Quantifier Elimination for Linear Arithmetic

Tobias Nipkow

February 6, 2026

Abstract

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

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

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

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

Contents

1 Logic **2**
1.1 Atoms . . . 5

2 Quantifier elimination **8**
2.1 No Equality . . . 8
2.1.1 DNF-based . . . 8
2.1.2 NNF-based . . . 11
2.2 With equality . . . 13

3 DLO **14**
3.1 Basics . . . 14
3.2 DNF-based quantifier elimination . . . 20
3.3 Examples . . . 23
3.4 Interior Point Method . . . 24
3.5 Quantifier elimination with infinitesimals . . . 29

4 Linear real arithmetic **33**
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

5 Presburger arithmetic **55**
5.1 Syntax . . . 55
5.2 LCM and lemmas . . . 58
5.3 Setting coeffiencients 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* φ_1 φ_2 ≡ *Or* (*Neg* φ_1) φ_2
abbreviation *AllQ* **where** *AllQ* φ ≡ *Neg*(*ExQ*(*Neg* φ))

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

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

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

if φ_1=*TrueF* $\vee$ φ_2=*TrueF then TrueF else Or* φ_1 φ_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* ($\langle amap_{fm} \rangle$) **where**
$amap_{fm}$ *h TrueF* = *TrueF* |
$amap_{fm}$ *h FalseF* = *FalseF* |
$amap_{fm}$ *h* (*Atom a*) = *h a* |
$amap_{fm}$ *h* (*And* φ_1 φ_2) = *and* ($amap_{fm}$ *h* φ_1) ($amap_{fm}$ *h* φ_2) |
$amap_{fm}$ *h* (*Or* φ_1 φ_2) = *or* ($amap_{fm}$ *h* φ_1) ($amap_{fm}$ *h* φ_2) |
$amap_{fm}$ *h* (*Neg* φ) = *neg* ($amap_{fm}$ *h* φ)

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

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

lemma *qfree-and*[*simp*]: $[\![$ *qfree* φ_1; *qfree* φ_2 $]\!]$ $\Longrightarrow$ *qfree*(*and* φ_1 φ_2)
by(*simp add*:*and-def*)

lemma *qfree-or*[*simp*]: $[\![$ *qfree* φ_1; *qfree* φ_2 $]\!]$ $\Longrightarrow$ *qfree*(*or* φ_1 φ_2)
by(*simp add*:*or-def*)

lemma *qfree-neg*[*simp*]: *qfree*(*neg* φ) = *qfree* φ
by(*simp add*:*neg-def*)

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

lemma *qfree-list-conj*[*simp*]:
assumes $\forall \varphi \in$ *set fs*. *qfree* φ **shows** *qfree*(*list-conj fs*)
proof −
{ fix *fs* φ
have $[\![\forall \varphi \in$ *set fs*. *qfree* φ; *qfree* φ $]\!]$ $\Longrightarrow$ *qfree*(*foldr and fs* φ)

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

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

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

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

lemma *atoms-list-conjE*:
$a \in$ *atoms*(*list-conj fs*) $\Longrightarrow$ $a \in (\bigcup \varphi \in$ *set fs. atoms* $\varphi)$
apply(*induct fs*)
apply (*simp add*:*list-conj-def*)
apply (*auto simp add*:*list-conj-def Logic.and-def split*:*if-split-asm*)
done

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

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

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

lemma *nqfree-map-fm*: *nqfree* (*map*$_{fm}$ *f* φ) = *nqfree* φ
by (*induct* φ) *simp-all*

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

1.1 Atoms

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

locale *ATOM* =
fixes *aneg* :: '*a* ⇒ '*a fm*
fixes *anormal* :: '*a* ⇒ *bool*
assumes *nqfree-aneg*: *nqfree*(*aneg a*)
assumes *anormal-aneg*: *anormal a* ⟹ ∀ *b*∈*atoms*(*aneg a*). *anormal b*

fixes I_a :: '*a* ⇒ '*b list* ⇒ *bool*
assumes I_a-*aneg*: *interpret* I_a (*aneg a*) *xs* = (¬ I_a *a xs*)

fixes *depends*$_0$ :: '*a* ⇒ *bool*
and *decr* :: '*a* ⇒ '*a*
assumes *not-dep-decr*: ¬*depends*$_0$ *a* ⟹ I_a *a* (*x*#*xs*) = I_a (*decr a*) *xs*
assumes *anormal-decr*: ¬ *depends*$_0$ *a* ⟹ *anormal a* ⟹ *anormal*(*decr a*)

begin

fun *atoms*$_0$ :: '*a fm* ⇒ '*a list* **where**
atoms$_0$ *TrueF* = [] |
atoms$_0$ *FalseF* = [] |
atoms$_0$ (*Atom a*) = (*if depends*$_0$ *a then* [*a*] *else* []) |
atoms$_0$ (*And* φ_1 φ_2) = *atoms*$_0$ φ_1 @ *atoms*$_0$ φ_2 |
atoms$_0$ (*Or* φ_1 φ_2) = *atoms*$_0$ φ_1 @ *atoms*$_0$ φ_2 |
atoms$_0$ (*Neg* φ) = *atoms*$_0$ φ

abbreviation *I* **where** *I* ≡ *interpret* I_a

fun *nnf* :: '*a fm* ⇒ '*a fm* **where**
nnf (*And* φ_1 φ_2) = *And* (*nnf* φ_1) (*nnf* φ_2) |
nnf (*Or* φ_1 φ_2) = *Or* (*nnf* φ_1) (*nnf* φ_2) |
nnf (*Neg TrueF*) = *FalseF* |
nnf (*Neg FalseF*) = *TrueF* |

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

lemma *nqfree-nnf*: $qfree\ \varphi \Longrightarrow nqfree(nnf\ \varphi)$
by(*induct* φ *rule*:*nnf.induct*)
(*simp-all add*: *nqfree-aneg and-def or-def*)

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

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

lemma *I-dnf*:
$nqfree\ \varphi \Longrightarrow (\exists as{\in}set\ (dnf\ \varphi).\ \forall a{\in}set\ as.\ I_a\ a\ xs) = I\ \varphi\ xs$
by (*induct* φ) (*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* φ_1 φ_2) $\longleftrightarrow$ *normal* φ_1 $\wedge$ *normal* φ_2
normal (*Or* φ_1 φ_2) $\longleftrightarrow$ *normal* φ_1 $\wedge$ *normal* φ_2
normal (*Neg* φ) $\longleftrightarrow$ *normal* φ
normal (*ExQ* φ) $\longleftrightarrow$ *normal* φ
by(*auto simp*:*normal-def*)

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

lemma *normal-and*[*simp*]:
normal φ_1 $\Longrightarrow$ *normal* φ_2 $\Longrightarrow$ *normal* (*and* φ_1 φ_2)
by (*simp add*:*Logic.and-def*)

lemma *normal-or*[*simp*]:
normal φ_1 $\Longrightarrow$ *normal* φ_2 $\Longrightarrow$ *normal* (*or* φ_1 φ_2)
by (*simp add*:*Logic.or-def*)

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

lemma *normal-nnf*: *normal* φ $\Longrightarrow$ *normal*(*nnf* φ)
by(*induct* φ *rule*:*nnf.induct*) *simp-all*

lemma *normal-map-fm*:
$\forall$ *a*. *anormal*(*f a*) = *anormal*(*a*) $\Longrightarrow$ *normal* (map_{fm} *f* φ) = *normal* φ
by(*induct* φ) *auto*

lemma *anormal-nnf*:
qfree φ $\Longrightarrow$ *normal* φ $\Longrightarrow$ $\forall$ *a*$\in$*atoms*(*nnf* φ). *anormal a*
apply(*induct* φ *rule*:*nnf.induct*)
apply(*unfold normal-def*)
apply(*simp-all*)
apply (*blast dest*:*anormal-aneg*)+
done

lemma *atoms-dnf*: *nqfree* φ $\Longrightarrow$ *as* $\in$ *set*(*dnf* φ) $\Longrightarrow$ *a* $\in$ *set as* $\Longrightarrow$ *a* $\in$ *atoms* φ
by(*induct* φ *arbitrary*: *as a rule*:*nqfree.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*. *depends*$_0$ *a*];
indep = [*Atom*(*decr a*). *a*←*as*, ¬ *depends*$_0$ *a*]
in and qf (*list-conj indep*))

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

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

lemma *I-qelim*:
assumes *qe*: ⋀*as*. (∀ *a* ∈ *set as*. *depends*$_0$ *a*) ⟹ *is-dnf-qe qe as*
shows *is-dnf-qe* (*qelim qe*) *as* (**is** ∀ *xs*. *?P xs*)
proof

fix *xs*
let *?as0* = *filter* $depends_0$ *as*
let *?as1* = *filter* (*Not* ∘ $depends_0$) *as*
have *I* (*qelim qe as*) *xs* =
(*I* (*qe ?as0*) *xs* ∧ (∀ *a*∈*set*(*map decr ?as1*). I_a *a xs*))
(**is** - = (- ∧ *?B*)) **by**(*force simp add*:*qelim-def*)
also have ... = ((∃ *x*. ∀ *a* ∈ *set ?as0*. I_a *a* (*x*#*xs*)) ∧ *?B*)
by(*simp add*:*qe not-dep-decr*)
also have ... = (∃ *x*. (∀ *a* ∈ *set ?as0*. I_a *a* (*x*#*xs*)) ∧ *?B*) **by** *blast*
also have *?B* = (∀ *a* ∈ *set ?as1*. I_a (*decr a*) *xs*) **by** *simp*
also have (∃ *x*. (∀ *a* ∈ *set ?as0*. I_a *a* (*x*#*xs*)) ∧ ...) =
(∃ *x*. (∀ *a* ∈ *set ?as0*. I_a *a* (*x*#*xs*)) ∧
(∀ *a* ∈ *set ?as1*. I_a *a* (*x*#*xs*)))
by(*simp add*: *not-dep-decr*)
also have ... = (∃ *x*. ∀ *a* ∈ *set*(*?as0* @ *?as1*). I_a *a* (*x*#*xs*))
by (*simp add*:*ball-Un*)
also have ... = (∃ *x*. ∀ *a* ∈ *set*(*as*). I_a *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* φ_1 φ_2) = *and* (*lift-dnf-qe qe* φ_1) (*lift-dnf-qe qe* φ_2) |
lift-dnf-qe qe (*Or* φ_1 φ_2) = *or* (*lift-dnf-qe qe* φ_1) (*lift-dnf-qe qe* φ_2) |
lift-dnf-qe qe (*Neg* φ) = *neg*(*lift-dnf-qe qe* φ) |
lift-dnf-qe qe (*ExQ* φ) = *Disj* (*dnf*(*nnf*(*lift-dnf-qe qe* φ))) (*qelim qe*) |
lift-dnf-qe qe φ = φ

lemma *qfree-lift-dnf-qe*: (⋀*as*. (∀ *a*∈*set as*. $depends_0$ *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* |$depends_0$| → |*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*. $depends_0$ *a*) ⟹ *qfree*(*qe as*)
and ⋀*as*. (∀ *a* ∈ *set as*. $depends_0$ *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* ∈ *lists* |*depends*$_0$| → |*qfree*|
and ∀ *as* ∈ *lists* |*depends*$_0$|. *is-dnf-qe qe as*
shows *I* (*lift-dnf-qe qe* φ) *xs* = *I* φ *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*: ⋀*xs as*. ∀ *a* ∈ *set as*. *depends*$_0$ *a* ∧ *anormal a* ⟹ *is-dnf-qe qe as*
and *nm*: ∀ *a* ∈ *set as*. *anormal a*
shows *I* (*qelim qe as*) *xs* = (∃ *x*. ∀ *a*∈*set as*. I_a *a* (*x*#*xs*))
proof −
  let *?as0* = *filter depends*$_0$ *as*
  let *?as1* = *filter* (*Not* ∘ *depends*$_0$) *as*
  have *I* (*qelim qe as*) *xs* =
      (*I* (*qe ?as0*) *xs* ∧ (∀ *a*∈*set*(*map decr ?as1*). I_a *a xs*))
    (**is** - = (- ∧ *?B*)) **by**(*force simp add*:*qelim-def*)
  also have ... = ((∃ *x*. ∀ *a* ∈ *set ?as0*. I_a *a* (*x*#*xs*)) ∧ *?B*)
    by(*simp add*:*qe nm not-dep-decr*)
  also have ... = (∃ *x*. (∀ *a* ∈ *set ?as0*. I_a *a* (*x*#*xs*)) ∧ *?B*) **by** *blast*
  also have *?B* = (∀ *a* ∈ *set ?as1*. I_a (*decr a*) *xs*) **by** *simp*
  also have (∃ *x*. (∀ *a* ∈ *set ?as0*. I_a *a* (*x*#*xs*)) ∧ ...) =
          (∃ *x*. (∀ *a* ∈ *set ?as0*. I_a *a* (*x*#*xs*)) ∧
              (∀ *a* ∈ *set ?as1*. I_a *a* (*x*#*xs*)))
    by(*simp add*: *not-dep-decr*)
  also have ... = (∃ *x*. ∀ *a* ∈ *set*(*?as0* @ *?as1*). I_a *a* (*x*#*xs*))
    by (*simp add*:*ball-Un*)
  also have ... = (∃ *x*. ∀ *a* ∈ *set*(*as*). I_a *a* (*x*#*xs*))
    by *simp blast*
  finally show *?thesis* **.**
qed

