Quantifier Elimination for Linear Arithmetic

Tobias Nipkow

August 28, 2014

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 9
2.1.2 NNF-based 12
2.2 With equality 13

3 DLO 15
3.1 Basics 15
3.2 DNF-based quantifier elimination 20
3.3 Examples 23
3.4 Interior Point Method 25
3.5 Quantifier elimination with infinitesimals 29
1 Logic

theory Logic
imports Main 
/

/* src */
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 'a fm =
  TrueF | FalseF | Atom 'a | And 'a fm 'a fm | Or 'a fm 'a fm |
  Neg 'a fm | ExQ 'a fm

abbreviation Imp where Imp ϕ1 ϕ2 ≡ Or (Neg ϕ1) ϕ2
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 ϕ1 ϕ2 =
  (if ϕ1=TrueF then ϕ2 else if ϕ2=TrueF then ϕ1 else
  if ϕ1=FalseF ∨ ϕ2=TrueF then FalseF else And ϕ1 ϕ2)

definition or :: 'a fm ⇒ 'a fm ⇒ 'a fm where
  or ϕ1 ϕ2 =
  (if ϕ1=FalseF then ϕ2 else if ϕ2=FalseF then ϕ1 else
  if ϕ1=TrueF ∨ ϕ2=TrueF then TrueF else Or ϕ1 ϕ2)

2
**definition** list-conj :: 'a fm list ⇒ 'a fm where
list-conj fs = foldr and fs TrueF

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

**abbreviation** Disj is f ≡ list-disj (map f is)

**fun** atoms :: 'a fm ⇒ 'a set where
atoms TrueF = { } |
atoms FalseF = { } |
atoms (Atom a) = { a } |
atoms (And φ₁ φ₂) = atoms φ₁ ∪ atoms φ₂ |
atoms (Or φ₁ φ₂) = atoms φ₁ ∪ atoms φ₂ |
atoms (Neg φ) = atoms φ |
atoms (ExQ φ) = atoms φ

**fun** map-fm :: (′a ⇒ ′b fm) ⇒ 'a fm ⇒ 'b fm (map-fm) where
map-fm h TrueF = TrueF |
map-fm h FalseF = FalseF |
map-fm h (Atom a) = Atom(h a) |
map-fm h (And φ₁ φ₂) = And (map-fm h φ₁) (map-fm h φ₂) |
map-fm h (Or φ₁ φ₂) = Or (map-fm h φ₁) (map-fm h φ₂) |
map-fm h (Neg φ) = Neg (map-fm h φ) |
map-fm h (ExQ φ) = ExQ (map-fm h φ)

**lemma** atoms-map-fm[simp]: atoms(map-fm f φ) = f ' atoms φ
by(induct φ) auto

**fun** amap-fm :: ('a ⇒ 'b fm) ⇒ 'a fm ⇒ 'b fm (amap-fm) where
amap-fm h TrueF = TrueF |
amap-fm h FalseF = FalseF |
amap-fm h (Atom a) = h a |
amap-fm h (And φ₁ φ₂) = and (amap-fm h φ₁) (amap-fm h φ₂) |
amap-fm h (Or φ₁ φ₂) = or (amap-fm h φ₁) (amap-fm h φ₂) |
amap-fm h (Neg φ) = neg (amap-fm h φ) |
amap-fm h (ExQ φ) = ExQ (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 ⇒ bool where
qfree(ExQ f) = False |
qfree(And φ₁ φ₂) = (qfree φ₁ ∧ qfree φ₂) |
qfree(Or φ₁ φ₂) = (qfree φ₁ ∧ qfree φ₂) |
qfree(Neg φ) = (qfree φ) |
qfree φ = True
lemma qfree-and[simp]: \[ qfree \varphi_1; qfree \varphi_2 \] \implies qfree(and \varphi_1 \varphi_2)  
by(simp add:and-def)

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

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

lemma qfree-foldr-Or[simp]:  
qfree(foldr Or fs \varphi) = (qfree \varphi \land (\forall \varphi \in \text{set fs. qfree} \varphi))  
by(induct fs) auto

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

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

lemma qfree-map-fm: qfree (mapfm f \varphi) = qfree \varphi  
by (induct \varphi) simp-all

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

lemma atoms-list-conjE:  
a : atoms(list-conj fs) \implies a : (\bigcup \varphi \in \text{set fs. atoms} \varphi)  
apply(induct fs)  
apply (simp add:list-conj-def)  
apply (auto simp add:list-conj-def Logic.and-def split:split-if asm)  
done
fun dnf :: 'a fm ⇒ 'a list list where
dnf TrueF = [[]] |
dnf FalseF = [[]] |
dnf (Atom ϕ) = [[ϕ]] |
dnf (And ϕ₁ ϕ₂) = [d₁ @ d₂. d₁ ← dnf ϕ₁, d₂ ← dnf ϕ₂] |
dnf (Or ϕ₁ ϕ₂) = dnf ϕ₁ @ dnf ϕ₂

fun nqfree :: 'a fm ⇒ bool where
nqfree (Atom a) = True |
nqfree TrueF = True |
nqfree FalseF = True |
nqfree (And ϕ₁ ϕ₂) = (nqfree ϕ₁ ∧ nqfree ϕ₂) |
nqfree (Or ϕ₁ ϕ₂) = (nqfree ϕ₁ ∧ nqfree ϕ₂) |
nqfree ϕ = False

lemma nqfree-qfree|simp|: nqfree ϕ ⇒ 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 ϕ₁ ϕ₂) xs = (interpret h ϕ₁ xs ∧ interpret h ϕ₂ xs) |
interpret h (Or ϕ₁ ϕ₂) xs = (interpret h ϕ₁ xs | interpret h ϕ₂ 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 I₀ :: 'a ⇒ 'b list ⇒ bool
assumes I₀-aneg: interpret I₀ (aneg a) xs = (¬ I₀ a a xs)

fixes depends₀ :: 'a ⇒ bool
and decr :: 'a ⇒ 'a
assumes not-dep-decr: ¬depends₀ a ⇒ I₀ a (x#xs) = I₀ (decr a) xs
assumes anormal-decr: ¬ depends₀ a ⇒ anormal a ⇒ anormal (decr a)
**fun** atoms₀ :: 'a fm ⇒ 'a list where
atoms₀ TrueF = []
atoms₀ FalseF = []
atoms₀ (Atom a) = (if depends₀ a then [a] else [])
atoms₀ (And ϕ₁ ϕ₂) = atoms₀ ϕ₁ @ atoms₀ ϕ₂
atoms₀ (Or ϕ₁ ϕ₂) = atoms₀ ϕ₁ @ atoms₀ ϕ₂
atoms₀ (Neg ϕ) = atoms₀ ϕ

**abbreviation** I where I ≡ interpret I₀

**fun** nnf :: 'a fm ⇒ 'a fm where
nnf (And ϕ₁ ϕ₂) = And (nnf ϕ₁) (nnf ϕ₂)
nnf (Or ϕ₁ ϕ₂) = Or (nnf ϕ₁) (nnf ϕ₂)
nnf (Neg TrueF) = FalseF
nnf (Neg FalseF) = TrueF
nnf (Neg (Neg ϕ)) = (nnf ϕ)
nnf (Neg (And ϕ₁ ϕ₂)) = (Or (nnf (Neg ϕ₁)) (nnf (Neg ϕ₂)))
nnf (Neg (Or ϕ₁ ϕ₂)) = (And (nnf (Neg ϕ₁)) (nnf (Neg ϕ₂)))
nnf (Neg (Atom a)) = aneg a
nnf ϕ = ϕ

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

**lemma** qfree-nnf[simp]: qfree(nnf ϕ) = qfree ϕ
by (induct ϕ rule: nnf.induct)(simp-all add: nqfree-aneg)

**lemma** I-neg[simp]: I (neg ϕ) xs = I (Neg ϕ) xs
by (simp add: neg-def)

**lemma** I-and[simp]: I (and ϕ₁ ϕ₂) xs = I (And ϕ₁ ϕ₂) xs
by (simp add: and-def)

**lemma** I-list-conj[simp]:
I (list-conj fs) xs = (∀ϕ ∈ set fs. I ϕ xs)
proof –
  { fix fs ϕ
    have I (foldr and fs ϕ) xs = (I ϕ xs ∧ (∀ϕ ∈ set fs. I ϕ xs))
      by (induct fs) auto
  } thus thesis by (simp add: list-conj-def)
qed

**lemma** I-or[simp]: I (or ϕ₁ ϕ₂) xs = I (Or ϕ₁ ϕ₂) xs
by (simp add: or-def)

6
lemma I-list-disj[simp]:
I (list-disj fs) xs = (∃ϕ ∈ set fs. I ϕ xs)

proof -
  { fix fs ϕ
    have I (foldr or fs ϕ) xs = (I ϕ xs ∨ (∃ϕ ∈ set fs. I ϕ xs))
      by (induct fs) auto
  } thus ?thesis by(simp add:list-disj-def)
qed

lemma I-nnf: I (nnf ϕ) xs = I ϕ xs
by(induct rule:nnf.induct)(simp-all add:I a-aneg)

lemma I-dnf:
nfree ϕ ⇒ (∃as∈set (dnf ϕ). ∀a∈set as. I_a a xs) = I ϕ xs
by (induct ϕ) (fastforce+)

definition normal ϕ = (∀a ∈ atoms ϕ. anormal a)

lemma normal-simps[simp]:
normal TrueF
normal FalseF
normal (Atom a) ←→ anormal a
normal (And ϕ₁ ϕ₂) ←→ normal ϕ₁ ∧ normal ϕ₂
normal (Or ϕ₁ ϕ₂) ←→ normal ϕ₁ ∧ normal ϕ₂
normal (Neg ϕ) ←→ normal ϕ
normal (ExQ ϕ) ←→ normal ϕ
by (auto simp:normal-def)

lemma normal-aneg[simp]: anormal a ⇒ normal (aneg a)
by (simp add:anormal-aneg normal-def)

lemma normal-and[simp]:
normal ϕ₁ ⇒ normal ϕ₂ ⇒ normal (and ϕ₁ ϕ₂)
by (simp add:Logic.and-def)

lemma normal-or[simp]:
normal ϕ₁ ⇒ normal ϕ₂ ⇒ normal (or ϕ₁ ϕ₂)
by (simp add:Logic.or-def)

lemma normal-list-disj[simp]:
∀ϕ∈set fs. normal ϕ ⇒ 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 ⇒ normal(nnf ϕ)
by(induct ϕ rule:nnf.induct) simp-all
lemma normal-map-fm:
\( \forall a. \text{anormal}(f a) = \text{anormal}(a) \implies \text{normal}(\text{map}_f \varphi) = \text{normal} \varphi \)
by (induct \( \varphi \)) auto

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

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

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

2.1 No Equality

context ATOM
begin
2.1.1 DNF-based

Taking care of atoms independent of variable 0:

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

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

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

lemma I-qelim:
assumes qe: (∀a ∈ set as. depends0 a) ⇒ is-dnf-qe qe as
shows is-dnf-qe (qelim qe) as (is ∀xs. ?P xs)

proof
fix xs
let ?as0 = filter depends0 as
let ?as1 = filter (Not a depends0) 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 φ₁ φ₂) = and (lift-dnf-qe qe φ₁) (lift-dnf-qe qe φ₂) |
lift-dnf-qe qe (Or φ₁ φ₂) = or (lift-dnf-qe qe φ₁) (lift-dnf-qe qe φ₂) |
lift-dnf-qe qe (Neg φ) = neg(lift-dnf-qe qe φ) |
lift-dnf-qe qe (ExQ φ) = Disj (dnf(nnf(lift-dnf-qe qe φ))) (qelim qe) |
lift-dnf-qe qe φ = φ

lemma qfree-lift-dnf-qe: (∀as. (∀a∈set as. depends0 a) ⇒ qfree(qe as))
  ⇒ qfree(lift-dnf-qe qe φ)
by (induct \varphi) (simp-all add:qelim-def)

lemma qfree-lift-dnf-\varphi qe2: \text{qe} : \text{lists} \to \text{qfree}
\Rightarrow \text{qfree}(\text{liftdnf qe} \varphi)
using in-lists-conv-set[where \?'a = 'a]
by (simp add:P-def qfree-lift-dnf-\varphi qe)

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

lemma I-lift-dnf-\varphi qe:
assumes \bigwedge as. (\forall a \in \text{set as}. \text{depends}_a a) \Rightarrow \text{qfree(qe as)}
and \bigwedge as. (\forall a \in \text{set as}. \text{depends}_a a) \Rightarrow \text{is-dnf-\varphi qe as}
shows I (\text{liftdnf qe } \varphi \text{ } xs) = I \varphi \text{ } xs
proof(induct \varphi \text{ arbitrary:xs})
\text{case ExQ thus \?'case}
by (simp add: assms I-qelim lem I-dnf qfree-nnf qfree-lift-dnf-\varphi qe I-nnf)

qed simp-all

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

Quantifier elimination with invariant (needed for Presburger):

lemma I-qelim-anormal:
assumes \text{qe} : \\exists \text{as}. \forall a \in \text{set as}. \text{depends}_a a \land \text{anormal a} \Rightarrow \text{is-dnf-\varphi qe as}
and \text{nn}: \forall a \in \text{set as}. \text{anormal a}
shows I (\text{qelim qe} \text{ as}) \text{ } xs = (\exists x. \forall a \in \text{set as}. I_a a (x\#xs))
proof
- let \?'as0 = \text{filter depends}_0 as
- let \?'as1 = \text{filter (Not o depends}_0 as)
- have I (\text{qelim qe} \text{ as}) \text{ } xs =
  (I (\text{qe } \?'as0) \text{ } xs \land (\forall a \in \text{set(map decre \?'as1)}. I_a a \text{ } xs))
  (is - = (\cdots \text{ ?B}) by (force simp add:qelim-def)
also have \ldots = ((\exists x. \forall a \in \text{set ?as0}. I_a a (x\#xs)) \land \text{ ?B})
  by(simp add:qe nn not-dep-decr)
also have \ldots = (\exists x. (\forall a \in \text{set ?as0}. I_a a (x\#xs)) \land \text{ ?B}) by blast
also have ?B = (\forall a \in \text{set ?as1}. I_a (decr a) \text{ } xs) by simp
also have (\exists x. (\forall a \in \text{set ?as0}. I_a a (x\#xs)) \land \ldots) =
  (\exists x. (\forall a \in \text{set ?as0}. I_a a (x\#xs)) \land
  (\forall a \in \text{set ?as1}. I_a a (x\#xs)))
  by(simp add: not-dep-decr)
also have \ldots = (\exists x. \forall a \in \text{set(?as0 @ ?as1)}. I_a a (x\#xs))
  by (simp add:ball-Un)
also have \ldots = (\exists x. \forall a \in \text{set}(as). I_a a (x\#xs))
  by simp blast
finally show \texttt{thesis} .

qed

declare \llbracket \text{simp-depth-limit = 5} \rrbracket

\textbf{lemma} \texttt{anormal-atoms-qelim:}

\begin{align*}
& (\forall \mathbf{as}, \, \forall a \in \text{set as}. \, \text{depends}_0 a \wedge \text{anormal a} \Rightarrow \text{normal(qe as)}) \Rightarrow \\
& \forall a \in \text{set as}, \, \text{anormal a} \Rightarrow a : \text{atoms(qelim qe as)} \Rightarrow \text{anormal a}
\end{align*}

apply\texttt{(auto simp add:qelim-def and-def normal-def split:split-if-asm)}
apply\texttt{(auto simp add:anormal-decr dest:! atoms-list-conjE)}
apply\texttt{(erule-tac \texttt{x=filter depends} 0 \texttt{as in meta-allE})}
apply\texttt{(simp)}
apply\texttt{(erule-tac \texttt{x=filter depends} 0 \texttt{as in meta-allE})}
apply\texttt{(simp)}
done

\textbf{lemma} \texttt{normal-lift-dnf-qe:}

\begin{align*}
& \text{assumes } (\forall \mathbf{as}. \, \forall a \in \text{set as}. \, \text{depends}_0 a \Rightarrow \text{qfree(qe as)}) \\
& \text{and } (\forall \mathbf{as}. \, \forall a \in \text{set as}. \, \text{depends}_0 a \wedge \text{anormal a} \Rightarrow \text{normal(qe as)}) \\
& \text{shows } \text{normal } f \Rightarrow \text{I (lift-dnf-qe qe f) xs = I f xs}
\end{align*}

proof\texttt{(simp add:normal-def, induct \varphi)}
\texttt{case ExQ thus } \texttt{?case using normal-lift-dnf-qe[of qe]}\texttt{by (simp add:assms simplified normal-def anormal-dnf-nnf I-qelim-anormal I-dnf nqfree-nnf qfree-lift-dnf-qe I-nnf normal-def Ball-def)}
done

