

Undecidability Results on Orienting Single Rewrite Rules*

René Thiemann, Fabian Mitterwallner, and Aart Middeldorp

University of Innsbruck

March 17, 2025

Abstract

We formalize several undecidability results on termination for *one-rule* term rewrite systems by means of simple reductions from Hilbert’s 10th problem. To be more precise, for a class C of reduction orders, we consider the question for a given rewrite rule $\ell \rightarrow r$, whether there is some reduction order $\succ \in C$ such that $\ell \succ r$. We include undecidability results for each of the following classes C :

- the class of *linear* polynomial interpretations over the natural numbers,
- the class of linear polynomial interpretations over the natural numbers in the *weakly monotone* setting,
- the class of Knuth–Bendix orders with *subterm coefficients*,
- the class of *non-linear* polynomial interpretations over the natural numbers, and
- the class of non-linear polynomial interpretations over the *rational* and *real* numbers.

Contents

1	Introduction	2
2	Preliminaries: Extending the Library on Multivariate Polynomials	2
2.1	Part 1 – Extensions Without Importing Univariate Polynomials	2
2.2	Part 2 – Extensions With Importing Univariate Polynomials	15
3	Definition of Monotone Algebras and Polynomial Interpretations	37

*This research was supported by the Austrian Science Fund (FWF) project I 5943.

4	Hilbert's 10th Problem to Linear Inequality	43
5	Undecidability of Linear Polynomial Termination	48
6	Undecidability of KBO with Subterm Coefficients	82
7	Undecidability of Polynomial Termination over Integers	90
8	Undecidability of Polynomial Termination using δ -Orders	124

1 Introduction

The main part of this paper is about one of the earliest termination methods for term rewrite systems: using a polynomial interpretation over the natural numbers, which goes back to Lankford [1].

In a recent paper [3] it was shown that this and other related techniques are undecidable, even for one-rule rewrite systems. This AFP entry formally proves the results in [3]. These are all based on reduction from a variant of Hilbert's 10th problem, which was shown to be undecidable by Matiyasevich [2].

2 Preliminaries: Extending the Library on Multivariate Polynomials

2.1 Part 1 – Extensions Without Importing Univariate Polynomials

```

theory Preliminaries-on-Polynomials-1
  imports
    Polynomials.More-MPoly-Type
    Polynomials.MPoly-Type-Class-FMap
  begin

  type-synonym var = nat
  type-synonym monom = var  $\Rightarrow_0$  nat

  definition substitute :: (var  $\Rightarrow$  'a mpoly)  $\Rightarrow$  'a :: comm-semiring-1 mpoly  $\Rightarrow$  'a
    mpoly where
      substitute  $\sigma$  p = insertion  $\sigma$  (replace-coeff Const p)

  lemma Const-0: Const 0 = 0
    by (transfer, simp add: Const0-zero)

  lemma Const-1: Const 1 = 1
    by (transfer, simp add: Const0-one)

```

lemma *insertion-Var*: $\text{insertion } \alpha (\text{Var } x) = \alpha x$
apply *transfer*
by (*metis One-nat-def Var₀-def insertion.abs-eq insertion-single mapping-of-inverse monom.rep-eq mult.right-neutral mult-1 power.simps(2) power-0*)

lemma *insertion-Const*: $\text{insertion } \alpha (\text{Const } a) = a$
by (*metis Const.abs-eq Const₀-def insertion-single monom.abs-eq mult.right-neutral power-0 single-zero*)

lemma *insertion-power*: $\text{insertion } \alpha (p^{\wedge}n) = (\text{insertion } \alpha p)^{\wedge}n$
by (*induct n, auto simp: insertion-mult*)

lemma *insertion-monom-add*: $\text{insertion } \alpha (\text{monom } (f + g) a) = \text{insertion } \alpha (\text{monom } f 1) * \text{insertion } \alpha (\text{monom } g a)$
by (*metis insertion-mult mult-1 mult-monom*)