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

lemma *anormal-atoms-qelim*:
  (⋀*as*. ∀ *a* ∈ *set as*. *depends*$_0$ *a* ∧ *anormal a* ⟹ *normal*(*qe as*)) ⟹
   ∀ *a* ∈ *set as*. *anormal a* ⟹ *a* ∈ *atoms*(*qelim qe as*) ⟹ *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*$_0$ *as* **in** *meta-allE*)
 apply(*simp*)
apply(*erule-tac x* = *filter depends*$_0$ *as* **in** *meta-allE*)
apply(*simp*)
done

lemma *normal-lift-dnf-qe*:

assumes $\bigwedge as.\ \forall\, a \in set\ as.\ depends_0\ a \Longrightarrow qfree(qe\ as)$
and $\bigwedge as.\ \forall\, a \in set\ as.\ depends_0\ a \wedge anormal\ a \Longrightarrow normal(qe\ as)$
shows $normal\ \varphi \Longrightarrow normal(lift\text{-}dnf\text{-}qe\ qe\ \varphi)$
proof(*simp add*:*normal-def*, *induct* φ)
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 $\bigwedge as.\ \forall\, a \in set\ as.\ depends_0\ a \Longrightarrow qfree(qe\ as)$
and $\bigwedge as.\ \forall\, a \in set\ as.\ depends_0\ a \wedge anormal\ a \Longrightarrow normal(qe\ as)$
and $\bigwedge xs\ as.\ \forall\, a \in set\ as.\ depends_0\ a \wedge anormal\ a \Longrightarrow is\text{-}dnf\text{-}qe\ qe\ as$
shows $normal\ f \Longrightarrow I\ (lift\text{-}dnf\text{-}qe\ qe\ f)\ xs = I\ f\ xs$
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 $qe \in lists\ |depends_0| \to |qfree|$
and $qe \in lists\ (\ |depends_0| \cap |anormal|\ ) \to |normal|$
and $\forall\, as \in lists(\ |depends_0| \cap |anormal|\ ).\ is\text{-}dnf\text{-}qe\ qe\ as$
shows $normal\ f \Longrightarrow I\ (lift\text{-}dnf\text{-}qe\ qe\ f)\ xs = I\ f\ xs$
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* ⇒ *'a fm*) ⇒ *'a fm* ⇒ *'a fm* **where**
lift-nnf-qe qe (*And* φ_1 φ_2) = *and* (*lift-nnf-qe qe* φ_1) (*lift-nnf-qe qe* φ_2) |
lift-nnf-qe qe (*Or* φ_1 φ_2) = *or* (*lift-nnf-qe qe* φ_1) (*lift-nnf-qe qe* φ_2) |
lift-nnf-qe qe (*Neg* φ) = *neg*(*lift-nnf-qe qe* φ) |
lift-nnf-qe qe (*ExQ* φ) = *qe*(*nnf*(*lift-nnf-qe qe* φ)) |
lift-nnf-qe qe φ = φ

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

$\Longrightarrow$ *qfree*(*lift-nnf-qe qe* φ)
by (*induct* φ) (*simp-all add*:*nqfree-nnf*)

lemma *qfree-lift-nnf-qe2*:
  qe $\in$ |*nqfree*| $\rightarrow$ |*qfree*| $\Longrightarrow$ *qfree*(*lift-nnf-qe qe* φ)
by(*simp add*:*Pi-def qfree-lift-nnf-qe*)

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

lemma *I-lift-nnf-qe2*:
assumes *qe* $\in$ |*nqfree*| $\rightarrow$ |*qfree*|
and $\forall\, \varphi \in$ |*nqfree*|. $\forall\, xs.\ I\ (qe\ \varphi)\ xs = (\exists\, x.\ I\ \varphi\ (x\#xs))$
shows $I\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
using *assms* **by**(*simp add*:*Pi-def I-lift-nnf-qe*)

lemma *normal-lift-nnf-qe*:
assumes $\bigwedge\varphi$. *nqfree* $\varphi \Longrightarrow$ *qfree*(*qe* φ)
and $\bigwedge\varphi$. *nqfree* $\varphi \Longrightarrow$ *normal* $\varphi \Longrightarrow$ *normal*(*qe* φ)
shows *normal* $\varphi \Longrightarrow$ *normal*(*lift-nnf-qe qe* φ)
by (*induct* φ)
  (*simp-all add*: *assms Logic.neg-def normal-nnf*
    nqfree-nnf qfree-lift-nnf-qe)

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

lemma *I-lift-nnf-qe-normal2*:
assumes *qe* $\in$ |*nqfree*| $\rightarrow$ |*qfree*|
and *qe* $\in$ |*nqfree*| $\cap$ |*normal*| $\rightarrow$ |*normal*|
and $\forall\, \varphi \in$ |*normal*| $\cap$ |*nqfree*|. $\forall\, xs.\ I\ (qe\ \varphi)\ xs = (\exists\, x.\ I\ \varphi\ (x\#xs))$
shows *normal* $\varphi \Longrightarrow I\ (lift\text{-}nnf\text{-}qe\ qe\ \varphi)\ xs = I\ \varphi\ xs$
using *assms* **by**(*simp add*:*Pi-def I-lift-nnf-qe-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 $solvable_0$ :: $'a \Rightarrow bool$
and *trivial* :: $'a \Rightarrow bool$
and $subst_0$ :: $'a \Rightarrow 'a \Rightarrow 'a$
assumes $subst_0$:
$[\![solvable_0\ eq;\ \neg trivial\ eq;\ I_a\ eq\ (x\#xs);\ depends_0\ a]\!]$
$\Longrightarrow I_a\ (subst_0\ eq\ a)\ xs = I_a\ a\ (x\#xs)$
and *trivial*: $trivial\ eq \Longrightarrow I_a\ eq\ xs$
and *solvable*: $solvable_0\ eq \Longrightarrow \exists\, x.\ I_a\ eq\ (x\#xs)$
and *is-triv-self-subst*: $solvable_0\ eq \Longrightarrow trivial\ (subst_0\ eq\ eq)$

begin

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

theorem *I-lift-eq-qe*:
assumes *dep*: $\forall\, a \in set\ as.\ depends_0\ a$
assumes *qe*: $\bigwedge as.\ (\forall\, a \in set\ as.\ depends_0\ a \wedge \neg\ solvable_0\ a) \Longrightarrow$
$I\ (qe\ as)\ xs = (\exists\, x.\ \forall\, a \in set\ as.\ I_a\ a\ (x\#xs))$
shows $I\ (lift\text{-}eq\text{-}qe\ qe\ as)\ xs = (\exists\, x.\ \forall\, a \in set\ as.\ I_a\ a\ (x\#xs))$
(**is** *?L* = *?R*)
proof −
let *?as* = [$a \leftarrow as.\ \neg\ trivial\ a$]
show *?thesis*
proof (*cases* [$a \leftarrow ?as.\ solvable_0\ a$])
case *Nil*
hence $\forall\, a \in set\ as.\ \neg\ trivial\ a \longrightarrow \neg\ solvable_0\ a$
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 $eq \in set\ as\ solvable_0\ eq\ \neg\ trivial\ eq$
by (*auto simp*: *filter-eq-Cons-iff*)
then obtain *e* **where** $I_a\ eq\ (e\#xs)$ **by**(*metis solvable*)
have $\forall\, a \in set\ as.\ I_a\ a\ (e\ \#\ xs) = I_a\ (subst_0\ eq\ a)\ xs$
by(*simp add*: $subst_0[OF\ \langle solvable_0\ eq\rangle\ \langle\neg\ trivial\ eq\rangle\ \langle I_a\ eq\ (e\#xs)\rangle]$ *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 subst*$_0$)
done
qed
qed

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

lemma *qfree-lift-eq-qe*:
(⋀*as*. ∀ *a*∈*set as*. *depends*$_0$ *a* ⟹ *qfree* (*qe as*)) ⟹
∀ *a*∈*set as*. *depends*$_0$ *a* ⟹ *qfree*(*lift-eq-qe qe as*)
by(*simp add*:*lift-eq-qe-def ball-Un split*:*list.split*)

lemma *qfree-lift-dnfeq-qe*: (⋀*as*. (∀ *a*∈*set as*. *depends*$_0$ *a*) ⟹ *qfree*(*qe as*))
⟹ *qfree*(*lift-dnfeq-qe qe* φ)
by(*simp add*: *lift-dnfeq-qe-def qfree-lift-dnf-qe qfree-lift-eq-qe*)

lemma *I-lift-dnfeq-qe*:
(⋀*as*. (∀ *a* ∈ *set as*. *depends*$_0$ *a*) ⟹ *qfree*(*qe as*)) ⟹
(⋀*as*. (∀ *a* ∈ *set as*. *depends*$_0$ *a* ∧ ¬ *solvable*$_0$ *a*) ⟹ *is-dnf-qe qe as*) ⟹
I (*lift-dnfeq-qe qe* φ) *xs* = *I* φ *xs*
by(*simp add*:*lift-dnfeq-qe-def I-lift-dnf-qe qfree-lift-eq-qe I-lift-eq-qe*)

lemma *I-lift-dnfeq-qe2*:
qe ∈ *lists* |*depends*$_0$| → |*qfree*| ⟹
(∀ *as* ∈ *lists*(|*depends*$_0$| ∩ − |*solvable*$_0$|). *is-dnf-qe qe as*) ⟹
I (*lift-dnfeq-qe qe* φ) *xs* = *I* φ *xs*
using *in-lists-conv-set*[**where** *?'a* = *'a*]
by(*simp add*:*Pi-def I-lift-dnfeq-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* ⟹ ∃ *y*. *x* < *y* ∧ *y* < *z*
and *no-ub*: ∃ *u*. *x* < *u* **and** *no-lb*: ∃ *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 neg_{dlo} :: *atom* ⇒ *atom fm* **where**
neg_{dlo} (*Less i j*) = *Or* (*Atom*(*Less j i*)) (*Atom*(*Eq i j*)) |
neg_{dlo} (*Eq i j*) = *Or* (*Atom*(*Less i j*)) (*Atom*(*Less j i*))

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

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

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

global-interpretation *DLO*:

ATOM neg_{dlo} ‹λ-. *True*› I_{dlo} $depends_{dlo}$ $decr_{dlo}$
defines *nnf* = *DLO.nnf*
and *qelim* = *DLO.qelim*
and *lift-dnf-qe* = *DLO.lift-dnf-qe*
and *lift-nnf-qe* = *DLO.lift-nnf-qe*
proof
show ‹*nqfree* (neg_{dlo} *a*)› **for** *a*
by (*cases a*) *simp-all*
show ‹*interpret* I_{dlo} (neg_{dlo} *a*) *xs* $\longleftrightarrow$ $\neg$ I_{dlo} *a xs*› **for** *a* **and** *xs* :: ‹'*a list*›
by (*cases a*) *auto*
show ‹I_{dlo} *a* (*x* # *xs*) $\longleftrightarrow$ I_{dlo} ($decr_{dlo}$ *a*) *xs*› **if** ‹$\neg$ $depends_{dlo}$ *a*› **for** *a* **and** *x* **and** *xs* :: ‹'*a list*›
using *that* **by** (*cases a*) *simp-all*
qed *simp-all*

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

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

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

lemma *EQ-conv-set-ebounds*:
x ∈ *EQ f xs* = (∃ *e*∈*set*(*ebounds*($DLO.atoms_0$ *f*)). *x* = *xs*!*e*)
by(*auto simp*: *EQ-def set-ebounds*)

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

fun *asubst* :: *nat* ⇒ *atom* ⇒ *atom* **where**
asubst k (*Less i j*) = *Less* (*isubst k i*) (*isubst k j*)|
asubst k (*Eq i j*) = *Eq* (*isubst k i*) (*isubst k j*)

abbreviation *subst* φ *k* ≡ map_{fm} (*asubst k*) φ

lemma *I-subst*:
qfree f ⟹ *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* ⇒ *atom fm* **where**
amin-inf (*Less - 0*) = *FalseF* |
amin-inf (*Less 0 -*) = *TrueF* |
amin-inf (*Less* (*Suc i*) (*Suc j*)) = *Atom*(*Less i j*) |
amin-inf (*Eq 0 0*) = *TrueF* |
amin-inf (*Eq 0 -*) = *FalseF* |
amin-inf (*Eq - 0*) = *FalseF* |
amin-inf (*Eq* (*Suc i*) (*Suc j*)) = *Atom*(*Eq i j*)

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

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

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

lemma *min-inf*:
nqfree f ⟹ ∃ *x*. ∀ *y*≤*x*. *DLO.I* (inf_- *f*) *xs* = *DLO.I f* (*y* # *xs*)
(**is** - ⟹ ∃ *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:
  nqfree f ⟹ ∃ x.∀ y≥x. DLO.I (inf₊ f) xs = DLO.I f (y # xs)
  (is - ⟹ ∃ x. ?P f x)
proof (induct f)
  have dlo-bound: ⋀z::'a. ∃ x. ∀ y≥x. y > z
  proof −
    fix z
    from no-ub obtain w :: 'a where w > z ..
    then have ∀ y≥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.atoms_0 f))
    by (auto simp:set-lbounds image-def)
  thus ?thesis by simp
qed

lemma finite-UB: finite(UB f xs)
proof −
  have UB f xs = (λk.  xs!k) ‘ set(ubounds(DLO.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* $\varphi \Longrightarrow$ *qfree*(*inf*$_-$ φ)
by(*induct* φ)(*simp-all add*:*qfree-amin-inf*)

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

lemma *qfree-plus-inf*: *nqfree* $\varphi \Longrightarrow$ *qfree*(*inf*$_+$ φ)
by(*induct* φ)(*simp-all add*:*qfree-aplus-inf*)