\textbf{declare} \llbracket \text{simp-depth-limit = 9} \rrbracket

\textbf{lemma} \texttt{I-lift-dnf-qe-anormal2:}

\begin{align*}
& \text{assumes } \texttt{qe : lists |depends}_0 \rightarrow |\text{qfree}| \\
& \text{and } \texttt{qe : lists ( |depends}_0 | \texttt{\cap |anormal| ) \rightarrow |normal|} \\
& \text{and } \forall \mathbf{as} \in \text{lists}(\texttt{|depends}_0 \texttt{\cap |anormal|). is-dnf-qe qe as} \\
& \text{shows } \text{normal } f \Rightarrow \text{I (lift-dnf-qe qe f) xs = I f xs}
\end{align*}

using \texttt{assms in-lists-conv-set(where \texttt{'a = 'a]}

\textbf{declare} \llbracket \text{simp-depth-limit = 50} \rrbracket

\textbf{lemma} \texttt{I-lift-dnf-qe-anormal2:}

\begin{align*}
& \text{assumes } \texttt{qe : lists |depends}_0 \rightarrow |\text{qfree}| \\
& \text{and } \texttt{qe : lists ( |depends}_0 | \texttt{\cap |anormal| ) \rightarrow |normal|} \\
& \text{and } \forall \mathbf{as} \in \text{lists}(\texttt{|depends}_0 \texttt{\cap |anormal|). is-dnf-qe qe as} \\
& \text{shows } \text{normal } f \Rightarrow \text{I (lift-dnf-qe qe f) xs = I f xs}
\end{align*}

using \texttt{assms in-lists-conv-set(where \texttt{'a = 'a]}

11
2.1.2 NNF-based

fun lift-nnf-qe :: ('a fm ⇒ 'a fm) ⇒ 'a fm where
lift-nnf-qe qe (And ϕ₁ ϕ₂) = and (lift-nnf-qe qe ϕ₁) (lift-nnf-qe qe ϕ₂) |
lift-nnf-qe qe (Or ϕ₁ ϕ₂) = or (lift-nnf-qe qe ϕ₁) (lift-nnf-qe qe ϕ₂) |
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: (⋀ϕ. nqfree ϕ ⇒ qfree(qe ϕ))
⇒ qfree(lift-nnf-qe qe ϕ)
by (induct ϕ) (simp-all add: nqfree-nnf qfree-lift-nnf-qe I-nnf)

lemma qfree-lift-nnf-qe2: qe : |nqfree| → |qfree| ⇒ qfree(lift-nnf-qe qe ϕ)
by (simp add: Pi-def qfree-lift-nnf-qe)

lemma lift-nnf-qe:
assumes □(ϕ. nqfree ϕ ⇒ qfree(qe ϕ))
and □(ϕ. nqfree ϕ ⇒ I (qe ϕ) xs = (∃x. I ϕ (x#xs)))
shows I (lift-nnf-qe qe ϕ) xs = I ϕ xs
proof (induct ϕ arbitrary:xs)
  case ExQ thus ?case
    by (simp add: assms Logic. neg-def normal-nnf nqfree-nnf qfree-lift-nnf-qe)
qed simp-all

lemma lift-nnf-qe2:
assumes qe : |nqfree| → |qfree|
and □(ϕ. nqfree ϕ ⇒ I (qe ϕ) xs = (∃x. I ϕ (x#xs)))
shows I (lift-nnf-qe qe ϕ) xs = I ϕ xs
using assms
by (simp add: Pi-def I-lift-nnf-qe)

lemma normal-lift-nnf-qe:
assumes □(ϕ. nqfree ϕ ⇒ qfree(qe ϕ))
and □(ϕ. nqfree ϕ ⇒ normal ϕ ⇒ normal(qe ϕ))
shows normal ϕ ⇒ 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 lift-nnf-qe-normal:
assumes □(ϕ. nqfree ϕ ⇒ qfree(qe ϕ))
and □(ϕ. nqfree ϕ ⇒ normal ϕ ⇒ normal(qe ϕ))
and □(xs ϕ. normal ϕ ⇒ nqfree ϕ ⇒ I (qe ϕ) xs = (∃x. I ϕ (x#xs)))
shows normal ϕ ⇒ I (lift-nnf-qe qe ϕ) xs = I ϕ xs
proof (induct ϕ arbitrary:xs)
  case ExQ thus ?case
qed
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 : |nqfree| → |qfree|
  and qe : |nqfree| ∩ |normal| → |normal|
  and ALL φ : |normal| Int |nqfree|. ALL xs. I (qe φ) xs = (∃ x. I φ (x#xs))
  shows normal φ =⇒ I (lift-nnf-qe qe φ) xs = I φ 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 solvable0 :: 'a ⇒ bool
  and trivial :: 'a ⇒ bool
  and subst0 :: 'a ⇒ 'a ⇒ 'a
  assumes subst0: [solvable0 eq; ¬ trivial eq; I a eq (x#xs); depends0 a] =⇒ I a (subst0 eq a) xs = I a a (x#xs)
  and trivial: trivial eq =⇒ I a eq xs
  and solvable: solvable0 eq =⇒ ∃ x. I a eq (x#xs)
  and is-triv-self-subst: solvable0 eq =⇒ trivial (subst0 eq eq)

begin

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

theorem I-lift-eq-qe:
  assumes dep: ∀ a∈set as. depends0 a
  assumes qe: (∀ a∈set as. depends0 a ∧ ¬ solvable0 a) =⇒ I (qe as) xs = (∃ x. ∀ a∈set as. I a a (x#xs))
  shows I (lift-eq-qe qe as) xs = (∃ x. ∀ a∈set as. I a a (x#xs))
  (is ?L = ?R)
  proof –
    let ?as = [a←as. ¬ trivial a]
    show ?thesis
    proof (cases [a←?as. solvable0 a])

13
case Nil
hence ∀ a∈set as. ¬ trivial a —> ¬ solvable0 a
by(auto simp: filter-empty-cone)
thus ⊤L = ⊤R
by(simp add:lift-eq-qi-conv dep qe cong:conj-cong) (metis trivial)

next
case (Cons eq -)
then have eq : set as solvable0 eq ¬ trivial eq
by (auto simp: filter-eq-Cons-iff)
then obtain e where I a eq (e # xs) = I a (subst0 eq a) xs
by(simp add: subst0 (OF solvable0 eq) ¬ trivial eq I a eq (e # xs)) [dep]
thus ?thesis using Cons dep
apply(simp add: lift-eq-qi-conv clarsimp simp: filter-eq-Cons-iff ball-Un)
apply(rule iffI)
apply(fastforce intro:exI[of - e] simp: trivial is-triv-self-subst)
apply (metis subst0)
done
qed

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

lemma qfree-lift-eq-qe: (\as. (∀ a∈set as. depends0 a => qfree (qe as))) => (\as. (∀ a∈set as. depends0 a => qfree(lift-eq-qi qi qe as)) => qfree(lift-eq-qi qi qe as))
by(simp add:lift-eq-qi-conv qe cong:ball-conj Un split:split)

lemma qfree-lift-dnfeq-qe: (\as. (∀ a∈set as. depends0 a => qfree(qe as))) => qfree(lift-dnfeq-qe qi qe qe "e"")
by(simp add: lift-dnfeq-qi-conv qe cong:qfree-qe cong:ball-conj Un split:split)

lemma I-lift-dnfeq-qe:
(\as. (∀ a∈set as. depends0 a => qfree(qe as))) =>
(\as. (∀ a∈set as. depends0 a ∧ ¬ solvable0 a) => is-dnf-qi qi qe as) => I (lift-dnfeq-qi qi qe "e") xs = I "e" xs

lemma I-lift-dnfeq-qe2:
qe : lists (depends0) -> |qe| =>
(\as ∈ lists ( depends0 ∩ ¬ solvable0 ). is-dnf-qi qi qe as) => I (lift-dnfeq-qi qi qe "e") xs = I "e" xs
using in-lists-conv-set[where ?'a = 'a]
by(simp add:Pi-def I-lift-dnfeq-qi qe cong:is-dnf-qi cong:ball-conj Un split:split)

end

end
3 DLO

theory DLO
imports QE Complex-Main
begin

3.1 Basics

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

instance real :: dlo

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

datatype atom = Less nat nat | Eq nat nat

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

abbreviation is-Eq \equiv\ Not o is-Less

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

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

fun I_dlo :: atom \Rightarrow 'a::dlo list \Rightarrow bool where
I_dlo (Eq i j) xs = (xs!i = xs!j) |
\( I_{\Delta_0} (\text{Less } i \ j) \ xs = (xs!i < xs!j) \)

\[
\text{fun depends}_{\Delta_0} :: \text{atom} \Rightarrow \text{bool} \quad \text{where} \\
\text{depends}_{\Delta_0} (\text{Eq } i \ j) = (i = 0 \mid j = 0) \\
\text{depends}_{\Delta_0} (\text{Less } i \ j) = (i = 0 \mid j = 0)
\]

\[
\text{fun decr}_{\Delta_0} :: \text{atom} \Rightarrow \text{atom} \quad \text{where} \\
\text{decr}_{\Delta_0} (\text{Less } i \ j) = \text{Less } (i - 1) (j - 1) \\
\text{decr}_{\Delta_0} (\text{Eq } i \ j) = \text{Eq } (i - 1) (j - 1)
\]

\[
\begin{align*}
\text{definition} & \quad \begin{cases} 
\text{[code del]}: \text{nff} = \text{ATOM.nnf neg}_{\Delta_0} \\
\text{[code del]}: \text{qelim} = \text{ATOM.qelim depends}_{\Delta_0} \text{decr}_{\Delta_0} \\
\text{[code del]}: \text{lift-dnf-qe} = \text{ATOM.lift-dnf-qe neg}_{\Delta_0} \text{depends}_{\Delta_0} \text{decr}_{\Delta_0} \\
\text{[code del]}: \text{lift-nnf-qe} = \text{ATOM.lift-nnf-qe neg}_{\Delta_0}
\end{cases} \\
\text{hide-const} & \quad \text{nff qelim lift-dnf-qe lift-nnf-qe}
\end{align*}
\]

\[
\begin{align*}
\text{lemmas} & \quad \text{DLO-code-lemmas} = \text{nff-def qelim-def lift-dnf-qe-def lift-nnf-qe-def} \\
\text{interpretation} & \quad \text{DLO!}:
\begin{align*}
\text{ATOM neg}_{\Delta_0} (\lambda a. \text{True}) & = I_{\Delta_0} \text{depends}_{\Delta_0} \text{decr}_{\Delta_0} \\
\text{apply} & \quad \text{(unfold-locales)} \\
\text{apply} & \quad \text{(case-tac } a) \\
\text{apply} & \quad \text{(simp-all)} \\
\text{apply} & \quad \text{(case-tac } a) \\
\text{apply} & \quad \text{(simp-all add:linorder-class.not-less-iff-gr-or-eq}
\text{linorder-not-less linorder-neq-iff)} \\
\text{apply} & \quad \text{(case-tac } a) \\
\text{apply} & \quad \text{(simp-all add:nth-Cons')} \\
\text{done}
\end{align*}
\]

\[
\begin{align*}
\text{lemmas} & \quad \begin{cases} 
\text{[folded DLO-code-lemmas, code]} = \\
\text{DLO.nnf.simps DLO.qelim-def DLO.lift-dnf-qe.simps DLO.lift-dnf-qe.simps}
\end{cases} \\
\text{setup} & \quad \langle \langle \text{Sign.revert-abbrev} @\{\text{const-abbrev DLO.I}\} \rangle \rangle
\end{align*}
\]

\[
\begin{align*}
\text{definition} & \quad \text{bounds where } lbounds as = [i. \text{Less } (\text{Suc } i) \ 0 \leftarrow as] \\
\text{definition} & \quad \text{ubounds where } ubounds as = [i. \text{Less } 0 (\text{Suc } i) \leftarrow as] \\
\text{definition} & \quad \text{ebounds where } ebounds as = [i. \text{Eq } (\text{Suc } i) \ 0 \leftarrow as] @ [i. \text{Eq } 0 (\text{Suc } i) \leftarrow as]
\end{align*}
\]

\[
\begin{align*}
\text{lemma} & \quad \text{set-lbounds: set(lbounds as) = } \{i. \text{Less } (\text{Suc } i) \ 0 \ : \ \text{set as}\} \\
& \quad \text{by(auto simp: lbounds-def split:nat.splits atom.splits)} \\
\text{lemma} & \quad \text{set-ubounds: set(ubounds as) = } \{i. \text{Less } 0 (\text{Suc } i) \ : \ \text{set as}\} \\
& \quad \text{by(auto simp: ubounds-def split:nat.splits atom.splits)} \\
\text{lemma} & \quad \text{set-ebounds: set(ebounds as) = } \{k. \text{Eq } (\text{Suc } k) \ 0 \ : \ \text{set as} \lor \text{Eq } 0 (\text{Suc } k) \ : \ \text{set as}\} \\
& \quad \text{by(auto simp: ebounds-def split: atom.splits nat.splits)}
\end{align*}
\]
abbreviation \( LB f xs \equiv \{xs!i|i. \ Less (Suc i) 0 : \set(DLO.atoms_0 f)\} \)
abbreviation \( UB f xs \equiv \{xs!i|i. \ Less 0 (Suc i) : \set(DLO.atoms_0 f)\} \)
definition \( EQ f xs \equiv \{xs!k|k. \ Eq (Suc k) 0 : \set(DLO.atoms_0 f) \lor Eq 0 (Suc k) : \set(DLO.atoms_0 f)\} \)

lemma \( EQ \And\{simp\}: EQ (\And f g) xs = (EQ f xs \Un EQ g xs) \)
by(auto simp:EQ-def)

lemma \( EQ \Or\{simp\}: EQ (\Or f g) xs = (EQ f xs \Un EQ g xs) \)
by(auto simp:EQ-def)

lemma \( EQ-conv-set-ebounds: x \in EQ f xs = (\exists e \in \set(\text{ebounds}(DLO.atoms_0 f)). x = xs!e) \)
by(auto simp:EQ-def set-ebounds)

fun isubst where isubst k 0 = k | isubst k (Suc i) = i
fun asubst :: nat \Rightarrow atom \Rightarrow 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 \( \varphi \ k \equiv \mapfm(asubst k) \varphi \)

lemma \( I-subst\):
qfree f \Rightarrow DLO.I \ (\text{subst } f \ k) \ xs = DLO.I \ (xs!k \ # \ xs) 
apply(induct f)
apply(simp-all)
apply(case-tac a)
apply(simp-all add:nth.simps split:nat.splits)
done

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

abbreviation min-inf :: atom fm \Rightarrow atom fm (inf _) where
inf _ \equiv amapfm amin-inf

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

abbreviation plus-inf :: atom fm ⇒ 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 $\text{thesis}$ $z$ ..
qed

case (Atom $a$)
show $\text{case}
proof (cases a rule: aplus-inf.cases)
case 1 thus $\text{thesis}$
  by (simp add: nth-Cons') (metis linorder-not-less)
next
case 2 thus $\text{thesis}$ by (auto intro: dlo-bound)
next
case 5 thus $\text{thesis}$
  by simp (metis dlo-bound less-imp-neq)
next
case 6 thus $\text{thesis}$
  by simp (metis dlo-bound less-imp-neq)
qed simp-all

next
case (And $f_1$ $f_2$)
then obtain $x_1$ $x_2$ where $\text{thesis}$ by fastforce+
hence $\text{thesis}$ (max $x_1$ $x_2$) by (force simp: and-def)
thus $\text{thesis}$ ..
next
case (Or $f_1$ $f_2$)
then obtain $x_1$ $x_2$ where $\text{thesis}$ by fastforce+
hence $\text{thesis}$ (max $x_1$ $x_2$) by (force simp: or-def)
thus $\text{thesis}$ ..
qed simp-all

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

lemma $LBex$:
[ $\text{nf} f$; $\text{DLO} f$ $x$#$xs$; $\text{DLO} f$ $x$#xs $\notin$ $\text{EQ} f$ $xs$ ]
$\implies \exists l \in LB f$ $xs$. $l < x$
proof (induct $f$)
case (Atom $a$) thus $\text{thesis}$
  by (cases a rule: amin-inf.cases)
    (simp-all add: nth.simps EQ-def split: nat.splits)
qed auto

lemma $UBex$:
[ $\text{nf} f$; $\text{DLO} f$ $x$#$xs$; $\text{DLO} f$ $x$#xs $\notin$ $\text{EQ} f$ $xs$ ]
$\implies \exists u \in UB f$ $xs$. $x < u$
proof (induct $f$)
case (Atom $a$) thus $\text{thesis}$
  by (cases a rule: aplus-inf.cases)
    (simp-all add: nth.simps EQ-def split: nat.splits)
qed auto

declare[[simp-depth-limit=50]]
lemma finite-LB: finite(LB f xs)
proof -
  have LB f xs = (λk. xs!k) ∈ set(lbounds(DLO.atoms0 f))
    by (auto simp:set-lbounds image-def)
  thus ?thesis by simp
qed

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

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

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

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

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

end

theory QEdlo
imports DLO
begin

3.2 DNF-based quantifier elimination

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

theorem I-qe-dlo1:
assumes less: ∀a ∈ set as. is-Less a and dep: ∀a ∈ set as. depends_dlo a
shows DLO.I (qe-dlo1 as) xs = (∃x. ∀a ∈ set as. I_dlo a (x#xs))
(is ?L = ?R)
proof
  let ?lbs = [i. Less (Suc i) 0 ← as]
let \( \mathcal{U}_b \) = \( \{ j \mid \text{Less } 0 \ (\text{Suc } j) \leftarrow \text{as} \} \)
let \( \mathcal{L}_s \) = set \( \mathcal{U}_b \) let \( \mathcal{U}_a \) = set \( \mathcal{U}_b \)
let \( \mathcal{U}_b = \text{Max} \left( \bigcup x \in \mathcal{L}_s. \{ x!x \} \right) \)
let \( \mathcal{U}_a = \text{Min} \left( \bigcup x \in \mathcal{L}_a. \{ x!x \} \right) \)

have 2: \( \text{Less } 0 \ 0 \ \notin \text{set as} \implies \forall a \in \text{set as} \)

\( (\exists i \in \mathcal{L}_s. \ a = \text{Less } (\text{Suc } i) \ 0) \lor (\exists i \in \mathcal{U}_a. \ a = \text{Less } 0 \ (\text{Suc } i)) \)

proof

fix a assume \( \text{Less } 0 \ 0 \ \notin \text{set as} \) as \( a \in \text{set as} \)
then obtain \( i \ \text{where} \ [\text{simp}]: \ a = \text{Less } i \ j \)
  using \text{less} by (force simp:is-Less-iff)
with \text{dep} obtain \( k \ \text{where} \ i = 0 \ \land \ j = \text{Suc } k \ \lor \ i = \text{Suc } k \ \land \ j = 0 \)
  using (Less 0 0 \ \notin \text{set as}) \ \langle a \in \text{set as} \rangle
by auto (metis \text{Nat.nat.chotomy} \text{depends}_{\text{dlo}}.\text{simps}(2))
moreover hence \( i=0 \ \land \ k \in \ ?U_a \ \lor \ j=0 \ \land \ k \in \ ?L_s \)
  using \( \langle a \in \text{set as} \rangle \) by force
ultimately show \( (\exists i \in \mathcal{L}_s. \ a=\text{Less } (\text{Suc } i) \ 0) \lor (\exists i \in \mathcal{U}_a. \ a=\text{Less } 0 \ (\text{Suc } i)) \)
by force

qed

assume \( \text{qe1} \): \( ?L \)
hence 0: \( \text{Less } 0 \ 0 \ \notin \text{set as} \) by (auto simp:qe-dlo1-def)
with \text{qe1} have 1: \( \forall x \in \mathcal{L}_s. \ \forall y \in \mathcal{U}_a. \ x!x \ < \ x!y \ y \)
  by (fastforce simp:qe-dlo1-def)

have finite: finite ?Ls finite ?U_a by (rule finite-set)+

\{ fix \( i \ \text{x} \)
assume \( \text{Less } i \ 0 \in \text{set as} \) \| \( \text{Less } 0 \ 0 \ in \text{set as} \)
moreover hence \( i \neq 0 \) using 0 by 
ultimately have \( (x\#xs)!i = xs!(i-1) \) by (simp add: nth-Cons')
\}

note this(simp)

\{ assume nonempty: \( \mathcal{L}_s \neq \{ \} \ \land \ \mathcal{U}_a \neq \{ \} \)
hence \( \text{Max} \left( \bigcup x \in \mathcal{L}_s. \{ x!x \} \right) < \text{Min} \left( \bigcup x \in \mathcal{U}_a. \{ x!x \} \right) \)
  using \text{finite} by auto
then obtain \( m \ \text{where} \ ?ub < m \ \land \ m < \ ?ub \) using \text{dense} by blast
hence \( \forall i \in \mathcal{L}_s. \ x!i < m \ \land \ \forall j \in \mathcal{U}_a. \ m < x!j \)
  using \text{nonempty} \text{finite} by auto
hence \( \forall a \in \text{set as}. \ I_dlo \ a \ (m \ # \ xs) \) using \( 2 \left( \text{OF } 0 \right) \) by(\text{auto simp:less})
hence \( \text{R} \ .. \) \}

moreover

\{ assume \text{asm}: \( \mathcal{L}_s \neq \{ \} \ \land \ \mathcal{U}_a = \{ \} \)
then obtain \( m \ \text{where} \ ?ub < m \) using \text{no-ab} by blast
hence \( \forall a \in \text{set as}. \ I_dlo \ a \ (m \ # \ xs) \) using \( 2 \left( \text{OF } 0 \right) \) \text{asm} \text{ finite} by auto
hence \( \text{R} \ .. \) \}

moreover

\{ assume \text{asm}: \( \mathcal{L}_s = \{ \} \ \land \ \mathcal{U}_a \neq \{ \} \)
then obtain \( m \ \text{where} \ m < \ ?ub \) using \text{no-lb} by blast
hence \( \forall a \in \text{set as}. \ I_dlo \ a \ (m \ # \ xs) \) using \( 2 \left( \text{OF } 0 \right) \) \text{asm} \text{ finite} by auto
hence \( \text{R} \ .. \) \}

moreover

\{ assume \( \mathcal{L}_s = \{ \} \ \land \ \mathcal{U}_a = \{ \} \)
hence \( \text{R} \) using \( 2 \left( \text{OF } 0 \right) \) by (\text{auto simp add:less})

