi xs]$   $\langle nqfree \varphi \rangle$ 
  by( $auto simp add:qe-interior_1-def amap-fm-list-disj$ )
} 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  $\langle ?QE \rangle$   $\langle nqfree \varphi \rangle$  obtain  $l u$ 
  where  $DLO.I (subst_2 l u \varphi) xs$  and  $xs!l < xs!u$ 
  by( $fastforce simp: qe-interior_1-def set-lbounds set-ubounds I-subst EQ-conv-set-ebounds$ )
  moreover then obtain  $x$  where  $xs!l < x \wedge x < xs!u$  by( $metis dense$ )
  ultimately have  $DLO.I \varphi (x \# xs)$ 
  using  $\langle nqfree \varphi \rangle$   $I-subst_21[OF \langle nqfree \varphi \rangle \langle xs!l < xs!u \rangle]$  by  $simp$ 
  hence  $?EX ..$  }
ultimately show  $?EX$  by  $blast$ 
next
let  $?as = DLO.atoms_0 \varphi$  let  $?E = set(ebounds ?as)$ 
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  $\langle nqfree \varphi \rangle$  by( $auto simp:qe-interior_1-def$ )
} moreover
{ assume  $\exists k \in ?E. DLO.I (subst \varphi k) xs$ 
  hence  $?QE$  by( $force simp:qe-interior_1-def$ ) } 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  $\langle nqfree \varphi \rangle$  by ( $force simp:set-ebounds EQ-def I-subst$ )
  hence  $x \notin EQ \varphi xs$  using  $x$  by  $fastforce$ 
  obtain  $l$  where  $l \in LB \varphi xs$   $l < x$ 
  using  $LBex[OF \langle nqfree \varphi \rangle x \langle \neg DLO.I(inf_- \varphi) xs \rangle \langle x \notin EQ \varphi xs \rangle]$  ..
  obtain  $u$  where  $u \in UB \varphi xs$   $x < u$ 
  using  $UBex[OF \langle nqfree \varphi \rangle x \langle \neg DLO.I(inf_+ \varphi) xs \rangle \langle x \notin EQ \varphi xs \rangle]$  ..
  have  $\exists l \in LB \varphi xs. \exists u \in UB \varphi xs. l < x \wedge x < u \wedge nolub \varphi xs l x u \wedge (\forall y. l < y$ 
 $\wedge y < u \longrightarrow DLO.I \varphi (y \# xs))$ 
  using  $dense-interval[where P = \lambda x. DLO.I \varphi (x \# xs), OF finite-LB finite-UB$ 
 $\langle l:LB \varphi xs \rangle \langle u:UB \varphi xs \rangle \langle l < x \rangle \langle x < u \rangle x]$   $x$   $innermost-intvl[OF \langle nqfree \varphi \rangle - - - \langle x$ 
 $\notin EQ \varphi xs \rangle]$ 
  by ( $simp add:nolub-def$ )
  then obtain  $m n$  where
   $Less (Suc m) 0 \in set ?as$   $Less 0 (Suc n) \in set ?as$ 
   $xs!m < x \wedge x < xs!n$ 
   $nolub \varphi xs (xs!m) x (xs!n)$ 
   $\forall y. xs!m < y \wedge y < xs!n \longrightarrow DLO.I \varphi (y \# xs)$ 
  by  $blast$ 
  moreover
  hence  $DLO.I (subst_2 m n \varphi) xs$  using  $noE$ 
  by( $force intro!: I-subst_22[OF \langle nqfree \varphi \rangle]$ )
  ultimately have  $?QE$ 

```

```

    by(fastforce simp add:qe-interior1-def bex-Un set-lbounds set-ubounds)
  } ultimately show ?QE by blast
qed

```

```

lemma qfree-asubst2: qfree (asubst2 l u a)
by(cases (l,u,a) rule:asubst2.cases) simp-all

```

```

lemma qfree-subst2: nqfree  $\varphi \implies$  qfree (subst2 l u  $\varphi$ )
by(induct  $\varphi$ ) (simp-all add:qfree-asubst2)

```

```

lemma qfree-interior1: nqfree  $\varphi \implies$  qfree(qe-interior1  $\varphi$ )
apply(simp add:qe-interior1-def)
apply(rule qfree-list-disj)
apply (auto simp:qfree-min-inf qfree-plus-inf qfree-subst2 qfree-map-fm)
done

```

```

definition qe-interior = DLO.lift-nnf-qe qe-interior1

```

```

lemma qfree-qe-interior: qfree(qe-interior  $\varphi$ )
by(simp add: qe-interior-def DLO.qfree-lift-nnf-qe qfree-interior1)

```

```

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-interior1 I-interior1)

```

```

end

```

```

theory QEdlo-inf
imports DLO
begin

```

### 3.5 Quantifier elimination with infinitesimals

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

```

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

```

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

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

**lemma**  $nolb\text{-}And[simp]$ :

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

**apply**  $(clarsimp\ simp:nolb-def)$

**apply**  $blast$

**done**

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

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

**apply**  $(clarsimp\ simp:nolb-def)$

**apply**  $blast$

**done**

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

**begin**

**lemma**  $innermost\text{-}intvl$ :

$[[\ nqfree\ \varphi; nolb\ \varphi\ xs\ l\ x; l < x; x \notin EQ\ \varphi\ xs; DLO.I\ \varphi\ (x\#\!xs); l < y; y \leq x\ ]]$   
 $\implies DLO.I\ \varphi\ (y\#\!xs)$

**proof**  $(induct\ \varphi)$

**case**  $(Atom\ a)$

**show**  $?case$

**proof**  $(cases\ a)$

**case**  $(Less\ i\ j)$

**then show**  $?thesis\ using\ Atom$

**unfolding**  $nolb-def$

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

$(metis\ not-le-imp-less\ order-antisym\ order-less-trans)+$

**next**

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

**apply**  $(clarsimp\ simp:EQ-def\ nolb-def\ nth-Cons')$

**apply**  $(case-tac\ i=0 \wedge j=0)$  **apply**  $simp$

**apply**  $(case-tac\ i\neq 0 \wedge j\neq 0)$  **apply**  $simp$

**apply**  $(case-tac\ i=0 \wedge j\neq 0)$  **apply**  $(fastforce\ split:if-split-asm)$

**apply**  $(case-tac\ i\neq 0 \wedge j=0)$  **apply**  $(fastforce\ split:if-split-asm)$

**apply**  $arith$

**done**

**qed**

**next**

**case**  $And$  **thus**  $?case\ by\ (fastforce)$

**next**

**case**  $Or$  **thus**  $?case\ by\ (fastforce)$

**qed**  $simp+$

**lemma** *I-subst-peps2*:  
 $ngfree\ \varphi \implies xs!l < x \implies nolb\ \varphi\ xs\ (xs!l)\ x \implies x \notin EQ\ \varphi\ xs$   
 $\implies \forall y \in \{xs!l <.. x\}. DLO.I\ \varphi\ (y\#xs)$   
 $\implies DLO.I\ (subst_+\ \varphi\ l)\ xs$   
**proof**(*induct*  $\varphi$ )  
  **case** *FalseF* **thus** *?case*  
    **by** *simp* (*metis 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 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*  
  
**end**

**lemma** *dense-interval*:  
**assumes** *finite L l ∈ L l < x P(x::'a::dlo)*  
**and** *dense:  $\bigwedge y l. [\forall y \in \{l <.. <x\}. y \notin L; l < x; l < y; y \leq x] \implies P\ y$*   
**shows**  $\exists l \in L. l < x \wedge (\forall y \in \{l <.. <x\}. y \notin L) \wedge (\forall y. l < y \wedge y \leq x \longrightarrow P\ y)$   
**proof** –  
  **let**  $?L = \{l \in L. l < x\}$   
  **let**  $?ll = Max\ ?L$   
  **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**  $\langle finite\ L \rangle$  **by** *force*  
  **moreover** **have**  $?ll \in L$   
  **proof**  
    **show**  $?ll \in ?L$  **using**  $\langle finite\ L \rangle\ Max-in[OF - \langle ?L \neq \{\} \rangle]$  **by** *simp*  
    **show**  $?L \subseteq L$  **by** *blast*  
  **qed**  
  **moreover** **have**  $?ll < x$  **using**  $\langle finite\ L \rangle \langle ?L \neq \{\} \rangle$  **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*:  
 $ngfree\ \varphi \implies DLO.I\ (subst_+\ \varphi\ l)\ xs \longrightarrow$   
 $(\exists leps > xs!l. \forall x. xs!l < x \wedge x \leq leps \longrightarrow DLO.I\ \varphi\ (x\#xs))$   
**proof**(*induct*  $\varphi$ )  
  **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(7))
    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.atoms0  $\varphi$  let ?ebs = ebounds ?as
assume ?QE
{ assume DLO.I (inf_  $\varphi$ ) xs
  hence ?EX using <?QE> min-inf[of  $\varphi$  xs] <ngfree  $\varphi$ >
    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 <?QE> <ngfree  $\varphi$ > obtain l where DLO.I (subst+  $\varphi$  l) xs