end

theory *QEdlo*
imports *DLO*
begin

3.2 DNF-based quantifier elimination

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

theorem *I-qe-dlo*$_1$:
assumes *less*: $\forall\, a \in$ *set as*. *is-Less a* **and** *dep*: $\forall\, a \in$ *set as*. *depends*$_{dlo}$ *a*
shows *DLO.I* (*qe-dlo*$_1$ *as*) *xs* = ($\exists\, x$. $\forall\, a \in$ *set as*. I_{dlo} *a* (*x*#*xs*))
(**is** *?L* = *?R*)
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 *2*: *Less 0 0* $\notin$ *set as* $\Longrightarrow \forall\, a \in$ *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 ‹*Less 0 0* $\notin$ *set as*› ‹*a* $\in$ *set as*›
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 ‹*a* $\in$ *set as*› **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 ∉ set as by (auto simp:qe-dlo_1-def)
  with qe1 have 1: ∀ x∈?Ls. ∀ y∈?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 ∈ set as | Less 0 i ∈ set as
    moreover hence i ≠ 0 using 0 by iprover
    ultimately have (x#xs) ! i = xs!(i − 1) by (simp add: nth-Cons')
  } note this[simp]
  { assume nonempty: ?Ls ≠ {} ∧ ?Us ≠ {}
    hence Max (⋃x∈?Ls. {xs!x}) < Min (⋃x∈?Us. {xs!x})
      using 1 finite by auto
    then obtain m where ?lb < m ∧ m < ?ub using dense by blast
    hence ∀ i∈?Ls. xs!i < m and ∀ j∈?Us. m < xs!j
      using nonempty finite by auto
    hence ∀ a ∈ set as. I_dlo a (m # xs) using 2[OF 0] by(auto simp:less)
    hence ?R .. }
  moreover
  { assume asm: ?Ls ≠ {} ∧ ?Us = {}
    then obtain m where ?lb < m using no-ub by blast
    hence ∀ a∈ set as. I_dlo a (m # xs) using 2[OF 0] asm finite by auto
    hence ?R .. }
  moreover
  { assume asm: ?Ls = {} ∧ ?Us ≠ {}
    then obtain m where m < ?ub using no-lb by blast
    hence ∀ a∈ set as. I_dlo a (m # xs) using 2[OF 0] asm finite by auto
    hence ?R .. }
  moreover
  { assume ?Ls = {} ∧ ?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: ∀ a∈ set as. I_dlo a (x # xs) ..
  hence 0: Less 0 0 ∉ set as by auto
  { fix i j
    assume asm: Less i 0 ∈ set as Less 0 j ∈ set as
    hence (x#xs)!i < x x < (x#xs)!j using 1 by auto+
    hence (x#xs)!i < (x#xs)!j by(rule order-less-trans)
    moreover have ¬(i = 0 | 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_1-def is-Less-iff split:atom.splits nat.splits)
qed
```

lemma *I-qe-dlo$_1$-pretty*:
$\forall a \in set\ as.\ is\text{-}Less\ a \wedge depends_{dlo}\ a \Longrightarrow DLO.is\text{-}dnf\text{-}qe$ - $qe\text{-}dlo_1\ as$
by(*metis I-qe-dlo$_1$*)

definition *subst* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
subst i j k = (*if k=0 then if i=0 then j else i else k*) − *1*
fun *subst$_0$* :: *atom* ⇒ *atom* ⇒ *atom* **where**
subst$_0$ (*Eq i j*) *a* = (*case a of*
Less m n ⇒ *Less* (*subst i j m*) (*subst i j n*)
| *Eq m n* ⇒ *Eq* (*subst i j m*) (*subst i j n*))

lemma *subst$_0$-pretty*:
subst$_0$ (*Eq i j*) (*Less m n*) = *Less* (*subst i j m*) (*subst i j n*)
subst$_0$ (*Eq i j*) (*Eq m n*) = *Eq* (*subst i j m*) (*subst i j n*)
by *auto*

global-interpretation *DLO$_e$*:
ATOM-EQ neg$_{dlo}$ ‹*λ-. True*› *I$_{dlo}$ depends$_{dlo}$ decr$_{dlo}$*
‹*λEq i j* ⇒ *i=0* ∨ *j=0* | *a* ⇒ *False*›
‹*λEq i j* ⇒ *i=j* | *a* ⇒ *False*› *subst$_0$*
defines *lift-dnfeq-qe* = ‹*DLO$_e$.lift-dnfeq-qe*›
and *lift-eq-qe* = ‹*DLO$_e$.lift-eq-qe*›
by *standard* (*auto simp add*: *nth-Cons′ subst-def split*: *atom.splits*)

setup ‹*Sign.revert-abbrev* @{*const-abbrev DLO$_e$.lift-dnfeq-qe*}›

definition *qe-dlo* = *DLO$_e$.lift-dnfeq-qe qe-dlo$_1$*

lemma *qfree-qe-dlo$_1$*: *qfree* (*qe-dlo$_1$ as*)
by(*auto simp*:*qe-dlo$_1$-def intro*!:*qfree-list-conj*)

theorem *I-qe-dlo*: *DLO.I* (*qe-dlo φ*) *xs* = *DLO.I φ xs*
unfolding *qe-dlo-def*
by(*fastforce intro*!: *I-qe-dlo$_1$ qfree-qe-dlo$_1$ DLO$_e$.I-lift-dnfeq-qe*
simp: *is-Less-iff not-is-Eq-iff split*:*atom.splits cong*:*conj-cong*)

theorem *qfree-qe-dlo*: *qfree* (*qe-dlo φ*)
by(*simp add*:*qe-dlo-def DLO$_e$.qfree-lift-dnfeq-qe qfree-qe-dlo$_1$*)

end

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

definition *interpret* :: *atom fm* ⇒ *'a*::*dlo list* ⇒ *bool* **where**
interpret = *Logic.interpret I$_{dlo}$*

lemma *interpret-Atoms*:
interpret (*Atom* (*Eq i j*)) *xs* = (*xs*!*i* = *xs*!*j*)
interpret (*Atom* (*Less i j*)) *xs* = (*xs*!*i* < *xs*!*j*)
by(*simp-all add*:*interpret-def*)

lemma *interpret-others*:
interpret (*Neg*(*ExQ* (*Neg f*))) *xs* = (∀ *x*. *interpret f* (*x*#*xs*))
interpret (*Or* (*Neg f1*) *f2*) *xs* = (*interpret f1 xs* ⟶ *interpret f2 xs*)
by(*simp-all add*:*interpret-def*)

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

method-setup *dlo-reify* = ‹
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*}])))
› *dlo reification*

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

3.3 Examples

lemma ∀ *x*::*real*. ¬ *x* < *x*
apply *dlo-reify*
apply (*subst I-qe-dlo* [*symmetric*])
by *eval*

lemma ∀ *x y*::*real*. ∃ *z*. *x* < *y* ⟶ *x* < *z* ∧ *z* < *y*
apply *dlo-reify*
apply (*subst I-qe-dlo* [*symmetric*])
by *eval*

lemma ∃ *x*::*real*. *a*+*b* < *x* ∧ *x* < *c*∗*d*
apply *dlo-reify*
apply (*subst I-qe-dlo* [*symmetric*])
apply *normalization*
oops

lemma ∀ *x*::*real*. ¬ *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 $asubst_2 :: nat \Rightarrow nat \Rightarrow atom \Rightarrow atom\ fm$ **where**
$asubst_2\ l\ u\ (Less\ 0\ 0) = FalseF$ |
$asubst_2\ l\ u\ (Less\ 0\ (Suc\ j)) = Or\ (Atom(Less\ u\ j))\ (Atom(Eq\ u\ j))$ |
$asubst_2\ l\ u\ (Less\ (Suc\ i)\ 0) = Or\ (Atom(Less\ i\ l))\ (Atom(Eq\ i\ l))$ |
$asubst_2\ l\ u\ (Less\ (Suc\ i)\ (Suc\ j)) = Atom(Less\ i\ j)$ |
$asubst_2\ l\ u\ (Eq\ 0\ 0) = TrueF$ |
$asubst_2\ l\ u\ (Eq\ 0\ \text{-}) = FalseF$ |
$asubst_2\ l\ u\ (Eq\ \text{-}\ 0) = FalseF$ |
$asubst_2\ l\ u\ (Eq\ (Suc\ i)\ (Suc\ j)) = Atom(Eq\ i\ j)$

abbreviation $subst_2\ l\ u \equiv amap_{fm}\ (asubst_2\ l\ u)$

lemma *I-subst₂1*:
$nqfree\ f \Longrightarrow xs!l < xs!u \Longrightarrow DLO.I\ (subst_2\ l\ u\ f)\ xs$
$\Longrightarrow xs!l < x \Longrightarrow x < xs!u \Longrightarrow DLO.I\ f\ (x\#xs)$
proof(*induct f arbitrary*: *x*)
case (*Atom a*) **thus** *?case*
by (*cases* (*l*,*u*,*a*) *rule*: $asubst_2.cases$) *auto*
qed *auto*

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

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

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

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

lemma *innermost-intvl*:
$[\![\ nqfree\ f;\ nolub\ f\ xs\ l\ x\ u;\ l < x;\ x < u;\ x \notin EQ\ f\ xs;$
$DLO.I\ f\ (x\#xs);\ l < y;\ y < u]\!]$
$\Longrightarrow DLO.I\ f\ (y\#xs)$
proof(*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-subst_2 2:
 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 (subst_2 l u f) xs
proof (induct f)
  case (Atom a) show ?case
    apply (cases (l,u,a) rule: asubst_2.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-interior_1 φ =
(let as = DLO.atoms_0 φ; lbs = lbounds as; ubs = ubounds as; ebs = ebounds as;
     intrs = [And (Atom(Less l u)) (subst_2 l u φ). l←lbs, u←ubs]
 in list-disj (inf_- φ # inf_+ φ # intrs @ map (subst φ) ebs))

lemma dense-interval:
assumes finite L finite U l ∈ L u ∈ U l < x x < u P(x::'a::dlo)
and dense: ⋀y l u. ⟦ ∀ y∈{l<..<x}. y ∉ L; ∀ y∈{x<..<u}. y ∉ U;
```

```
                    l<x;x<u; l<y;y<u ⟧ ⟹ P y
shows ∃ l∈L.∃ u∈U. l<x ∧ x<u ∧ (∀ y∈{l<..<x}. y∉L) ∧ (∀ y∈{x<..<u}. y∉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 simp
  ultimately show ?thesis using ‹l < x› ‹x < u› ‹?L ≠ {}› ‹?U ≠ {}›
    by(blast intro!:dense greaterThanLessThan-iff[THEN iffD1])
qed


theorem I-interior1:
assumes nqfree φ shows DLO.I (qe-interior₁ φ) xs = (∃ x. DLO.I φ (x#xs))
  (is ?QE = ?EX)
proof
  assume ?QE
  { assume DLO.I (inf₋ φ) xs
    hence ?EX using ‹?QE› min-inf[of φ xs] ‹nqfree φ›
      by(auto simp add:qe-interior₁-def amap-fm-list-disj)
  } moreover
  { assume  DLO.I (inf₊ φ) xs
    hence ?EX using ‹?QE› plus-inf[of φ xs] ‹nqfree φ›
      by(auto simp add:qe-interior₁-def amap-fm-list-disj)
  } moreover
  { assume ¬DLO.I (inf₋ φ) xs ∧ ¬DLO.I (inf₊ φ) xs ∧
          (∀ x∈EQ φ xs. ¬DLO.I φ (x#xs))
    with ‹?QE› ‹nqfree φ› obtain l u
      where DLO.I (subst₂ l u φ) xs and xs!l < xs!u
    by(fastforce simp: qe-interior₁-def set-lbounds set-ubounds I-subst EQ-conv-set-ebounds)
    moreover then obtain x where xs!l < x ∧ x < xs!u by(metis dense)
    ultimately have DLO.I φ (x # xs)
      using ‹nqfree φ› I-subst₂1[OF ‹nqfree φ› ‹xs!l < xs!u›] by simp
```

```
    hence ?EX .. }
  ultimately show ?EX by blast
next
  let ?as = DLO.atoms₀ φ let ?E = set(ebounds ?as)
  assume ?EX
  then obtain x where x: DLO.I φ (x#xs) ..
  { assume DLO.I (inf₋ φ) xs ∨ DLO.I (inf₊ φ) xs
    hence ?QE using ‹nqfree φ› by(auto simp:qe-interior₁-def)
  } moreover
  { assume ∃k ∈ ?E. DLO.I (subst φ k) xs
    hence ?QE by(force simp:qe-interior₁-def) } moreover
  { assume ¬ DLO.I (inf₋ φ) xs and ¬ DLO.I (inf₊ φ) xs
    and ∀k ∈ ?E. ¬ DLO.I (subst φ k) xs
    hence noE: ∀e ∈ EQ φ xs. ¬ DLO.I φ (e#xs)
      using ‹nqfree φ› by (force simp:set-ebounds EQ-def I-subst)
    hence x ∉ EQ φ xs using x by fastforce
    obtain l where l ∈ LB φ xs l < x
      using LBex[OF ‹nqfree φ› x ‹¬ DLO.I(inf₋ φ) xs› ‹x ∉ EQ φ xs›] ..
    obtain u where u ∈ UB φ xs x < u
      using UBex[OF ‹nqfree φ› x ‹¬ DLO.I(inf₊ φ) xs› ‹x ∉ EQ φ xs›] ..
    have ∃l∈LB φ xs. ∃u∈UB φ xs. l<x ∧ x<u ∧ nolub φ xs l x u ∧ (∀y. l < y
∧ y < u ⟶ DLO.I φ (y#xs))
     using dense-interval[where P = λx. DLO.I φ (x#xs), OF finite-LB finite-UB
‹l:LB φ xs› ‹u:UB φ xs› ‹l<x› ‹x<u› x] x innermost-intvl[OF ‹nqfree φ› - - - ‹x
∉ EQ φ xs›]
      by (simp add:nolub-def)
    then obtain m n where
      Less (Suc m) 0 ∈ set ?as Less 0 (Suc n) ∈ set ?as
      xs!m < x ∧ x < xs!n
      nolub φ xs (xs!m) x (xs!n)
      ∀y. xs!m < y ∧ y < xs!n ⟶ DLO.I φ (y#xs)
      by blast
    moreover
    hence DLO.I (subst₂ m n φ) xs using noE
      by(force intro!: I-subst₂2[OF ‹nqfree φ›])
    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 φ ⟹ qfree (subst₂ l u φ)
by(induct φ) (simp-all add:qfree-asubst₂)

lemma qfree-interior1: nqfree φ ⟹ qfree(qe-interior₁ φ)
apply(simp add:qe-interior₁-def)
apply(rule qfree-list-disj)
```

apply (*auto simp*:*qfree-min-inf qfree-plus-inf qfree-subst*$_2$ *qfree-map-fm*)
done

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

lemma *qfree-qe-interior*: *qfree*(*qe-interior* φ)
by(*simp add*: *qe-interior-def DLO.qfree-lift-nnf-qe qfree-interior1*)

lemma *I-qe-interior*: *DLO.I* (*qe-interior* φ) *xs* = *DLO.I* φ *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* ⇒ *atom* ⇒ *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* ⇒ *nat* ⇒ *atom fm* (‹*subst*$_+$›) **where**
subst$_+$ φ *k* ≡ *amap*$_{fm}$ (*asubst*$_+$ *k*) φ