21
ultimately show $R$ by blast

next

assume $R$
then obtain $x$ where \[ \forall a \in \text{set as } I_{dlo} a (x \# xs) \] hence $0$: Less $0 \ 0 \notin \text{set as by auto}$

\{ fix $i \ j$
assume asm: Less $i \ 0 \in \text{set as Less } 0 \ j \in \text{set as}$
hence $(x\#xs)!i < x x < (x\#xs)!j$ using $I$ by auto+
hence $(x\#xs)!i < (x\#xs)!j$ by (rule order-less-trans)
moreover have $\neg(i = 0 \mid j = 0)$ using $0$ asm by blast
ultimately have $xs \ I (i - 1) < xs \ (j - 1)$ by (simp add: nth-Cons') \}

thus $L$ using $0$ less
by (fastforce simp: qe-dlo1-def is-Less-iff split:atom.splits nat.splits)

qed

lemma I-qe-dlo1-prett:
$\forall a \in \text{set as. is-Less a \ \& \ \text{depends}_{dlo} a \ \Rightarrow \ DLO\text{-dnf-qe - qe-dlo1 as by (metis I-qe-dlo1)}}$

definition subst :: nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat where
$\text{subst } i \ j \ k = (\text{if } k=0 \ \text{then if } i=0 \ \text{then } j \ \text{else } i \ \text{else } k) - 1$

fun subst0 :: atom $\Rightarrow$ atom $\Rightarrow$ atom where
$\text{subst0 } (\text{Eq } i \ j) \ a = (\text{case } a \ \text{of}$
$\text{Less } m \ n \Rightarrow \text{Less } (\text{subst } i \ j \ m ) (\text{subst } i \ j \ n )$
$| \text{Eq } m \ n \Rightarrow \text{Eq } (\text{subst } i \ j \ m ) (\text{subst } i \ j \ n ))$

lemma subst0-prett:
$\text{subst0 } (\text{Eq } i \ j ) (\text{Less } m \ n ) = \text{Less } (\text{subst } i \ j \ m ) (\text{subst } i \ j \ n )$
$\text{subst0 } (\text{Eq } i \ j ) (\text{Eq } m \ n ) = \text{Eq } (\text{subst } i \ j \ m ) (\text{subst } i \ j \ n )$

by auto

interpretation $DLO_e$!
ATOM-EQ neg_{dlo} ($\lambda a. \ True$) $I_{dlo}$ depends_{dlo} decr_{dlo}
$\text{Subst}$ ($\text{Eq } i \ j \Rightarrow i=0 \ \vee \ j=0 \ \mid \ a \Rightarrow \text{False}$)
$\text{Subst}$ ($\text{Eq } i \ j \Rightarrow i=j \ \mid \ a \Rightarrow \text{False}$) subst0
apply (unfold-locale)
apply (fastforce simp: subst-def nth-Cons' split:atom.splits split-if-asm)
apply (simp add:subst-def split:atom.splits)
apply (fastforce simp: subst-def nth-Cons' split:atom.splits)
apply (fastforce simp add: subst-def split:atom.splits)
done

setup "$\langle \langle \text{Sign.revert-abbrev @\{const-abbrev DLO_e.lift-dnfeq-qe \} \rangle} \rangle$

definition qe-dlo = $DLO_e$.lift-dnfeq-qe qe-dlo1
lemma qfree-qe-dlo1: qfree (qe-dlo1 as)
by (auto simp: qe-dlo1-def intro!: qfree-list-conj)
theorem I-qe-dlo: DLO.I (qe-dlo \( \varphi \)) \( \text{xs} = \) DLO.I \( \varphi \) \( \text{xs} \)
unfolding qe-dlo-def
by (fastforce intro!: I-qe-dloDLO qfree-qe-dlo\( _{\text{I}} \) DLO.e.I-lift-dnfeq-qe
  simp: is-Less-iff not-is-Eq-iff split:atom.split cong:conj-cong)

theorem qfree-qe-dlo: qfree (qe-dlo \( \varphi \))
by (simp add:qe-dlo-def DLO.e.qfree-lift-dnfeq-qe qfree-qe-dlo\( _{\text{I}} \))
end

definition interpret :: atom fm \( \Rightarrow \) \text{'a::dlo list} \( \Rightarrow \) bool where
  interpret = Logic.interpret I_dlo

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

lemma interpret-others:
  interpret (Neg (ExQ (Neg f))) \( \text{xs} = \) (\( \forall \) x. interpret f (x\#xs))
  interpret (Or (Neg f1) f2) \( \text{xs} = \) (interpret f1 \( \text{xs} \) \( \Rightarrow \) interpret f2 \( \text{xs} \))
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[code del]
declare Logic.interpret.simps(1-2)[code]

3.3 Examples
lemma \( \forall \text{x::real}. \neg \text{x} < \text{x} \)

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::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 y z. ∀ u::real. x < x | ¬ x<u | x<y & y<z & ¬ x<z)
apply dlo-reify
apply (subst I-qe-dlo [symmetric])
by eval

lemma qe-dlo(AllQ (Imp (Atom(Less 0 1)) (Atom(Less 0 1)))) = 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 (Imp (Atom(Less 2 1)) (Atom(Less 1 0))))) = FalseF
by eval

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

lemma qe-dlo(AllQ(AllQ(ExQ (Imp (Atom(Less 1 0)) (Atom(Less 0 2))))) = FalseF
by eval

24
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 ⇒ nat ⇒ atom ⇒ atom fm where
asubst_2 l u (Less 0 0) = FalseF |
asubst_2 l u (Less 0 (Suc j)) = Or (Atom(Less u j)) (Atom(Eq u j)) |
asubst_2 l u (Less (Suc i) 0) = Or (Atom(Less i l)) (Atom(Eq i l)) |
asubst_2 l u (Less (Suc i) (Suc j)) = Atom(Less i j) |
asubst_2 l u (Eq 0 0) = TrueF |
asubst_2 l u (Eq 0 -) = FalseF |
asubst_2 l u (Eq - 0) = FalseF |
asubst_2 l u (Eq (Suc i) (Suc j)) = Atom(Eq i j)

abbreviation subst_2 l u ≡ amap_fm (asubst_2 l u)

lemma I-subst_2_1:
 nqfree f ⇒ xs!l < xs!u ⇒ DLO.I (subst_2 l u f) xs
⇒ xs!l < x ⇒ x < xs!u ⇒ DLO.I f (x#xs)
proof(induct f arbitrary: x)
case (Atom a) thus ?case
  by (cases (l,u,a) rule: asubst_2.cases) auto
qed auto

definition nolub f xs l x u ←→ (∀ y∈{l..<x}. y ∉ LB f xs) ∧ (∀ y∈{x..<u}. y ∉ UB f xs)

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

lemma nolub-Or[simp]:
\texttt{nolub (Or f g) xs l x u = (nolub f xs l x u \land nolub g xs l x u)}

\texttt{by(auto simp:nolub-def)}

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

\texttt{lemma innermost-intvl:}

\texttt{[ \ nqfree f; nolub f xs l x u; l < x; x < u; x \notin EQ f xs; DLO.I f (x\#xs); l < y; y < u ]} \implies \texttt{DLO.I f (y\#xs)}

\texttt{proof(induct f)}

\texttt{case (Atom a)}

\texttt{show \ ?case}

\texttt{proof (cases a)}

\texttt{case (Less i j)}

\texttt{then show \ ?thesis using Atom}

\texttt{unfolding nolub-def}

\texttt{by (clarisimp simp; nth.simps Ball-def split:split-if_asm nat.splits)}

\texttt{(metis not-leE order-antisym order-less-trans)+}

\texttt{next}

\texttt{case (Eq i j)[simp]}

\texttt{show \ ?thesis}

\texttt{proof (cases i)}

\texttt{case 0[simp]}

\texttt{show \ ?thesis}

\texttt{proof (cases j)}

\texttt{case 0 thus \ ?thesis using Atom by simp}

\texttt{next}

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

\texttt{qed}

\texttt{next}

\texttt{case Suc[simp]}

\texttt{show \ ?thesis}

\texttt{proof (cases j)}

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

\texttt{next}

\texttt{case Suc thus \ ?thesis using Atom by simp}

\texttt{qed}

\texttt{qed}

\texttt{next}

\texttt{case (And f1 f2) thus \ ?case by (fastforce)}

\texttt{next}

\texttt{case (Or f1 f2) thus \ ?case by (fastforce)}

\texttt{qed simp+}

\texttt{lemma I-subst2:}

\texttt{\ nqfree f \implies \ \forall x. x \in \{xs!l <..< xs!u\} \implies nolub f xs (xs!u) x (xs!l)}

\texttt{\implies \ \forall x. x \notin \{xs!l <..< xs!u\}. DLO.I f (x\#xs) \land x \notin EQ f xs}

\texttt{\implies DLO.I (subst2 l u f) xs}

\texttt{proof (induct f)}
proof shows \( \exists n q \text{free} \) assumes \( I \text{-interior} \) theorem qed

next case Or thus ?case by (simp add: Ball-def)(metis innermost-intvl) qed auto declare[[simp-depth-limit=50]]

definition \( qe \text{-interior} _1 \) \( \varphi \) =
(\{ let as = DLO.atoms _0 \( \varphi \); lbs = lbounds as; ub = ubounds as; ebs = ebounds as; intrs = [And (Atom (Less l u)) (subst t l u \( \varphi \)). l \( \leftarrow \) lbs, u \( \leftarrow \) ub] in list-disj (inf \( \varphi \) # inf \( \varphi \) # intrs @ map (subst \( \varphi \) ebs))
)

lemma dense-interval:
assumes \( \text{finite } L \) \( \text{finite } U \) \( l : L \ u : U \ l < x < u P(x::a:dlo) \)
and dense: \( \exists y l u. [ \forall y \in \{l<..<x\} \ y \notin L \ \forall y \in \{x<..<u\} \ y \notin U \ l < x < u \ l < y < u \ \Longrightarrow \ y \ l] \)
shows \( \exists l \in L . \exists u \in U . l < x \land x < u \ \land (\forall y \in \{l<..<x\} \ y \notin L \ \land \forall y \in \{x<..<u\} \ y \notin U \)
\land (\forall y \ l < y \land y < u \ \Longrightarrow \ y \ l) \)
proof –
let ?L = \{l: L. l < x \} let ?U = \{u : U. x < u \}
have ?L \( \neq \) \{ \} using \( l : L \) \( l < x \) by (blast intro:order-less-imp-le)
moreover have ?U \( \neq \) \{ \} using \( u : U \) \( x < u \) by (blast intro:order-less-imp-le)
ultimately have \( \forall y . \ ?U \ y < y < u \ \Longrightarrow \ y \ ?U \ \forall y . x < y \land y < ?u < u \ \Longrightarrow \ y \ ?U \)
using \( \text{finite } L \) \( \text{finite } U \) by force+
moreover have \( \forall l : L \)
proof
show \( \exists l : L . \ ?L \) using \( \text{finite } L \) \( \text{Max-in}[OF - \ ?L \neq \{\}] \) by simp
show \( \exists l \subseteq L \) by blast
qed
moreover have \( \forall uu : U \)
proof
show \( \exists u : U . \ ?U \) using \( \text{finite } U \) \( \text{Min-in}[OF - \ ?U \neq \{\}] \) by simp
show \( \exists u \subseteq U \) by blast
qed
moreover have \( \forall l < x \) using \( \text{finite } L \) \( \forall \ ?L \neq \{\} \) by simp
moreover have \( x < \ ?uu \) using \( \text{finite } U \) \( \forall \ ?U \neq \{\} \) by simp
moreover have \( \forall l < \ ?uu \) using \( \ ?l \ ?u \ ?x \ ?u u \) by simp
ultimately show \( \text{thesis} \) using \( l < x \) \( x < u \) \( \forall \ ?L \neq \{\} \ ?U \neq \{\} \)
by(blast intro!:dense greaterThanLessThan-iff[THEN iffD1])
qed

theorem \( I \text{-interior} _1 \): 
asummary{\( qe \text{-interior} _1 \) \( \varphi \) shows DLO.I \( (qe \text{-interior} _1 \) \( \varphi \) \) xs = (EX x. DLO.I \( \varphi \) (xs #xs))
(is \( ?QE \ = \ ?EX \))

27
proof

assume \( ?QE \)

\{ assume \( DLO.I \ (\text{inf} \_ \varphi) \ xs \)

\( \text{hence } ?EX \text{ using } \langle ?QE \rangle \ \text{min-inf[of } \varphi \ xs \rangle \ \langle \text{nagfree } \varphi \rangle \)

\( \text{by(auto simp add:qe-interior1-def amap-fm-list-disj)} \}

\} moreover

\{ assume \( DLO.I \ (\text{inf}_+ \varphi) \ xs \)

\( \text{hence } ?EX \text{ using } \langle ?QE \rangle \ \text{plus-inf[of } \varphi \ xs \rangle \ \langle \text{nagfree } \varphi \rangle \)

\( \text{by(auto simp add:qe-interior1-def amap-fm-list-disj)} \}

\} moreover

\{ assume \( \neg DLO.I \ (\text{inf} \_ \varphi) \ xs \wedge \neg DLO.I \ (\text{inf}_+ \varphi) \ xs \wedge \\
\( \forall x \in EQ \ \varphi \ xs \). \neg DLO.I \ \varphi \ (x\#xs) \)

with \( ?QE \) \langle \text{nagfree } \varphi \rangle \ obtain \ l \ u \)

where \( DLO.I \ (\text{subst}_2 \ l \ u \varphi) \ xs \) and \( \text{xs}!l < \text{xs}!u \)

\( \text{by(fastforce simp: qe-interior1-def set-lbounds set-ubounds I-subst EQ-conv-set-ebounds)} \)

\} moreover then \( \text{obtain } x \) where \( \text{xs}!l < x \wedge x < \text{xs}!u \) by(\text{metis dense})

\( \text{ultimately have } DLO.I \ \varphi \ (x \neq xs) \)

\( \text{using } \langle \text{nagfree } \varphi \rangle \ \text{I-subst}_2 \ [\text{OF } \langle \text{nagfree } \varphi \rangle \ \text{xs}!l < \text{xs}!u] \) \( \text{by simp} \)

\( \text{hence } ?EX .. \} \)

\( \text{ultimately show } ?EX \text{ by blast} \)

next

let \( ?as = DLO.\text{atoms}_0 \ \varphi \) let \( ?E = \text{set(ebounds } \?as) \)

assume \( ?EX \)

then \( \text{obtain } x \) where \( x: DLO.I \ \varphi \ (x\#xs) .. \)

\{ assume \( DLO.I \ (\text{inf} \_ \varphi) \ xs \vee DLO.I \ (\text{inf}_+ \varphi) \ xs \)

\( \text{hence } ?QE \text{ using } \langle \text{nagfree } \varphi \rangle \ \text{by(auto simp:qe-interior1-def)} \}

\} moreover

\{ assume \( \exists E \ ?E. \ \ DLO.I \ (\text{subst } \varphi \ k) \ xs \)

\( \text{hence } ?QE \text{ by(force simp:qe-interior1-def)} \} \)

\} moreover

\{ assume \( \neg E \ \forall e \in EQ \ \varphi \ xs. \neg DLO.I \ (\text{inf}_+ \varphi) \ xs \)

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

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

\( \text{using } \langle \text{nagfree } \varphi \rangle \ \text{by(force simp:set-ebounds EQ-def I-subst)} \)

\( \text{hence } x \notin EQ \ \varphi \ xs \) \( \text{using } x \text{ by fastforce} \)

\( \text{obtain } l \) where \( l: LB \ \varphi \ xs \ l < x \)

\( \text{using } \text{LBex}[OF \ \langle \text{nagfree } \varphi \rangle \ \text{xs} \ (\neg DLO.I \ (\text{inf} \_ \varphi) \ xs) \ (x \notin EQ \ \varphi \ xs)] .. \)

\( \text{obtain } u \) where \( u: UB \ \varphi \ xs \ x < u \)

\( \text{using } \text{UBex}[OF \ \langle \text{nagfree } \varphi \rangle \ \text{xs} \ (\neg DLO.I \ (\text{inf}_+ \varphi) \ xs) \ (x \notin EQ \ \varphi \ xs)] .. \)

\( \exists \exists l \in LB \ \varphi \ xs. \exists u \in UB \ \varphi \ xs. l < x \wedge x < u \wedge \text{nolub } \varphi \ xs \ l \ x \ u \wedge (\forall y. \ l < y \wedge y < u \longrightarrow DLO.I \ \varphi \ (y\#xs)) \)

\( \text{using } \text{dense-interval[where } P = \lambda x. \ DLO.I \ \varphi \ (x\#xs), \ \text{OF finite-LB finite-UB} \ l:LB \ \varphi \ xs \ (u:UB \ \varphi \ xs) \ (l < x) \wedge (x < u) \wedge (\forall y. \ l < y \wedge y < u \longrightarrow DLO.I \ \varphi \ (y\#xs)) \]

\( \text{by(simp add:nolub-def)} \)

then \( \text{obtain } m \ n \) where

\( \text{Less } (Suc \ m) \ 0 : \text{set } \?as \ \text{Less } 0 \ (Suc \ n) : \text{set } \?as \)

\( \text{xs}!m < x \wedge x < \text{xs}!n \)

\( \text{nolub } \varphi \ xs \ (\text{xs}!m) \ x \ (\text{xs}!n) \)

\( \forall y. \ \text{xs}!m < y \wedge y < \text{xs}!n \longrightarrow DLO.I \ \varphi \ (y\#xs) \)

28
by blast
moreover
hence DLO.I (subst₂ m n ϕ) xs using noE
  by(force intro!: subst₂[OF ϕ])
ultimately have ?QE
  by(fastforce simp add:qe-interior₁-def bex-U n 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-subst₂ qfree-map-fm)
done

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

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) |

\(\text{asubst-peps } k \ (\text{Eq } 0 \ 0) = \text{TrueF} \ |
\text{asubst-peps } k \ (\text{Eq } 0 \ -) = \text{FalseF} \ |
\text{asubst-peps } k \ (\text{Eq } - \ 0) = \text{FalseF} \ |
\text{asubst-peps } k \ (\text{Eq } (\text{Suc } i) \ (\text{Suc } j)) = \text{Atom(Eq } i \ j)\)

**abbreviation** \(\text{subst-peps} :: \text{atom fm } \Rightarrow \text{nat } \Rightarrow \text{atom fm (subst_+)}\) where
\[
\text{subst_+ } \varphi \ k \equiv \text{amap fm (asubst_+ k) } \varphi
\]

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

**lemma** \(\text{nolb-And [simp]}:\)
\[
\text{nolb}(\text{And } \varphi_1 \varphi_2) \ xs l x = (\text{nolb } \varphi_1 \ xs l x \wedge \text{nolb } \varphi_2 \ xs l x)
\]
**apply** (clarsimp simp:nolb-def)
**apply** blast
**done**