```



```

    by(fastforce 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 fastforce
  hence ?EX .. }
ultimately show ?EX by blast
next
let ?as = DLO.atoms0  $\varphi$  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:if-split-asm)
  hence  $x \notin$  EQ  $\varphi$  xs using x by fastforce
  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  $\wedge$  nolb  $\varphi$  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:nolb-def)
  then obtain m
    where *: Less (Suc m) 0  $\in$  set ?as  $\wedge$  xs!m < x  $\wedge$  nolb  $\varphi$  xs (xs!m) x
       $\wedge$  ( $\forall y. xs!m < y \wedge y \leq x \longrightarrow$  DLO.I  $\varphi$  (y#xs))
    by blast
  then have DLO.I (subst+  $\varphi$  m) xs
    using noE by(auto intro!: I-subst-peps2[OF  $\langle$ ngfree  $\varphi$  $\rangle$ ])
  with * 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<sub>+</sub> k a)  
 by(cases (k,a) rule:astubst-peps.cases) simp-all

**lemma** qfree-subst-peps: ngfree  $\varphi \implies$  qfree (subst<sub>+</sub>  $\varphi$  k)  
 by(induct  $\varphi$ ) (simp-all add:qfree-astubst-peps)

**lemma** qfree-qe-eps<sub>1</sub>: ngfree  $\varphi \implies$  qfree(qe-eps<sub>1</sub>  $\varphi$ )  
 apply(simp add:qe-eps<sub>1</sub>-def)  
 apply(rule qfree-list-disj)  
 apply (auto simp:qfree-min-inf qfree-subst-peps qfree-map-fm)  
 done

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

**lemma**  $qfree-qe-eps: qfree(qe-eps\ \varphi)$   
**by**( $simp\ add: qe-eps-def\ DLO.qfree-lift-nnf-qe\ qfree-qe-eps_1$ )

**lemma**  $I-qe-eps: DLO.I\ (qe-eps\ \varphi)\ xs = DLO.I\ \varphi\ xs$   
**by**( $simp\ add: qe-eps-def\ DLO.I-lift-nnf-qe\ qfree-qe-eps_1\ I-qe-eps_1$ )

**end**

## 4 Linear real arithmetic

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

**declare**  $iprod-assoc[simp]$

### 4.1 Basics

#### 4.1.1 Syntax and Semantics

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

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

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

**lemma**  $is-Less-iff: is-Less\ f = (\exists\ r\ rs.\ f = Less\ r\ rs)$   
**by**( $induct\ f$ ) *auto*

**lemma**  $is-Eq-iff: (\forall\ i\ j.\ a \neq Less\ i\ j) = (\exists\ i\ j.\ a = Eq\ i\ j)$   
**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**  $I_R :: \text{atom} \Rightarrow \text{real list} \Rightarrow \text{bool}$  **where**  
 $I_R (\text{Less } r \text{ } cs) \text{ } xs = (r < \langle cs, xs \rangle) \mid$   
 $I_R (\text{Eq } r \text{ } cs) \text{ } xs = (r = \langle cs, xs \rangle)$

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

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

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

**proof** *goal-cases*

**case** 1

**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<sub>R</sub>-def split:list.splits*)

**done**

**next**

**case** 2

**thus** *?case by(simp add:atoms<sub>0</sub>-def)*

**qed**

**setup**  $\langle \text{Sign.revert-abbrev } @\{\text{const-abbrev } R.I\} \rangle$

**setup**  $\langle \text{Sign.revert-abbrev } @\{\text{const-abbrev } R.\text{lift-nnf-qe}\} \rangle$

#### 4.1.2 Shared constructions

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

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

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

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

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

**lemma** *set-lbounds:*

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

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

**lemma** *set-ubounds:*

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

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

**lemma** *set-ebounds:*

$set(ebounds\ as) = \{(r/c, (-1/c) *s\ cs) \mid r\ c\ cs.\ Eq\ r\ (c\#cs) \in\ set\ as \wedge\ c \neq 0\}$   
**by**(force simp: ebounds-def split:list.splits atom.splits if-splits)

**abbreviation EQ where**

$EQ\ f\ xs \equiv \{(r - \langle cs, xs \rangle) / c \mid r\ c\ cs.\ Eq\ r\ (c\#cs) \in\ set(R.atoms_0\ f) \wedge\ c \neq 0\}$

**abbreviation LB where**

$LB\ f\ xs \equiv \{(r - \langle cs, xs \rangle) / c \mid r\ c\ cs.\ Less\ r\ (c\#cs) \in\ set(R.atoms_0\ f) \wedge\ c > 0\}$

**abbreviation UB where**

$UB\ f\ xs \equiv \{(r - \langle cs, xs \rangle) / c \mid r\ c\ cs.\ Less\ r\ (c\#cs) \in\ set(R.atoms_0\ f) \wedge\ c < 0\}$

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

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

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

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

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

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

**definition eval :: real \* real list  $\Rightarrow$  real list  $\Rightarrow$  real where**

$eval\ rcs\ xs = fst\ rcs + \langle snd\ rcs, xs \rangle$

**lemma I-asubst:**

$I_R\ (asubst\ t\ a)\ xs = I_R\ a\ (eval\ t\ xs\ \# xs)$

**proof**(cases a)

**case** (Less r cs)

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

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

arith

**next**

**case** (Eq r cs)

**thus** ?thesis

**by**(cases t, cases cs, simp-all add:eval-def distrib-left iprod-left-add-distrib)

arith

**qed**

**lemma I-subst:**

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

**by**(induct  $\varphi$ )(simp-all add:I-asubst)

**lemma I-subst-pretty:**

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

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

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

$inf\_ (And\ \varphi_1\ \varphi_2) = and\ (inf\_ \varphi_1)\ (inf\_ \varphi_2) \mid$

$inf\_ (Or\ \varphi_1\ \varphi_2) = or\ (inf\_ \varphi_1)\ (inf\_ \varphi_2) \mid$

$inf\_ (Atom(Less\ r\ (c\#cs))) =$

(if  $c < 0$  then TrueF else if  $c > 0$  then FalseF else Atom(Less r cs)) |  
 $\text{inf}_- (\text{Atom}(\text{Eq } r (c \# cs))) = (\text{if } c = 0 \text{ then Atom}(\text{Eq } r cs) \text{ else FalseF})$  |  
 $\text{inf}_- \varphi = \varphi$

**fun plus-inf :: atom fm  $\Rightarrow$  atom fm (inf<sub>+</sub>) where**  
 $\text{inf}_+ (\text{And } \varphi_1 \varphi_2) = \text{and } (\text{inf}_+ \varphi_1) (\text{inf}_+ \varphi_2)$  |  
 $\text{inf}_+ (\text{Or } \varphi_1 \varphi_2) = \text{or } (\text{inf}_+ \varphi_1) (\text{inf}_+ \varphi_2)$  |  
 $\text{inf}_+ (\text{Atom}(\text{Less } r (c \# cs))) =$   
 (if  $c > 0$  then TrueF else if  $c < 0$  then FalseF else Atom(Less r cs)) |  
 $\text{inf}_+ (\text{Atom}(\text{Eq } r (c \# cs))) = (\text{if } c = 0 \text{ then Atom}(\text{Eq } r cs) \text{ else FalseF})$  |  
 $\text{inf}_+ \varphi = \varphi$

**lemma qfree-min-inf:**  $\text{qfree } \varphi \Longrightarrow \text{qfree}(\text{inf}_- \varphi)$   
**by**(induct  $\varphi$  rule:min-inf.induct) simp-all

**lemma qfree-plus-inf:**  $\text{qfree } \varphi \Longrightarrow \text{qfree}(\text{inf}_+ \varphi)$   
**by**(induct  $\varphi$  rule:plus-inf.induct) simp-all

**lemma min-inf:**  
 $\text{ngfree } f \Longrightarrow \exists x. \forall y \leq x. R.I (\text{inf}_- f) xs = R.I f (y \# xs)$   
 (is -  $\Longrightarrow \exists x. ?P f x$ )

**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 fastforce+
  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 fastforce+
  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) (( $r - \langle cs, xs \rangle - 1$ )/ $c$ ) using Less Cons
        by(auto simp add: field-simps)
        hence ?thesis .. }
      moreover
      { assume  $c>0$ 

```

```

      hence ?P (Atom a) ((r - ⟨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 fastforce+
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 fastforce+
hence ?P (Or f1 f2) (max x1 x2) by(force simp:or-def)
thus ?case ..
qed simp-all

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

lemma LBe:
  [[ 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(rename-tac a)
apply(case-tac a)

```

```

apply(auto simp add: dependsR-def field-simps split:if-splits list.splits)
apply fastforce+
done

```

```

lemma UBex:
  [[ nqfree f; R.I f (x#xs);  $\neg R.I (inf_+ f) xs$ ;  $x \notin EQ f xs$  ]]
   $\implies \exists u \in UB f xs. x < u$ 
apply(induct f)
apply simp
apply simp
apply(rename-tac a)
apply(case-tac a)
apply(auto simp add: dependsR-def field-simps split:if-splits list.splits)
apply fastforce+
done

```

**end**

```

lemma finite-LB: finite(LB f xs)
proof -
  have  $LB f xs = (\lambda(r,cs). r + \langle cs,xs \rangle) \text{ ` } set(lbounds(R.atoms_0 f))$ 
    by (force simp:set-lbounds image-def field-simps)
  thus ?thesis by simp
qed

```

```

lemma finite-UB: finite(UB f xs)
proof -
  have  $UB f xs = (\lambda(r,cs). r + \langle cs,xs \rangle) \text{ ` } set(ubounds(R.atoms_0 f))$ 
    by (force simp:set-ubounds image-def field-simps)
  thus ?thesis by simp
qed

```

**end**

```

theory QElin
imports LinArith
begin

```

## 4.2 Fourier

```

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

```

```

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

```



```

let ?Ls = set(lbounds as) let ?Us = set(ubounds as)
let ?lbs = UN (r,cs):?Ls. {r + ⟨cs,xs⟩}
let ?ubs = UN (r,cs):?Us. {r + ⟨cs,xs⟩}
have fins: finite ?lbs ∧ finite ?ubs by auto
have 2: ∀f ∈ set as. ∃ r c cs. f = Less r (c#cs) ∧
      (c>0 ∧ (r/c,(-1/c)*s cs) ∈ ?Ls ∨ c<0 ∧ (r/c,(-1/c)*s cs) ∈ ?Us)
  using dep less
  by(fastforce simp:set-lbounds set-ubounds is-Less-iff dependsR-def
      split:list.splits)
assume ?L
have 1: ∀x ∈ ?lbs. ∀y ∈ ?ubs. x < y
proof (rule ballI)+
  fix x y assume x ∈ ?lbs y ∈ ?ubs
  then obtain r cs
    where (r,cs) ∈ ?Ls ∧ x = r + ⟨cs,xs⟩ by fastforce
  moreover from ⟨y ∈ ?ubs⟩ obtain s ds
    where (s,ds) ∈ ?Us ∧ y = s + ⟨ds,xs⟩ by fastforce
  ultimately show x < y using ⟨?L⟩
    by(fastforce simp:qe-FM1-def algebra-simps iprod-left-diff-distrib)
qed
{ assume nonempty: ?lbs ≠ {} ∧ ?ubs ≠ {}
  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 ∧ m < Min ?ubs
    using dense[where 'a = real] by blast
  hence ∀ a ∈ set as. IR a (m#xs) using 2 nonempty
    apply (auto simp: Ball-def)
    apply (auto simp: Bex-def)
    apply (fastforce simp: field-simps)
    done
  hence ?R .. }
moreover
{ assume asm: ?lbs ≠ {} ∧ ?ubs = {}
  have ∀ a ∈ set as. IR a ((Max ?lbs + 1) # xs)
  proof
    fix a assume a ∈ set as
    then obtain r c cs
      where a = Less r (c#cs) c>0 (r/c,(-1/c)*s cs) ∈ ?Ls
      using asm 2
      by (fastforce simp: field-simps)
    moreover hence (r - ⟨cs,xs⟩)/c ≤ Max ?lbs
      using asm fins
      by(auto intro!: Max-ge-iff[THEN iffD2])
        (force simp add:field-simps)
    ultimately show IR a ((Max ?lbs + 1) # xs) by (simp add: field-simps)
  qed
  hence ?R .. }
moreover
{ assume asm: ?lbs = {} ∧ ?ubs ≠ {}