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

lemma *nolb-And*[*simp*]:
  nolb (*And* φ_1 φ_2) *xs l x* = (*nolb* φ_1 *xs l x* ∧ *nolb* φ_2 *xs l x*)
apply(*clarsimp simp*:*nolb-def*)
apply *blast*
done

lemma *nolb-Or*[*simp*]:
  nolb (*Or* φ_1 φ_2) *xs l x* = (*nolb* φ_1 *xs l x* ∧ *nolb* φ_2 *xs l x*)

apply(*clarsimp simp*:*nolb-def*)
apply *blast*
done

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

lemma *innermost-intvl*:
⟦ *nqfree* φ; *nolb* φ *xs* *l* *x*; $l < x$; $x \notin$ *EQ* φ *xs*; *DLO.I* φ (x#*xs*); $l < y$; $y \leq x$⟧
$\Longrightarrow$ *DLO.I* φ (y#*xs*)
proof(*induct* φ)
case (*Atom* *a*)
show *?case*
proof (*cases* *a*)
case (*Less* *i* *j*)
then show *?thesis* **using** *Atom*
unfolding *nolb-def*
by (*clarsimp simp*: *nth.simps Ball-def split*:*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*:
nqfree $\varphi \Longrightarrow$ *xs*!*l* $< x \Longrightarrow$ *nolb* φ *xs* (*xs*!*l*) $x \Longrightarrow x \notin$ *EQ* φ *xs*
$\Longrightarrow \forall y \in \{$*xs*!*l* <.. $x\}$. *DLO.I* φ (y#*xs*)
$\Longrightarrow$ *DLO.I* (*subst*$_+$ φ *l*) *xs*
proof(*induct* φ)
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: ⋀y l. ⟦ ∀ y∈{l<..<x}. y ∉ L; l<x; l<y; y≤x ⟧ ⟹ P y
shows ∃ l∈L. l<x ∧ (∀ y∈{l<..<x}. y ∉ L) ∧ (∀ y. l<y ∧ y≤x ⟶ P y)
proof −
  let ?L = {l∈L. l < x}
  let ?ll = Max ?L
  have ?L ≠ {} using ‹l ∈ L› ‹l<x› by (blast intro:order-less-imp-le)
  hence ∀ y. ?ll<y ∧ y<x ⟶ y ∉ L using ‹finite L› by force
  moreover have ?ll ∈ L
  proof
    show ?ll ∈ ?L using ‹finite L› Max-in[OF - ‹?L ≠ {}›] by simp
    show ?L ⊆ L by blast
  qed
  moreover have ?ll < x using ‹finite L› ‹?L ≠ {}› by simp
  ultimately show ?thesis using ‹l < x› ‹?L ≠ {}›
    by(blast intro!:dense greaterThanLessThan-iff[THEN iffD1])
qed


lemma I-subst-peps:
  nqfree φ ⟹ DLO.I (subst+ φ l) xs ⟶
  (∃ leps>xs!l. ∀ x. xs!l < x ∧ x ≤ leps ⟶ DLO.I φ (x#xs))
proof(induct φ)
  case TrueF thus ?case by simp (metis no-ub)
next
  case (Atom a)
  show ?case
  proof (cases (l,a) rule: asubst-peps.cases)
    case 2 thus ?thesis using Atom
      apply(auto)
      apply(drule dense)
      apply(metis One-nat-def xt1(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-eps1(φ) =
(let as = DLO.atoms0 φ; lbs = lbounds as; ebs = ebounds as
 in list-disj (inf_ φ # map (subst+ φ) lbs @ map (subst φ) ebs))

theorem I-qe-eps1:
assumes nqfree φ shows DLO.I (qe-eps1 φ) xs = (∃x. DLO.I φ (x#xs))
  (is ?QE = ?EX)
proof
  let ?as = DLO.atoms0 φ let ?ebs = ebounds ?as
  assume ?QE
  { assume DLO.I (inf_ φ) xs
    hence ?EX using ‹?QE› min-inf[of φ xs] ‹nqfree φ›
      by(auto simp add:qe-eps1-def amap-fm-list-disj)
  } moreover
  { assume ∀i ∈ set ?ebs. ¬DLO.I φ (xs!i # xs)
          ¬ DLO.I (inf_ φ) xs
    with ‹?QE› ‹nqfree φ› obtain l where DLO.I (subst+ φ l) xs
      by(fastforce simp: I-subst qe-eps1-def set-ebounds set-lbounds)
    then obtain leps where DLO.I φ (leps#xs)
      using I-subst-peps[OF ‹nqfree φ›] by fastforce
    hence ?EX .. }
  ultimately show ?EX by blast
next
  let ?as = DLO.atoms0 φ let ?ebs = ebounds ?as
  assume ?EX
  then obtain x where x: DLO.I φ (x#xs) ..
  { assume DLO.I (inf_ φ) xs
    hence ?QE using ‹nqfree φ› by(auto simp:qe-eps1-def)
  } moreover
  { assume ∃k ∈ set ?ebs. DLO.I (subst φ k) xs
```

hence *?QE* **by**(*auto simp*:*qe-eps*$_1$*-def*) **}** **moreover**
{ assume ¬ *DLO.I* (*inf*$_-$ *φ*) *xs*
and ∀ *k* ∈ *set ?ebs*. ¬ *DLO.I* (*subst φ k*) *xs*
hence *noE*: ∀ *e* ∈ *EQ φ xs*. ¬ *DLO.I φ* (*e*#*xs*) **using** ‹*nqfree φ*›
by (*auto simp*:*set-ebounds EQ-def I-subst nth-Cons*′ *split*:*if-split-asm*)
hence *x* ∉ *EQ φ xs* **using** *x* **by** *fastforce*
obtain *l* **where** *l* ∈ *LB φ xs l* < *x*
using *LBex*[*OF* ‹*nqfree φ*› *x* ‹¬ *DLO.I*(*inf*$_-$ *φ*) *xs*› ‹*x* ∉ *EQ φ xs*›] **..**
have ∃ *l*∈*LB φ xs*. *l*<*x* ∧ *nolb φ xs l x* ∧
(∀ *y*. *l* < *y* ∧ *y* ≤ *x* ⟶ *DLO.I φ* (*y*#*xs*))
using *dense-interval*[**where** *P* = λ*x*. *DLO.I φ* (*x*#*xs*), *OF finite-LB* ‹*l*∈*LB φ xs*› ‹*l*<*x*› *x*] *x innermost-intvl*[*OF* ‹*nqfree φ*› - - ‹*x* ∉ *EQ φ xs*›]
by (*simp add*:*nolb-def*)
then obtain *m*
where ∗: *Less* (*Suc m*) *0* ∈ *set ?as* ∧ *xs*!*m* < *x* ∧ *nolb φ xs* (*xs*!*m*) *x*
∧ (∀ *y*. *xs*!*m* < *y* ∧ *y* ≤ *x* ⟶ *DLO.I φ* (*y*#*xs*))
by *blast*
then have *DLO.I* (*subst*$_+$ *φ m*) *xs*
using *noE* **by**(*auto intro*!: *I-subst-peps2*[*OF* ‹*nqfree φ*›])
with ∗ **have** *?QE*
by(*simp add*:*qe-eps*$_1$*-def bex-Un set-lbounds set-ebounds*) *metis*
} ultimately show *?QE* **by** *blast*
qed

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

lemma *qfree-subst-peps*: *nqfree φ* ⟹ *qfree* (*subst*$_+$ *φ k*)
by(*induct φ*) (*simp-all add*:*qfree-asubst-peps*)

lemma *qfree-qe-eps*$_1$: *nqfree φ* ⟹ *qfree*(*qe-eps*$_1$ *φ*)
apply(*simp add*:*qe-eps*$_1$*-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 φ*)
by(*simp add*: *qe-eps-def DLO.qfree-lift-nnf-qe qfree-qe-eps*$_1$)

lemma *I-qe-eps*: *DLO.I* (*qe-eps φ*) *xs* = *DLO.I φ xs*
by(*simp add*:*qe-eps-def DLO.I-lift-nnf-qe qfree-qe-eps*$_1$ *I-qe-eps1*)

end

4 Linear real arithmetic

theory *LinArith*

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

declare *iprod-assoc*[*simp*]

4.1 Basics

4.1.1 Syntax and Semantics

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

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

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

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

lemma *is-Eq-iff*: (∀ *i j*. *a* ≠ *Less i j*) = (∃ *i j*. *a* = *Eq i j*)
by(*cases a*) *auto*

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

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

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

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

fun I_R :: *atom* ⇒ *real list* ⇒ *bool* **where**
I_R (*Less r cs*) *xs* = (*r* < ⟨*cs*,*xs*⟩) |
I_R (*Eq r cs*) *xs* = (*r* = ⟨*cs*,*xs*⟩)

definition $atoms_0$ = *ATOM*.$atoms_0$ $depends_R$

interpretation *R*: *ATOM* neg_R (λ*a*. *True*) I_R $depends_R$ $decr_R$
rewrites *ATOM*.$atoms_0$ $depends_R$ = $atoms_0$
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 add*:$depends_R$*-def split*:*list.splits*)
done
next
case *2*
thus *?case* **by**(*simp add*:$atoms_0$*-def*)
qed

setup ‹*Sign.revert-abbrev* @{*const-abbrev R.I*}›
setup ‹*Sign.revert-abbrev* @{*const-abbrev R.lift-nnf-qe*}›

4.1.2 Shared constructions

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

definition *lbounds as* = [(*r*/*c*, (−*1*/*c*) ∗$_s$ *cs*). *Less r* (*c*#*cs*) ← *as*, *c*>*0*]
definition *ubounds as* = [(*r*/*c*, (−*1*/*c*) ∗$_s$ *cs*). *Less r* (*c*#*cs*) ← *as*, *c*<*0*]
definition *ebounds as* = [(*r*/*c*, (−*1*/*c*) ∗$_s$ *cs*). *Eq r* (*c*#*cs*) ← *as*, *c*≠*0*]

lemma *set-lbounds*:
set(*lbounds as*) = {(*r*/*c*, (−*1*/*c*) ∗$_s$ *cs*)|*r c cs*. *Less r* (*c*#*cs*) ∈ *set as* ∧ *c*>*0*}
by(*force simp*: *lbounds-def split*:*list.splits atom.splits if-splits*)
lemma *set-ubounds*:
set(*ubounds as*) = {(*r*/*c*, (−*1*/*c*) ∗$_s$ *cs*)|*r c cs*. *Less r* (*c*#*cs*) ∈ *set as* ∧ *c*<*0*}
by(*force simp*: *ubounds-def split*:*list.splits atom.splits if-splits*)
lemma *set-ebounds*:
set(*ebounds as*) = {(*r*/*c*, (−*1*/*c*) ∗$_s$ *cs*)|*r c cs*. *Eq r* (*c*#*cs*) ∈ *set as* ∧ *c*≠*0*}
by(*force simp*: *ebounds-def split*:*list.splits atom.splits if-splits*)

abbreviation *EQ* **where**
EQ f xs ≡ {(*r* − ⟨*cs*,*xs*⟩)/*c*|*r c cs*. *Eq r* (*c*#*cs*) ∈ *set*($R.atoms_0$ *f*) ∧ *c*≠*0*}
abbreviation *LB* **where**
LB f xs ≡ {(*r* − ⟨*cs*,*xs*⟩)/*c*|*r c cs*. *Less r* (*c*#*cs*) ∈ *set*($R.atoms_0$ *f*) ∧ *c*>*0*}
abbreviation *UB* **where**
UB f xs ≡ {(*r* − ⟨*cs*,*xs*⟩)/*c*|*r c cs*. *Less r* (*c*#*cs*) ∈ *set*($R.atoms_0$ *f*) ∧ *c*<*0*}

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

```
asubst (r,cs) (Eq s (d#ds)) = Eq (s − d∗r) (d ∗s cs + ds) |
asubst (r,cs) (Less s []) = Less s [] |
asubst (r,cs) (Eq s []) = Eq s []

abbreviation subst φ rcs ≡ map_fm (asubst rcs) φ

definition eval :: real ∗ real list ⇒ real list ⇒ real where
eval rcs xs = fst rcs + ⟨snd rcs,xs⟩

lemma I-asubst:
 I_R (asubst t a) xs = I_R a (eval t xs # xs)
proof(cases a)
  case (Less r cs)
  thus ?thesis by(cases t, cases cs,
    simp-all add:eval-def 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 φ ⟹ R.I (subst φ t) xs = R.I φ (eval t xs # xs)
by(induct φ)(simp-all add:I-asubst)
lemma I-subst-pretty:
  qfree φ ⟹ R.I (subst φ (r,cs)) xs = R.I φ ((r + ⟨cs,xs⟩) # xs)
by(simp add:I-subst eval-def)


fun min-inf :: atom fm ⇒ atom fm (‹inf_›) where
inf_ (And φ1 φ2) = and (inf_ φ1) (inf_ φ2) |
inf_ (Or φ1 φ2) = or (inf_ φ1) (inf_ φ2) |
inf_ (Atom(Less r (c#cs))) =
  (if c<0 then TrueF else if c>0 then FalseF else Atom(Less r cs)) |
inf_ (Atom(Eq r (c#cs))) = (if c=0 then Atom(Eq r cs) else FalseF) |
inf_ φ = φ

fun plus-inf :: atom fm ⇒ atom fm (‹inf_+›) where
inf_+ (And φ1 φ2) = and (inf_+ φ1) (inf_+ φ2) |
inf_+ (Or φ1 φ2) = or (inf_+ φ1) (inf_+ φ2) |
inf_+ (Atom(Less r (c#cs))) =
  (if c>0 then TrueF else if c<0 then FalseF else Atom(Less r cs)) |
inf_+ (Atom(Eq r (c#cs))) = (if c=0 then Atom(Eq r cs) else FalseF) |
inf_+ φ = φ

lemma qfree-min-inf: qfree φ ⟹ qfree(inf_ φ)
by(induct φ rule:min-inf.induct) simp-all
```

```
lemma qfree-plus-inf: qfree φ ⟹ qfree(inf₊ φ)
by(induct φ rule:plus-inf.induct) simp-all

lemma min-inf:
  nqfree f ⟹ ∃ x. ∀ y≤x. R.I (inf₋ f) xs = R.I f (y # xs)
  (is - ⟹ ∃ x. ?P f x)
proof(induct f)
  case (Atom a)
  show ?case
  proof (cases a)
    case (Less r cs)
    show ?thesis
    proof(cases cs)
      case Nil thus ?thesis using Less by simp
    next
      case (Cons c cs)
      { assume c=0 hence ?thesis using Less Cons by simp }
      moreover
      { assume c<0
        hence ?P (Atom a) ((r − ⟨cs,xs⟩ + 1)/c) using Less Cons
          by(auto simp add: field-simps)
        hence ?thesis .. }
      moreover
      { assume c>0
        hence ?P (Atom a) ((r − ⟨cs,xs⟩ − 1)/c) using Less Cons
          by(auto simp add: field-simps)
        hence ?thesis .. }
      ultimately show ?thesis by force
    qed
  next
    case (Eq r cs)
    show ?thesis
    proof(cases cs)
      case Nil thus ?thesis using Eq by simp
    next
      case (Cons c cs)
      { assume c=0 hence ?thesis using Eq Cons by simp }
      moreover
      { assume c<0
        hence ?P (Atom a) ((r − ⟨cs,xs⟩ + 1)/c) using Eq Cons
          by(auto simp add: field-simps)
        hence ?thesis .. }
      moreover
      { assume c>0
        hence ?P (Atom a) ((r − ⟨cs,xs⟩ − 1)/c) using Eq Cons
          by(auto simp add: field-simps)
        hence ?thesis .. }
      ultimately show ?thesis by force
```