**lemma** \(\text{nolb-Or [simp]}:\)
\[
\text{nolb}(\text{Or } \varphi_1 \varphi_2) \ xs l x = (\text{nolb } \varphi_1 \ xs l x \wedge \text{nolb } \varphi_2 \ xs l x)
\]
**apply** (clarsimp simp:nolb-def)
**apply** blast
**done**

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

**lemma** innermost-intvl:
\[
[\text{nfree } \varphi; \text{nolb } \varphi \ xs l x; l < x; x \notin \text{EQ } \varphi \ xs; \text{DLO.I } \varphi (x\#xs); l < y; y \leq x] \implies \text{DLO.I } \varphi (y\#xs)
\]
**proof** (induct \(\varphi\))
**case** (Atom \(a\))
**show** ?case
**proof** (cases \(a\))
**case** (Less \(i \ j\))
**then** show ?thesis using Atom unfolding nolb-def
**by** (clarsimp simp: nth.simps Ball-def split:split-if-asmp nat.splits)
(metis not-leE order-antisym order-less-trans)+

**next**
**case** (Eq \(i \ j\)) \textbf{thus} ?thesis using Atom
**apply** (clarsimp simp:EQ-def nolb-def nth-Cons’)
**apply** (case-tac \(i=0 \land j=0\)) **apply** simp
**apply** (case-tac \(i\neq0 \land j\neq0\)) **apply** simp
**apply** (case-tac \(i=0 \land j\neq0\)) **apply** (fastforce split:split-if-asmp)
**apply** (case-tac \(i\neq0 \land j=0\)) **apply** (fastforce split:split-if-asmp)
**apply** arith
**done**

**Qed**
**next**
**case** And \textbf{thus} ?case \textbf{by} (fastforce)
**next**
**case** Or \textbf{thus} ?case \textbf{by} (fastforce)
qed simp+

lemma I-subst-peps2:
\[ \text{nqfree } \varphi \implies \\text{xs}(x) < x \implies \text{nolb } \varphi \text{ xs } (xs!l) \implies x \notin EQ \varphi \text{ xs} \]
\[ \implies \forall y \in \{xs!l <.. x\}, DLO.I \varphi (y#xs) \]
\[ \implies DLO.I (\text{subst}_+ \varphi l) \text{ xs} \]
proof (induct \( \varphi \))
  case FalseF thus ?case by simp
proof (cases (l,a) rule:subst-peps.cases)
  case 3 thus ?thesis using Atom by (auto simp: nolb-def EQ-def Ball-def)
qed (insert Atom, auto simp: nolb-def EQ-def Ball-def)

next
  case Or thus ?case by (simp add: Ball-def)
qed simp-all

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

lemma dense-interval:
assumes finite L l \( \in \) L l < x P(x::'a::dlo)
and dense: \( \\forall \) l y. \( \{\langle l, y\rangle | l<x\} \)
shows \( \exists \) l \( \in \) L l<x y l<y y<x \implies P y
proof
  let ?L = \{l\in L. l < x\}
  let ?ll = Max ?L
  have ?L \( \neq \) {} using \( \langle l, y\rangle | l<x\) by (blast intro:order-less-imp-le)
  hence \( \forall \) y. ?ll<y y<x \implies y \notin L using \( \langle l, y\rangle | l<x\) by force
  moreover have ?ll \in L
  proof
    show ?ll \in ?L using \( \langle l, y\rangle | l<x\) Max-in[of - ?L]
    show ?L \subseteq L by blast
  qed
  moreover have ?ll < x using \( \langle l, y\rangle | l<x\) by simp
  ultimately show ?thesis using \( \langle l, y\rangle | l<x\) by simp
  qed

lemma I-subst-peps:
\[ \text{nqfree } \varphi \implies DLO.I (\text{subst}_+ \varphi l) \text{ xs} \implies \]
\[ (\exists \text{leps}>xs!l. \forall x. xs!l < x \land x \leq \text{leps} \implies DLO.I \varphi (x#xs)) \]
proof (induct \( \varphi \))
  case TrueF thus ?case by simp (metis no-ub)
next

31
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)
next
  case 5 thus ?thesis using Atom by (auto)
next
  case 8 thus ?thesis using Atom by (auto)
qed (insert Atom, auto)
next
  case And thus ?case
    apply clarsimp
    apply(rule-tac x = min leps lepsa in exI)
    apply simp
    done
next
  case Or thus ?case by force
qed simp-all

definition
qe-eps_1(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-eps_1 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-eps_1-def amap-fm-list-disj)
  } moreover
  { assume ∀i ∈ set ?ebs. ¬DLO.I ϕ (xs!i # xs)
\[ \neg \text{DLO}.I \ (\text{inf}_- \varphi) \ x s \]

with \(\text{OF} \ (\text{nafree} \ \varphi) \ \text{obtain} \ l \ \text{where} \ \text{DLO}.I \ (\text{subst}_+ \ \varphi \ l) \ x s \)
by (fastforce simp \(\text{I-subst \ qe-eps}_1\)-def \(\text{set-lbounds} \ \text{set-lbounds}\))
then obtain \(\text{leps} \ \text{where} \ \text{DLO}.I \ \varphi \ (\text{leps}#xs)\)
using \(\text{I-subst-eps}_1\) by fastforce
hence \(\text{?EX} \ \ldots \ \}
ultimately show \(\text{?EX} \ \text{by blast}\)

next
let \(\text{?as} = \text{DLO}.\text{atoms}_0 \ \varphi \ \text{let} \ \text{?ebs} = \text{ebounds} \ ?\text{as} \)
assume \(\text{?EX} \)
then obtain \(x \ \text{where} \ \text{DLO}.I \ \varphi \ (x#xs) \ \ldots \)
\{ assume \(\text{DLO}.I \ (\text{inf}_- \varphi) \ x s \)
hence \(\text{?QE} \ \text{using} \ (\text{nafree} \ \varphi) \ \text{by} (\text{auto simp:qe-eps}_1\)-def) \}
moreover
\{ assume \(\exists k \ \in \ \text{set} \ ?\text{ebs} \ \text{DLO}.I \ (\text{subst} \ \varphi \ k) \ x s \)
hence \(\text{?QE} \ \text{by} (\text{auto simp:qe-eps}_1\)-def) \}
moreover
\{ assume \(\neg \text{DLO}.I \ (\text{inf}_- \varphi) \ x s \)
and \(\forall k \ \in \ \text{set} \ ?\text{ebs} \ \neg \text{DLO}.I \ (\text{subst} \ \varphi \ k) \ x s \)
hence \(\text{noE:} \ \forall e \ \in \ \text{EQ} \ \varphi \ xs. \ \neg \text{DLO}.I \ \varphi \ (e#xs) \ \text{using} \ (\text{nafree} \ \varphi) \)
by (auto simp: \(\text{set-lbounds} \ \text{set-ebounds} \ \text{EQ-def} \ \text{I-subst nth-Cons'} \ \text{split}\)-split-if-asm)
hence \(x \ \notin \ \text{EQ} \ \varphi \ xs \ \text{using} \ x \ \text{by fastforce} \)
obtain \(l \ \text{where} \ l \ \in \ \text{LB} \ \varphi \ xs \ l < x \)
using \(\text{LBex}[\text{OF} \ (\text{nafree} \ \varphi) \ x (\neg \text{DLO}.I.(\text{inf}_- \varphi) \ x s) \ (x \ \notin \ \text{EQ} \ \varphi \ xs)] \ \ldots \)
have \(\exists l \in \text{LB} \ \varphi \ xs. \ l < x \ \land \ \text{nolb} \ \varphi \ xs \ l x \ 
\ (\forall y. \ l < y \ \land \ y \leq x \ \rightarrow \ \text{DLO}.I \ \varphi \ (y#xs)) \)
using \(\text{dense-interval}[\text{where} \ P = \lambda x. \ \text{DLO}.I \ \varphi \ (x#xs), \ \text{OF} \ \text{finite-LB} \ \langle l \in \text{LB} \ \varphi \ xs \rangle \ (l < x) \ x \ \text{innermost-intel}[\text{OF} \ (\text{nafree} \ \varphi) \ \text{-} \ \neg \ (x \ \notin \ \text{EQ} \ \varphi \ xs)] \)
by (simp add: \(\text{nolb-def}\))
then obtain \(m \)
where \(\vdash \text{Less} \ (\text{Suc} \ m) \ 0 \ \in \ \text{set} \ ?a s \ \land \ x s!m < x \ \land \ \text{nolb} \ \varphi \ xs \ (xs!m) \ x \ 
\ (\forall y. \ x s!m < y \ \land \ y \leq x \ \rightarrow \ \text{DLO}.I \ \varphi \ (y#xs)) \)
by blast
then have \(\text{DLO}.I \ (\text{subst}_+ \ \varphi \ m) \ x s \)
using \(\text{noE} \ \text{by} (\text{auto intro!} \ \text{I-subst-eps}_2[\text{OF} \ (\text{nafree} \ \varphi)]) \)
with \(\ast \ \text{have} \ \text{?QE} \)
by (simp add: \(\text{qe-eps}_1\)-def \(\text{bez-Un} \ \text{set-lbounds} \ \text{set-ebounds}\) \ \text{metis})
\}
ultimately show \(\text{?QE} \ \text{by blast}\)

qed

lemma \(\text{qfree-asubst-eps}_1\): \(\text{qfree} \ \varphi \ \Longrightarrow \ \text{qfree} \ (\text{subst}_+ \ \varphi \ k) \)
by (induct \ \varphi) \ (\text{simp-all} \ \text{add} : \text{qfree-asubst-eps}_1\))

lemma \(\text{qfree-eps}_1\): \(\text{qfree} \ \varphi \ \Longrightarrow \ \text{qfree} \ (\text{qe-eps}_1 \ \varphi) \)
apply (simp add: \(\text{qe-eps}_1\)-def)
apply (rule \(\text{qfree-list-disj}\))
apply (auto simp: \(\text{qfree-min-inf} \ \text{qfree-asubst-eps} \ \text{qfree-map-fm}\))
definition $\text{qe-eps} = DLO.\text{.lift-nnf-qe}\ \text{qe-eps}_1$

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

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

end

4 Linear real arithmetic

theory LinArith
imports QE ~/~/src/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 o 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 \( R \) (\( \text{Less} \ r \ rs \)) = \( \text{Less} \ r \ (\text{tl} \ rs) \) |
\( \text{decr} \ R \) (\( \text{Eq} \ r \ rs \)) = \( \text{Eq} \ r \ (\text{tl} \ rs) \)

**fun** \( I_R :: \text{atom} \Rightarrow \text{real list} \Rightarrow \text{bool} \) where
\( I_R \) (\( \text{Less} \ r \ cs \)) \( xs = (r < (cs, xs)) \) |
\( I_R \) (\( \text{Eq} \ r \ cs \)) \( xs = (r = (cs, xs)) \)

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

**interpretation** \( R! :: \text{ATOM} \ \text{neg}_R (\lambda. \text{True}) \) \( I_R \ \text{depends}_R \ \text{decr}_R \)
where \( \text{ATOM}.\text{atoms}_0 \ \text{depends}_R = \text{atoms}_0 \)

**proof** –
**case** \( \text{goal}1 \)
**thus** ?case
**apply** (unfold-locales)
- **apply** (case-tac a)
- **apply** simp-all
- **apply** (case-tac a)
- **apply** simp-all
- **apply** arith
- **apply** arith
- **apply** (case-tac a)
- **apply** (simp-all add:depends_R-def split:list.splits)
done
**next**
**case** \( \text{goal}2 \)
**thus** ?case **by** (simp add:atoms_0-def)
**qed**

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

### 4.1.2 Shared constructions

**fun** \( \text{combine} :: (\text{real} \times \text{real list}) \Rightarrow (\text{real} \times \text{real list}) \Rightarrow \text{atom} \) where
\( \text{combine} \ (r_1, cs_1) \ (r_2, cs_2) = \text{Less} \ (r_1-r_2) \ (cs_2 - cs_1) \)

**definition** \( \text{lbounds} as = [(r/c, (-1/c) * s cs), \text{Less} \ r \ (c\#cs) \Irightarrow as, c>0] \)
**definition** \( \text{ubounds} as = [(r/c, (-1/c) * s cs), \text{Less} \ r \ (c\#cs) \Irightarrow as, c<0] \)
**definition** \( \text{ebounds} as = [(r/c, (-1/c) * s cs), \text{Eq} \ r \ (c\#cs) \Irightarrow as, c\neq 0] \)

**lemma** \( \text{set-lbounds} : \text{set}, \text{lbounds} as \) = \{ \( (r/c, (-1/c) * s cs) | r c cs. \text{Less} \ r \ (c\#cs) : \text{set} \ \wedge c>0 \} \)
**by** (force simp: lbounds-def split:list.splits atom.splits if-splits)
**lemma** \( \text{set-ubounds} : \text{set}, \text{ubounds} as \) = \{ \( (r/c, (-1/c) * s cs) | r c cs. \text{Less} \ r \ (c\#cs) : \text{set} \ \wedge c<0 \} \)
by (force simp: inbounds-def split:list.splits atom.splits if-splits)

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

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

abbreviation LB where
LB f xs \equiv \{(r - \langle cs, xs \rangle)/c | r c cs. Less r (c#cs) : set(R.atoms0 f) \land c > 0\}

abbreviation UB where
UB f xs \equiv \{(r - \langle cs, xs \rangle)/c | r c cs. Less r (c#cs) : set(R.atoms0 f) \land 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 [] []) = Less s [] |
asubst (r, cs) (Eq [] []) = Eq s []

abbreviation subst \phi rcs \equiv map fm (asubst rcs) \phi

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 \phi \Longrightarrow R.I (subst \phi t) xs = R.I \phi (eval t xs \# xs)
by (induct \phi)(simp-all add: I-asubst)

lemma I-subst-pretty:
qfree \phi \Longrightarrow R.I (subst \phi (r, cs)) xs = R.I \phi ((r + \langle cs, xs \rangle) \# xs)
by (simp add: I-subst eval-def)

fun min-inf :: atom fm => atom fm (inf -) where
inf - (And \ phi1 \ phi2) = and (inf - \ phi1) (inf - \ phi2) |
\[
\begin{align*}
\inf_-(\text{Or } \varphi_1 \varphi_2) &= \text{or} (\inf_- \varphi_1) (\inf_- \varphi_2) | \\
\inf_- (\text{Atom} (\text{Less } r (c \# cs))) &= \\
&\text{if } c < 0 \text{ then TrueF else if } c > 0 \text{ then FalseF else } \text{Atom} (\text{Less } r cs) | \\
\inf_- (\text{Atom} (\text{Eq } r (c \# cs))) &= \text{if } c = 0 \text{ then } \text{Atom} (\text{Eq } r cs) \text{ else FalseF}) | \\
\inf_- \varphi &= \varphi
\end{align*}
\]

fun plus-inf :: atom fm ⇒ atom fm (inf+) where
\[
\begin{align*}
\inf_+ (\text{And } \varphi_1 \varphi_2) &= \text{and} (\inf_+ \varphi_1) (\inf_+ \varphi_2) | \\
\inf_+ (\text{Or } \varphi_1 \varphi_2) &= \text{or} (\inf_+ \varphi_1) (\inf_+ \varphi_2) | \\
\inf_+ (\text{Atom} (\text{Less } r (c \# cs))) &= \\
&\text{if } c > 0 \text{ then TrueF else if } c < 0 \text{ then FalseF else } \text{Atom} (\text{Less } r cs) | \\
\inf_+ (\text{Atom} (\text{Eq } r (c \# cs))) &= \text{if } c = 0 \text{ then } \text{Atom} (\text{Eq } r cs) \text{ else FalseF}) | \\
\inf_+ \varphi &= \varphi
\end{align*}
\]

lemma qfree-min-inf: qfree \(\varphi\) ⇒ qfree(\inf_- \varphi)
by(induct \(\varphi\) rule:min-inf.induct) simp-all

lemma qfree-plus-inf: qfree \(\varphi\) ⇒ qfree(\inf_+ \varphi)
by(induct \(\varphi\) rule:plus-inf.induct) simp-all

lemma min-inf:
\[
\begin{align*}
n\text{qfree } f \Rightarrow \exists x. \forall y \leq x. R.I (\inf_- f) xs &= R.I (\inf_- f (y \# xs)) | \\
(R\text{-is } - \Rightarrow \exists x. \text{?P } f x)
\end{align*}
\]
proof(induct f)
case (Atom a)
show ?case
proof (cases a)
case (Less r cs)
show ?thesis
proof(cases cs)
case Nil thus ?thesis using Less by simp
next
case (Cons c cs)
{ assume c=0 hence ?thesis using Less Cons by simp }
moreover
{ assume c<0 hence ?P (Atom a) ((r - \langle cs, xs \rangle + 1)/c) using Less Cons
by(auto simp add: field-simps)
hence ?thesis .. }
moreover
{ assume c>0 hence ?P (Atom a) ((r - \langle cs, xs \rangle - 1)/c) using Less Cons
by(auto simp add: field-simps)
hence ?thesis .. }
ultimately show ?thesis by force
qed
next
case (Eq r cs)
show ?thesis
proof (cases cs)
  case Nil thus ?thesis using Eq by simp
next
  case (Cons c cs)
  { assume c=0 hence ?thesis using Eq by simp }
  moreover
  { assume c<0 hence ?thesis using Eq Cons
    by (auto simp add: field-simps)
  }
  moreover
  { assume c>0 hence ?thesis using Eq Cons
    by (auto simp add: field-simps)
  }
  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:
  \(\text{nqfree } f \implies \exists x. \forall y \geq x. R.I (\inf f) \ x s = R.I f \ (y \# \ x s)\)
  (is - \implies \exists x. ?P f x)

proof (induct f)
  case (Atom a)
  show ?case
  proof (cases a)
    case (Less r cs)
    show ?thesis
  proof (cases cs)
    case Nil thus ?thesis using Less by simp
next
    case (Cons c cs)
    { assume c=0 hence ?thesis using Less Cons by simp }
    moreover
    { assume c<0 hence ?thesis using Less Cons
      by (auto simp add: field-simps)
    }
  qed
moreover
{ assume \( c > 0 \)
  hence \(?P\ (\text{Atom } a) \ ((r - \langle cs, xs \rangle) + 1)/c\) using \(\text{Less Cons}\)
  by (auto simp add: \(\text{field-simps}\) )
  hence \(?thesis\ ..\ )
} ultimately show \(?thesis\ by\ force\)
qed

next
case \((\text{Eq } r \ cs)\)
show \(?thesis\)
proof (cases \(cs\) )
case \(\text{Nil}\) thus \(?thesis\) using \(\text{Eq}\) by simp
next
case \((\text{Cons } c \ cs)\)
{ assume \( c = 0 \) hence \(?thesis\ using \(\text{Eq}\) by simp \) }
moreover
{ assume \( c < 0 \)
  hence \(?P\ (\text{Atom } a) \ ((r - \langle cs, xs \rangle) - 1)/c\) using \(\text{Eq Cons}\)
  by (auto simp add: \(\text{field-simps}\) )
  hence \(?thesis\ ..\ )
} moreover
{ assume \( c > 0 \)
  hence \(?P\ (\text{Atom } a) \ ((r - \langle cs, xs \rangle) + 1)/c\) using \(\text{Eq Cons}\)
  by (auto simp add: \(\text{field-simps}\) )
  hence \(?thesis\ ..\ )
} ultimately show \(?thesis\ by\ force\)
qed

next
case \((\text{And } f_1 f_2)\)
then obtain \(x_1 x_2\) where \(?P\ f_1 x_1 \ ?P\ f_2 x_2\) by \(\text{fastforce}\)+
hence \(?P\ (\text{And } f_1 f_2) \ (\text{max } x_1 x_2)\) by (force simp: \(\text{and-def}\) )
thus \(\text{case } ..\)
next
case \((\text{Or } f_1 f_2)\)
then obtain \(x_1 x_2\) where \(?P\ f_1 x_1 \ ?P\ f_2 x_2\) by \(\text{fastforce}\)+
hence \(?P\ (\text{Or } f_1 f_2) \ (\text{max } x_1 x_2)\) by (force simp: \(\text{or-def}\) )
thus \(\text{case } ..\)
qed simp-all

declare \([\text{simp-depth-limit = 4}]\)

lemma \(\text{LBex}\): 
\[
\begin{array}{rcl}
  & \begin{array}{l}
    \text{nfree } f; \text{R.I } f \ (x \# xs); \, \neg \text{R.I } (\text{inf } f) \ xs; \, x \notin EQ f xs \\