```

**have**  $\forall a \in \text{set } as. I_R a ((\text{Min } ?ubs - 1) \# xs)$   
**proof**  
**fix**  $a$  **assume**  $a \in \text{set } as$   
**then obtain**  $r \ c \ cs$   
**where**  $a = \text{Less } r (c \# cs) \ c < 0 \ (r/c, (-1/c) *_s cs) \in ?Us$   
**using**  $asm \ 2$  **by**  $\text{fastforce}$   
**moreover hence**  $\text{Min } ?ubs \leq (r - \langle cs, xs \rangle) / c$   
**using**  $asm \ fins$   
**by**( $\text{auto intro!}: \text{Min-le-iff}[\text{THEN } \text{iffD2}]$ )  
( $\text{force simp add:field-simps}$ )  
**ultimately show**  $I_R a ((\text{Min } ?ubs - 1) \# xs)$  **by** ( $\text{simp add: field-simps}$ )  
**qed**  
**hence**  $?R \ .. \}$   
**moreover**  
**{ assume**  $?lbs = \{\} \wedge ?ubs = \{\}$   
**hence**  $?R$  **using**  $2$  **less by**  $\text{auto (rule, fast)}$   
**}**  
**ultimately show**  $?R$  **by**  $\text{blast}$   
**next**  
**let**  $?Ls = \text{set}(lbounds \ as)$  **let**  $?Us = \text{set}(ubounds \ as)$   
**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**  $\text{Less } r (c \# cs) \in \text{set } as \ 0 < c \ \text{Less } s (d \# ds) \in \text{set } as \ d < 0$   
**hence**  $r < c*x + \langle cs, xs \rangle \ s < d*x + \langle ds, xs \rangle \ c > 0 \ d < 0$   
**using**  $1$  **by**  $\text{auto}$   
**hence**  $(r - \langle cs, xs \rangle) / c < x \ x < (s - \langle ds, xs \rangle) / d$  **by**( $\text{simp add:field-simps}$ )  
**hence**  $(r - \langle cs, xs \rangle) / c < (s - \langle ds, xs \rangle) / d$  **by**  $\text{arith}$   
**}**  
**thus**  $?L$  **by** ( $\text{auto simp: } \text{qe-FM}_1\text{-def } \text{iproduct-left-diff-distrib } \text{less field-simps } \text{set-lbounds } \text{set-ubounds}$ )  
**qed**

**corollary**  $I\text{-qe-FM}_1\text{-pretty}$ :

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

**fun**  $\text{subst}_0 :: \text{atom} \Rightarrow \text{atom} \Rightarrow \text{atom}$  **where**

$\text{subst}_0 (Eq \ r \ (c \# cs)) \ a = (\text{case } a \ \text{of}$   
 $\text{Less } s \ (d \# ds) \Rightarrow \text{Less } (s - (r*d)/c) \ (ds - (d/c) *_s cs)$   
 $| \ Eq \ s \ (d \# ds) \Rightarrow \text{Eq } (s - (r*d)/c) \ (ds - (d/c) *_s cs)$

**lemma**  $\text{subst}_0\text{-pretty}$ :

$\text{subst}_0 (Eq \ r \ (c \# cs)) (\text{Less } s \ (d \# ds)) = \text{Less } (s - (r*d)/c) \ (ds - (d/c) *_s cs)$   
 $\text{subst}_0 (Eq \ r \ (c \# cs)) (Eq \ s \ (d \# ds)) = \text{Eq } (s - (r*d)/c) \ (ds - (d/c) *_s cs)$   
**by**  $\text{auto}$

**lemma**  $I\text{-subst}_0: \text{depends}_R a \implies c \neq 0 \implies$

$I_R (\text{subst}_0 (Eq\ r\ (c\#\#cs))\ a)\ xs = I_R\ a\ ((r - \langle cs, xs \rangle) / c\ \#\#\ xs)$   
**apply**(cases a)  
**by** (auto simp add: depends<sub>R</sub>-def iprod-left-diff-distrib algebra-simps diff-divide-distrib  
split:list.splits)

**interpretation**  $R_e$ :

**ATOM-EQ** neg<sub>R</sub> (λa. True) I<sub>R</sub> depends<sub>R</sub> decr<sub>R</sub>  
(λEq - (c#-) ⇒ c ≠ 0 | - ⇒ False)  
(λEq r cs ⇒ r=0 ∧ (∀ c ∈ set cs. c=0) | - ⇒ False) subst<sub>0</sub>  
**apply**(unfold-locales)  
**apply**(simp del:subst<sub>0</sub>.simps add:I-subst<sub>0</sub> 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 - ⟨cs,xs⟩) / 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**  $qe\text{-}FM = R_e.\text{lift-dnfeq-qe}\ qe\text{-}FM_1$

**lemma**  $qfree\text{-}qe\text{-}FM_1$ :  $qfree\ (qe\text{-}FM_1\ as)$   
**by**(auto simp:qe-FM<sub>1</sub>-def intro!:qfree-list-conj)

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

**unfolding**  $qe\text{-}FM\text{-}def$   
**apply**(rule  $R_e.I\text{-}lift\text{-}dnfeq\text{-}qe$ )  
**apply**(rule  $qfree\text{-}qe\text{-}FM_1$ )  
**apply**(rule allI)  
**apply**(rule  $I\text{-}qe\text{-}FM_1$ )  
**prefer** 2 **apply** blast  
**apply**(clarify)  
**apply**(drule-tac x=a in bspec) **apply** simp  
**apply**(simp add: depends<sub>R</sub>-def split:atom.splits list.splits)  
**done**

**theorem**  $qfree\text{-}qe\text{-}FM$ :  $qfree\ (qe\text{-}FM\ f)$   
**by**(simp add:qe-FM-def  $R_e.qfree\text{-}lift\text{-}dnfeq\text{-}qe\ qfree\text{-}qe\text{-}FM_1$ )

#### 4.2.1 Tests

**lemmas**  $qesimps = qe\text{-}FM\text{-}def\ R_e.\text{lift-dnfeq-qe}\text{-}def\ R_e.\text{lift-eq-qe}\text{-}def\ R.qelim\text{-}def\ qe\text{-}FM_1\text{-}def$   
 $lbounds\text{-}def\ ubounds\text{-}def\ list\text{-}conj\text{-}def\ list\text{-}disj\text{-}def\ and\text{-}def\ or\text{-}def\ depends_R\text{-}def$

**lemma**  $qe\text{-}FM(TrueF) = TrueF$   
**by**(simp add:qesimps)

**lemma**

$qe\text{-}FM(ExQ (And (Atom(Less 0 [1])) (Atom(Less 0 [-1]))) = Atom(Less 0 []))$   
**by**(*simp add:qesimps*)

**lemma**

$qe\text{-}FM(ExQ (And (Atom(Less 0 [1])) (Atom(Less (- 1) [-1]))) = Atom(Less (- 1) []))$   
**by**(*simp add:qesimps*)

**end**

**theory** *QElin-opt*

**imports** *QElin*

**begin**

#### 4.2.2 An optimization

Atoms are simplified asap.

**definition**

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

**lemma** *asimp-pretty*:

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

**by**(*auto simp:asimp-def*)

**definition**  $qe\text{-}FMo_1 :: atom\ list \Rightarrow atom\ fm$  **where**

$qe\text{-}FMo_1\ as = list\ conj\ [asimp(combine\ p\ q).\ p \leftarrow lbounds\ as,\ q \leftarrow ubounds\ as]$

**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*:  $R.I\ (qe\text{-}FMo_1\ as)\ xs = R.I\ (qe\text{-}FM_1\ as)\ xs$

**by**(*simp add:qe-FM1-def qe-FMo1-def I-asimp*)

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

```

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 Re.I-lift-dnfeq-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: dependsR-def split:atom.splits list.splits)
done

```

```

theorem qfree-qe-FMo: qfree (qe-FMo f)
by(simp add:qe-FMo-def Re.qfree-lift-dnfeq-qe qfree-qe-FMo1)

```

**end**

```

theory FRE
imports LinArith
begin

```

### 4.3 Ferrante-Rackoff

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

```

fun between :: real * real list ⇒ real * real list ⇒ real * real list
where between (r,cs) (s,ds) = ((r+s)/2, (1/2) *s (cs+ds))

```

```

definition FR1 :: atom fm ⇒ atom fm where
FR1 φ =
(let as = R.atoms0 φ; lbs = lbounds as; ub = ubounds as; ebs = ebounds as;
  intrs = [subst φ (between l u) . l ← lbs, u ← ub]
  in list-disj (inf- φ # inf+ φ # intrs @ map (subst φ) ebs))

```

**lemma** *dense-interval*:

```

assumes finite L finite U l ∈ L u ∈ U l < x x < u P(x::real)
and dense: ∧y l u. [∀y∈{l<..
  l < x; x < u; l < y; y < u] ⇒ P y
shows ∃l∈L. ∃u∈U. l < u ∧ (∀y. l < y ∧ y < u → 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 ≠ {} using ⟨l ∈ L⟩ ⟨l < x⟩ by (blast intro:order-less-imp-le)
moreover have ?U ≠ {} using ⟨u:U⟩ ⟨x < u⟩ by (blast intro:order-less-imp-le)
ultimately have ∀ y. ?ll < y ∧ y < x ⟶ y ∉ L ∀ y. x < y ∧ y < ?uu ⟶ y ∉ U
  using ⟨finite L⟩ ⟨finite U⟩ 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
moreover have ?uu ∈ U
proof
  show ?uu ∈ ?U using ⟨finite U⟩ Min-in[OF - ⟨?U ≠ {}⟩] by simp
  show ?U ⊆ U by blast
qed
moreover have ?ll < x using ⟨finite L⟩ ⟨?L ≠ {}⟩ by simp
moreover have x < ?uu using ⟨finite U⟩ ⟨?U ≠ {}⟩ by simp
moreover have ?ll < ?uu using ⟨?ll < x⟩ ⟨x < ?uu⟩ by arith
ultimately show ?thesis using ⟨l < x⟩ ⟨x < u⟩ ⟨?L ≠ {}⟩ ⟨?U ≠ {}⟩
  by (blast intro!:dense greaterThanLessThan-iff[THEN iffD1])
qed