```
    qed
  qed
next
  case (And f1 f2)
  then obtain x1 x2 where ?P f1 x1 ?P f2 x2 by 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:
  nqfree f ⟹ ∃ x. ∀ y≥x. R.I (inf₊ f) xs = R.I f (y # xs)
  (is - ⟹ ∃ x. ?P f x)
proof(induct f)
  case (Atom a)
  show ?case
  proof (cases a)
    case (Less r cs)
    show ?thesis
    proof(cases cs)
      case Nil thus ?thesis using Less by simp
    next
      case (Cons c cs)
      { assume c=0 hence ?thesis using Less Cons by simp }
      moreover
      { assume c<0
        hence ?P (Atom a) ((r − ⟨cs,xs⟩ − 1)/c) using Less Cons
          by(auto simp add: field-simps)
        hence ?thesis .. }
      moreover
      { assume c>0
        hence ?P (Atom a) ((r − ⟨cs,xs⟩ + 1)/c) using Less Cons
          by(auto simp add: field-simps)
        hence ?thesis .. }
      ultimately show ?thesis by force
    qed
  next
    case (Eq r cs)
    show ?thesis
    proof(cases cs)
      case Nil thus ?thesis using Eq by simp
    next
      case (Cons c cs)
      { assume c=0 hence ?thesis using Eq Cons by simp }
      moreover
```

```
      { assume c<0
        hence ?P (Atom a) ((r − ⟨cs,xs⟩ − 1)/c) using Eq Cons
          by(auto simp add: field-simps)
        hence ?thesis .. }
      moreover
      { assume c>0
        hence ?P (Atom a) ((r − ⟨cs,xs⟩ + 1)/c) using Eq Cons
          by(auto simp add: field-simps)
        hence ?thesis .. }
      ultimately show ?thesis by force
    qed
  qed
next
  case (And f1 f2)
  then obtain x1 x2 where ?P f1 x1 ?P f2 x2 by 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 LBex:
 ⟦ nqfree f; R.I f (x#xs); ¬R.I (inf_ f) xs; x ∉ EQ f xs ⟧
  ⟹ ∃ l∈ LB f xs. l < x
apply(induct f)
apply simp
apply simp
apply(rename-tac a)
apply(case-tac a)
apply(auto simp add: depends_R-def field-simps split:if-splits list.splits)
apply fastforce+
done

lemma UBex:
 ⟦ nqfree f; R.I f (x#xs); ¬R.I (inf_+ f) xs; x ∉ EQ f xs ⟧
  ⟹ ∃ u∈ UB f xs. x < u
apply(induct f)
apply simp
apply simp
apply(rename-tac a)
apply(case-tac a)
apply(auto simp add: depends_R-def field-simps split:if-splits list.splits)
apply fastforce+
```

done

end

lemma *finite-LB*: *finite*(*LB f xs*)
proof −
  have *LB f xs* = (λ(*r*,*cs*). *r* + ⟨*cs*,*xs*⟩) ‘ *set*(*lbounds*(*R.atoms*$_0$ *f*))
    by (*force simp*:*set-lbounds image-def field-simps*)
  thus *?thesis* **by** *simp*
qed

lemma *finite-UB*: *finite*(*UB f xs*)
proof −
  have *UB f xs* = (λ(*r*,*cs*). *r* + ⟨*cs*,*xs*⟩) ‘ *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-FM*$_1$ :: *atom list* ⇒ *atom fm* **where**
qe-FM$_1$ *as* = *list-conj* [*Atom*(*combine p q*). *p*←*lbounds as*, *q*←*ubounds as*]

theorem *I-qe-FM*$_1$:
assumes *less*: ∀ *a* ∈ *set as*. *is-Less a* **and** *dep*: ∀ *a* ∈ *set as*. *depends*$_R$ *a*
shows *R.I* (*qe-FM*$_1$ *as*) *xs* = (∃ *x*. ∀ *a* ∈ *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 depends*$_R$*-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. I_R 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. I_R 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 I_R a ((Max ?lbs + 1) # xs) by (simp add: field-simps)
  qed
  hence ?R .. }
moreover
{ assume asm: ?lbs = {} ∧ ?ubs ≠ {}
  have ∀ a ∈ set as. I_R a ((Min ?ubs − 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) ∈ ?Us
      using asm 2 by fastforce
    moreover hence Min ?ubs ≤ (r − ⟨cs,xs⟩)/c
      using asm fins
      by(auto intro!: Min-le-iff[THEN iffD2])
        (force simp add:field-simps)
    ultimately show I_R a ((Min ?ubs − 1) # xs) by (simp add: field-simps)
  qed
  hence ?R .. }
moreover
```

{ assume *?lbs* = {} ∧ *?ubs* = {}
hence *?R* **using** *2 less* **by** *auto* (*rule*, *fast*)
}
ultimately show *?R* **by** *blast*
next
let *?Ls* = *set*(*lbounds as*) **let** *?Us* = *set*(*ubounds as*)
assume *?R*
then obtain *x* **where** *1*: ∀ *a*∈ *set as*. I_R *a* (*x*#*xs*) **..**
{ fix *r c cs s d ds*
assume *Less r* (*c*#*cs*) ∈ *set as* 0 < *c Less s* (*d*#*ds*) ∈ *set as d* < 0
hence *r* < *c*∗*x* + ⟨*cs*,*xs*⟩ *s* < *d*∗*x* + ⟨*ds*,*xs*⟩ *c* > 0 *d* < 0
using *1* **by** *auto*
hence (*r* − ⟨*cs*,*xs*⟩)/*c* < *x x* < (*s* − ⟨*ds*,*xs*⟩)/*d* **by**(*simp add*:*field-simps*)+
hence (*r* − ⟨*cs*,*xs*⟩)/*c* < (*s* − ⟨*ds*,*xs*⟩)/*d* **by** *arith*
}
thus *?L* **by** (*auto simp*: *qe-FM*$_1$*-def iprod-left-diff-distrib less field-simps set-lbounds set-ubounds*)
qed

corollary *I-qe-FM*$_1$*-pretty*:
∀ *a* ∈ *set as*. *is-Less a* ∧ *depends*$_R$ *a* ⟹ *R.is-dnf-qe qe-FM*$_1$ *as*
by(*metis I-qe-FM*$_1$)

fun *subst*$_0$ :: *atom* ⇒ *atom* ⇒ *atom* **where**
subst$_0$ (*Eq r* (*c*#*cs*)) *a* = (*case a of*
Less s (*d*#*ds*) ⇒ *Less* (*s* − (*r*∗*d*)/*c*) (*ds* − (*d*/*c*) ∗$_s$ *cs*)
| *Eq s* (*d*#*ds*) ⇒ *Eq* (*s* − (*r*∗*d*)/*c*) (*ds* − (*d*/*c*) ∗$_s$ *cs*))

lemma *subst*$_0$*-pretty*:
subst$_0$ (*Eq r* (*c*#*cs*)) (*Less s* (*d*#*ds*)) = *Less* (*s* − (*r*∗*d*)/*c*) (*ds* − (*d*/*c*) ∗$_s$ *cs*)
subst$_0$ (*Eq r* (*c*#*cs*)) (*Eq s* (*d*#*ds*)) = *Eq* (*s* − (*r*∗*d*)/*c*) (*ds* − (*d*/*c*) ∗$_s$ *cs*)
by *auto*

lemma *I-subst*$_0$: *depends*$_R$ *a* ⟹ *c* ≠ 0 ⟹
I_R (*subst*$_0$ (*Eq r* (*c*#*cs*)) *a*) *xs* = I_R *a* ((*r* − ⟨*cs*,*xs*⟩)/*c* # *xs*)
apply(*cases a*)
by (*auto simp add*: *depends*$_R$*-def iprod-left-diff-distrib algebra-simps diff-divide-distrib split*:*list.splits*)

interpretation R_e:
ATOM-EQ neg$_R$ (λ*a*. *True*) I_R *depends*$_R$ *decr*$_R$
(λ*Eq* - (*c*#-) ⇒ *c* ≠ 0 | - ⇒ *False*)
(λ*Eq r cs* ⇒ *r*=0 ∧ (∀ *c*∈ *set cs*. *c*=0) | - ⇒ *False*) *subst*$_0$
apply(*unfold-locales*)
apply(*simp del*:*subst*$_0$*.simps add*:*I-subst*$_0$ *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-FM* = R_e.*lift-dnfeq-qe qe-FM*$_1$

lemma *qfree-qe-FM*$_1$: *qfree* (*qe-FM*$_1$ *as*)
by(*auto simp*:*qe-FM*$_1$*-def intro*!:*qfree-list-conj*)

corollary *I-qe-FM*: *R.I* (*qe-FM* φ) *xs* = *R.I* φ *xs*
unfolding *qe-FM-def*
apply(*rule* R_e.*I-lift-dnfeq-qe*)
apply(*rule qfree-qe-FM*$_1$)
apply(*rule allI*)
apply(*rule I-qe-FM*$_1$)
prefer *2* **apply** *blast*
apply(*clarify*)
apply(*drule-tac x*=*a* **in** *bspec*) **apply** *simp*
apply(*simp add*: *depends*$_R$*-def split*:*atom.splits list.splits*)
done

theorem *qfree-qe-FM*: *qfree* (*qe-FM f*)
by(*simp add*:*qe-FM-def* R_e.*qfree-lift-dnfeq-qe qfree-qe-FM*$_1$)

4.2.1 Tests

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

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

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

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

end

theory *QElin-opt*

imports *QElin*
begin

4.2.2 An optimization

Atoms are simplified asap.

definition
asimp a = (*case a of*
Less r cs ⇒ (*if* ∀ *c*∈ *set cs. c* = *0*
then if r<0 then TrueF else FalseF
else Atom a) |
Eq r cs ⇒ (*if* ∀ *c*∈ *set cs. c* = *0*
then if r=0 then TrueF else FalseF else Atom a))

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

definition *qe-FMo*$_1$ :: *atom list* ⇒ *atom fm* **where**
qe-FMo$_1$ *as* = *list-conj* [*asimp*(*combine p q*). *p*←*lbounds as*, *q*←*ubounds as*]

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

lemma *I-qe-FMo*$_1$: *R.I* (*qe-FMo*$_1$ *as*) *xs* = *R.I* (*qe-FM*$_1$ *as*) *xs*
by(*simp add*:*qe-FM*$_1$*-def qe-FMo*$_1$*-def I-asimp*)

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

lemma *qfree-qe-FMo*$_1$: *qfree* (*qe-FMo*$_1$ *as*)
by(*auto simp*:*qe-FM*$_1$*-def qe-FMo*$_1$*-def asimp-def intro*!:*qfree-list-conj*
split:*atom.split*)

corollary *I-qe-FMo*: *R.I* (*qe-FMo* φ) *xs* = *R.I* φ *xs*
unfolding *qe-FMo-def*
apply(*rule* R_e*.I-lift-dnfeq-qe*)
apply(*rule qfree-qe-FMo*$_1$)
apply(*rule allI*)
apply(*subst I-qe-FMo*$_1$)
apply(*rule I-qe-FM*$_1$)
prefer *2* **apply** *blast*
apply(*clarify*)

apply(*drule-tac x=a* **in** *bspec*) **apply** *simp*
apply(*simp add*: $depends_R$*-def split*:*atom.splits list.splits*)
done

theorem *qfree-qe-FMo*: *qfree* (*qe-FMo f*)
by(*simp add*:*qe-FMo-def* R_e*.qfree-lift-dnfeq-qe qfree-qe-FMo*$_1$)

end

theory *FRE*
imports *LinArith*
begin

4.3 Ferrante-Rackoff

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

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

definition FR_1 :: *atom fm* $\Rightarrow$ *atom fm* **where**
$FR_1\ \varphi =$
(*let as* = $R.atoms_0\ \varphi$; *lbs* = *lbounds as*; *ubs* = *ubounds as*; *ebs* = *ebounds as*;
intrs = [*subst* φ (*between l u*) . *l* $\leftarrow$ *lbs*, *u* $\leftarrow$ *ubs*]
in list-disj ($inf_-\ \varphi$ # $inf_+\ \varphi$ # *intrs* @ *map* (*subst* φ) *ebs*))