  \Rightarrow \exists l \in LB f xs. \ l < x
\end{array} \\
\end{array}
\]
apply (induct \(f\) )
apply simp
apply simp

39
apply (case-tac a)
apply(auto simp add: dependsR-def field-simps split:if-splits list.splits)
apply fastforce+
done

lemma UBex:
\[
\text{\texttt{\texttt{context-free \textbf{f}; R.I (x\#xs); \neg R.I (inf_+ f) \; xs; x \notin EQ f \; xs}} \Rightarrow \exists u \in UB f \; xs. \; x < u}
\]
apply(induct f)
apply simp
apply simp
apply(case-tac a)
apply(auto simp add: dependsR-def field-simps split:if-splits list.splits)
apply fastforce+
done

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

lemma finite-LB: finite(LB f xs)
proof
  have LB f xs = (\lambda(r,cs). r + \langle cs, xs \rangle) ' set(lbounds(R.atoms0 f))
    by (force simp: set-lbounds image-def field-simps)
  thus ?thesis by simp
qed

lemma finite-UB: finite(UB f xs)
proof
  have UB f xs = (\lambda(r,cs). r + \langle cs, xs \rangle) ' set(ubounds(R.atoms0 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 \Rightarrow atom fm where
qe-FM_1 as = list-conj [Atom(combine p q), pt-lbounds as, qe-ubounds as]

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

let \(?Ls = set(\text{bounds as})\) let \(?Us = set(\text{ubounds as})\)

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

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

have \(\text{fins: finite ?lbs} \land \text{finite ?ubs by auto}\)

have \(2: \forall f \in \text{set as}. \exists r \in cs. f = \text{Less} r (c \# cs) \land \) \(c > 0 \land (r/c, (-1/c) \times cs) \in ?Ls \lor c < 0 \land (r/c, (-1/c) \times cs) \in ?Us\)

using \(\text{dep less}\)

by (\(\text{fastforce simp: set-bounds set-ubounds is-Less-iff depends_R-def}\)

\(\text{split: list splits}\))

assume \(?L\)

have \(1: \forall x \in ?lbs. \forall y \in ?ubs. x < y\)

proof (\(\text{rule ballI}+)\)

\(\text{fix} \ x \ y \ \text{assume} \ x \in ?lbs \ \lor \ y \in ?ubs\)

then obtain \(r \in cs\)

where \((r, cs) \in ?Ls \land x = r + \langle cs, xs\rangle\) by \(\text{fastforce}\)

moreover from \((y \in ?ubs) \ \text{obtain} \ s \in ds\)

where \((s, ds) \in ?Us \land y = s + \langle ds, xs\rangle\) by \(\text{fastforce}\)

ultimately show \(x < y\)

using \(\langle ?L\rangle\)

by (\(\text{fastforce simp: qc-FM1-def algebra-simps iprod-left-diff-distrib}\))

qed

\{ assume \(\text{nonempty: ?lbs \neq \{\} \land ?ubs \neq \{\}\}\)

hence \(\text{Max ?lbs < Min ?ubs using fins 1}\)

by (\(\text{blast intro: Max-less-iff THEN iffD2 Min-gr-iff THEN iffD2}\))

then obtain \(m \in ?ubs\)

where \(\text{Max ?lbs < m} \land m < \text{Min ?ubs}\)

using dense [where \(\text{'a = real}\)] by \(\text{blast}\)

hence \(\forall a \in \text{set as}. I_R a (m \# xs)\)

using \(\text{2 nonempty}\)

by (\(\text{auto simp: Ball-def Bex-def (fastforce simp: field-simps)}\))

hence \(?R .. \)\}

moreover

\{ assume \(\text{asm: ?lbs = \{\} \land ?ubs = \{\}\}\)

have \(\forall a \in \text{set as}. I_R a ((\text{Max ?lbs + 1}) \# xs)\)

proof

fix \(a\) assume \(a \in \text{set as}\)

then obtain \(r \in cs\)

where \(a = \text{Less} r (c \# cs) c > 0 \ (r/c, (-1/c) \times cs) \in ?Ls\)

using \(\text{asm 2 by fastforce}\)

moreover hence \(r - \langle cs, xs\rangle / c \leq \text{Max ?lbs}\)

using \(\text{asm fins}\)

by (\(\text{auto intro!: Max-ge-iff THEN iffD2}\))

(force simp add: field-simps)

ultimately show \(I_R a ((\text{Max ?lbs + 1}) \# xs)\)

by (simp add: field-simps)

qed

hence \(?R .. \)\}

moreover

\{ assume \(\text{asm: ?lbs = \{\} \land ?ubs = \{\}\}\)

have \(\forall a \in \text{set as}. I_R a ((\text{Min ?ubs - 1}) \# xs)\)

proof

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

\textbf{corollary} \text{I-\text{qe-FM}_1-pretty:}
\forall a \in \text{set as}. \text{is-Less } a \land \text{depends}_R a \implies R.\text{is-dnf-qe \text{qe-FM}_1} as
by(metis \text{I-\text{qe-FM}_1})

\textbf{fun} \text{subst}_0 :: \text{atom} \Rightarrow \text{atom} \Rightarrow \text{atom} where
\text{subst}_0 (\text{Eq } r (c \cdot cs)) a = (\text{case } a \text{ of }
  \text{Less } s (d \cdot ds) \Rightarrow \text{Less } (s - (r \cdot d) / c) (ds - (d / c) \cdot cs)
  | \text{Eq } s (d \cdot ds) \Rightarrow \text{Eq } (s - (r \cdot d) / c) (ds - (d / c) \cdot cs))

\textbf{lemma} \text{subst}_0\cdot\text{-pretty:}
\text{subst}_0 (\text{Eq } r (c \cdot cs)) \langle \text{Less } s (d \cdot ds) \rangle = \text{Less } (s - (r \cdot d) / c) (ds - (d / c) \cdot cs)
\text{subst}_0 (\text{Eq } r (c \cdot cs)) \langle \text{Eq } s (d \cdot ds) \rangle = \text{Eq } (s - (r \cdot d) / c) (ds - (d / c) \cdot cs)
by auto

\textbf{lemma} \text{I-subst}_0: \text{depends}_R a \implies c \neq 0 \implies
I_R (\text{subst}_0 (\text{Eq } r (c \cdot cs)) a) xs = I_R a \langle (r - \langle cs, xs \rangle) / c \# xs \rangle
apply(cases a)
by (auto simp add: \text{depends}_R\cdot\text{-def iprod-left-diff-distrib algebra-simps diff-divide-distrib)
interpretation $R_c$:

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

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

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

apply (unfold-locales)

apply (simp del: subst, simps add: I-subst, split: atom.splits list.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 = $R_c$.lift-dnfeq-qe qe-FM

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

corollary I-qe-FM: $R_I$ (qe-FM $\varphi$) $xs$ = $R_I$ $\varphi$ $xs$

unfolding qe-FM-def
apply (rule $R_c$.I-lift-dnfeq-qe)

apply (rule qfree-qe-FM)

apply (rule allI)

apply (rule I-qe-FM)

prefer 2 apply blast

apply (clarify)

apply (drule-tac $x$ $= a$ in bspec) apply simp

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

done

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

4.2.1 Tests

lemmas qesims = qe-FM-def $R_c$.lift-dnfeq-qe-def $R_c$.lift-eq-qe-def $R_c$.qelim-def qe-FM-def

lbounds-def ubounds-def list-conj-def list-disj-def and-def or-def dependsR-def

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

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 QEin-opt
imports QEin
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 True else False
else Atom a) |
Eq r cs ⇒ (if ∀ c ∈ set cs. c = 0
then if r = 0 then True else False
else Atom a))

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

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

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

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

definition qe-FM0 = R_e.lift-dnfeq-qe qe-FMo1

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

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

**unfolding** qe-FMo-def

apply(rule R_e.I-lift-dnfeq-qe)

apply(rule qfree-qe-FMo1)

apply(rule allI)

apply(subst I-qe-FMo)

prefer 2 apply blast

apply(clarify)

apply(drule-tac x=a in bspec) apply simp

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

done

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

by(simp add:qe-FMo-def R_e.qfree-lift-dnfeq-qe qfree-qe-FMo1)

end

theory FRE

imports LinArith

begin

4.3 Ferrante-Rackoff

This section formalizes a slight variant of Ferrante and Rackoff’s algorithm [2].

We consider equalities separately, which improves performance.

fun between :: real * real list ⇒ real * real list ⇒ real * real list

where between (r,cs) (s,ds) = ((r+s)/2, (1/2) * (cs+ds))

definition FR1 :: atom fm ⇒ atom fm where

\( FR_1 \varphi = \)

(let as = R.atoms0 \( \varphi \); lbs = lbounds as; ubs = ubounds as; ebs = ebounds as;

intrs = [subst \( \varphi \) (between l u) . l ← lbs, u ← u bs]

in list-disj (inf - \( \varphi \) # inf + \( \varphi \) # intrs @ map (subst \( \varphi \)) ebs))

lemma dense-interval:

assumes finite L finite U l : L l < x x < u P(x::real)

and dense: \( \forall y l u. [ \forall y \{l<x\}, y \notin L; \forall y \{x<u\}, y \notin U; \]

\( l<x; x<u; l<y; y<u \] → P y)

shows \( \exists l \in L. \exists u \in U. l<u \land (\forall y. l<y \land y<u \rightarrow P y) \)

proof –

let \( ?L = \{l:L. l < x\} \) let \( ?U = \{u:U. x < u\} \)

let \( ?l = \text{Max } ?L \) let \( ?u = \text{Min } ?U \)

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

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

45
ultimately have \( \forall y. \ ?ll < y \land y < x \implies y \notin L \forall y. \ x < y \land y < ?uu \implies y \notin U \)

moreover have \( ?ll : L \)

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

moreover have \( ?uu : U \)

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

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

qed

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

lemma dense:
\[
\begin{align*}
  \neg \text{qfree } f; \forall y \in \{ l <..< x \}. \ y \notin LB f xs; \forall y \in \{ x <..< u \}. \ y \notin UB f xs; \\
  l < x; x < u; x \notin EQ f xs; \ R.1 \ f (x#xs); \ l < y; y < u \\
\implies R.1 \ f (y#xs)
\end{align*}
\]

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 \( R.1 \)-def)
    next
      case (Cons c cs)
      hence \( r < c\ast x + \langle cs, xs \rangle \) using Atom Less by simp
      \{ assume \( c=0 \) hence ?thesis using Atom Less Cons by simp \} 
      moreover
      \{ assume \( c<0 \) \\
      hence \( x < (r - \langle cs, xs \rangle)/c \) using \( r < c\ast x + \langle cs, xs \rangle \) \( \text{by (simp add: field-simps)} \) \\
      have ?thesis
      proof (rule ccontr)
        assume \( \neg R.1 \ (Atom \ a) \ (y#xs) \) \\
        hence \( ?u \leq y \) using Atom Less Cons \( \langle c<0 \rangle \) \\
        by (auto simp add: field-simps)
      hence \( ?u < u \) using \( y<u \) by simp
      with \( c<0 \) show False using Atom Less Cons \( \langle c<0 \rangle \) \\
      by (auto simp: depends \( R.1 \)-def)
    
    qed
  qed

  qed

next
  case (Cons c cs)
  hence \( r < c\ast x + \langle cs, xs \rangle \) using Atom Less by simp
  \{ assume \( c=0 \) hence ?thesis using Atom Less Cons by simp \} 
  moreover
  \{ assume \( c<0 \) \\
  hence \( x < (r - \langle cs, xs \rangle)/c \) using \( r < c\ast x + \langle cs, xs \rangle \) \( \text{by (simp add: field-simps)} \) \\
  have ?thesis
  proof (rule ccontr)
    assume \( \neg R.1 \ (Atom \ a) \ (y#xs) \) \\
    hence ?u \leq y using Atom Less Cons \( \langle c<0 \rangle \) \\
    by (auto simp add: field-simps)
  hence ?u < u using \( y<u \) by simp
  with \( c<0 \) show False using Atom Less Cons \( \langle c<0 \rangle \) \\
  by (auto simp: depends \( R.1 \)-def)
\end{align*}
\]

46
qed } moreover
{ assume \( c > 0 \)
  hence \( x \geq \frac{(r - \langle cs, xs \rangle)}{c} \) (is \( > \) ?l) using \( (r < c * x + \langle cs, xs \rangle) \)
  by (simp add: field-simps)
have ?thesis
proof (rule ccontr)
  assume \( \neg R.I \ (Atom \ a) \ (y \# xs) \)
  hence \( ?l \geq y \) using Atom Less Cons \( (c > 0) \)
  by (auto simp add: field-simps)
  hence \( ?l > l \) using \( (y > b) \) by simp
  with \( (?l < x) \) show False using Atom Cons \( (c > 0) \)
  by (auto simp: dependsR_def)
qed }
ultimately show ?thesis by force
qed

next
case (Eq r cs)
show ?thesis
proof (cases cs)
case Nil thus ?thesis using Atom Eq by (simp add: dependsR_def)
next
case (Cons c cs)
  hence \( r = c * x + \langle cs, xs \rangle \) using Atom Eq by simp
  { assume \( c = 0 \)
    hence ?thesis using Atom Eq Cons by simp }
  moreover
  { assume \( c \neq 0 \)
    hence ?thesis using \( (r = c * x + \langle cs, xs \rangle) \) Atom Eq Cons \( (l < y) \) \( (y < u) \)
    by(auto simp: ac-simps dependsR_def split:if_splits) }
ultimately show ?thesis by force
qed

next
case (And f1 f2) thus ?case by (fastforce simp: Ball-def)
next
case (Or f1 f2) thus ?case by (fastforce simp: Ball-def)
qed fastforce

theorem I-FR1:
assumes \( \text{nqfree } \varphi \) shows \( R.I \ (FR_1 \ \varphi) \ xs = (\exists x. \ R.I \ \varphi \ (x \# xs)) \)
(is \( ?FR = ?EX \))
proof
assume \( ?FR \)
{ assume \( R.I \ (\inf - \varphi) \ xs \)
  hence \( ?EX \) using \( (?FR) \ ) min-inf[OF \ (\text{nqfree } \varphi), \ where \ xs=x] \)
  by(auto simp add:FR1_def)
} moreover
{ assume \( R.I \ (\inf + \varphi) \ xs \)
  hence \( ?EX \) using \( (?FR) \ ) plus-inf[OF \ (\text{nqfree } \varphi), \ where \ xs=x] \)
  by(auto simp add:FR1_def)

47
moreover
{ assume $\exists x \in \text{EQ} \varphi \, \text{xs}$, $R.I \, \varphi \, (x\#\text{xs})$

hence $\exists \text{EX}$ using $\exists \text{FR}$ by(auto simp add:FR1-def)

} moreover
{ assume $\neg R.I \, (\inf_- \varphi) \, \text{xs}$ \& $\neg R.I \, (\inf_+ \varphi) \, \text{xs}$ \&

$(\forall x \in \text{EQ} \varphi \, \text{xs}) \, \neg R.I \, \varphi \, (x\#\text{xs})$

with $\exists \text{FR}$ obtain $r \, \text{cs} \, \text{ds}$

where $R.I \, (\text{subst} \, (\text{between} \, (r, \text{cs}) \, (s, \text{ds})) \, \text{xs})$

by(auto simp: FR1-def eval-def
diff-divide-distrib set-ebounds l-subst \langle nqfree \varphi \rangle blast

hence $R.I \, \varphi \, (\text{eval} \, (\text{between} \, (r, \text{cs}) \, (s, \text{ds})) \, \text{xs} \# \, \text{xs})$

by(simp add: l-subst \langle nqfree \varphi \rangle)

hence $\exists \text{EX}$ .. }

ultimately show $\exists \text{EX}$ by blast

next
assume $\exists \text{EX}$

then obtain $x$ where $x: R.I \, \varphi \, (x\#\text{xs})$ ..