```

lemma dense:

```

[[ nqfree f; ∀ y ∈ {l <..<x}. y ∉ LB f xs; ∀ y ∈ {x <..<u}. y ∉ UB f xs;
  l < x; x < u; x ∉ EQ f xs; R.I f (x#xs); l < y; y < u]
⇒ 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 + ⟨cs,xs⟩ using Atom Less by simp
{ assume c=0 hence ?thesis using Atom Less Cons by simp }
moreover
{ assume c<0
  hence x < (r - ⟨cs,xs⟩)/c (is - < ?u) using ⟨r < c*x + ⟨cs,xs⟩⟩
    by (simp add: field-simps)
  have ?thesis
proof (rule ccontr)
  assume ¬ R.I (Atom a) (y#xs)
  hence ?u ≤ y using Atom Less Cons ⟨c<0⟩
    by (auto simp add: field-simps)
  hence ?u < u using ⟨y < u⟩ by simp
  with ⟨x < ?u⟩ show False using Atom Less Cons ⟨c<0⟩

```

```

      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  $\langle r = c * x + \langle cs, xs \rangle \rangle$  Atom Eq Cons  $\langle l < y \rangle \langle y < u \rangle$ 
      by (auto simp: dependsR-def split: if-splits) }
  ultimately show ?thesis by force
qed
qed
next
case (And f1 f2) thus ?case
  by auto (metis (no-types, opaque-lifting))+
next
case (Or f1 f2) thus ?case
  by auto (metis (no-types, opaque-lifting))+
qed fastforce+

theorem I-FR1:
assumes nqfree  $\varphi$  shows R.I (FR1  $\varphi$ )  $xs = (\exists x. R.I \varphi (x \# xs))$ 
(is ?FR = ?EX)
proof
  assume ?FR
  { assume R.I (inf_  $\varphi$ )  $xs$ 
    hence ?EX using  $\langle ?FR \rangle$  min-inf[OF  $\langle nqfree \varphi \rangle$ , where  $xs = xs$ ]
      by(auto simp add:FR1-def)
  } moreover

```

```

{ assume  $R.I (inf_+ \varphi) xs$ 
  hence  $?EX$  using  $\langle ?FR \rangle plus-inf[OF \langle ngfree \varphi \rangle, \text{where } xs=xs]$ 
    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-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  $x \in EQ \varphi xs$ 
  then obtain  $r cs$ 
    where  $(r,cs) \in set(ebounds(R.atoms_0 \varphi)) \wedge x = r + \langle cs,xs \rangle$ 
    by(force simp:set-ebounds field-simps)
  moreover hence  $R.I (subst \varphi (r,cs)) xs$  using  $x$ 
    by(auto simp: I-subst  $\langle ngfree \varphi \rangle$  eval-def)
  ultimately have  $?FR$  by(force simp:FR1-def) } moreover
{ assume  $\neg R.I (inf_- \varphi) xs$  and  $\neg R.I (inf_+ \varphi) xs$  and  $x \notin EQ \varphi xs$ 
  obtain  $l$  where  $l \in LB \varphi xs$   $l < x$ 
    using  $LBex[OF \langle ngfree \varphi \rangle x \langle \neg R.I (inf_- \varphi) xs \rangle \langle x \notin EQ \varphi xs \rangle] ..$ 
  obtain  $u$  where  $u \in UB \varphi xs$   $x < u$ 
    using  $UBex[OF \langle ngfree \varphi \rangle x \langle \neg R.I (inf_+ \varphi) xs \rangle \langle x \notin EQ \varphi xs \rangle] ..$ 
  have  $\exists l \in LB \varphi xs. \exists u \in UB \varphi xs. l < u \wedge (\forall y. l < y \wedge y < u \longrightarrow R.I \varphi (y\#xs))$ 
    using dense-interval[where  $P = \lambda x. R.I \varphi (x\#xs)$ ,  $OF$  finite-LB finite-UB
 $\langle l:LB \varphi xs \rangle \langle u:UB \varphi xs \rangle \langle l < x \rangle \langle x < u \rangle$ ]  $x$  dense[OF  $\langle ngfree \varphi \rangle$  - - -  $\langle x \notin EQ \varphi xs \rangle$ ]
  by simp
  then obtain  $r c cs s d ds$ 
    where  $Less r (c \# cs) \in set (R.atoms_0 \varphi)$   $Less s (d \# ds) \in set (R.atoms_0 \varphi)$ 
     $\wedge y. (r - \langle cs,xs \rangle) / c < y \implies y < (s - \langle ds,xs \rangle) / d \implies R.I \varphi (y \# xs)$ 
    and  $*: c > 0$   $d < 0$   $(r - \langle cs,xs \rangle) / c < (s - \langle ds,xs \rangle) / d$ 
    by blast
  moreover
  have  $(r - \langle cs,xs \rangle) / c < eval (between (r / c, (-1 / c) *_s cs) (s / d, (-1 / d) *_s ds)) xs$  (is  $?P$ )
    and  $eval (between (r / c, (-1 / c) *_s cs) (s / d, (-1 / d) *_s ds)) xs < (s$ 

```



```

- ⟨ds,xs⟩ / d (is ?Q)
  proof -
    from * have [simp]: c * (c * (d * (d * 4))) > 0
      by (simp add: algebra-split-simps)
    from * have c * s + d * ⟨cs,xs⟩ < d * r + c * ⟨ds,xs⟩
      by (simp add: field-simps)
    with * have (2 * c * c * d) * (d * r + c * ⟨ds,xs⟩)
      < (2 * c * c * d) * (c * s + d * ⟨cs,xs⟩)
      and (2 * c * d * d) * (c * s + d * ⟨cs,xs⟩)
      < (2 * c * d * d) * (d * r + c * ⟨ds,xs⟩) by simp-all
    with * show ?P and ?Q by (auto simp add: field-simps eval-def iprod-left-add-distrib)
  qed
  ultimately have ?FR
    by (fastforce simp: FR1-def bex-Un set-lbounds set-ubounds set-ebounds I-subst
      ⟨ngfree φ⟩)
  } ultimately show ?FR by blast
qed

```

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

```

lemma qfree-FR1: ngfree φ ⇒ qfree (FR1 φ)
apply(simp add:FR1-def)
apply(rule qfree-list-disj)
apply(auto simp:qfree-min-inf qfree-plus-inf set-ubounds set-lbounds set-ebounds
  image-def qfree-map-fm)
done

```

```

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

```

```

theorem qfree-FR: qfree (FR φ)
by(simp add:FR-def R.qfree-lift-nnf-qe qfree-FR1)

```

**end**

```

theory QElin-inf
imports LinArith
begin

```

#### 4.4 Quantifier elimination with infinitesimals

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

```

fun asubst-peps :: real * real list ⇒ atom ⇒ atom fm (asubst+) where
  asubst-peps (r,cs) (Less s (d#ds)) =
    (if d=0 then Atom(Less s ds) else

```

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

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

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

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

$nolb\ (And\ f\ g)\ xs\ l\ x = (nolb\ f\ xs\ l\ x \wedge nolb\ g\ xs\ l\ x)$

**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**

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

**begin**

**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**  $[simp]: (Less\ r\ cs)$

**show**  $?thesis$

**proof**  $(cases\ cs)$

**case**  $Nil$  **thus**  $?thesis$  **using**  $Atom$  **by**  $(simp\ add:depends_R-def)$

**next**

**case**  $[simp]: (Cons\ c\ cs)$

**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 ⟨x<?u⟩ show False using Atom ⟨c<0⟩
      by(auto simp:dependsR-def)
  qed } moreover
{ assume c>0
  hence x > (r - ⟨cs,xs⟩)/c (is - > ?l) using ⟨r < c*x + ⟨cs,xs⟩⟩
    by (simp add: field-simps)
  then have ?l < y using Atom ⟨c>0⟩
    by (auto simp:dependsR-def Ball-def nolb-def)
      (metis linorder-not-le antisym order-less-trans)
  hence ?thesis using ⟨c>0⟩ by (simp add: field-simps)
} ultimately show ?thesis by force
qed
next
case [simp]: (Eq r cs)
show ?thesis
proof (cases cs)
  case Nil thus ?thesis using Atom by (simp add:dependsR-def)
next
  case [simp]: (Cons c cs)
  hence r = c*x + ⟨cs,xs⟩ using Atom by simp
  { assume c=0 hence ?thesis using Atom by simp }
  moreover
  { assume c≠0
    hence ?thesis using ⟨r = c*x + ⟨cs,xs⟩⟩ Atom
      by (auto simp: dependsR-def split: if-splits) }
  ultimately show ?thesis by force
qed
qed
next
  case (And f1 f2) thus ?case by (fastforce)
next
  case (Or f1 f2) thus ?case by (fastforce)
qed simp+

definition EQ2 = EQ

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

lemma EQ2-And[simp]: EQ2 (And f g) xs = (EQ2 f xs ∪ 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

```

```

 $\implies \forall y \in \{r + \langle cs, xs \rangle <.. x\}. R.I f (y \# xs) \wedge y \notin EQ2 f xs$ 
 $\implies R.I (subst_+ f (r, cs)) xs$ 
proof(induct f)
  case FalseF thus ?case
    by simp (metis 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 + \langle ds, xs \rangle using Atom 1 by simp
      moreover have d*x < d*(r + \langle cs, xs \rangle) using \langle d<0 \rangle Atom 1
        by (simp add: mult-strict-left-mono-neg)
      ultimately have s < d * (r + \langle cs, xs \rangle) + \langle ds, xs \rangle by(simp add: algebra-simps)
      hence ?thesis using 1
        by (auto simp add: iprod-left-add-distrib algebra-simps)
    } moreover
    { let ?L = (s - \langle ds, xs \rangle) / d let ?U = r + \langle cs, xs \rangle
      assume d>0
      hence ?U < x and  $\forall y. ?U < y \wedge y < x \longrightarrow y \neq ?L$ 
        and  $\forall y. ?U < y \wedge y \leq x \longrightarrow ?L < y$  using Atom 1
          by(simp-all add: nolb-def depends_R-def Ball-def field-simps)
      hence ?L < ?U  $\vee$  ?L = ?U
        by (metis linorder-neqE-linordered-idom order-refl)
      hence ?thesis using Atom 1 \langle d>0 \rangle
        by (simp add: iprod-left-add-distrib field-simps)
    } ultimately show ?thesis by force
  }
next
  case 2 thus ?thesis using Atom
  by (fastforce simp: nolb-def EQ2-def depends_R-def field-simps split: if-split-asm)
  qed (insert Atom, auto)
next
  case Or thus ?case by(simp add: Ball-def)(metis order-refl innermost-intvl2)
qed simp-all