lemma *dense-interval*:
assumes *finite L finite U l* $\in$ *L u* $\in$ *U l* $<$ *x x* $<$ *u P*(*x*::*real*)
and *dense*: $\bigwedge y\ l\ u.\ [\![\forall y{\in}\{l{<}..{<}x\}.\ y \notin L;\ \forall y{\in}\{x{<}..{<}u\}.\ y \notin U;$
$l{<}x; x{<}u;\ l{<}y; y{<}u\ ]\!] \Longrightarrow P\ y$
shows $\exists l{\in}L. \exists u{\in}U.\ l{<}u \wedge (\forall y.\ l{<}y \wedge y{<}u \longrightarrow P\ y)$
proof −
let *?L* = {*l*:*L*. *l* $<$ *x*} **let** *?U* = {*u*:*U*. *x* $<$ *u*}
let *?ll* = *Max ?L* **let** *?uu* = *Min ?U*
have *?L* $\neq$ {} **using** ‹*l* $\in$ *L*› ‹*l*$<$*x*› **by** (*blast intro*:*order-less-imp-le*)
moreover have *?U* $\neq$ {} **using** ‹*u*:*U*› ‹*x*$<$*u*› **by** (*blast intro*:*order-less-imp-le*)
ultimately have $\forall y.$ *?ll*$<$*y* $\wedge$ *y*$<$*x* $\longrightarrow$ *y* $\notin$ *L* $\forall y.$ *x*$<$*y* $\wedge$ *y*$<$*?uu* $\longrightarrow$ *y* $\notin$ *U*
using ‹*finite L*› ‹*finite U*› **by** *force*+
moreover have *?ll* $\in$ *L*
proof
show *?ll* $\in$ *?L* **using** ‹*finite L*› *Max-in*[*OF* - ‹*?L* $\neq$ {}›] **by** *simp*
show *?L* $\subseteq$ *L* **by** *blast*
qed
moreover have *?uu* $\in$ *U*
proof
show *?uu* $\in$ *?U* **using** ‹*finite U*› *Min-in*[*OF* - ‹*?U* $\neq$ {}›] **by** *simp*
show *?U* $\subseteq$ *U* **by** *blast*

```
  qed
  moreover have ?ll < x using ‹finite L› ‹?L ≠ {}› 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:depends_R-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:depends_R-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)
        have ?thesis
        proof (rule ccontr)
          assume ¬ R.I (Atom a) (y#xs)
          hence ?l ≥ y using Atom Less Cons ‹c>0›
            by (auto simp add: field-simps)
          hence ?l > l using ‹y>l› by simp
          with ‹?l<x› show False using Atom Less Cons ‹c>0›
            by (auto simp:depends_R-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:depends_R-def)
   next
    case (Cons c cs)
    hence r = c*x + ⟨cs,xs⟩ using Atom Eq by simp
    { assume c=0 hence ?thesis using Atom Eq Cons by simp }
    moreover
    { assume c≠0
     hence ?thesis using ‹r = c*x + ⟨cs,xs⟩› Atom Eq Cons ‹l<y› ‹y<u›
      by (auto simp: depends_R-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-FR_1:
assumes nqfree φ shows R.I (FR_1 φ) xs = (∃ x. R.I φ (x#xs))
  (is ?FR = ?EX)
proof
  assume ?FR
  { assume R.I (inf_- φ) xs
   hence ?EX using ‹?FR› min-inf[OF ‹nqfree φ›, where xs=xs]
    by(auto simp add:FR_1-def)
  } moreover
  { assume R.I (inf_+ φ) xs
   hence ?EX using ‹?FR› plus-inf[OF ‹nqfree φ›, where xs=xs]
    by(auto simp add:FR_1-def)
  } moreover
  { assume ∃ x ∈ EQ φ xs. R.I φ (x#xs)
   hence ?EX using ‹?FR› by(auto simp add:FR_1-def)
  } moreover
  { assume ¬R.I (inf_- φ) xs ∧ ¬R.I (inf_+ φ) xs ∧
          (∀ x∈EQ φ xs. ¬R.I φ (x#xs))
   with ‹?FR› obtain r cs s ds
    where R.I (subst φ (between (r,cs) (s,ds))) xs
    by(auto simp: FR_1-def eval-def
      diff-divide-distrib set-ebounds I-subst ‹nqfree φ›) blast
   hence R.I φ (eval (between (r,cs) (s,ds)) xs # xs)
```

```
    by(simp add:I-subst ‹nqfree φ›)
   hence ?EX .. }
  ultimately show ?EX by blast
next
  assume ?EX
  then obtain x where x: R.I φ (x#xs) ..
  { assume R.I (inf_- φ) xs ∨ R.I (inf_+ φ) xs
   hence ?FR by(auto simp:FR_1-def)
  } moreover
  { assume x ∈ EQ φ xs
   then obtain r cs
     where (r,cs) ∈ set(ebounds(R.atoms_0 φ)) ∧ x = r + ⟨cs,xs⟩
     by(force simp:set-ebounds field-simps)
   moreover hence R.I (subst φ (r,cs)) xs using x
     by(auto simp: I-subst ‹nqfree φ› eval-def)
   ultimately have ?FR by(force simp:FR_1-def) } moreover
  { assume ¬ R.I (inf_- φ) xs and ¬ R.I (inf_+ φ) xs and x ∉ EQ φ xs
   obtain l where l ∈ LB φ xs l < x
     using LBex[OF ‹nqfree φ› x ‹¬ R.I (inf_- φ) xs› ‹x ∉ EQ φ xs›] ..
   obtain u where u ∈ UB φ xs x < u
     using UBex[OF ‹nqfree φ› x ‹¬ R.I (inf_+ φ) xs› ‹x ∉ EQ φ xs›] ..
   have ∃ l∈LB φ xs. ∃ u∈UB φ xs. l<u ∧ (∀ y. l < y ∧ y < u ⟶ R.I φ (y#xs))
     using dense-interval[where P = λx. R.I φ (x#xs), OF finite-LB finite-UB
‹l:LB φ xs› ‹u:UB φ xs› ‹l<x› ‹x<u› x] x dense[OF ‹nqfree φ› - - - - ‹x ∉ EQ φ
xs›] by simp
   then obtain r c cs s d ds
     where Less r (c # cs) ∈ set (R.atoms_0 φ) Less s (d # ds) ∈ set (R.atoms_0
φ)
         ⋀y. (r - ⟨cs,xs⟩) / c < y ⟹ y < (s - ⟨ds,xs⟩) / d ⟹ R.I φ (y # xs)
       and *: c > 0 d < 0 (r - ⟨cs,xs⟩) / c < (s - ⟨ds,xs⟩) / d
     by blast
   moreover
     have (r - ⟨cs,xs⟩) / 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: FR_1-def bex-Un set-lbounds set-ubounds set-ebounds I-subst
```

‹*nqfree* φ›)
} **ultimately show** *?FR* **by** *blast*
qed

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

lemma $qfree\text{-}FR_1$: *nqfree* $\varphi \Longrightarrow$ *qfree* $(FR_1\ \varphi)$
apply(*simp add*:FR_1*-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*: $R.I\ (FR\ \varphi)\ xs = R.I\ \varphi\ xs$
by(*simp add*:$I\text{-}FR_1$ *FR-def R.I-lift-nnf-qe* $qfree\text{-}FR_1$)

theorem *qfree-FR*: *qfree* $(FR\ \varphi)$
by(*simp add*:*FR-def R.qfree-lift-nnf-qe* $qfree\text{-}FR_1$)

end

theory *QElin-inf*
imports *LinArith*
begin

4.4 Quantifier elimination with infinitesimals

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

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

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

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* ∧ *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* ∉ *EQ f xs*; *R.I f* (*x*#*xs*); *l* < *y*; *y* ≤ *x*⟧
⟹ *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* + ⟨*cs*,*xs*⟩ **using** *Atom* **by** *simp*
{ assume *c*=*0* **hence** *?thesis* **using** *Atom* **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* ‹*c*<*0*›
by (*auto simp add*: *field-simps*)
with ‹*x*<*?u*› **show** *False* **using** *Atom* ‹*c*<*0*›
by(*auto simp*:*depends*$_R$*-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*:*depends*$_R$*-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:depends_R-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: depends_R-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
 ⟹ ∀ y ∈ {r+⟨cs,xs⟩ <.. x}. R.I f (y#xs) ∧ y ∉ EQ2 f xs
 ⟹ R.I (subst_+ f (r,cs)) xs
proof(induct f)
  case FalseF thus ?case
    by simp (metis antisym-conv1 linorder-neq-iff)
next
  case (Atom a)
  show ?case
  proof(cases ((r,cs),a) rule:asubst-peps.cases)
    case (1 r cs s d ds)
    { assume d=0 hence ?thesis using Atom 1 by auto }
    moreover
    { assume d<0
      have s < d*x + ⟨ds,xs⟩ using Atom 1 by simp
```

```
    moreover have d*x < d*(r + ⟨cs,xs⟩) using ‹d<0› Atom 1
      by (simp add: mult-strict-left-mono-neg)
    ultimately have s < d * (r + ⟨cs,xs⟩) + ⟨ds,xs⟩ by(simp add:algebra-simps)
    hence ?thesis using 1
      by (auto simp add: iprod-left-add-distrib algebra-simps)
   } moreover
   { let ?L = (s - ⟨ds,xs⟩) / d let ?U = r + ⟨cs,xs⟩
    assume d>0
    hence ?U < x and ∀ y. ?U < y ∧ y < x ⟶ y ≠ ?L
      and ∀ y. ?U < y ∧ y ≤ x ⟶ ?L < y using Atom 1
      by(simp-all add:nolb-def depends_R-def Ball-def field-simps)
    hence ?L < ?U ∨ ?L = ?U
      by (metis linorder-neqE-linordered-idom order-refl)
    hence ?thesis using Atom 1 ‹d>0›
      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:
  nqfree f ⟹ R.I (subst_+ f (r,cs)) xs ⟹
  (∃ leps>r+⟨cs,xs⟩. ∀ x. r+⟨cs,xs⟩ < x ∧ x ≤ leps ⟶ 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: ⋀y l. ⟦ ∀ y∈{l<..<x}. y ∉ L; l<x; l<y; y≤x ⟧ ⟹ P y
shows ∃ l∈L. l<x ∧ (∀ y∈{l<..<x}. y ∉ L) ∧ (∀ y. l<y ∧ y≤x ⟶ P y)
proof −
  let ?L = {l∈L. l < x}
  let ?ll = Max ?L
  have ?L ≠ {} using ‹l ∈ L› ‹l<x› by (blast intro:order-less-imp-le)
  hence ∀ y. ?ll<y ∧ y<x ⟶ y ∉ L using ‹finite L› by force
  moreover have ?ll ∈ L
  proof
    show ?ll ∈ ?L using ‹finite L› Max-in[OF - ‹?L ≠ {}›] by simp
    show ?L ⊆ L by blast
  qed
  moreover have ?ll < x using ‹finite L› ‹?L ≠ {}› by simp
  ultimately show ?thesis using ‹l < x› ‹?L ≠ {}›
    by(blast intro!:dense greaterThanLessThan-iff[THEN iffD1])
qed
```

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

theorem *I-eps1*:
assumes *nqfree f* **shows** $R.I\ (qe\text{-}eps_1\ f)\ xs = (\exists\, x.\ R.I\ f\ (x\#xs))$
(**is** *?QE = ?EX*)
proof
let *?as* $= R.atoms_0\ f$ **let** *?ebs = ebounds ?as*
assume *?QE*
{ assume $R.I\ (inf_-\ f)\ xs$
hence *?EX* **using** ‹*?QE*› *min-inf*[*of f xs*] ‹*nqfree f*›
by(*auto simp add*:$qe\text{-}eps_1$*-def amap-fm-list-disj*)
} moreover
{ assume $\forall\, x \in EQ\ f\ xs.\ \neg R.I\ f\ (x\#xs)$
$\neg\ R.I\ (inf_-\ f)\ xs$
with ‹*?QE*› ‹*nqfree f*› **obtain** *r cs* **where** $R.I\ (subst_+\ f\ (r,cs))\ xs$
by (*fastforce simp*: $qe\text{-}eps_1$*-def set-ebounds diff-divide-distrib eval-def I-subst* ‹*nqfree f*›)
then obtain *leps* **where** $R.I\ f\ (leps\#xs)$
using *I-subst-peps*[*OF* ‹*nqfree f*›] **by** *fastforce*
hence *?EX* **.. }**
ultimately show *?EX* **by** *blast*
next
let *?as* $= R.atoms_0\ f$ **let** *?ebs = ebounds ?as*
assume *?EX*
then obtain *x* **where** *x*: $R.I\ f\ (x\#xs)$ **..**
{ assume $R.I\ (inf_-\ f)\ xs$
hence *?QE* **using** ‹*nqfree f*› **by**(*auto simp*:$qe\text{-}eps_1$*-def*)
} moreover
{ assume $\exists\, rcs \in set\ ?ebs.\ R.I\ (subst\ f\ rcs)\ xs$
hence *?QE* **by**(*auto simp*:$qe\text{-}eps_1$*-def*) **} moreover**
{ assume $\neg\ R.I\ (inf_-\ f)\ xs$
and $\forall\, rcs \in set\ ?ebs.\ \neg\ R.I\ (subst\ f\ rcs)\ xs$
hence *noE*: $\forall\, e \in EQ\ f\ xs.\ \neg\ R.I\ f\ (e\#xs)$ **using** ‹*nqfree f*›
by (*force simp*:*set-ebounds I-subst diff-divide-distrib eval-def split*:*if-split-asm*)
hence $x \notin EQ\ f\ xs$ **using** *x* **by** *fastforce*
obtain *l* **where** $l \in LB\ f\ xs\ l < x$
using *LBex*[*OF* ‹*nqfree f*› *x* ‹$\neg\ R.I(inf_-\ f)\ xs$› ‹$x \notin EQ\ f\ xs$›] **..**
have $\exists\, l{\in}LB\ f\ xs.\ l{<}x \wedge nolb\ f\ xs\ l\ x \wedge$
$(\forall\, y.\ l < y \wedge y \leq x \longrightarrow R.I\ f\ (y\#xs))$
using *dense-interval*[**where** $P = \lambda x.\ R.I\ f\ (x\#xs)$, *OF finite-LB* ‹$l{\in}LB\ f\ xs$› ‹$l{<}x$› *x*] *x innermost-intvl*[*OF* ‹*nqfree f*› - - ‹$x \notin EQ\ f\ xs$›]
by (*simp add*:*nolb-def*)
then obtain *r c cs*
where $*$: $Less\ r\ (c\#cs) \in set(R.atoms_0\ f) \wedge c{>}0 \wedge$
$(r - \langle cs,xs\rangle)/c < x \wedge nolb\ f\ xs\ ((r - \langle cs,xs\rangle)/c)\ x$

$\wedge$ ($\forall$ *y*. (*r* − ⟨*cs*,*xs*⟩)/*c* < *y* ∧ *y* ≤ *x* ⟶ *R.I f* (*y*#*xs*))
by *blast*
then have *R.I* (*subst*$_+$ *f* (*r*/*c*, (−*1*/*c*) ∗$_s$ *cs*)) *xs* **using** *noE*
by(*auto intro*!: *I-subst-peps2*[*OF* ‹*nqfree f*›]
simp:*EQ2-def diff-divide-distrib algebra-simps*)
with ∗ **have** *?QE*
by(*simp add*:*qe-eps*$_1$*-def bex-Un set-lbounds*) *metis*
} **ultimately show** *?QE* **by** *blast*
qed

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

lemma *qfree-subst-peps*: *nqfree* φ ⟹ *qfree* (*subst*$_+$ φ *rcs*)
by(*induct* φ) (*simp-all add*:*qfree-asubst-peps*)

lemma *qfree-qe-eps*$_1$: *nqfree* φ ⟹ *qfree*(*qe-eps*$_1$ φ)
apply(*simp add*:*qe-eps*$_1$*-def*)
apply(*rule qfree-list-disj*)
apply (*auto simp*:*qfree-min-inf qfree-subst-peps qfree-map-fm*)
done

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

lemma *qfree-qe-eps*: *qfree*(*qe-eps* φ)
by(*simp add*: *qe-eps-def R.qfree-lift-nnf-qe qfree-qe-eps*$_1$)

lemma *I-qe-eps*: *R.I* (*qe-eps* φ) *xs* = *R.I* φ *xs*
by(*simp add*:*qe-eps-def R.I-lift-nnf-qe qfree-qe-eps*$_1$ *I-eps1*)

end

5 Presburger arithmetic

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

declare *iprod-assoc*[*simp*]

5.1 Syntax

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

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

fun neg_Z :: $atom \Rightarrow atom\ fm$ **where**
$neg_Z\ (Le\ i\ ks) = Atom(Le\ (1-i)\ (-ks))$ |
$neg_Z\ (Dvd\ d\ i\ ks) = Atom(NDvd\ d\ i\ ks)$ |
$neg_Z\ (NDvd\ d\ i\ ks) = Atom(Dvd\ d\ i\ ks)$

fun *hd-coeff* :: $atom \Rightarrow int$ **where**
hd-coeff $(Le\ i\ ks) = (case\ ks\ of\ [] \Rightarrow 0 \mid k\#\text{-} \Rightarrow k)$ |
hd-coeff $(Dvd\ d\ i\ ks) = (case\ ks\ of\ [] \Rightarrow 0 \mid k\#\text{-} \Rightarrow k)$ |
hd-coeff $(NDvd\ d\ i\ ks) = (case\ ks\ of\ [] \Rightarrow 0 \mid k\#\text{-} \Rightarrow k)$

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

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

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

interpretation *Z*:
ATOM $neg_Z\ (\lambda a.\ divisor\ a \neq 0)\ I_Z\ (\lambda a.\ hd\text{-}coeff\ a \neq 0)\ decr_Z$
rewrites $ATOM.atoms_0\ (\lambda a.\ hd\text{-}coeff\ a \neq 0) = atoms_0$
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(*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*:$atoms_0$-*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*]: *divisor* (*asubst i ks a*) = *divisor a*
by(*induct i ks a rule*:*asubst.induct*) *auto*

definition *lbounds as* = [(*i*,*ks*). *Le i* (*k*#*ks*) ← *as*, *k*>*0*]
definition *ubounds as* = [(*i*,*ks*). *Le i* (*k*#*ks*) ← *as*, *k*<*0*]
lemma *set-lbounds*:
set(*lbounds as*) = {(*i*,*ks*)|*i k ks*. *Le i* (*k*#*ks*) ∈ *set as* ∧ *k*>*0*}
by(*auto simp*: *lbounds-def split*:*list.splits atom.splits if-splits*)
lemma *set-ubounds*:
set(*ubounds as*) = {(*i*,*ks*)|*i k ks*. *Le i* (*k*#*ks*) ∈ *set as* ∧ *k*<*0*}
by(*auto simp*: *ubounds-def split*:*list.splits atom.splits if-splits*)

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

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

5.2 LCM and lemmas

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

lemma *dvd-zlcms*: *i* ∈ *set is* ⟹ *i dvd zlcms is*
by(*induct is*) *auto*

lemma *zlcms-pos*: ∀ *i* ∈ *set is*. *i*≠*0* ⟹ *zlcms is* > *0*
by(*induct is*)(*auto simp*:*lcm-pos-int*)

lemma *zlcms0-iff*[*simp*]: (*zlcms is* = *0*) = (*0* ∈ *set is*)
by (*metis mod-by-0 dvd-eq-mod-eq-0 dvd-zlcms zlcms-pos less-le*)

lemma *elem-le-zlcms*: ∀ *i* ∈ *set is*. *i* ≠ *0* ⟹ *i* ∈ *set is* ⟹ *i* ≤ *zlcms is*
by (*metis dvd-zlcms zdvd-imp-le zlcms-pos*)

5.3 Setting coeffiencients to 1 or -1

fun *hd-coeff1* :: *int* ⇒ *atom* ⇒ *atom* **where**
hd-coeff1 m (*Le i* (*k*#*ks*)) =
  (*if k*=*0 then Le i* (*k*#*ks*)
   else let m′ = *m div* (*abs k*) *in Le* (*m*′∗*i*) (*sgn k* # (*m*′ ∗*s ks*))) |
hd-coeff1 m (*Dvd d i* (*k*#*ks*)) =
  (*if k*=*0 then Dvd d i* (*k*#*ks*)
   else let m′ = *m div k in Dvd* (*m*′∗*d*) (*m*′∗*i*) (*1* # (*m*′ ∗*s ks*))) |
hd-coeff1 m (*NDvd d i* (*k*#*ks*)) =
  (*if k*=*0 then NDvd d i* (*k*#*ks*)
   else let m′ = *m div k in NDvd* (*m*′∗*d*) (*m*′∗*i*) (*1* # (*m*′ ∗*s ks*))) |
hd-coeff1 - *a* = *a*

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

lemma ⟦ *k* ≠ *0*; *k dvd m* ⟧ ⟹
  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* $\Longrightarrow$ 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≠0*
with *Le* **have** |*k*| *dvd m* **by** *simp*
let *?m' = m div |k|*
have *?m' > 0* **using** ‹|*k*| *dvd m*› *pos-imp-zdiv-pos-iff* ‹*m>0*› ‹*k≠0*›
by(*simp add*:*zdvd-imp-le*)
have *1*: *k*(*x*?m'*) = *sgn k * x * m*
proof −
have *k*(*x*?m'*) = (*sgn k * abs k*) * (*x * ?m'*)
by(*simp only*: *mult-sgn-abs*)
also have ... = *sgn k * x * (abs k * ?m'*) **by** *simp*
also have ... = *sgn k * 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* (*Le i ks*)) (*m*x#xs*) $\longleftrightarrow$
(*i*?m' ≤ sgn k * m*x + ?m' * ⟨*ks'*,*xs*⟩)
using ‹*k≠0*› **by**(*simp add*: *algebra-simps*)
also have ... $\longleftrightarrow$ *?m'*i* ≤ *?m'* * (*k*x* + ⟨*ks'*,*xs*⟩) **using** *1*
by(*simp* (*no-asm-simp*) *add*:*algebra-simps*)
also have ... $\longleftrightarrow$ *i* ≤ *k*x* + ⟨*ks'*,*xs*⟩ **using** ‹*?m'>0*›
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 zdvd-imp-le)
      have 1: k*(x*?m') = x * m
      proof −
        have k*(x*?m') = x*(k*?m') by(simp add:algebra-simps)
        also have ... = x*m using 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 zdvd-imp-le)
      have 1: k*(x*?m') = x * m
      proof −
        have k*(x*?m') = x*(k*?m') by(simp add:algebra-simps)
        also have ... = x*m using 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* $\varphi \Longrightarrow \forall\ a \in set(Z.atoms_0\ \varphi).$ *hd-coeff a dvd m* $\Longrightarrow$
Z.I $(map_{fm}$ *(hd-coeff1 m)* $\varphi)$ *(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*: $\forall$ *a∈set as. hd-coeff a* $\neq$ *0* **shows**
($\exists$ *x.* $\forall$ *a* $\in$ *set*(*hd-coeffs1 as*). I_Z *a* (*x#xs*)) =
($\exists$ *x.* $\forall$ *a* $\in$ *set as.* I_Z *a* (*x#xs*)) (**is** *?B = ?A*)
proof −
let *?m = zlcms*(*map hd-coeff as*)
have *?m>0* **using** *0* **by**(*simp add*:*zlcms-pos*)
have *?A* = ($\exists$ *x.* $\forall$ *a* $\in$ *set as.* I_Z (*hd-coeff1 ?m a*) (*?m∗x#xs*))
by (*simp add*:*I-hd-coeff1-mult-a*[*OF* ‹*?m>0*›] *dvd-zlcms 0*)
also have ... = ($\exists$ *x. ?m dvd x+0* $\wedge$ ($\forall$ *a* $\in$ *set as.* I_Z (*hd-coeff1 ?m a*) (*x#xs*)))
by(*rule unity-coeff-ex*[*THEN meta-eq-to-obj-eq*])
finally show *?thesis* **by**(*simp add*:*hd-coeffs1-def*)
qed

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

definition
$qe\text{-}pres_1$ *as =*
(*let ds = filter is-dvd as*; (*d*::*int*) = *zlcms*(*map divisor ds*); *ls = lbounds as*
in if ls = []

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

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