{ assume $R.I \, (\inf_- \varphi) \, \text{xs} \nu R.I \, (\inf_+ \varphi) \, \text{xs}$

hence $\exists \text{FR}$ by(auto simp:FR1-def)

} moreover
{ assume $\neg R.I \, (\inf_- \varphi) \, \text{xs}$ \& $\neg R.I \, (\inf_+ \varphi) \, \text{xs}$ \& $x \notin \text{EQ} \, \varphi \, \text{xs}$

obtain $l$ where $l: \text{LB} \, \varphi \, \text{xs} \, l < x$

using LBex[OF \langle nqfree \varphi \rangle \, \text{xs} \, \langle x \notin \text{EQ} \, \varphi \, \text{xs} \rangle] ..

obtain $u$ where $u: \text{UB} \, \varphi \, \text{xs} \, u < x$

using UBex[OF \langle nqfree \varphi \rangle \, \text{xs} \, \langle x \notin \text{EQ} \, \varphi \, \text{xs} \rangle] ..

have $\exists l \, \text{LB} \, \varphi \, \text{xs}, \, \exists u \in \text{UB} \, \varphi \, \text{xs}, \, l < u \land (\forall y, \, l < y \land y < u \rightarrow R.I \, \varphi \, (y\#\text{xs}))$

using dense-interval[where $P = \lambda x. R.I \, \varphi \, (x\#\text{xs})$, \text{OF} \, \text{finite-LB} \, \text{finite-UB}

l:LB \, \varphi \, \text{xs} \, \langle u: \text{UB} \, \varphi \, \text{xs} \rangle \, \langle l < x \rangle \, \langle x < u \rangle \, x \, \text{dense}[\text{OF} \, \langle \text{nqfree} \, \varphi \rangle, \ldots, \langle x \notin \text{EQ} \, \varphi \, \text{xs} \rangle] \, \text{by simp

then obtain $r \, \text{c} \, \text{cs} \, \text{sd} \, \text{ds}$

where $\text{Less} \, r \, (\text{c} \# \text{cs}) : \text{set} \, (\text{R.atoms}_0 \, \varphi)$ \text{Less} \, $s \, (d \# \text{ds}) : \text{set} \, (\text{R.atoms}_0 \, \varphi)$

$(\forall y. \, (r - \langle \text{cs}, \text{xs} \rangle) / c < y \rightarrow y < (s - \langle \text{ds}, \text{xs} \rangle) / d \rightarrow R.I \, \varphi \, (y \# \text{xs})$

and $*: \, c > 0 \, d < 0 \, (r - \langle \text{cs}, \text{xs} \rangle) / c < (s - \langle \text{ds}, \text{xs} \rangle) / d$

by blast

moreover
have $(r - \langle \text{cs}, \text{xs} \rangle) / c < \text{eval} \, (\text{between} \, (r / c, \, (\text{\text{-}1} / c) \, * s \, \text{cs}) \, (s / d, \, (\text{\text{-}1} / d) \, * s \, \text{ds}) \, \text{xs} \, \text{is} \, \text{?P}$

and eval (between (r / c, (\text{-}1 / c) * s cs) (s / d, (\text{-}1 / d) * s ds)) xs < (s - \langle ds, xs \rangle) / d \, (\text{is} \, \text{?Q})

proof -

from * have [simp]: $c \ast (c \ast (d \ast (d \ast 4))) > 0$ by (auto simp add: sign-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)
ultimately show ?FR by blast
qed

definition FR = R.lift-nnf-qe FR1

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

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

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

end

theory QElin-inf
imports LinArith
begin

4.4 Quantifier elimination with infinitesimals

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

fun asubst-peps :: real * real list ⇒ atom ⇒ atom fm (asubst+)
where
asubst-peps (r, cs) (Less s (d#ds)) =
  (if d=0 then Atom(Less s ds) else
    let u = s - d*r; v = d * s + 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 ⇒ real * real list ⇒ atom fm ( subst+)
where subst+ ϕ rcs ≡ amap fm (asubst+ rcs) ϕ

definition nolb f xs l x = (∀ y∈{l<..<x}, y /∈ LB f xs)

lemma nolb-And[simp]:
nolb (And f g) xs l x = (nolb f xs l x ∧ 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

declare[[simp-depth-limit=4]]

lemma innermost-intvl:
[ nafree 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 (Less r cs)[simp]
show ?thesis
proof (cases cs)
case Nil thus ?thesis using Atom by (simp add:dependsR-def)
next
case (Cons c cs)[simp]
hence r < c+x + ⟨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:dependsR-def)
  qed } moreover
{ assume c>0
  hence x > (r − ⟨cs,xs⟩)/c (is - > ?l) using ⟨r < c+x + ⟨cs,xs⟩

by (simp add: field-simps)
then have \( ?! < y \) 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 (Eq r cs)[simp]
show \( ?thesis \)
proof (cases cs)
case Nil thus \( ?thesis \) using Atom by (simp add: depends_R-def)
next
case (Cons c cs)[simp]
hence \( r = c*x + \langle cs, xs \rangle \) using Atom by simp
  { assume c=0 hence \( ?thesis \) using Atom by simp }
moreover
  { assume c\#0
    hence \( ?thesis \) using \( \langle r = c*x + \langle cs, xs \rangle \rangle \) Atom
      by(auto simp: ac-simps 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]: \( EQ2 \ (Or \ f \ g) \ xs = (EQ2 \ f \ xs \ Un \ EQ2 \ g \ xs) \)
by(auto simp:EQ2-def)

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

lemma innermost-intvl2:
\[
\begin{align*}
& \text{nffree } f; \ nolb \ f \ xs \ l \ x; \ l < x; \ x \notin EQ2 \ f \ xs; \ R.I \ f \ (x\#xs); \ l \ < \ y; \ y \leq x
\Rightarrow R.I \ f \ (y\#xs)
\end{align*}
\]

unfolding \( EQ2\)-def by(blast intro:innermost-intvl)

lemma \( I\)-subst-peps2:
\[
\begin{align*}
& \text{nffree } f \Rightarrow r+\langle cs, xs \rangle < x \Rightarrow nolb \ f \ xs \ (r+\langle cs, xs \rangle) x
\Rightarrow \forall y \in \{r+\langle cs, xs \rangle <.. x \}. \ R.I \ f \ (y\#xs) \wedge y \notin EQ2 \ f \ xs
\Rightarrow R.I \ (\text{subst}+ \ f \ (r, cs)) \ xs
\end{align*}
\]

proof(induct \( f \))
case FalseF thus \( ?case \)
  by simp (metis linorder-antisym-conv1 linorder-neq-iff)
next
  case (Atom a)
  show ?case
  proof (cases ((r, cs), a) rule: asubst-peps_cases)
  case (1 (r cs s d ds)
  { assume d=0 hence ?thesis using Atom 1 by auto }
  moreover
  { assume d<0
    have s < d*x + ⟨ds, xs⟩ using Atom 1 by simp
    moreover have d*x < d*(r + ⟨cs, xs⟩) using ⟨d<0⟩ Atom 1
    by (simp add: mult-strict-left-mono-neg)
    ultimately have s < d * (r + ⟨cs, xs⟩) + ⟨ds, xs⟩ by (simp add: algebra-simps)
    hence ?thesis using Atom 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)
  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 Ball-def field-simps split: split-if-asm)
qed (insert Atom, auto)
next
  case Or thus ?case by simp (metis order-refl innermost-intvl2)
qed simp-all
declare[[simp-depth-limit=50]]

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 $a < x \land x < b$ by (metis dense)
hence $\forall y. a < y \land y \leq x \rightarrow s < d\cdot y + (d\cdot s,x)$
    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 $?a = s - d \cdot r$ let $?b = (d \cdot s \cdot c) + (d \cdot s,x)$
assume $d > 0$
with Atom 1 have $?a < ?b \lor ?a = ?b$ 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 $r + (c\cdot s,x) < x \land x \leq r + (c\cdot s,x) + 1$
    hence $d\cdot (r + (c\cdot s,x)) < d\cdot x$
      using $(d > 0)$ by (metis mult-strict-left-mono)
    hence $s < d\cdot x + (d\cdot s, x)$ 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:split-if-asm intro: less-add-one)
next
case And thus $?case$
  apply clarsimp
  apply (rule-tac $x = \min l \leq a$ lepsa in $exf$)
  apply simp
  done
next
case Or thus $?case$ by force
qed simp-all

lemma dense-interval:
assumes finite $L$ $l \in L$ $l < x$ $P(x::real)$
and dense: $\forall y l. [ \forall y \in l < x \cdot y \notin L \land l < y \land y \leq x ] \Longrightarrow P y$
shows $\exists l \in L. l < x \land (\forall y \in l < x \cdot y \notin L) \land (\forall y. l < y \land y \leq x \rightarrow P y)$
proof
  let $?L = \{ l \in L. l < x \}$
  let $?l = \text{Max} \ ?L$
  have $?L \neq \{ \}$ using $(l \in L \land l < x)$ by (blast intro:order-less-imp-le)
  hence $\forall y. ?l < y \land y < x \rightarrow y \notin L$ using $(\text{finite} \ ?L)$ by force
  moreover have $?l \in L$
proof
  show \( \forall l \in \mathcal{L} \) using \((\text{finite } \mathcal{L} \land \text{Max-in}[\text{OF} - \{\mathcal{L}\}])\) by simp
  show \( \mathcal{L} \subseteq \mathcal{L} \) by blast
  qed
moreover have \( \forall l < x \) using \((\text{finite } \mathcal{L} \land \{\mathcal{L}\} \neq \{\})\) by simp
ultimately show \( \text{thesis} \) using \((l < x) \land \{\mathcal{L}\} \neq \{\})\)
  by(blast intro!: dense GreaterThanLessThan_iff [THEN iffD1])
qed

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

theorem I-eps1:
  assumes \( nqfree f \) shows \( R.1 \ (\text{qe-eps}_1 f) \ xs = (\exists x. R.1 f (x#xs)) \)
  (is \( \text{?QE} = \text{?EX} \))
proof
  let \( \text{?as} = R.\text{atoms}_0 f \) let \( \text{?ebs} = \text{ebounds ?as} \)
  assume \( \text{?QE} \)
  \{ assume \( R.1 (\text{inf}_- f) \) \xs \\
  \quad hence \( \text{?EX \ using} \ (\text{?QE \ min-inf [of f xs]} \ nqfree f) \) \\
  \quad by(auto simp add:qe-eps1_def amap-fm-list-disj) \\
  \} moreover
  \{ assume \( \forall x \in \text{EQ f xs}. \neg R.1 f (x#xs) \) \\
  \quad \( \sim R.1 (\text{inf}_- f) \) \xs \\
  \quad with \( \text{?QE \ nqfree f} \) obtain \( r \) \( cs \) where \( R.1 (\text{subst}_+ f (r,cs)) \) \xs \\
  \quad by (fastforce simp: qe-eps1_def set-ebounds diff-divide-distrib eval-def I-subst nqfree f) \\
  \quad then obtain \( \text{leps} \) where \( R.1 f (\text{leps}#xs) \) \\
  \quad using \( \text{I-subst-peps[OF nqfree f]} \) by fastforce \\
  \quad hence \( \text{?EX ..} \) \}
ultimately show \( \text{?EX \ by blast} \)
next
let \( \text{?as} = R.\text{atoms}_0 f \) let \( \text{?ebs} = \text{ebounds ?as} \)
assume \( \text{?EX} \)
then obtain \( x \) where \( x: R.1 f (x#xs) \) ..
  \{ assume \( R.1 (\text{inf}_- f) \) \xs \\
  \quad hence \( \text{?QE \ using} \ nqfree f \) by(auto simp:qe-eps1_def) \\
  \} moreover
  \{ assume \( \exists rcs \in \text{set ?ebs}. R.1 (\text{subst} f rcs) \) \xs \\
  \quad hence \( \text{?QE \ by(auto simp:qe-eps1_def)} \) \}
  \quad moreover
  \{ assume \( \neg R.1 (\text{inf}_- f) \) \xs \\
  \quad and \( \forall rcs \in \text{set ?ebs}. \sim R.1 (\text{subst} f rcs) \) \xs \\
  \quad hence \( \text{noE}: \forall e \in \text{EQ f xs}. \sim R.1 f (e#xs) \) \( \text{using} \ nqfree f \) \\
  \quad by (force simp: set-ebounds I-subst diff-divide-distrib eval-def split:split-if-asm) \\
  \quad hence \( x \notin \text{EQ f xs} \) using \( x \) by fastforce \\
  \quad obtain \( l \) where \( l \in \text{LB f xs} l < x \) \\
  \quad using LBex[OF nqfree f] \( (\sim R.1(\text{inf}_- f) \) \xs \( (x \notin \text{EQ f xs}) \) ..
  \}
\]

have \( \exists l \in LB \, f \, xs \cdot l < x \wedge \text{nolb} \, f \, xs \, l \wedge \)
\((\forall y. \, l < y \wedge y \leq x \rightarrow (\text{R.I} \, f \, (y \# xs)))\)
using dense-interval\(\) where \( P = \lambda x. \, \text{R.I} \, f \, (x \# xs) \), \( OF \, \text{finite-LB} \, (l \in LB \, f \, xs) \)
\(\langle l < x \rangle \) \( x \) innermost-intvl\(\) \( [\text{OF} \, \langle \text{nqfree} \, f \rangle \cdot \cdot (x \notin \text{EQ} \, f \, xs)]\)
by (simp add: nolb-def)
then obtain \( r \, c \, cs \) where \( \ast: \, \text{Less} \, r \, (c \# cs) \in \text{set}(\text{R.atoms}_0 \, f) \wedge c > 0 \wedge \)
\((r - \langle cs, xs\rangle)/c < x \wedge \text{nolb} \, f \, xs \, ((r - \langle cs, xs\rangle)/c) \, x \wedge (\forall y. \, (r - \langle cs, xs\rangle)/c < y \wedge y \leq x \rightarrow (\text{R.I} \, f \, (y \# xs)))\)
by blast
then have \( \text{R.I} \, (\text{subst}_+ \, f \, (r/c, \, ((-1/c) \ast c) \, cs)) \, xs \) using noE
by(auto intro!: I-subst-eps2[of \( \langle \text{nqfree} \, f \rangle \)]
simp:EQ2-def diff-divide-distrib algebra-simps)
with \( \ast \) have \( ?QE \)
by(simp add:qe-eps1-def bex-Un set-lbounds) metis
} ultimately show \( ?QE \) by blast
qed

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

lemma qfree-subst-peps: nqfree \( \varphi \) \( \Rightarrow \) qfree (subst_+ \, \varphi \, \text{rcs})
by(induct \( \varphi \) ) (simp-all add:qfree-asubst-peps)

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

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

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

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

end

5 Presburger arithmetic

theory PresArith
imports GCD QE ~~/src/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 ⇒ 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 ⇒ int where
  hd-coeff (Le i ks) = (case ks of [] ⇒ 0 | k#_ ⇒ k) |
  hd-coeff (Dvd d i ks) = (case ks of [] ⇒ 0 | k#_ ⇒ k) |
  hd-coeff (NDvd d i ks) = (case ks of [] ⇒ 0 | k#_ ⇒ k)

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

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

interpretation Z!:
  ATOM neg Z (λa. divisor a ≠ 0) I Z (λa. hd-coeff a ≠ 0) decr Z
  where ATOM.atoms0 (λa. hd-coeff a ≠ 0) = atoms0

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

setup ⟨⟨ Sign.revert-abbrev @\{const-abbrev Z.I\} ⟩⟩
setup ⟨⟨ Sign.revert-abbrev @\{const-abbrev Z.lift-dnf-qe\} ⟩⟩

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

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

abbreviation subst :: int ⇒ int list ⇒ atom fm ⇒ atom fm
where subst i ks ⇓ map fm (asubst i ks)

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

lemma I-subst:
qfree 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:
\[
\text{set}(\text{ubounds as}) = \{(i, k) | i < k < 0\}
\]

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

\begin{verbatim}
lemma lbounds-append[simp]: lbounds(as @ bs) = lbounds as @ lbounds bs
by (simp add:lbounds-def)
\end{verbatim}

5.2 LCM and lemmas

\begin{verbatim}
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)
\end{verbatim}

5.3 Setting coefficients to 1 or -1

\begin{verbatim}
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
\end{verbatim}

The def of \textit{hd-coeff1} on \textit{Dvd} and \textit{NDvd} is different from the \textit{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:

\begin{verbatim}
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 (metis dvd-minus2-eq-if dvd-eq-mod-eq-0 THEN iffD1 algebra-simps)
apply (metis diff-convs-conv-left conv-left commute dvd-minus-iff minus-add-distrib)
apply (metis dvd-minus2-eq-if dvd-eq-mod-eq-0 THEN iffD1 algebra-simps)
apply (metis diff-convs-conv-left conv-left commute dvd-minus-iff minus-add-distrib)
\end{verbatim}

58
lemma I-hd-coeff1-mult-a: assumes $m > 0$
shows $\text{hd-coeff a dvd m} \iff I_Z (\text{hd-coeff1 m a}) (m \ast x \# xs) = I_Z a (x \# xs)$
proof (induct a)
  case (Le i ks)[simp]
  show ?case
  proof (cases ks)
    case Nil thus ?thesis by simp
  next
    case (Cons k ks′)[simp]
    show ?thesis proof cases
      assume $k = 0$ thus ?thesis by simp
    next
      assume $k \neq 0$
      with Le have $|k| dvd m$ by simp
      let $?m′ = m \div |k|
      have $?m′ > 0$ using $|k| dvd m$ pos-imp-zdiv-pos-iff $m > 0$ $k \neq 0$
        by (simp add: zdvd-imp-le)
      have 1: $k \ast (x \ast ?m′) = sgn k \ast x \ast m$
        proof
          have $k \ast (x \ast ?m′) = (sgn k \ast abs k) \ast (x \ast ?m′)$
            by (simp only: mult-sgn-abs)
          also have $\ldots = sgn k \ast x \ast (abs k \ast ?m′)$ by simp
          also have $\ldots = sgn k \ast x \ast m$
            using dvd-mult-cancel[OF $|k| dvd m$] by (simp add: algebra-simps)
          finally show ?thesis .
        qed
      have $I_Z (\text{hd-coeff1 m (Le i ks)}) (m \ast x \# xs) \iff$ ($i \ast ?m′ \leq sgn k \ast m \ast x + ?m′ \ast (ks′, xs)$)
        using $|k| \neq 0$ by (simp add: algebra-simps)
      also have $\ldots \iff ?m′ \ast i \leq ?m′ \ast (k \ast x + (ks′, xs))$ using 1
        by (simp (no-asm-simp) add: algebra-simps)
      also have $\ldots \iff i \leq k \ast x + (ks′, xs)$ using $?m′ > 0$
        by simp
      finally show ?thesis by(simp)
    qed
  qed
next
  case (Dvd d i ks)[simp]
  show ?case
  proof (cases ks)
    case Nil thus ?thesis by simp
  next
    case (Cons k ks′)[simp]
    show ?thesis
proof cases
  assume \(k = 0\) thus \(\text{thesis by simp}\)
next
  assume \(k \neq 0\)
  with \(Dvd\) have \(k \vdots m\) by simp
  let \(?m' = m \div k\)
  have \(?m' \neq 0\) using \(\langle k \vdots m \rangle\) \(zdiv-eq-0-iff\) \(\langle m > 0 \rangle\) \(\langle k \neq 0 \rangle\)
    by (simp add: linorder-not-less zdvd-imp-le)
  have 1: \(k \ast (x \ast ?m') = x \ast m\)
    proof
      have \(k \ast (x \ast ?m') = x \ast (k \ast ?m')\) by (simp add: algebra-simps)
    also have \(\ldots = x \ast m\) using dvd-mult-cancel \(\langle k \vdots m \rangle\)
    by (simp add: algebra-simps)
    finally show \(\text{thesis }\).
  qed

  have \(I Z (hd-coeff1 m \langle Dvd d i ks \rangle) \langle m \ast x \# xs \rangle \iff \neg (\langle m \ast d \vdots ?m' \ast i + m \ast x + ?m' \ast \langle ks', xs \rangle \rangle)\) using \(\langle k \neq 0 \rangle\) by (simp)
  qed

next
  case (NDvd d i ks)[simp]
  show \(?case\)
    proof (cases ks)
      case Nil thus \(\text{thesis by simp}\)
    next
      case (Cons k ks')[simp]
      show \(?thesis\)
        proof cases
          assume \(k = 0\) thus \(\text{thesis by simp}\)
        next
          assume \(k \neq 0\)
          with \(NDvd\) have \(k \vdots m\) by simp
          let \(?m' = m \div k\)
          have \(?m' \neq 0\) using \(\langle k \vdots m \rangle\) \(zdiv-eq-0-iff\) \(\langle m > 0 \rangle\) \(\langle k \neq 0 \rangle\)
            by (simp add: linorder-not-less zdvd-imp-le)
          have 1: \(k \ast (x \ast ?m') = x \ast m\)
            proof
              have \(k \ast (x \ast ?m') = x \ast (k \ast ?m')\) by (simp add: algebra-simps)
              also have \(\ldots = x \ast m\) using dvd-mult-cancel \(\langle k \vdots m \rangle\)
                by (simp add: algebra-simps)
              finally show \(\text{thesis }\).
            qed
          qed
          have \(I Z (hd-coeff1 m \langle NDvd d i ks \rangle) \langle m \ast x \# xs \rangle \iff \neg (\langle m \ast d \vdots ?m' \ast i + m \ast x + ?m' \ast \langle ks', xs \rangle \rangle)\)
using \(|k \neq 0\) by (simp add: algebra-simps)
also have \(\ldots \iff \nexists\ ?m' dvd \ ?m' * (i + k * x + \langle ks', xs \rangle)\) using 1
by (simp (no_asm-simp) add: algebra-simps)
also have \(\ldots \iff \nexists\ d \ dvd i + k * x + \langle ks', xs \rangle\) using \(\langle \nexists \ ?m' \neq 0 \rangle\) by (simp)
finally show \(?\thesis\) by (simp add: algebra-simps)
qed
qed
qed

lemma 1-hd-coeff1-mult: assumes \(m > 0\)
shows \(\text{qfree } \varphi \iff \forall \ a \in \text{set}(\mathbb{Z}.\text{atoms}_0 \ \varphi). \ \text{hd-coeff a dvd m} \implies \ Z.I \ (\text{map}_\varphi (\text{hd-coeff1 m} \ \varphi)) \ (m * x # xs) = Z.I \ \varphi \ (x # xs)\)
proof (induct \(\varphi\))
case (Atom a)
thus \(?\case\) using 1-hd-coeff1-mult-a \[OF \langle m > 0 \rangle\] by auto
qed simp-all

end

theory QEpres
imports PresArith
begin

5.4 DNF-based quantifier elimination
definition
hd-coeffs1 as =
(let \(m = \text{zlcms}(\text{map hd-coeff as})\)
in \(\text{Dvd m 0 [1] # map (hd-coeff1 m) as}\))

lemma 1-hd-coeffs1:
assumes \(0: \forall a \in \text{set as}. \ \text{hd-coeff a \neq 0}\)
shows \((\exists x. \forall a \in \text{set}(\text{hd-coeffs1 as}). \ I_Z \ a \ (x#xs)) =
(\exists x. \forall a \in \text{set as}. \ I_Z \ a \ (x#xs)) \ (\text{is } ?B = ?A)\)
proof –
let \(?m = \text{zlcms}(\text{map hd-coeff as})\)
have \(?m > 0\) using \(0\) by (simp add: zlcms-pos)
have \(?A = (\exists x. \forall a \in \text{set as}. \ I_Z \ (\text{hd-coeff1 m} \ ?m a) \ (\ ?m * x # xs))\)
by (simp add: 1-hd-coeff1-mult-a[OF \(m > 0\)] dvd-zlcms 0)
also have \(\ldots = (\exists x. \ ?m \ dvd x + 0 \ \land \ (\forall a \in \text{set as}. \ I_Z \ (\text{hd-coeff1 m} \ ?m a) \ (x # xs)))\)
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