end

lemma I-subst-peps:
  ngfree f \implies R.I (subst_+ f (r, cs)) xs \implies
  ( $\exists leps > r + \langle cs, xs \rangle. \forall x. r + \langle cs, xs \rangle < x \wedge x \leq leps \longrightarrow R.I f (x \# xs)$ )
proof(induct f)
  case TrueF thus ?case by simp (metis 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 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 ∀y. ?a < y ∧ y ≤ x ⟶ s < d*y + ⟨ds,xs⟩
  using ⟨d<0⟩ by (simp add:field-simps)
  (metis add-le-cancel-right mult-le-cancel-left order-antisym linear 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 ∨ ?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 distrib-left mult.commute mult-strict-left-mono)
  next
  assume ?a < ?b
  { fix x assume r+⟨cs,xs⟩ < x ∧ x ≤ r+⟨cs,xs⟩ + 1
    hence d*(r + ⟨cs,xs⟩) < d*x
    using ⟨d>0⟩ by(metis mult-strict-left-mono)
    hence s < d*x + ⟨ds,xs⟩ using ⟨d>0⟩ ⟨?a < ?b⟩
    by (simp add:algebra-simps iprod-left-add-distrib)
  }
  thus ?thesis using 1 ⟨d>0⟩
  by(force simp: iprod-left-add-distrib)
} qed
} ultimately show ?thesis by (metis less-linear)
qed (insert Atom, auto split:if-split-asm intro: less-add-one)
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

```

**lemma dense-interval:**

**assumes** finite L l ∈ L l < x P(x::real)

**and dense:**  $\bigwedge y l. [\forall y \in \{l \dots x\}. y \notin L; l < x; l < y; y \leq x] \implies P y$

**shows**  $\exists l \in L. l < x \wedge (\forall y \in \{l \dots x\}. y \notin L) \wedge (\forall y. l < y \wedge y \leq x \implies P y)$

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

**definition**

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

**theorem**  $I\text{-eps}_1$ :

**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**  $\langle ?QE \rangle \text{min-inf}[of f xs] \langle ngfree f \rangle$   
**by**  $(\text{auto simp add:qe-eps}_1\text{-def amap-fm-list-disj})$   
**}** **moreover**  
**{** **assume**  $\forall x \in EQ f xs. \neg R.I f (x \# xs)$   
 $\neg R.I (\text{inf}_- f) xs$   
**with**  $\langle ?QE \rangle \langle ngfree f \rangle$  **obtain**  $r cs$  **where**  $R.I (\text{subst}_+ f (r, cs)) xs$   
**by**  $(\text{fastforce simp: qe-eps}_1\text{-def set-ebounds diff-divide-distrib eval-def I-subst}$   
 $\langle ngfree f \rangle)$   
**then obtain**  $leps$  **where**  $R.I f (leps \# xs)$   
**using**  $I\text{-subst-peps}[OF \langle ngfree f \rangle]$  **by fastforce**  
**hence**  $?EX ..$  **}**  
**ultimately show**  $?EX$  **by blast**

**next**

**let**  $?as = R.\text{atoms}_0 f$  **let**  $?ebs = \text{ebounds } ?as$   
**assume**  $?EX$   
**then obtain**  $x$  **where**  $x: R.I f (x \# xs) ..$   
**{** **assume**  $R.I (\text{inf}_- f) xs$   
**hence**  $?QE$  **using**  $\langle ngfree f \rangle$  **by**  $(\text{auto simp:qe-eps}_1\text{-def})$   
**}** **moreover**  
**{** **assume**  $\exists rcs \in \text{set } ?ebs. R.I (\text{subst } f rcs) xs$   
**hence**  $?QE$  **by**  $(\text{auto simp:qe-eps}_1\text{-def})$  **}** **moreover**  
**{** **assume**  $\neg R.I (\text{inf}_- f) xs$

**and**  $\forall rcs \in \text{set } ?\text{ebs}. \neg R.I (\text{subst } f \text{ rcs}) \text{ xs}$   
**hence**  $\text{noE}: \forall e \in EQ \text{ f xs}. \neg R.I \text{ f } (e\#xs)$  **using**  $\langle \text{ngfree } f \rangle$   
**by**  $(\text{force } \text{simp}:\text{set-ebounds } I\text{-subst } \text{diff-divide-distrib } \text{eval-def } \text{split}:\text{if-split-asm})$   
**hence**  $x \notin EQ \text{ f xs}$  **using**  $x$  **by**  $\text{fastforce}$   
**obtain**  $l$  **where**  $l \in LB \text{ f xs } l < x$   
**using**  $LBex[OF \langle \text{ngfree } f \rangle x \langle \neg R.I(\text{inf}_- f) \text{ xs} \rangle \langle x \notin EQ \text{ f xs} \rangle] ..$   
**have**  $\exists l \in LB \text{ f xs}. l < x \wedge \text{nolb } f \text{ xs } l \text{ x} \wedge$   
 $(\forall y. l < y \wedge y \leq x \longrightarrow R.I \text{ f } (y\#xs))$   
**using**  $\text{dense-interval}[\text{where } P = \lambda x. R.I \text{ f } (x\#xs), OF \text{ finite-LB } \langle l \in LB \text{ f xs} \rangle$   
 $\langle l < x \rangle x \text{ innermost-intvl}[OF \langle \text{ngfree } f \rangle - - \langle x \notin EQ \text{ f xs} \rangle]$   
**by**  $(\text{simp } \text{add}:\text{nolb-def})$   
**then obtain**  $r \text{ c cs}$   
**where**  $*$ :  $\text{Less } r (c\#cs) \in \text{set}(R.\text{atoms}_0 \text{ f}) \wedge c > 0 \wedge$   
 $(r - \langle cs, xs \rangle) / c < x \wedge \text{nolb } f \text{ xs } ((r - \langle cs, xs \rangle) / c) \text{ x}$   
 $\wedge (\forall y. (r - \langle cs, xs \rangle) / c < y \wedge y \leq x \longrightarrow R.I \text{ f } (y\#xs))$   
**by**  $\text{blast}$   
**then have**  $R.I (\text{subst}_+ \text{ f } (r/c, (-1/c) *_s \text{ cs})) \text{ xs}$  **using**  $\text{noE}$   
**by**  $(\text{auto } \text{intro}!: I\text{-subst-peps2}[OF \langle \text{ngfree } f \rangle]$   
 $\text{simp}:\text{EQ2-def } \text{diff-divide-distrib } \text{algebra-simps})$   
**with**  $*$  **have**  $?QE$   
**by**  $(\text{simp } \text{add}:\text{qe-eps}_1\text{-def } \text{bex-Un } \text{set-lbounds}) \text{metis}$   
**} ultimately show**  $?QE$  **by**  $\text{blast}$   
**qed**

**lemma**  $\text{qfree-asubst-peps}: \text{qfree } (a\text{subst}_+ \text{ rcs } a)$   
**by**  $(\text{cases } (rcs, a) \text{ rule}:\text{asubst-peps.cases}) \text{simp-all}$

**lemma**  $\text{qfree-subst-peps}: \text{ngfree } \varphi \implies \text{qfree } (\text{subst}_+ \varphi \text{ rcs})$   
**by**  $(\text{induct } \varphi) (\text{simp-all } \text{add}:\text{qfree-asubst-peps})$

**lemma**  $\text{qfree-qe-eps}_1: \text{ngfree } \varphi \implies \text{qfree}(\text{qe-eps}_1 \varphi)$   
**apply**  $(\text{simp } \text{add}:\text{qe-eps}_1\text{-def})$   
**apply**  $(\text{rule } \text{qfree-list-disj})$   
**apply**  $(\text{auto } \text{simp}:\text{qfree-min-inf } \text{qfree-subst-peps } \text{qfree-map-fm})$   
**done**

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

**lemma**  $\text{qfree-qe-eps}: \text{qfree}(\text{qe-eps } \varphi)$   
**by**  $(\text{simp } \text{add}:\text{qe-eps-def } R.\text{qfree-lift-nnf-qe } \text{qfree-qe-eps}_1)$

**lemma**  $I\text{-qe-eps}: R.I (\text{qe-eps } \varphi) \text{ xs} = R.I \varphi \text{ xs}$   
**by**  $(\text{simp } \text{add}:\text{qe-eps-def } R.I\text{-lift-nnf-qe } \text{qfree-qe-eps}_1 \text{I-eps1})$

**end**

## 5 Presburger arithmetic

**theory**  $\text{PresArith}$

```
imports QE HOL-Library.ListVector
begin
```

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

## 5.1 Syntax

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

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

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

```
fun decrZ :: atom ⇒ atom where
  decrZ (Le i ks) = Le i (tl ks) |
  decrZ (Dvd d i ks) = Dvd d i (tl ks) |
  decrZ (NDvd d i ks) = NDvd d i (tl ks)
```

```
fun IZ :: atom ⇒ int list ⇒ bool where
  IZ (Le i ks) xs = (i ≤ ⟨ks,xs⟩) |
  IZ (Dvd d i ks) xs = (d dvd i+⟨ks,xs⟩) |
  IZ (NDvd d i ks) xs = (¬ d dvd i+⟨ks,xs⟩)
```

```
definition atoms0 = ATOM.atoms0 (λa. hd-coeff a ≠ 0)
```

**interpretation Z:**

```
ATOM negZ (λa. divisor a ≠ 0) IZ (λa. hd-coeff a ≠ 0) decrZ
rewrites ATOM.atoms0 (λa. hd-coeff a ≠ 0) = atoms0
```

**proof goal-cases**

```
case 1
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 2
  thus ?case by(simp add:atoms0-def)
qed

setup <Sign.revert-abbrev @{const-abbrev Z.I}>
setup <Sign.revert-abbrev @{const-abbrev Z.lift-dnf-qe}>

```

### abbreviation

```

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

```

```

fun asubst :: int ⇒ int list ⇒ atom ⇒ atom where
  asubst i' ks' (Le i (k#ks)) = Le (i - k*i') (k *_s ks' + ks) |
  asubst i' ks' (Dvd d i (k#ks)) = Dvd d (i + k*i') (k *_s ks' + ks) |
  asubst i' ks' (NDvd d i (k#ks)) = NDvd d (i + k*i') (k *_s ks' + ks) |
  asubst i' ks' a = a

```

```

abbreviation subst :: int ⇒ int list ⇒ atom fm ⇒ atom fm
where subst i ks ≡ map_fm (asubst i ks)

```

```

lemma IZ-asubst: I_Z (asubst i ks a) xs = I_Z a ((i + ⟨ks,xs⟩) # xs)
apply (cases a)
apply (rename-tac list)
apply (case-tac list)
apply (simp-all add:algebra-simps iprod-left-add-distrib)
apply (rename-tac list)
apply (case-tac list)
apply (simp-all add:algebra-simps iprod-left-add-distrib)
apply (rename-tac list)
apply (case-tac list)
apply (simp-all add:algebra-simps iprod-left-add-distrib)
done

```

### lemma I-subst:

```

  qfree φ ⇒ Z.I φ ((i + ⟨ks,xs⟩) # xs) = Z.I (subst i ks φ) xs
by (induct φ) (simp-all add:IZ-asubst)

```

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

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

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

**lemma** *set-lbounds*:

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

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

**lemma** *set-ubounds*:

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

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

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

**by**(*simp add:lbounds-def*)

## 5.2 LCM and lemmas

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

*zlcms* [] = 1 |

*zlcms* (i#is) = *lcm* i (*zlcms* is)

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

**by**(*induct is*) *auto*

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

**by**(*induct is*)(*auto simp:lcm-pos-int*)

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

**by** (*metis mod-by-0 dvd-eq-mod-eq-0 dvd-zlcms zlcms-pos less-le*)

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

**by** (*metis dvd-zlcms zdvd-imp-le zlcms-pos*)

## 5.3 Setting coefficients to 1 or -1

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

*hd-coeff1* *m* (*Le* *i* (*k*#*ks*)) =

(*if* *k*=0 *then* *Le* *i* (*k*#*ks*)

*else* *let* *m'* = *m* *div* (*abs* *k*) *in* *Le* (*m'*\**i*) (*sgn* *k* # (*m'* \*<sub>s</sub> *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'* \*<sub>s</sub> *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'* \*<sub>s</sub> *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 [  $k \neq 0$ ;  $k \text{ dvd } m$  ]  $\implies$ 
   $I_Z$  (hd-coeff1 m (Dvd d i (k#ks))) (x#e) = (let m' = m div (abs k) in
   $I_Z$  (Dvd (m'*d) (m'*i) (sgn k # (m' *_s ks))) (x#e))
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-conv-add-uminus add.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-conv-add-uminus add.left-commute dvd-minus-iff minus-add-distrib)
done

```

```

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

```

```

qed
next
case [simp]: (Dvd d i ks)
show ?case
proof(cases ks)
  case Nil thus ?thesis by simp
next
case [simp]: (Cons k ks')
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 zdiv-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 dvd-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 [simp]: (NDvd d i ks)
show ?case
proof(cases ks)
  case Nil thus ?thesis by simp
next
case [simp]: (Cons k ks')
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 zdiv-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 dvd-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) ‹↔›
  ¬(?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
qed

```

```

lemma I-hd-coeff1-mult: assumes m>0
shows qfree φ ‹⇒› ∀ a ∈ set(Z.atoms_0 φ). hd-coeff a dvd m ‹⇒›
  Z.I (map_fm (hd-coeff1 m) φ) (m*x#xs) = Z.I φ (x#xs)
proof(induct φ)
  case (Atom a)
  thus ?case using I-hd-coeff1-mult-a[OF ‹m>0›] by auto
qed simp-all
end

```

```

theory QEpres
imports PresArith
begin

```

## 5.4 DNF-based quantifier elimination

**definition**

```

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

```

**lemma I-hd-coeffs1:**

```

assumes 0: ∀ a ∈ set as. hd-coeff a ≠ 0 shows
  (∃ x. ∀ a ∈ set(hd-coeffs1 as). I_Z a (x#xs)) =
  (∃ x. ∀ a ∈ 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 = (∃ x. ∀ a ∈ 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** ... = ( $\exists x. ?m \text{ dvd } x + 0 \wedge (\forall a \in \text{set as. } I_Z (\text{hd-coeff1 } ?m \ a) (x \# xs))$ )  
**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-pres<sub>1</sub> as* =  
(*let ds = filter is-dvd as*; (*d::int*) = *zlcms*(*map divisor ds*); *ls = lbounds as*  
*in if* *ls = []*  
*then Disj [0..d - 1] ( $\lambda n. \text{list-conj}(\text{map} (\text{Atom} \circ \text{asubst } n \ []) \text{ ds})$ )*  
*else*  
*Disj ls ( $\lambda(li, lks). \text{Disj [0..d - 1] ( $\lambda n. \text{list-conj}(\text{map} (\text{Atom} \circ \text{asubst} (li + n) (-lks)) \text{ as}))$ )$ )*)

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

**lemma** *I-cyclic*:

**assumes** *is-dvd a* **and** *hd-coeff a = 1* **and** *i 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**  $\langle \text{hd-coeff } a = 1 \rangle$  **obtain** *ks'* **where** [*simp*]:  $ks = 1 \# ks'$

**by**(*simp split:list.splits*)

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

**proof** -

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

**by** (*simp add: mod-add-eq*)

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

**by** (*simp add: mod-add-eq*)

**also have**  $i \text{ mod } d = j \text{ mod } d$

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

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

**by**(*rule mod-add-eq*)

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

**by**(*rule mod-add-eq*)

**finally show** *?thesis* .

**qed**

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

**next**

**case** (*NDvd d l ks*)

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

**by**(*simp split:list.splits*)

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

**proof** -

```

have ?l = (l mod d + (i + ⟨ks',e⟩) mod d) mod d
  by (simp add: mod-add-eq)
also have (i + ⟨ks',e⟩) mod d = (i mod d + ⟨ks',e⟩ mod d) mod d
  by (simp add: mod-add-eq)
also have i mod d = j mod d
  using ⟨i mod divisor a = j mod divisor a⟩ NDvd by simp
also have (j mod d + ⟨ks',e⟩ mod d) mod d = (j + ⟨ks',e⟩) mod d
  by(rule mod-add-eq)
also have (l mod d + (j + ⟨ks',e⟩) mod d) mod d = ?r
  by(rule mod-add-eq)
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: ∀ a ∈ set as. divisor a ≠ 0
and hd: ∀ a ∈ set as. hd-coeff-is1 a
shows Z.I (qe-pres1 as) xs = (∃ x. ∀ a ∈ set as. IZ 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 rule: atom.exhaust)(auto split:list.splits)
    }
  moreover
  { fix x assume x ∈ ?S
    hence x ∈ set as by auto }
  ultimately show ?thesis by blast
qed
have 1: ∀ a ∈ ?Ds. hd-coeff a = 1 using hd by(fastforce split:atom.splits)
show ?thesis (is ?QE = (∃ x. ?P x))
proof
  assume ?QE
  { assume ?lbs = []
    with ⟨?QE⟩ obtain n where n < ?lcm and
      A: ∀ a ∈ ?Ds. IZ 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 ∃ x. ?P x
  }

```

**proof cases**  
**assume**  $?Us = \{\}$   
**with**  $\langle ?Ls = \{\} \rangle$  **have**  $set\ as = ?Ds$  **using**  $as$  **by**  $(simp\ (no-asm-use))blast$   
**hence**  $?P\ n$  **using**  $A$  **by**  $auto$   
**thus**  $?thesis\ ..$   
**next**  
**assume**  $?Us \neq \{\}$   
**let**  $?M = \{\langle tl\ ks,\ xs \rangle - i \mid ks\ i.\ Le\ i\ ks \in ?Us\}$  **let**  $?m = Min\ ?M$   
**have**  $finite\ ?M$   
**proof**  $-$   
**have**  $finite\ (\ (\lambda Le\ i\ ks \Rightarrow \langle tl\ ks,\ xs \rangle - i)\ \{a \in set\ as.\ \exists i\ k\ ks.\ k < 0 \wedge a = Le\ i\ (k \# ks)\})$   
**(is**  $finite\ ?B$   
**by**  $simp$   
**also** **have**  $?B = ?M$  **using**  $hd$   
**by**  $(fastforce\ simp:image-def\ neq-Nil-conv\ split:atom.splits\ list.splits)$   
**finally** **show**  $?thesis$  **by**  $auto$   
**qed**  
**have**  $?M \neq \{\}$   
**proof**  $-$   
**from**  $\langle ?Us \neq \{\} \rangle$  **obtain**  $i\ k\ ks$  **where**  $Le\ i\ (k \# ks) \in ?Us \wedge k < 0$   
**by**  $(fastforce\ split:atom.splits\ list.splits)$   
**thus**  $?thesis$  **by**  $auto$   
**qed**  
**let**  $?k = (n - ?m)\ div\ ?lcm + 1$  **let**  $?x = n - ?k * ?lcm$   
**have**  $\forall a \in ?Ds.\ I_Z\ a\ (?x \# xs)$   
**proof**  $(intro\ allI\ ballI)$   
**fix**  $a$  **assume**  $a \in ?Ds$   
**let**  $?d = divisor\ a$   
**have**  $2: ?d\ dvd\ ?lcm$  **using**  $\langle a \in ?Ds \rangle$  **by**  $(simp\ add:dvd-zlcm)$   
**have**  $?x\ mod\ ?d = n\ mod\ ?d$  **(is**  $?l = ?r$   
**proof**  $-$   
**have**  $?l = (?r - ((?k * ?lcm)\ mod\ ?d))\ mod\ ?d$   
**by**  $(simp\ add:\ mod-diff-eq)$   
**also** **have**  $(?k * ?lcm)\ mod\ ?d = 0$   
**by**  $(simp\ add:\ dvd-eq-mod-eq-0[symmetric]\ dvd-mult[OF\ 2])$   
**finally** **show**  $?thesis$  **by**  $simp$   
**qed**  
**thus**  $I_Z\ a\ (?x \# xs)$  **using**  $A\ I-cyclic[of\ a\ n\ ?x]\ \langle a \in ?Ds \rangle\ 1$  **by**  $auto$   
**qed**  
**moreover**  
**have**  $\forall a \in ?Us.\ I_Z\ a\ (?x \# xs)$   
**proof**  
**fix**  $a$  **assume**  $a \in ?Us$   
**then** **obtain**  $l\ ks$  **where**  $[simp]:\ a = Le\ l\ (-1 \# ks)$  **using**  $hd$   
**by**  $(fastforce\ split:atom.splits\ list.splits)$   
**have**  $?m \leq \langle ks, xs \rangle - l$   
**using**  $Min-le-iff[OF\ \langle finite\ ?M \rangle\ \langle ?M \neq \{\} \rangle]\ \langle a \in ?Us \rangle$  **by**  $fastforce$   
**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: minus-mod-eq-mult-div [symmetric] algebra-simps)
  qed
  moreover
  have set as = ?Ds  $\cup$  ?Us using as <?Ls = {}>
    by (simp (no-asm-use)) blast
  ultimately have ?P(?x) by auto
  thus ?thesis ..
  qed }
  moreover
  { assume ?lbs  $\neq$  []
    with <?QE> obtain il ksl m
      where  $\forall a \in \text{set } as. I_Z (asubst (il + m) ksl a) xs$ 
      by(auto simp:qe-pres1-def)
    hence ?P(il + m + <ksl,xs>) by(simp add:IZ-asubst)
    hence  $\exists x. ?P x ..$  }
  ultimately show  $\exists x. ?P x$  by blast
next
assume  $\exists x. ?P x$  then obtain x where  $x: ?P x ..$ 
show ?QE
proof cases
  assume ?lbs = []
  moreover
  have  $\exists x. 0 \leq x \wedge x < ?lcm \wedge (\forall a \in ?Ds. I_Z a (x \# xs))$ 
    (is  $\exists x. ?P x$ )
  proof
    { fix a assume  $a \in ?Ds$ 
      hence  $I_Z a ((x \bmod ?lcm) \# xs) = I_Z a (x \# xs)$  using 1
      by (fastforce 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  $\neq$  []
let ?L = {i - <ks,xs> | ks i. (i,ks)  $\in$  set(lbounds as)}
let ?lm = Max ?L
let ?n = (x - ?lm) mod ?lcm
have finite ?L
proof -
  have finite(( $\lambda(i,ks). i - \langle ks,xs \rangle$ ) ‘ set(lbounds as) ) (is finite ?B)
  by simp
  also have ?B = ?L by auto
  finally show ?thesis by auto
qed
moreover have ?L  $\neq$  {} using <?lbs  $\neq$  []>
  by(fastforce simp:neq-Nil-conv)
ultimately have ?lm  $\in$  ?L by(rule Max-in)