```
lemma I-cyclic:
assumes is-dvd a and hd-coeff a = 1 and i mod divisor a = j mod divisor a
shows I_Z a (i#e) = I_Z a (j#e)
proof (cases a)
  case (Dvd d l ks)
  with ‹hd-coeff a = 1› obtain ks' where [simp]: ks = 1#ks'
    by(simp split:list.splits)
  have (l + (i + ⟨ks',e⟩)) mod d = (l + (j + ⟨ks',e⟩)) 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› Dvd 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 Dvd by (simp add:dvd-eq-mod-eq-0)
next
  case (NDvd d l ks)
  with ‹hd-coeff a = 1› obtain ks' where [simp]: ks = 1#ks'
    by(simp split:list.splits)
  have (l + (i + ⟨ks',e⟩)) mod d = (l + (j + ⟨ks',e⟩)) 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-pres_1:
assumes norm: ∀ a ∈ set as. divisor a ≠ 0
and hd: ∀ a ∈ set as. hd-coeff-is1 a
shows Z.I (qe-pres_1 as) xs = (∃ x. ∀ a∈ set as. I_Z a (x#xs))
proof −
  let ?lbs = lbounds as
  let ?ds = filter is-dvd as
  let ?lcm = zlcms(map divisor ?ds)
  let ?Ds = {a ∈ set as. is-dvd a}
  let ?Us = {a ∈ set as. case a of Le - (k#-) ⇒ k < 0 | - ⇒ False}
  let ?Ls = {a ∈ set as. case a of Le - (k#-) ⇒ k > 0 | - ⇒ False}
  have as: set as = ?Ds ∪ ?Ls ∪ ?Us (is - = ?S)
  proof −
    { fix x assume x ∈ set as
      hence x ∈ ?S using hd by (cases x 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. I_Z a (n#xs) using 1
        by(auto simp:IZ-asubst qe-pres_1-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 ‹?Ls = {}› have set as = ?Ds using as by(simp (no-asm-use))blast
        hence ?P n using A by auto
        thus ?thesis ..
      next
        assume ?Us ≠ {}
        let ?M = {⟨tl ks, xs⟩ − i|ks i. Le i ks ∈ ?Us} let ?m = Min ?M
        have finite ?M
        proof −
          have finite ( (λLe i ks ⇒ ⟨tl ks, xs⟩ − i) ‘
                        {a∈set as. ∃ i k ks. k<0 ∧ a = Le i (k#ks)} )
            (is finite ?B)
```

```
      by simp
    also have ?B = ?M using hd
      by(fastforce simp:image-def neq-Nil-conv split:atom.splits list.splits)
    finally show ?thesis by auto
  qed
  have ?M ≠ {}
  proof −
    from ‹?Us ≠ {}› obtain i k ks where Le i (k#ks) ∈ ?Us ∧ k<0
      by (fastforce split:atom.splits list.splits)
    thus ?thesis by auto
  qed
  let ?k = (n − ?m) div ?lcm + 1 let ?x = n − ?k ∗ ?lcm
  have ∀ a ∈ ?Ds. I_Z a (?x # xs)
  proof (intro allI ballI)
    fix a assume a ∈ ?Ds
    let ?d = divisor a
    have 2: ?d dvd ?lcm using ‹a ∈ ?Ds› by(simp add:dvd-zlcms)
    have ?x mod ?d = n mod ?d (is ?l = ?r)
    proof −
      have ?l = (?r − ((?k ∗ ?lcm) mod ?d)) mod ?d
        by (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] ‹a ∈ ?Ds› 1 by auto
  qed
  moreover
  have ∀ a∈ ?Us. I_Z a (?x#xs)
  proof
    fix a assume a ∈ ?Us
    then obtain l ks where [simp]: a = Le l (−1#ks) using hd
      by(fastforce split:atom.splits list.splits)
    have ?m ≤ ⟨ks,xs⟩ − l
      using Min-le-iff[OF ‹finite ?M› ‹?M ≠ {}›] ‹a ∈ ?Us› 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 ∪ ?Us using as ‹?Ls = {}›
    by (simp (no-asm-use)) blast
  ultimately have ?P(?x) by auto
  thus ?thesis ..
 qed }
moreover
{ assume ?lbs ≠ []
 with ‹?QE› obtain il ksl m
```

```
    where ∀ a∈set as. I_Z (asubst (il + m) ksl a) xs
    by(auto simp:qe-pres_1-def)
  hence ?P(il + m + ⟨ksl,xs⟩) by(simp add:IZ-asubst)
  hence ∃ x. ?P x .. }
 ultimately show ∃ x. ?P x by blast
next
 assume ∃ x. ?P x then obtain x where x: ?P x ..
 show ?QE
 proof cases
  assume ?lbs = []
  moreover
  have ∃ x. 0 ≤ x ∧ x < ?lcm ∧ (∀ a ∈ ?Ds. I_Z a (x # xs))
   (is ∃ x. ?P x)
  proof
   { fix a assume a ∈ ?Ds
    hence I_Z a ((x mod ?lcm) # xs) = I_Z a (x # xs) using 1
     by (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-pres_1-def IZ-asubst)
 next
  assume ?lbs ≠ []
  let ?L = {i − ⟨ks,xs⟩ |ks i. (i,ks) ∈ set(lbounds as)}
  let ?lm = Max ?L
  let ?n = (x − ?lm) mod ?lcm
  have finite ?L
  proof −
   have finite((λ(i,ks). i−⟨ks,xs⟩) ‘ set(lbounds as) ) (is finite ?B)
    by simp
   also have ?B = ?L by auto
   finally show ?thesis by auto
  qed
  moreover have ?L ≠ {} using ‹?lbs ≠ []›
   by(fastforce simp:neq-Nil-conv)
  ultimately have ?lm ∈ ?L by(rule Max-in)
  then obtain li lks where (li,lks)∈ set ?lbs and lm: ?lm = li−⟨lks,xs⟩
   by blast
  moreover
  have n: 0 ≤ ?n ∧ ?n < ?lcm using norm by(simp add:zlcms-pos)
  moreover
  { fix a assume a ∈ set as
   with x have I_Z a (x # xs) by blast
   have I_Z a ((li + ?n − ⟨lks,xs⟩) # xs)
   proof −
    { assume a ∈ ?Ls
     then obtain i ks where [simp]: a = Le i (1#ks) using hd
      by(fastforce split:atom.splits list.splits)
     from ‹a ∈ ?Ls› have i−⟨ks,xs⟩ ∈ ?L by(fastforce simp:set-lbounds)
```