61
definition
qe-pres1 as =
(let ds = filter is-dvd a; (d::int) = zlems(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 IZ a (i#e) = IZ 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(rule mod-add-eq)
    also have (i + ⟨ks’,e⟩) mod d = (i mod d + ⟨ks’,e⟩) mod d mod d
      by(rule 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 = (j + ⟨ks’,e⟩) mod d
      by(rule mod-add-eq[asymmetric])
    also have (l mod d + (j + ⟨ks’,e⟩)) mod d mod d = ⟨r⟩
      by(rule mod-add-eq[asymmetric])
    finally show ⟨thesis⟩.
  qed
t hus ⟨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(rule mod-add-eq)
    also have (i + ⟨ks’,e⟩) mod d = (i mod d + ⟨ks’,e⟩) mod d mod d
      by(rule 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 = (j + ⟨ks’,e⟩) mod d
      by(rule mod-add-eq[asymmetric])
    also have (l mod d + (j + ⟨ks’,e⟩)) mod d mod d = ⟨r⟩
by (rule mod-add-eq[symmetric])
finally show ?thesis.

qed

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

next

\( \text{case } \text{Le} \text{ thus } \text{?thesis using } \langle \text{is-dvd } a \rangle \text{ by simp} \)

qed

lemma \(I\)-qe-pres1:

assumes \( \text{norm: } \forall a \in \text{set as. divisor } a \neq 0 \)

and \( \text{hd: } \forall a \in \text{set as. } \text{hd-coeff-is1 } a \)

displays \(Z.I \ (\text{qe-pres1 as} ) \times = (\exists x. \forall a \in \text{set as. } I_Z a (x \# xs)) \)

proof –

let \( ?lbs = \text{lbounds as} \)

let \( ?ds = \text{filter is-dvd as} \)

let \( ?lcm = \text{zlcms}(\text{map divisor } ?ds) \)

let \( ?Ds = \{ a \in \text{set as. case } a \text{ of Le - (k\#-) } \Rightarrow k < 0 \ | \ - \Rightarrow \text{False} \} \)

let \( ?Ls = \{ a \in \text{set as. case } a \text{ of Le - (k\#-) } \Rightarrow k > 0 \ | \ - \Rightarrow \text{False} \} \)

have as: \( \text{set as } = ?Ds \cup ?Ls \cup ?Us \ (\text{is } = \?S) \)

proof –

\{ fix x assume \( x \in \text{set as} \)

hence \( x \in ?S \text{ using hd by (cases x) (auto split: list.splits)} \}

moreover

\{ fix x assume \( x \in ?S \)

hence \( x \in \text{set as} \text{ by auto} \}

ultimately show ?thesis by blast

qed

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

show ?thesis \( \ (\text{is } ?QE = (\exists x. \ ?P x)) \)

proof

assume \( ?QE \)

\{ assume \( ?lbs = [] \)

with \( ?QE \) obtain \( n \) where \( n < ?lcm \) and

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

by (auto simp: IZ-asubst qe-pres1-def)

have \( ?Ls = {} \) using \( ?lbs = [] \text{ set-lbounds}[of as] \)

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

have \( \exists x. \ ?P x \)

proof cases

assume \( ?Us = {} \)

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

hence \( ?P n \text{ using A by auto} \)

thus ?thesis ..

next

assume \( ?Us \neq {} \)

let \( ?M = \langle \text{tl } ks, \ xs \rangle \ - i|ks i. \ Le i ks < \?Us \} \)

let \( ?m = \text{Min } ?M \)

have finite ?M

proof –

63
have \( \text{finite} \left( (\lambda i. k s \Rightarrow (t l k s, x s) - i) \right) \) ’
\( \{ a \in \text{set as. } \exists i k s. k < 0 \land a = \text{Le } i (k\#k s) \} \)
\( \text{by simp} \)
also have \( ?B = ?M \text{ using } \text{hd} \)
\( \text{by (fastforce simp: image-def neg-nil-conv split:atom.splits list.splits)} \)
finally show \( ?\text{thesis by auto} \)
qed

have \( ?M \neq \{ \} \)
\( \text{proof} - \)
\( \text{from } (?Us \neq \{ \}) \text{ obtain } i k s \text{ where } \text{Le } i (k\#k s) \in ?Us \land k < 0 \)
\( \text{by (fastforce split:atom.splits list.splits)} \)
thus \( ?\text{thesis by auto} \)
qed

let \( ?k = (n - ?m) \text{ div } ?\text{lcm} + 1 \text{ let } ?x = n - ?k * ?\text{lcm} \)
have \( \forall a \in ?Ds. I Z a (?x \# x s) \)
\( \text{proof (intro allI ballI)} \)
fix \( a \)
assume \( a \in ?Ds \)
let \( ?d = \text{divisor } a \)
have \( 2: ?d \text{ dvd } ?\text{lcm using } (a \in ?Ds) \text{ by (simp add: dvd-zlcms)} \)
have \( ?x \mod ?d = n \mod ?d \) \( \text{is } ?l = ?r \)
\( \text{proof} - \)
have \( ?l = ( ?r - ((?k * ?\text{lcm}) \mod ?d)) \mod ?d \)
\( \text{by (rule mod-diff-eq)} \)
also have \( (?k * ?\text{lcm}) \mod ?d = 0 \)
\( \text{by (simp add: dvd-eq-mod-eq-0 [symmetric] dvd-mult [OF 2])} \)
finally show \( ?\text{thesis by simp} \)
qed
thus \( I Z a (?x\#x s) \) using \( A \text{-cyclic} [of a n ?x] \) \( (a \in ?Ds) \text{ 1 by auto} \)
qed

moreover
have \( \forall a \in ?Us. I Z a (?x\#x s) \)
\( \text{proof} - \)
fix \( a \)
assume \( a \in ?Us \)
then obtain \( l k s \text{ where } [\text{simp}]: a = \text{Le } l (-1\#k s) \text{ using } \text{hd} \)
\( \text{by (fastforce split:atom.splits list.splits)} \)
have \( ?m \leq (k s, x s) - l \)
\( \text{using } \text{Min-le-iff} [\text{OF } \text{finite } ?M; (?M \neq \{ \}) \text{ (a } \in ?Us)] \text{ by fastforce} \)
moreover have \( (n - ?m) \text{ mod } ?\text{lcm} < ?\text{lcm} \)
\( \text{by (simp add: pos-mod-bound [OF zlcms-pos] norm)} \)
ultimately show \( I Z a (?x\#x s) \)
\( \text{by (simp add: zmult-div-cancel algebra-simps)} \)
qed

moreover
have \( \text{set as } = ?Ds \cup ?Us \text{ using } [\text{?Ls } = \{ \}] \)
\( \text{by (simp (no-asm-use)) blast} \)
ultimately have \( ?P(?x) \) by auto
thus \( ?\text{thesis ..} \)
qed
moreover  
\{ \text{ assume } \\textit{lbs} \neq [ ] \}  
with \langle \textit{QE} \rangle \textit{ obtain } il ksl m  
where \forall a \in set as. \textit{I}_{Z} (\textit{subst} (il + m) ksl a) xs  
by (auto simp:qe-pres$_1$-def)  
\textit{hence } \textit{P}(il + m + (ksl, xs)) \textit{ by (simp add: IZ-asubst)}  
\textit{hence } \exists x. \textit{P} x .. \}  
ultimately show \exists x. \textit{P} x by blast

next  
\text{ assume } \exists x. \textit{P} x then obtain x where x: \textit{P} x ..  
show \textit{QE}  
proof cases  
\text{ assume } \exists x. \textit{lbs} = [ ]  
moreover 
\textit{have } \exists x. \emptyset \leq x \land x < \?lcm \land (\forall a \in ?Ds. \textit{I}_{Z} a (x \neq xs))  
(is \exists x. \textit{P} x)  
proof  
\{ \text{ fix } a \textit{ assume } a \in ?Ds \textit{ hence } \textit{I}_{Z} a ((x \textit{ mod } \?lcm) \neq xs) = \textit{I}_{Z} a (x \neq xs) \textit{ using } 1 \textit{ by (fastforce del:iffI intro: I-cyclic simp: mod-mod-cancel dvd-zlcm:1) } \}  
thus \textit{P}(x \textit{ mod } \?lcm) \textit{ using } x \textit{ norm by (simp add: zlcm-pos)}  
qed  
ultimately show \textit{thesis} by (auto simp:qe-pres$_1$-def IZ-asubst)

next  
\text{ assume } \exists x. \textit{lbs} \neq [ ] \textit{ let } \?L = \{ i - \langle ksl, xs \rangle \mid ksl. (i, ksl) \in set(lbounds as) \} \textit{ let } \?lm = \textit{Max } \?L \textit{ let } \?n = (x - \?lm) \textit{ mod } \?lcm \textit{ have finite } \?L \textit{ proof –} \textit{ have } \textit{finite}((\lambda(i, ksl). i - \langle ksl, xs \rangle) \cdot \textit{set}(lbounds as)) \textit{ (is finite } \?B) \textit{ by simp} \textit{ also have } \?B = \?L \textit{ by auto} \textit{ finally show } \textit{thesis} by auto \textit{ qed} 
moreover have \?L \neq [ ] \textit{ using } \?lbs \neq [ ] \textit{ by (fastforce simp:neq-Nil-conv)}  
ultimately have \?lm \in ?L \textit{ by (rule Max-in)} \textit{ then obtain li lks where } (li, lks) \in set \?lbs \textit{ and } \?lm: \?lm = li - \langle lks, xs \rangle \textit{ by blast} \textit{ moreover } \textit{ have } n: \emptyset \leq \?n \land \?n < \?lcm \textit{ using norm by (simp add: zlcm-pos)} \textit{ moreover } 
\{ \text{ fix } a \textit{ assume } a \in set as \textit{ with } x \textit{ have } \textit{I}_{Z} a (x \neq xs) \textit{ by blast} \textit{ have } \textit{I}_{Z} a ((li + \?n - \langle lks, xs \rangle) \neq xs) \textit{ proof –} \textit{ assume } a \in ?Ls \textit{ qed}
then obtain $i \, ks$ where \[ a = Le i \ (1 # ks) \] using \( \text{hd} \)
by (fastforce split:atom.splits list.splits)
from \( a \in ?L \) have $i - (ks, xs) \in ?L$ by (fastforce simp:set-lbounds)
hence $i - (ks, xs) \leq li - (lks, xs)$
using \( \text{lm} \{\text{symmetric} \} \) \( \text{finite} ?L \) \( \{ ?L \neq \} \) by auto
hence \(?thesis \) using \( n \) by simp \} 
moreover \{ 
assume $a \in ?U$ 
then obtain $i \, ks$ where \[ a = Le i \ (-1 # ks) \] using \( \text{hd} \)
by (fastforce split:atom.splits list.splits)
have $li (1 # lks) \in \text{set as}$ using \( \langle li,lks \rangle \in \text{set ?lbs} \) \( \text{hd} \)
by (auto simp: set-lbounds)
hence $li - (lks, xs) \leq x$ using \( x \) by auto
hence $(x - ?lm) \mod ?lcm \leq x - ?lm$
using \( \text{lm} \) by (simp add: zmod-le-nonneg-dividend)
\(?thesis \) using \( IZ \) \( a (x \# xs) \) \( \text{lm} \) by auto \} 
moreover \{ 
assume $a \in ?D$ 
have \(?thesis \) proof (rule I-cyclic \[ \text{THEN iffd2} \), OF \ - \ - \ \langle IZ \ a (x \# xs) \rangle \]
show is-dvd a using \( a \in ?D \) by simp
show hd-coef a = 1 using \( a \in ?D \) \( \text{hd} \)
by (fastforce split:atom.splits list.splits)
have $li + (x - ?lm) \mod ?lcm - (lks, xs) = ?lm + (x - ?lm) \mod ?lcm$
using \( \text{lm} \) by arith
hence $(li + (x - ?lm) \mod ?lcm - (lks, xs)) \mod \text{divisor} \ a =$
\( (?lm + (x - ?lm) \mod ?lcm) \mod \text{divisor} \ a \) by (simp only:)
also have \( \ldots = \)
\( (?lm \mod \text{divisor} \ a + (x - ?lm) \mod ?lcm \mod \text{divisor} \ a) \mod \text{divisor} \ a \)
by (rule mod-add-eq)
also have \( \ldots = \)
\( (?lm \mod \text{divisor} \ a + (x - ?lm) \mod \text{divisor} \ a) \mod \text{divisor} \ a \)
using \( \text{is-dvd} \) \( a \in \text{set as} \)
by (simp add: mod-mod-cancel dvd-zlcm)
also have \( \ldots = \)
\( (?lm + (x - ?lm)) \mod \text{divisor} \ a \)
by (rule mod-add-eq [symmetric])
also have \( \ldots = x \mod \text{divisor} \ a \) by simp
finally
show $(li + ?n - (lks, xs)) \mod \text{divisor} \ a = x \mod \text{divisor} \ a$
using norm by (auto simp: zlcm-pos)
\( \text{qed} \) 
ultimately show \(?thesis \) using \( a \in \text{set as} \) as by blast 
\( \text{qed} \) 
\} 
ultimately show \(?thesis \) using \( \langle ?lbs \neq \rangle \)
by (simp (no-asm-simp) add:qe-pres1-def IZ-asubst split-def)
(force simp del:int-nat-eq) 
\( \text{qed} \)
lemma \text{divisors-hd-coeffs1}:
assumes \text{div0: } \forall a \in \text{set as. divisor } a \neq 0 \text{ and } \text{hd0: } \forall a \in \text{set as. hd-coeff } a \neq 0
and \text{ a: } a \in \text{set (hd-coeffs1 as) shows divisor } a \neq 0
proof
\begin{align*}
\text{let } \vartriangleleft m \text{ = zlcms}(\text{map hd-coeff as}) \\
\text{from a have } a = \text{Dvd } \vartriangleleft m 0 \ [1] \lor (\exists b \in \text{set as. } a = \text{hd-coeff1 } \?m \ b) \\
\quad \text{(is } \?A \lor \?B) \\
\quad \text{by(auto simp:hd-coeffs1-def)} \\
\text{thus } \?thesis
\end{align*}

proof
\begin{align*}
\text{assume } \?A \text{ thus } \?thesis \text{ using } \text{hd0 by(auto)} \\
\text{next}
\text{assume } \?B \\
\text{then obtain } b \text{ where } b \in \text{set as and } \text{simp: } a = \text{hd-coeff1 } \?m \ b .. \\
\text{hence } b: \text{hd-coeff } b \neq 0 \text{ divisor } b \neq 0 \text{ using div0 hd0 by auto} \\
\text{show } \?thesis
\end{align*}

proof \text{(cases } b) \\
\text{case (Le i ks) thus } \?thesis \text{ using } b \text{ by(auto split:list.splits)} \\
\text{next}
\text{case (Dvd d i ks)[simp]}
\text{then obtain } k \text{ ks where } \text{simp: } ks = k \# ks \text{ using } b \\
\quad \text{by(auto split:list.splits)} \\
\text{have } k: k \in \text{set (map hd-coeff as) using } b \in \text{set as by force} \\
\text{have zlcms (map hd-coeff as) div } k \neq 0 \\
\text{using } b \text{ hd0 dvd-zlcms[OF } k] \\
\quad \text{by(auto simp add:dvd-def)} \\
\text{thus } \?thesis \text{ using } b \text{ by (simp)} \\
\text{next}
\text{case (NDvd d i ks)[simp]}
\text{then obtain } k \text{ ks where } \text{simp: } ks = k \# ks \text{ using } b \\
\quad \text{by(auto split:list.splits)} \\
\text{have } k: k \in \text{set (map hd-coeff as) using } b \in \text{set as by force} \\
\text{have zlcms (map hd-coeff as) div } k \neq 0 \\
\text{using } b \text{ hd0 dvd-zlcms[OF } k] \\
\quad \text{by(auto simp add:dvd-def)} \\
\text{thus } \?thesis \text{ using } b \text{ by (simp)}
\end{proof}

qed

lemma \text{hd-coeff-is1-hd-coeffs1}:
assumes \text{hd0: } \forall a \in \text{set as. hd-coeff } a \neq 0 \\
and \text{ a: } a \in \text{set (hd-coeffs1 as) shows } \text{hd-coeff-is1 } a
proof
\begin{align*}
\text{let } \vartriangleleft m = \text{zlcms}(\text{map hd-coeff as}) \\
\text{from a have } a = \text{Dvd } \vartriangleleft m 0 \ [1] \lor (\exists b \in \text{set as. } a = \text{hd-coeff1 } \?m \ b) \\
\quad \text{(is } \?A \lor \?B)
\end{align*}
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 [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-pres1-o:
[ ∀ a ∈ set as. divisor a ≠ 0; ∀ a∈set as. hd-coeff a ≠ 0 ] ⇒
Z.I ((qe-pres1 o hd-coeffs1) as) e = (∃ x. ∀ a∈ set as. I_Z a (x#e))
apply (simp)
apply (subst I-qe-pres1)
  apply (simp add: divisors-hd-coeffs1)
  apply (simp add: hd-coeffs1-def)
  apply (simp)
done

definition qe-pres = Z.lift-dnf-qe (qe-pres1 o hd-coeffs1)

lemma qfree-qe-pres-o: qfree ((qe-pres1 o hd-coeffs1) as)
by (auto simp: qe-pres1-def intro!: qfree-list-disj)

lemma normal-qe-pres1-o:
∀ a ∈ set as. hd-coeff a ≠ 0 ∧ divisor a ≠ 0 ⇒
Z.normal ((qe-pres1 o hd-coeffs1) as)
apply (auto simp: qe-pres1-def Z.normal-def
  dest!: atoms-list-disjE atoms-list-conjE)
apply (simp add: hd-coeffs1-def)
apply (erule disjE) apply fastforce
apply (clarsimp)
apply (erule disjE xa)
apply (case-tac list) apply fastforce apply (simp split: split-if-asm)
apply (case-tac list) apply fastforce
apply (simp split: split-if-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 v∈ set (map hd-coeff as). i ≠ 0)
prefer 2 apply simp