```

**then obtain**  $li\ lks$  **where**  $(li, lks) \in set\ ?lbs$  **and**  $lm: ?lm = li - \langle lks, xs \rangle$   
**by** *blast*  
**moreover**  
**have**  $n: 0 \leq ?n \wedge ?n < ?lcm$  **using** *norm* **by**  $(simp\ add: zlcms\ pos)$   
**moreover**  
**{** **fix**  $a$  **assume**  $a \in set\ as$   
**with**  $x$  **have**  $I_Z\ a\ (x \# xs)$  **by** *blast*  
**have**  $I_Z\ a\ ((li + ?n - \langle lks, xs \rangle) \# xs)$   
**proof**  $-$   
**{** **assume**  $a \in ?Ls$   
**then obtain**  $i\ ks$  **where**  $[simp]: a = Le\ i\ (1 \# ks)$  **using** *hd*  
**by**  $(fastforce\ split: atom.splits\ list.splits)$   
**from**  $\langle a \in ?Ls \rangle$  **have**  $i - \langle ks, xs \rangle \in ?L$  **by**  $(fastforce\ simp: set-lbounds)$   
**hence**  $i - \langle ks, xs \rangle \leq li - \langle lks, xs \rangle$   
**using**  $lm[symmetric]\ \langle finite\ ?L \rangle\ \langle ?L \neq \{ \} \rangle$  **by** *auto*  
**hence** *?thesis* **using**  $n$  **by** *simp* **}**  
**moreover**  
**{** **assume**  $a \in ?Us$   
**then obtain**  $i\ ks$  **where**  $[simp]: a = Le\ i\ (-1 \# ks)$  **using** *hd*  
**by**  $(fastforce\ split: atom.splits\ list.splits)$   
**have**  $Le\ li\ (1 \# lks) \in set\ as$  **using**  $\langle (li, lks) \in set\ ?lbs \rangle$  *hd*  
**by**  $(auto\ simp: set-lbounds)$   
**hence**  $li - \langle lks, xs \rangle \leq x$  **using**  $x$  **by** *auto*  
**hence**  $(x - ?lm) \bmod ?lcm \leq x - ?lm$   
**using**  $lm$  **by**  $(simp\ add: zmod-le-nonneg-dividend)$   
**hence** *?thesis* **using**  $\langle I_Z\ a\ (x \# xs) \rangle$   $lm$  **by** *auto* **}**  
**moreover**  
**{** **assume**  $a \in ?Ds$   
**have** *?thesis*  
**proof**  $(rule\ I-cyclic[THEN\ iffD2, OF\ - - - \langle I_Z\ a\ (x \# xs) \rangle])$   
**show** *is-dvd*  $a$  **using**  $\langle a \in ?Ds \rangle$  **by** *simp*  
**show** *hd-coeff*  $a = 1$  **using**  $\langle a \in ?Ds \rangle$  *hd*  
**by**  $(fastforce\ split: atom.splits\ list.splits)$   
**have**  $li + (x - ?lm) \bmod ?lcm - \langle lks, xs \rangle = ?lm + (x - ?lm) \bmod ?lcm$   
**using**  $lm$  **by** *arith*  
**hence**  $(li + (x - ?lm) \bmod ?lcm - \langle lks, xs \rangle) \bmod\ divisor\ a =$   
 $(?lm + (x - ?lm) \bmod ?lcm) \bmod\ divisor\ a$  **by**  $(simp\ only:)$   
**also have**  $\dots =$   
 $(?lm \bmod\ divisor\ a + (x - ?lm) \bmod\ divisor\ a) \bmod\ divisor\ a$   
**by**  $(simp\ add: mod-add-eq)$   
**also have**  
 $\dots = ( ?lm \bmod\ divisor\ a + (x - ?lm) \bmod\ divisor\ a) \bmod\ divisor\ a$   
**using**  $\langle is-dvd\ a \rangle\ \langle a \in set\ as \rangle$   
**by**  $(simp\ add: mod-mod-cancel\ dvd-zlcms)$   
**also have**  $\dots = (?lm + (x - ?lm)) \bmod\ divisor\ a$   
**by**  $(rule\ mod-add-eq)$   
**also have**  $\dots = x \bmod\ divisor\ a$  **by** *simp*  
**finally**  
**show**  $(li + ?n - \langle lks, xs \rangle) \bmod\ divisor\ a = x \bmod\ divisor\ a$

```

      using norm by(auto simp: 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* = *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*: *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 [*simp*]: (*Dvd d i ks*)

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 ‹*b* ∈ set as› 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 [*simp*]: (*NDvd d i ks*)

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 ‹*b* ∈ set as› 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-pres<sub>1</sub>-o*:  
   $\llbracket \forall a \in \text{set } as. \text{divisor } a \neq 0; \forall a \in \text{set } as. \text{hd-coeff } a \neq 0 \rrbracket \implies$   
   $Z.I ((\text{qe-pres}_1 \circ \text{hd-coeffs1}) \ as) \ e = (\exists x. \forall a \in \text{set } as. I_Z \ a \ (x\#e))$   
**apply**(*simp*)  
**apply**(*subst I-qe-pres<sub>1</sub>*)  
  **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.lift-dnf-qe* (*qe-pres<sub>1</sub>  $\circ$  hd-coeffs1*)

**lemma** *qfree-qe-pres-o*: *qfree* ((*qe-pres<sub>1</sub>  $\circ$  hd-coeffs1*) *as*)  
**by**(*auto simp:qe-pres<sub>1</sub>-def intro!:qfree-list-disj*)

**lemma** *normal-qe-pres<sub>1</sub>-o*:  
   $\forall a \in \text{set } as. \text{hd-coeff } a \neq 0 \wedge \text{divisor } a \neq 0 \implies$   
   $Z.normal ((\text{qe-pres}_1 \circ \text{hd-coeffs1}) \ as)$   
  **supply** *image-cong-simp* [*cong del*]  
**apply**(*auto simp:qe-pres<sub>1</sub>-def Z.normal-def*  
  *dest!:atoms-list-disjE atoms-list-conjE*)

**apply**(*simp add: hd-coeffs1-def*)  
**apply**(*erule disjE*) **apply** *fastforce*

```

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

```

```

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

**theorem** *I-pres-qe*:  $Z.normal\ \varphi \implies Z.I\ (qe-pres\ \varphi)\ xs = Z.I\ \varphi\ xs$   
**by**(*simp add:qe-pres-def Z.I-lift-dnf-qe-anormal I-qe-pres<sub>1</sub>-o qfree-qe-pres-o normal-qe-pres<sub>1</sub>-o del:o-apply*)

**theorem** *qfree-pres-qe*:  $qfree\ (qe-pres\ f)$   
**by**(*simp add:qe-pres-def Z.qfree-lift-dnf-qe qfree-qe-pres-o del:o-apply*)

**end**

**theory** *Cooper*  
**imports** *PresArith*  
**begin**

## 5.5 Cooper

This section formalizes Cooper's algorithm [1].

**lemma** *set-atoms0-iff*:  
 $qfree\ \varphi \implies a \in set(Z.atoms_0\ \varphi) \longleftrightarrow a \in atoms\ \varphi \wedge hd-coeff\ a \neq 0$   
**by**(*induct*  $\varphi$ ) (*auto split:if-split-asm*)

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

**lemma** *I-hd-coeffs1*:  
**assumes**  $qfree\ \varphi$   
**shows**  $(\exists x. Z.I\ (hd-coeffs1\ \varphi)\ (x\#xs)) = (\exists x. Z.I\ \varphi\ (x\#xs))$  (**is**  $?L = ?R$ )  
**proof** –

**let**  $?l = zlcms(map\ hd-coeff\ (Z.atoms_0\ \varphi))$   
**have**  $?l > 0$  **by**(*simp add: zlcms-pos set-atoms0-iff[OF <qfree  $\varphi$ >]*)  
**have**  $?L = (\exists x. ?l\ dvd\ x + 0 \wedge Z.I\ (map_{fm}\ (hd-coeff1\ ?l)\ \varphi)\ (x\#xs))$   
**by**(*simp add:hd-coeffs1-def*)  
**also have**  $\dots = (\exists x. Z.I\ (map_{fm}\ (hd-coeff1\ ?l)\ \varphi)\ (?l * x\#xs))$   
**by**(*rule unity-coeff-ex[THEN meta-eq-to-obj-eq,symmetric]*)  
**also have**  $\dots = ?R$   
**by**(*simp add: I-hd-coeff1-mult[OF <?l> <qfree  $\varphi$ >] dvd-zlcms*)  
**finally show** *?thesis* .  
**qed**

**fun** *min-inf* ::  $atom\ fm \Rightarrow atom\ fm\ (inf\_)$  **where**  
 $inf\_ (And\ \varphi_1\ \varphi_2) = and\ (inf\_ \varphi_1)\ (inf\_ \varphi_2) \mid$   
 $inf\_ (Or\ \varphi_1\ \varphi_2) = or\ (inf\_ \varphi_1)\ (inf\_ \varphi_2) \mid$

$\text{inf}_- (\text{Atom}(\text{Le } i (k\#ks))) =$   
 (if  $k < 0$  then  $\text{TrueF}$  else if  $k > 0$  then  $\text{FalseF}$  else  $\text{Atom}(\text{Le } i (0\#ks))) \mid$   
 $\text{inf}_- \varphi = \varphi$

**definition**

$\text{qe-cooper}_1 \varphi =$   
 (let  $as = Z.\text{atoms}_0 \varphi$ ;  $d = \text{zlcms}(\text{map } \text{divisor } as)$ ;  $ls = \text{lbounds } as$   
 in or ( $\text{Disj } [0..d - 1] (\lambda n. \text{subst } n [] (\text{inf}_- \varphi))$ )  
 ( $\text{Disj } ls (\lambda(i,ks). \text{Disj } [0..d - 1] (\lambda n. \text{subst } (i + n) (-ks) \varphi))$ ))