```
        hence i−⟨ks,xs⟩ ≤ li − ⟨lks,xs⟩
          using lm[symmetric] ‹finite ?L› ‹?L ≠ {}› by auto
        hence ?thesis using n by simp }
      moreover
      { assume a ∈ ?Us
        then obtain i ks where [simp]: a = Le i (−1#ks) using hd
          by(fastforce split:atom.splits list.splits)
        have Le li (1#lks) ∈ set as using ‹(li,lks) ∈ set ?lbs› hd
          by(auto simp:set-lbounds)
        hence li − ⟨lks,xs⟩ ≤ x using x by auto
        hence (x − ?lm) mod ?lcm ≤ x − ?lm
          using lm by(simp add: zmod-le-nonneg-dividend)
        hence ?thesis using ‹I_Z a (x # xs)› lm by auto }
      moreover
      { assume a ∈ ?Ds
        have ?thesis
        proof(rule I-cyclic[THEN iffD2, OF - - - ‹I_Z a (x # xs)›])
          show is-dvd a using ‹a ∈ ?Ds› by simp
          show hd-coeff a = 1 using ‹a ∈ ?Ds› hd
            by(fastforce split:atom.splits list.splits)
          have li + (x−?lm) mod ?lcm − ⟨lks,xs⟩ = ?lm + (x−?lm) mod ?lcm
            using lm by arith
          hence (li + (x−?lm) mod ?lcm − ⟨lks,xs⟩) mod divisor a =
                (?lm + (x−?lm) mod ?lcm) mod divisor a by (simp only:)
          also have ... =
    (?lm mod divisor a + (x−?lm) mod ?lcm mod divisor a) mod divisor a
            by (simp add: mod-add-eq)
          also have
    ... = (?lm mod divisor a + (x−?lm) mod divisor a) mod divisor a
            using ‹is-dvd a› ‹a∈ set as›
            by(simp add: mod-mod-cancel dvd-zlcms)
          also have ... = (?lm + (x−?lm)) mod divisor a
            by(rule mod-add-eq)
          also have ... = x mod divisor a by simp
          finally
          show (li + ?n − ⟨lks,xs⟩) mod divisor a = x mod divisor a
            using norm by(auto simp: 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-pres_1-def IZ-asubst split-def)
       (force simp del:int-nat-eq)
 qed
qed
qed

lemma divisors-hd-coeffs1:
```

assumes *div0*: $\forall$ *a*∈*set as*. *divisor a* ≠ *0* **and** *hd0*: $\forall$ *a*∈*set as*. *hd-coeff a* ≠ *0*
and *a*: *a*∈*set* (*hd-coeffs1 as*) **shows** *divisor a* ≠ *0*
proof −
let *?m* = *zlcms*(*map hd-coeff as*)
from *a* **have** *a* = *Dvd ?m 0* [*1*] ∨ (∃ *b* ∈ *set as*. *a* = *hd-coeff1 ?m b*)
(**is** *?A* ∨ *?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* ∈ *set as* **and** [*simp*]: *a* = *hd-coeff1 ?m b* **..**
hence *b*: *hd-coeff b* ≠ *0 divisor b* ≠ *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* ∈ *set*(*map hd-coeff as*) **using** ‹*b* ∈ *set as*› **by** *force*
have *zlcms* (*map hd-coeff as*) *div k* ≠ *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* ∈ *set*(*map hd-coeff as*) **using** ‹*b* ∈ *set as*› **by** *force*
have *zlcms* (*map hd-coeff as*) *div k* ≠ *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*∈*set as*. *hd-coeff a* ≠ *0*
and *a*: *a*∈*set* (*hd-coeffs1 as*) **shows** *hd-coeff-is1 a*
proof −
let *?m* = *zlcms*(*map hd-coeff as*)
from *a* **have** *a* = *Dvd ?m 0* [*1*] ∨ (∃ *b* ∈ *set as*. *a* = *hd-coeff1 ?m b*)
(**is** *?A* ∨ *?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* ∈ *set as* **and** [*simp*]: *a* = *hd-coeff1 ?m b* **..**
hence *b*: *hd-coeff b* ≠ *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*$_1$*-o*:
⟦ ∀ *a* ∈ *set as. divisor a* ≠ *0*; ∀ *a*∈*set as. hd-coeff a* ≠ *0* ⟧ ⟹
Z.I ((*qe-pres*$_1$ ∘ *hd-coeffs1*) *as*) *e* = (∃ *x*. ∀ *a*∈ *set as. I*$_Z$ *a* (*x*#*e*))
apply(*simp*)
apply(*subst I-qe-pres*$_1$)
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*$_1$ ∘ *hd-coeffs1*)

lemma *qfree-qe-pres-o*: *qfree* ((*qe-pres*$_1$ ∘ *hd-coeffs1*) *as*)
by(*auto simp*:*qe-pres*$_1$*-def intro*!:*qfree-list-disj*)

lemma *normal-qe-pres*$_1$*-o*:
∀ *a* ∈ *set as. hd-coeff a* ≠ *0* ∧ *divisor a* ≠ *0* ⟹
Z.normal ((*qe-pres*$_1$ ∘ *hd-coeffs1*) *as*)
supply *image-cong-simp* [*cong del*]
apply(*auto simp*:*qe-pres*$_1$*-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* ∈ *set*(*map hd-coeff as*))
prefer *2* **apply** *force*
apply(*subgoal-tac* ∀ *i*∈ *set*(*map hd-coeff as*). *i* ≠ *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* $\varphi \Longrightarrow$ *Z.I* (*qe-pres* φ) *xs* = *Z.I* φ *xs*
by(*simp add*:*qe-pres-def Z.I-lift-dnf-qe-anormal I-qe-pres*$_1$*-o qfree-qe-pres-o normal-qe-pres*$_1$*-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 $\varphi \Longrightarrow a \in set(Z.atoms_0\ \varphi) \longleftrightarrow a \in atoms\ \varphi \wedge hd\text{-}coeff\ a \neq 0$
by(*induct* φ) (*auto split*:*if-split-asm*)

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

lemma *I-hd-coeffs1*:
assumes *qfree* φ
shows ($\exists x.\ Z.I$ (*hd-coeffs1* φ) (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* φ›])
have *?L* = ($\exists x.$ *?l dvd x*+*0* $\wedge$ $Z.I$ (map_{fm} (*hd-coeff1 ?l*) φ) (x#*xs*))
by(*simp add*:*hd-coeffs1-def*)
also have ... = ($\exists x.\ Z.I$ (map_{fm} (*hd-coeff1 ?l*) φ) (*?l*∗x#*xs*))
by(*rule unity-coeff-ex*[*THEN meta-eq-to-obj-eq*,*symmetric*])
also have ... = *?R*
by(*simp add*: *I-hd-coeff1-mult*[*OF* ‹*?l*>*0*› ‹*qfree* φ›] *dvd-zlcms*)
finally show *?thesis* **.**
qed

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

definition
qe-cooper$_1$ φ =
(*let as* = $Z.atoms_0\ \varphi$; d = *zlcms*(*map divisor as*); *ls* = *lbounds as*
in or (*Disj* [*0*..d − *1*] (λn. *subst n* [] (*inf*_ φ)))
(*Disj ls* (λ(*i*,*ks*).
Disj [*0*..d − *1*] (λn. *subst* (i + n) (−*ks*) φ))))

lemma *min-inf*:
nqfree f $\Longrightarrow$ $\forall$ *a*$\in$*set*(*Z.atoms*$_0$ *f*). *hd-coeff-is1 a*
$\Longrightarrow$ $\exists$ *x*.$\forall$ *y*<*x*. *Z.I* (*inf*$_-$ *f*) (*y* # *xs*) = *Z.I f* (*y* # *xs*)
(**is** - $\Longrightarrow$ - $\Longrightarrow$ $\exists$ *x*. *?P f x*)
proof(*induct f rule*: *min-inf.induct*)
case (*3 i k ks*)
{ assume *k=0* **hence** *?case* **using** *3* **by** *simp* **}**
moreover
{ assume *k*= −*1*
hence *?P* (*Atom*(*Le i* (*k*#*ks*))) (−*i* + ⟨*ks*,*xs*⟩ − *1*) **using** *3* **by** *auto*
hence *?case* **.. }**
moreover
{ assume *k=1*
hence *?P* (*Atom*(*Le i* (*k*#*ks*))) (*i* − ⟨*ks*,*xs*⟩ − *1*) **using** *3* **by** *auto*
hence *?case* **.. }**
ultimately show *?case* **using** *3* **by** *auto*
next
case (*1 f1 f2*)
then obtain *x1 x2* **where** *?P f1 x1 ?P f2 x2* **by** *fastforce+*
hence *?P* (*And f1 f2*) (*min x1 x2*) **by** *simp*
thus *?case* **..**
next
case (*2 f1 f2*)
then obtain *x1 x2* **where** *?P f1 x1 ?P f2 x2* **by** *fastforce+*
hence *?P* (*Or f1 f2*) (*min x1 x2*) **by** *simp*
thus *?case* **..**
qed *simp-all*

lemma *min-inf-repeats*:
nqfree φ $\Longrightarrow$ $\forall$ *a*$\in$*set*(*Z.atoms*$_0$ φ). *divisor a dvd d* $\Longrightarrow$
Z.I (*inf*$_-$ φ) ((*x* − *k*∗*d*)#*xs*) = *Z.I* (*inf*$_-$ φ) (*x*#*xs*)
proof(*induct* φ *rule*:*min-inf.induct*)
case (*4-4 da i ks*)
show *?case*
proof (*cases ks*)
case *Nil* **thus** *?thesis* **by** *simp*
next
case (*Cons j js*)
show *?thesis*
proof *cases*
assume *j=0* **thus** *?thesis* **using** *Cons* **by** *simp*
next
assume *j*≠*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-distribs)
    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
      then show ?thesis using Cons by (simp add:ring-distribs)
    qed
  qed
qed simp-all


lemma atoms-subset: qfree f ⟹ set(Z.atoms₀(f::atom fm)) ≤ atoms f
by (induct f) auto


lemma β:
  ⟦ nqfree φ; ∀ a∈set(Z.atoms₀ φ). hd-coeff-is1 a;
    ∀ a∈set(Z.atoms₀ φ). divisor a dvd d; d > 0;
```

```
    ¬(∃ j∈{0 .. d − 1}. ∃ (i,ks) ∈ set(lbounds(Z.atoms₀ φ)).
        x = i − ⟨ks,xs⟩ + j); Z.I φ (x#xs) ⟧
  ⟹ Z.I φ ((x−d)#xs)
proof(induct φ)
  case (Atom a)
  show ?case
  proof (cases a)
    case (Le i js)
    show ?thesis
    proof (cases js)
      case Nil thus ?thesis using Le Atom by simp
    next
      case (Cons k ks) thus ?thesis using Le Atom
        by (auto simp:lbounds-def Ball-def split: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≠0
        hence m dvd d using Cons Dvd Atom by auto
        have m dvd i + (x + ⟨ks,xs⟩) ⟹ m dvd i + (x − d + ⟨ks,xs⟩)
          (is ?L ⟹ -)
        proof −
          assume ?L
          hence m dvd i + (x + ⟨ks,xs⟩) − d
            by (metis ‹m dvd d› 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 ⋀k. 0≤k ⟹ ∀ x. P x ⟶ 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 ≤ (|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* φ
and *u*: $\forall$ *a*$\in$*set*(*Z.atoms*$_0$ φ). *hd-coeff-is1 a*
and *d*: $\forall$ *a*$\in$*set*(*Z.atoms*$_0$ φ). *divisor a dvd d*
and *dp*: *d* > *0*
shows ($\exists$ *x*. *Z.I* φ (*x*#*xs*)) =
($\exists$ *j*$\in${*0*..*d* − *1*}. *Z.I* (*inf*$_-$ φ) (*j*#*xs*) ∨
($\exists$ (*i*,*ks*) ∈ *set*(*lbounds*(*Z.atoms*$_0$ φ)). *Z.I* φ ((*i* − ⟨*ks*,*xs*⟩ + *j*) # *xs*)))
(**is** ($\exists$ *x*. *?P* (*x*)) = ($\exists$ *j*∈ *?D*. *?M j* ∨ ($\exists$ (*i*,*ks*)∈ *?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* |*i ks*. (*i*,*ks*) ∈ *?B*}
have *BB*′: ($\exists$ *j*∈*?D*. $\exists$ (*i*,*ks*)∈ *?B*. *?P* (*?I i ks* + *j*)) = ($\exists$ *j* ∈ *?D*. $\exists$ *b* ∈ *?B*′. *?P* (*b* + *j*)) **by** *auto*
hence *th2*: $\forall$ *x*. ¬ ($\exists$ *j* ∈ *?D*. $\exists$ *b* ∈ *?B*′. *?P* ((*b* + *j*))) ⟶ *?P* (*x*) ⟶ *?P* ((*x* − *d*))
using *β*[*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* φ ⟹ *qfree* (*inf*$_-$ φ)
by (*induct* φ *rule*:*min-inf.induct*) *simp-all*

lemma *I-qe-cooper*$_1$:
assumes *norm*: $\forall$ *a*$\in$*atoms* φ. *divisor a* ≠ *0*
and *hd*: $\forall$ *a*$\in$*set*(*Z.atoms*$_0$ φ). *hd-coeff-is1 a* **and** *nqfree* φ
shows *Z.I* (*qe-cooper*$_1$ φ) *xs* = ($\exists$ *x*. *Z.I* φ (*x*#*xs*))
proof −
let *?as* = *Z.atoms*$_0$ φ
let *?d* = *zlcms*(*map divisor ?as*)
have *?d* > *0* **using** *norm atoms-subset*[*of* φ] ‹*nqfree* φ›
by(*fastforce intro*:*zlcms-pos*)
have *alld*: $\forall$ *a*$\in$*set*(*Z.atoms*$_0$ φ). *divisor a dvd ?d* **by**(*simp add*:*dvd-zlcms*)
from *cp-thm*[*OF* ‹*nqfree* φ› *hd alld* ‹*?d*>*0*›]
show *?thesis* **using** ‹*nqfree* φ›
by (*simp add*:*qe-cooper*$_1$-*def I-subst*[*symmetric*] *split-def algebra-simps*) *blast*
qed

lemma *divisor-hd-coeff1-neq0*:

qfree $\varphi \Longrightarrow a \in atoms\ \varphi \Longrightarrow divisor\ a \neq 0 \Longrightarrow$
divisor (*hd-coeff1* (*zlcms* (*map hd-coeff* ($Z.atoms_0$ φ))) *a*) $\neq$ *0*
apply (*case-tac a*)

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

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

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

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

lemma *I-cooper1-hd-coeffs1*: *Z.normal* $\varphi \Longrightarrow$ *nqfree* φ
$\Longrightarrow$ *Z.I* (*qe-cooper*$_1$(*hd-coeffs1* φ)) *xs* = ($\exists\, x.$ *Z.I* φ (*x* # *xs*))
apply(*simp add*:*Z.normal-def*)
apply(*subst I-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*$_1$ $\circ$ *hd-coeffs1*)

lemma *qfree-cooper1-hd-coeffs1*: *qfree* $\varphi \Longrightarrow$ *qfree* (*qe-cooper*$_1$ (*hd-coeffs1* φ))
by(*auto simp*:*qe-cooper*$_1$*-def hd-coeffs1-def qfree-map-fm*
intro!: *qfree-or qfree-and qfree-list-disj qfree-min-inf*)

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

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

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

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

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

end

References

[1] D.C. Cooper. Theorem proving in arithmetic without multiplication. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 7, pages 91–100. Edinburgh University Press, 1972.

[2] Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM J. Computing*, 4:69–76, 1975.

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

[4] Tobias Nipkow. Linear quantifier elimination. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning (IJCAR 2008)*, volume 5195 of *LNCS*, pages 18–33. Springer, 2008. www.in.tum.de/~nipkow/pubs/ijcar08.html.

[5] Tobias Nipkow. Reflecting quantifier elimination for linear arithmetic. In O. Grumberg, T. Nipkow, and C. Pfaller, editors, *Formal Logical Methods for System Security and Correctness*. IOS Press, 2008. Proc. Marktoberdorf Summer School 2007, to appear.