68
apply (metis elem-le-zlcms linorder-not-le zlcms-pos)
apply (case-tac list) apply fastforce
apply (simp split:split-if-asm) apply fastforce
apply (simp add: zdiv-eq-0-iff)
apply (subgoal-tac ∃ i ∈ set(map hd-coeff as). i ≠ 0)
prefer 2 apply simp
apply (subgoal-tac a ∈ 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 (case-tac list) apply fastforce apply (simp split:split-if-asm)
apply (case-tac list) apply fastforce
apply (simp split:split-if-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 (case-tac list) apply fastforce
apply (simp split:split-if-asm) apply fastforce
apply (simp add: zdiv-eq-0-iff)
apply (subgoal-tac ∃ i ∈ set(map hd-coeff as). i ≠ 0)
prefer 2 apply simp
apply (subgoal-tac a ∈ 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 ϕ → Z.I (qe-pres ϕ) xs = Z.I ϕ xs

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)
theory Cooper
imports PresArith
begin

5.5 Cooper

This section formalizes Cooper’s algorithm [1].

lemma set-atoms0-iff:
qfree ϕ =⇒ a : set(Z.atoms0 ϕ) ←→ a : atoms ϕ ∧ hd-coeff a ≠ 0
by (induct ϕ) (auto split:split-if-asm)

definition hd-coeffs1 ϕ =
(let m = zlcms(map hd-coeff (Z.atoms0 ϕ))
in And (Atom(Dvd m 0 [1])) (map fn (hd-coeff1 m) ϕ))

lemma I-hd-coeffs1:
assumes qfree ϕ
shows (∃ x. Z.I (hd-coeffs1 ϕ) (x#xs)) = (∃ x. Z.I ϕ (x#xs)) (is ?L = ?R)
proof –
let ?l = zlcms(map hd-coeff (Z.atoms0 ϕ))
have ?l>0 by (simp add: zlcms-pos set-atoms0-iff[OF ⟨qfree ϕ⟩])
have ?l = (∃ x. ?l dvd x + 0 ∧ Z.I (map fn (hd-coeff1 ?l) ϕ) (x#xs))
  by (simp add:hd-coeffs1-def)
also have ... = (∃ x. Z.I (map fn (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 fn (inf_) where
inf_ (And ϕ1 ϕ2) = and (inf_ ϕ1) (inf_ ϕ2) |
inf_ (Or ϕ1 ϕ2) = or (inf_ ϕ1) (inf_ ϕ2) |
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

definition qc-cooper1 ϕ =
(let as = Z.atoms0 ϕ; 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)))))

end
lemma min-inf:
\[ \text{nqfree } f \implies \forall a \in \text{set}(Z.\text{atoms}_0 f). \text{hd-coef-is1 } a \]
\[ \implies \exists x. \forall y < x. Z.I (\text{inf } f) (y \neq xs) = Z.I f (y \neq xs) \]
(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:
\[ \text{nqfree } \varphi \implies \forall a \in \text{set}(Z.\text{atoms}_0 \varphi). \text{divisor } a \text{ dvd } d \implies \]
\[ Z.I (\text{inf } \varphi) ((x − k*d)\#xs) = Z.I (\text{inf } \varphi) (x\#xs) \]
(proof (induct \varphi rule: min-inf.induct)
case (4-4 da i ks)
show ?case
proof (cases ks)
\{ case Nil thus ?thesis by simp \}
next
case (Cons j js)
show ?thesis
proof cases
\{ assume j=0 thus ?thesis using Cons by simp \}
next
\{ assume j≠0 hence da dvd d using Cons 4-4 by simp \}
hence da dvd i + (j * x + (j * d) + ⟨js, xs⟩) \leftrightarrow
 da dvd i + (j * x + ⟨js, xs⟩)
proof

have \( da \ dvd \ i + (j \cdot x - j \cdot (k \cdot d) + \langle js, xs \rangle) \ \longleftrightarrow \ \)
\( da \ dvd \ (i + j \cdot x + \langle js, xs \rangle) - (j \cdot k) \cdot d \)
by (simp add: algebra-simps)
also have \( \ldots \ \longleftrightarrow \ da \ dvd \ i + j \cdot x + \langle js, xs \rangle \) using \( da \ dvd \ d \)
by (metis dvd-diff zdvd-zdiffD dvd-mult mult.commute)
also have \( \ldots \ \longleftrightarrow \ da \ dvd \ i + (j \cdot x + \langle js, xs \rangle) \)
by (simp add: algebra-simps)
finally show \(?thesis\).

qed

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

qed

next

case \((4\cdot5 \ da \ i \ ks)\)
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 \(j\neq0\)
hence \(da \ dvd \ d\) using \(4\cdot5\) by simp
hence \(da \ dvd \ i + (j \cdot x - j \cdot (k \cdot d) + \langle js, xs \rangle) \ \longleftrightarrow \ \)
\(da \ dvd \ (i + j \cdot x + \langle js, xs \rangle) - (j \cdot k) \cdot d \)

proof

have \(da \ dvd \ i + (j \cdot x - j \cdot (k \cdot d) + \langle js, xs \rangle) \ \longleftrightarrow \ \)
\(da \ dvd \ (i + j \cdot x + \langle js, xs \rangle) - (j \cdot k) \cdot d \)
by (simp add: algebra-simps)
also have \(\ldots \ \longleftrightarrow \ da \ dvd \ i + j \cdot x + \langle js, xs \rangle\) using \(da \ dvd \ d\)
by (metis dvd-diff zdvd-zdiffD dvd-mult mult.commute)
also have \(\ldots \ \longleftrightarrow \ da \ dvd \ i + (j \cdot x + \langle js, xs \rangle)\)
by (simp add: algebra-simps)
finally show \(?thesis\).

qed

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

qed simp-all

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

lemma \(\beta\):

72
\[ \text{nagfree } \varphi; \ \forall a \in \text{set}(\text{Z.atoms}_0 \varphi). \text{hd-coeff-is1 } a; \\
\forall a \in \text{set}(\text{Z.atoms}_0 \varphi). \text{divisor } a \text{ dvd } d; \ d > 0; \\
\neg(\exists j \in \{0 .. \ d - 1\}; \exists (i,ks) \in \text{set}(\text{lbounds}(\text{Z.atoms}_0 \varphi))). \\
x = i - \langle ks, xs \rangle + j); \ Z.1 \varphi (x\#xs) \]

\( \Rightarrow \ Z.1 \varphi ((x-d)\#xs) \)

\textbf{proof} (\texttt{induct } \varphi)

\textbf{case} (\texttt{Atom } a)

\textbf{show } ?\texttt{case}

\textbf{proof } (\texttt{cases } a)

\textbf{case} (\texttt{Le } i js)

\textbf{show } ?\texttt{thesis}

\textbf{proof } (\texttt{cases } js)

\textbf{case } Nil \textbf{ thus } ?\texttt{thesis using } \texttt{Le Atom } by \texttt{ simp}

\textbf{next}

\textbf{case} (\texttt{Cons } k ks)

\textbf{thus } ?\texttt{thesis using } \texttt{Le Atom}

\textbf{by } (\texttt{auto simp: \texttt{lbounds-def Ball-def split:split-if-asm}}) arith

\textbf{qed}

\textbf{next}

\textbf{case} (\texttt{Dvd } m i js)

\textbf{show } ?\texttt{thesis}

\textbf{proof } (\texttt{cases } js)

\textbf{case } Nil \textbf{ thus } ?\texttt{thesis using } \texttt{Dvd Atom } by \texttt{ simp}

\textbf{next}

\textbf{case} (\texttt{Cons } k ks)

\textbf{show } ?\texttt{thesis}

\textbf{proof} cases

\textbf{assume } k=0 \textbf{ thus } ?\texttt{thesis using } \texttt{Cons Dvd Atom } by \texttt{ simp}

\textbf{next}

\textbf{assume} k\neq0

\textbf{hence } m \texttt{ dvd } d \textbf{ using } \texttt{Cons Dvd Atom } by \texttt{ auto}

\textbf{have} m \texttt{ dvd } i + (x + \langle ks, xs \rangle) \Longrightarrow m \texttt{ dvd } i + (x - d + \langle ks, xs \rangle)

(is \ ?L \Longrightarrow -)

\textbf{proof } -

\textbf{assume } ?L

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

\textbf{by } (\texttt{metis (m dvd d ) dvd-diff})

\textbf{thus } ?\texttt{thesis by}(\texttt{simp add:algebra-simps})

\textbf{qed}

\textbf{thus } ?\texttt{thesis using } \texttt{Atom Dvd Cons by(auto split:split-if-asm)}

\textbf{qed}

\textbf{qed}

\textbf{next}

\textbf{case} (\texttt{NDvd } m i js)

\textbf{show } ?\texttt{thesis}

\textbf{proof } (\texttt{cases } js)

\textbf{case } Nil \textbf{ thus } ?\texttt{thesis using } \texttt{NDvd Atom } by \texttt{ simp}

\textbf{next}

\textbf{case} (\texttt{Cons } k ks)

\textbf{show } ?\texttt{thesis}

73
proof cases
  assume k=0 thus ?thesis using Cons NDvd Atom by simp
next
  assume k\not=0
  hence m \mid d using Cons NDvd Atom by auto
  have m \mid d + (x - d + \langle ks, xs \rangle) \implies m \mid d + (x + \langle ks, xs \rangle)
    (is ?L \implies -)
  proof -
    assume ?L
    hence m \mid d + (x + \langle ks, xs \rangle) - d by (simp add: algebra-simps)
    thus ?thesis by (metis ⟨m \mid d⟩ zdvd-zdiffD)
  qed
  thus ?thesis using Atom NDvd Cons by (auto split: split-if asm)
  qed
  qed
  qed
  qed

lemma periodic-finite-ex:
  assumes dpos: (0::int) < d and modd: \forall x k. P x = P(x - k*d)
  shows (\exists x. P x) = (\exists j\in\{0..d-1\}. P j)
  (is ?LHS = ?RHS)
proof
  assume ?LHS
  then obtain x where P: P x ..
  have x mod d = x - (x div d)*d
    by (simp add: zmod-zdiv-equality 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

lemma cpmi-eq: (0::int) < D \implies (∃z. \forall x. x < z \implies (P x = P1 x))
  \implies \forall x. -(∃j\in\{0..D-1\}. ∃b\in B. P(b+j)) \implies P (x) \implies P (x - D)
  \implies \forall x. ∀ k. P1 x = P1(x-k*D)
  \implies (∃x. P(x)) = ((∃j\in\{0..D-1\}. P1(j)) ∨ (∃j\in\{0..D-1\}. ∃b\in B. P(b+j)))
apply (rule iffI)
prefer 2
apply (drule minusinfinity)
apply assumption+
apply (fastforce)
apply clarsimp
apply (subgoal-tac \\forall k. 0\leq k \implies \forall x. P x \implies P (x - k*D))
apply (rule-tac x = x and z=z in deer-lemma)
apply (subgoal-tac P1(x - (|x - z| + 1) * D))
prefer 2
apply (subgoal-tac 0 \leq (|x - z| + 1))
prefer 2 apply arith
apply fastforce
apply(drule (1) periodic-finite-ex)
apply blast
apply(blast dest:decr-mult-lemma)
done

theorem cp-thm:
assumes nq: nqfree \( \varphi \)
and u: \( \forall \, a \in \text{set}(Z, \text{atoms}_0 \, \varphi) \). \( \text{hd-coeff-is1} \, a \)
and d: \( \forall \, a \in \text{set}(Z, \text{atoms}_0 \, \varphi) \). \( \text{divisor} \, a \, \text{dvd} \, d \)
and dp: \( d > 0 \)
shows \((\exists \, x. \, Z, I \, \varphi \, (x \# xs)) = \)
\((\exists j \in \{0..d-1\}. \, Z, I \, (\text{inf} - \varphi) \, (j \# xs) \lor \)
\((\exists (i,ks) \in \text{set}(\text{lbounds}(Z, \text{atoms}_0 \, \varphi)). \, Z, I \, \varphi \, ((i - (\text{ks},xs) + j) \# xs))) \)
\((\text{is} \,(\exists \, x. \, ?P \, (x)) = (\exists \, j \in \, ?D. \, ?M \, j \lor (\exists (i,ks) \in \, ?B. \, ?P \,(\varphi \, I \, i \, ks \, j))) \))
proof –
from min-inf[OF nq u] have th: \( \exists \, z. \forall \, x < z. \, ?P \, x = ?M \, x \) by blast
let \( ?B' = \{ \, ?I \, i \, ks \mid i,ks, (i,ks) \in ?B \} \)
have BB': \((\exists j \in \, ?D. \, \exists (i,ks) \in \, ?B. \, ?P \,(\varphi \, I \, i \, ks \, j)) = (\exists j \in \, ?D. \, \exists b \in \, ?B'. \, ?P \,(b \, + \, j)) \) by auto
hence th2: \( \forall \, x, \, \neg (\exists j \in \?D. \, \exists b \in \, ?B'. \, ?P \,(b \, + \, j)) \longrightarrow \, ?P \,(x) \longrightarrow \, ?P \,(x \, \, d)) \)
  using \( \beta[\, \text{OF nq u d dp, of - xs} \, \text{by (simp add: Bex-def) metis} \)
from min-inf-repeats[OF nq d]
have th3: \( \forall \, x \, k. \, ?M \, x = ?M \,(x \, - \, k \, \times \, d) \) by simp
from cpmi-eq[OF dp th th2 th3] BB' show ?thesis by simp blast
qed

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

lemma I-qe-cooper\(_1\):
assumes norm: \( \forall \, a \in \text{atoms} \, \varphi. \, \text{divisor} \, a \neq 0 \)
and hd: \( \forall \, a \in \text{set}(Z, \text{atoms}_0 \, \varphi) \). \( \text{hd-coeff-is1} \, a \) and nqfree \( \varphi \)
shows \( Z, I \, (\text{qe-cooper}_1 \, \varphi) \, xs = (\exists \, x. \, Z, I \, \varphi \, (x \# xs)) \)
proof –
let \( ?a = Z, \text{atoms}_0 \, \varphi \)
let \( ?d = \text{zlcms}(\text{map} \, \text{divisor} \, ?a) \)
have \( ?d > 0 \) using norm atoms-subset[of \( \varphi \) (\( \neg \) qfree \( \varphi \) )]
by (fastforce intro:zlcms-pos)
have alld: \( \forall \, a \in \text{set}(Z, \text{atoms}_0 \, \varphi) \). \( \text{divisor} \, a \, \text{dvd} \, ?d \) by (simp add: dvd-zlcms)
from cp-thm[OF nqfree \( \varphi \) hd alld \( \?d > 0 \)]
show ?thesis using nqfree \( \varphi \)
  by (simp add: qe-cooper\(_1\)-def I-subst[symmetric] split-def algebra-simps) blast
qed
lemma \textit{divisor-hd-coeff1-neq0}:
\[ \text{qfree } \varphi \implies a \in \text{atoms } \varphi \implies \text{divisor } (\text{hd-coeff1} (\text{zlcms} (\text{map} \text{ hd-coeff} (Z.\text{atoms}_0 \varphi)))) a) \neq 0 \]
apply (case-tac a)

apply simp
apply (case-tac list) apply simp apply (simp split:split-if-asm)
apply simp
apply (case-tac list) apply simp
apply (clarsimp simp:split-if-asm)
apply (clarsimp split:split-if-asm)
apply (hypsubst-thin)
apply (clarsimp simp:split-if-asm)
apply (hypsubst-thin)
apply (clarsimp simp:split-if-asm)
apply (clarsimp simp:split-if-asm)
apply simp add: set-atoms0-iff
apply fastforce simp:image-def set-atoms0-iff Bex-def

apply simp
apply (case-tac list) apply simp
apply (clarsimp simp:split-if-asm)
apply (clarsimp simp:split-if-asm)
apply (clarsimp simp:split-if-asm)
apply simp add: set-atoms0-iff
apply fastforce simp:image-def set-atoms0-iff Bex-def

apply simp
apply (case-tac list) apply simp
apply (clarsimp simp:split-if-asm)
apply (clarsimp simp:split-if-asm)
apply (clarsimp simp:split-if-asm)
apply simp add: set-atoms0-iff
apply fastforce simp:image-def set-atoms0-iff Bex-def

done

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

lemma \textit{I-cooper1-hd-coeffs1}:
\[ Z.\text{normal } \varphi \implies \text{qfree } \varphi \implies Z.I (\text{qe-cooper}_1 (\text{hd-coeffs1} \varphi)) xs = (\exists x. Z.I \varphi (x \neq xs)) \]
apply (simp add:Z.normal-def)
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)
apply (clarsimp simp:hd-coeffs1-def qfree-map-fm)
apply (simp add: I-hd-coeffs1)

definition \textit{qe-cooper} = Z.lift-nnf-qe (qe-cooper1 \circ \text{hd-coeffs1})

lemma \textit{qfree-cooper1-hd-coeffs1}:
\[ \text{qfree } \varphi \implies \text{qfree } (\text{qe-cooper}_1 (\text{hd-coeffs1} \varphi)) \]
by (auto simp:qe-cooper1-def hd-coeffs1-def qfree-map-fm)
lemma normal-min-inf: Z.normal ϕ ⇒ Z.normal((inf - ϕ)
by(induct ϕ rule:min-inf.induct) simp-all

lemma normal-cooper1: Z.normal ϕ ⇒ Z.normal(qe-cooper1 ϕ)
by(simp add:qe-cooper1-def Logic.or-def Z.normal-map-fin normal-min-inf split-def)

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

theorem I-cooper: Z.normal ϕ ⇒ Z.I (qe-cooper ϕ) xs = Z.I ϕ xs
by(simp add:qe-cooper-def Z.I-lift-nnf-qe-normal qfree-cooper1-hd-coeffs1 I-cooper1-hd-coeffs1 normal-cooper1 normal-hd-coeffs1)

theorem qfree-cooper: qfree (qe-cooper ϕ)
by(simp add:qe-cooper-def Z.qfree-lift-nnf-qe qfree-cooper1-hd-coeffs1)

end

References