**lemma min-inf:**

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

**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**  $?P (\text{Atom}(\text{Le } i (k\#ks))) (-i + \langle ks, xs \rangle - 1)$  **using**  $\exists$  **by** *auto*  
**hence**  $?case ..$  }  
**moreover**  
 { **assume**  $k=1$   
**hence**  $?P (\text{Atom}(\text{Le } i (k\#ks))) (i - \langle ks, xs \rangle - 1)$  **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** *fastforce+*  
**hence**  $?P (\text{And } f1 f2) (\text{min } x1 x2)$  **by** *simp*  
**thus**  $?case ..$

**next**

**case** ( $2 f1 f2$ )  
**then obtain**  $x1 x2$  **where**  $?P f1 x1 ?P f2 x2$  **by** *fastforce+*  
**hence**  $?P (\text{Or } f1 f2) (\text{min } x1 x2)$  **by** *simp*  
**thus**  $?case ..$

**qed** *simp-all*

**lemma min-inf-repeats:**

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

**proof**(*induct  $\varphi$  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≠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 mult.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 mult.commute)
      also have ... ⟷ da dvd i + (j * x + ⟨js,xs⟩)
        by(simp add: algebra-simps)
      finally show ?thesis .
    qed
  qed

```



```

    then show ?thesis using Cons by (simp add:ring-distrib)
  qed
qed

```

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

**lemma**  $\beta$ :

```

[[ nqfree  $\varphi$ ;  $\forall a \in set(Z.atoms_0\ \varphi). hd-coeff-is1\ a$ ;
   $\forall a \in set(Z.atoms_0\ \varphi). divisor\ a\ dvd\ d$ ;  $d > 0$ ;
   $\neg(\exists j \in \{0 .. d - 1\}. \exists (i, ks) \in set(lbounds(Z.atoms_0\ \varphi)).$ 
     $x = i - \langle ks, xs \rangle + j$ );  $Z.I\ \varphi\ (x \# xs)$  ]]
 $\implies Z.I\ \varphi\ ((x-d) \# xs)$ 

```

**proof** (*induct  $\varphi$* )

**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:if-split-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**  $m\ dvd\ d$  **using** *Cons Dvd Atom* **by** *auto*

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

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

**proof** -

**assume** ?*L*

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

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

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

**qed**

```

      thus ?thesis using Atom Dvd Cons by(auto split:if-split-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⟩) ⇒ m dvd i + (x + ⟨ks,xs⟩)
    (is ?L ⇒ -)
  proof -
    assume ?L
    hence m dvd i + (x + ⟨ks,xs⟩) - d by(simp add:algebra-simps)
    thus ?thesis by (metis ⟨m dvd d⟩ zdvd-zdiffD)
  qed
  thus ?thesis using Atom NDvd Cons by(auto split:if-split-asm)
qed
qed
qed
qed force+

```

**lemma** *periodic-finite-ex*:

```

  assumes dpos: (0::int) < d and modd: ∀ x k. P x = P(x - k*d)
  shows (∃ x. P x) = (∃ j∈{0..d - 1}. P j)
  (is ?LHS = ?RHS)

```

**proof**

```

  assume ?LHS
  then obtain x where P: P x ..
  have x mod d = x - (x div d)*d
    by(simp add:mult-div-mod-eq [symmetric] ac-simps 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
  moreover have x mod d ∈ {0..d - 1} using dpos by auto
  ultimately show ?RHS ..

```

**qed** *auto*

```

lemma cpmi-eq: (0::int) < D ⇒ (∃ z. ∀ x. x < z ⇒ (P x = P1 x))
⇒ ∀ x. ¬(∃ j∈{0..D - 1}. ∃ b∈B. P(b+j)) ⇒ P(x) ⇒ P(x - D)
⇒ ∀ x. ∀ k. P1 x = P1(x - k*D)
⇒ (∃ x. P(x)) = ((∃ j∈{0..D - 1}. P1(j)) ∨ (∃ j∈{0..D - 1}. ∃ b∈B. P(b+j)))

```

```

apply(rule iffI)
prefer 2
apply(drule minusinfinity)
apply assumption+
apply(fastforce)
apply clarsimp
apply(subgoal-tac  $\bigwedge k. 0 \leq k \implies \forall x. P x \longrightarrow P (x - k * D)$ )
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 fastforce
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 (inf\_  $\varphi$ )  
**by** (induct  $\varphi$  rule:min-inf.induct) simp-all

**lemma** I-qe-cooper<sub>1</sub>:

```

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**  $Z.I (qe-cooper_1 \varphi) xs = (\exists x. Z.I \varphi (x\#xs))$   
**proof** –  
**let**  $?as = Z.atoms_0 \varphi$   
**let**  $?d = zlcms(\text{map } \text{divisor } ?as)$   
**have**  $?d > 0$  **using**  $\text{norm } \text{atoms-subset}[of \varphi] \langle \text{ngfree } \varphi \rangle$   
**by**  $(\text{fastforce } \text{intro:zlcms-pos})$   
**have**  $\text{alld: } \forall a \in \text{set}(Z.atoms_0 \varphi). \text{divisor } a \text{ dvd } ?d$  **by**  $(\text{simp } \text{add:dvd-zlcms})$   
**from**  $\text{cp-thm}[OF \langle \text{ngfree } \varphi \rangle \text{hd } \text{alld} \langle ?d > 0 \rangle]$   
**show**  $?thesis$  **using**  $\langle \text{ngfree } \varphi \rangle$   
**by**  $(\text{simp } \text{add:qe-cooper}_1\text{-def } I\text{-subst}[\text{symmetric}] \text{split-def } \text{algebra-simps})$  **blast**  
**qed**

**lemma**  $\text{divisor-hd-coeff1-neq0}$ :  
 $\text{ngfree } \varphi \implies a \in \text{atoms } \varphi \implies \text{divisor } a \neq 0 \implies$   
 $\text{divisor } (\text{hd-coeff1 } (zlcms (\text{map } \text{hd-coeff } (Z.atoms_0 \varphi))) a) \neq 0$   
**apply**  $(\text{case-tac } a)$

**apply**  $\text{simp}$   
**apply**  $(\text{rename-tac } \text{list})$  **apply**  $(\text{case-tac } \text{list})$  **apply**  $\text{simp}$  **apply**  $(\text{simp } \text{split:if-split-asm})$

**apply**  $\text{simp}$   
**apply**  $(\text{rename-tac } \text{list})$  **apply**  $(\text{case-tac } \text{list})$  **apply**  $\text{simp}$   
**apply**  $(\text{clarsimp } \text{split:if-split-asm})$   
**apply**  $(\text{hypsubst-thin})$   
**apply**  $(\text{subgoal-tac } a \in \text{set}(\text{map } \text{hd-coeff } (Z.atoms_0 \varphi)))$   
**apply**  $(\text{subgoal-tac } \forall i \in \text{set}(\text{map } \text{hd-coeff } (Z.atoms_0 \varphi)). i \neq 0)$   
**apply**  $(\text{metis } \text{dvd-zlcms } \text{mult-eq-0-iff } \text{dvd-mult-div-cancel } \text{zlcms0-iff})$   
**apply**  $(\text{simp } \text{add:set-atoms0-iff})$   
**apply**  $(\text{fastforce } \text{simp:image-def } \text{set-atoms0-iff } \text{Bex-def})$

**apply**  $\text{simp}$   
**apply**  $(\text{rename-tac } \text{list})$  **apply**  $(\text{case-tac } \text{list})$  **apply**  $\text{simp}$   
**apply**  $(\text{clarsimp } \text{split:if-split-asm})$   
**apply**  $(\text{hypsubst-thin})$   
**apply**  $(\text{subgoal-tac } a \in \text{set}(\text{map } \text{hd-coeff } (Z.atoms_0 \varphi)))$   
**apply**  $(\text{subgoal-tac } \forall i \in \text{set}(\text{map } \text{hd-coeff } (Z.atoms_0 \varphi)). i \neq 0)$   
**apply**  $(\text{metis } \text{dvd-zlcms } \text{mult-eq-0-iff } \text{dvd-mult-div-cancel } \text{zlcms0-iff})$   
**apply**  $(\text{simp } \text{add:set-atoms0-iff})$   
**apply**  $(\text{fastforce } \text{simp:image-def } \text{set-atoms0-iff } \text{Bex-def})$   
**done**

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

**lemma**  $I\text{-cooper1-hd-coeffs1}$ :  $Z.\text{normal } \varphi \implies \text{ngfree } \varphi$   
 $\implies Z.I (qe-cooper_1 (\text{hd-coeffs1 } \varphi)) xs = (\exists x. Z.I \varphi (x \# xs))$   
**apply**  $(\text{simp } \text{add:Z.normal-def})$   
**apply**  $(\text{subst } I\text{-qe-cooper}_1)$

**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*  
*hd-coeff-is1-hd-coeff1*)  
**apply**(*simp add:hd-coeffs1-def nqfree-map-fm*)  
**apply**(*simp add: I-hd-coeffs1*)  
**done**

**definition** *qe-cooper* = *Z.lift-nnf-qe (qe-cooper<sub>1</sub> ∘ hd-coeffs1)*

**lemma** *qfree-cooper1-hd-coeffs1*: *qfree φ ⇒ qfree (qe-cooper<sub>1</sub> (hd-coeffs1 φ))*  
**by**(*auto simp:qe-cooper<sub>1</sub>-def hd-coeffs1-def qfree-map-fm*  
*intro!: qfree-or qfree-and qfree-list-disj qfree-min-inf*)

**lemma** *normal-min-inf*: *Z.normal φ ⇒ Z.normal(inf<sub>-</sub> φ)*  
**by**(*induct φ rule:min-inf.induct*) *simp-all*

**lemma** *normal-cooper1*: *Z.normal φ ⇒ Z.normal(qe-cooper<sub>1</sub> φ)*  
**by**(*simp add:qe-cooper<sub>1</sub>-def Logic.or-def Z.normal-map-fm normal-min-inf split-def*)

**lemma** *normal-hd-coeffs1*: *qfree φ ⇒ Z.normal φ ⇒ Z.normal(hd-coeffs1 φ)*  
**by**(*auto simp: hd-coeffs1-def image-def set-atoms0-iff*  
*divisor-hd-coeff1-neq0 Z.normal-def*)

**theorem** *I-cooper*: *Z.normal φ ⇒ Z.I (qe-cooper φ) xs = Z.I φ 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 φ)*  
**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](http://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.