lemma *insertion-uminus*: $\text{insertion } \alpha (- p) = - \text{insertion } \alpha p$
by (*metis add-eq-0-iff insertion-add insertion-zero*)

lemma *insertion-sum-list*: $\text{insertion } \alpha (\text{sum-list } ps) = \text{sum-list } (\text{map } (\text{insertion } \alpha) ps)$
by (*induct ps, auto simp: insertion-add*)

lemma *coeff-uminus*: $\text{coeff } (- p) m = - \text{coeff } p m$
by (*simp add: coeff-def uminus-mpoly.rep-eq*)

lemma *insertion-substitute*: $\text{insertion } \alpha (\text{substitute } \sigma p) = \text{insertion } (\lambda x. \text{insertion } \alpha (\sigma x)) p$
unfolding *substitute-def*
proof (*induct p rule: mpoly-induct*)
case (*monom m a*)
show *?case*
apply (*subst replace-coeff-monom*)
subgoal by (*simp add: Const-0*)
subgoal proof (*induct m arbitrary: a rule: poly-mapping-induct*)
case (*single k v*)
show *?case by* (*simp add: insertion-mult insertion-Const insertion-power*)
next
case (*sum f g k v a*)
from *sum(1)[of 1] sum(2)[of a]* **show** *?case*
by (*simp add: insertion-monom-add insertion-mult Const-1*)
qed
done

next
case (*sum p1 p2 m a*)
then show *?case*
apply (*subst replace-coeff-add*)
subgoal by (*simp add: Const-0*)
subgoal by (*transfer', simp add: Const₀-def single-add*)

by (*simp add: insertion-add*)
qed

lemma Const-add: $Const (x + y) = Const x + Const y$
 by (*transfer, auto simp: Const₀-def single-add*)

lemma substitute-add[*simp*]: $substitute \sigma (p + q) = substitute \sigma p + substitute \sigma q$
unfolding *substitute-def insertion-add[symmetric]*
 by (*subst replace-coeff-add, auto simp: Const-0 Const-add*)

lemma Const-sum: $Const (sum f A) = sum (Const o f) A$
 by (*metis Const-0 Const-add sum-comp-morphism*)

lemma Const-sum-list: $Const (sum-list (map f xs)) = sum-list (map (Const o f) xs)$
 by (*induct xs, auto simp: Const-0 Const-add*)

lemma Const-0-eq[*simp*]: $Const x = 0 \longleftrightarrow x = 0$
 by (*smt (verit) Const.abs-eq Const₀-def coeff-monom monom.abs-eq single-zero when-def zero-mpoly-def*)

lemma Const-sum-any: $Const (Sum-any f) = Sum-any (Const o f)$
unfolding *Sum-any.expand-set Const-sum o-def*
 by (*intro sum.cong[OF - refl], auto simp: Const-0*)

lemma Const-mult: $Const (x * y) = Const x * Const y$
 by (*metis Const.abs-eq Const₀-def monom.abs-eq smult-conv-mult smult-monom*)

lemma Const-power: $Const (x \wedge e) = Const x \wedge e$
 by (*induct e, auto simp: Const-1 Const-mult*)

lemma lookup-replace-Const: $lookup (mapping-of (replace-coeff Const p)) l = Const (lookup (mapping-of p) l)$
 by (*metis Const-0 coeff-def coeff-replace-coeff*)

lemma replace-coeff-mult: $replace-coeff Const (p * q) = replace-coeff Const p * replace-coeff Const q$
apply (*subst coeff-eq[symmetric], intro ext, subst coeff-replace-coeff, rule Const-0*)
apply (*unfold coeff-def*)
apply (*unfold times-mpoly.rep-eq*)
apply (*unfold Poly-Mapping.lookup-mult*)
apply (*unfold Const-sum-any o-def Const-mult lookup-replace-Const*)
apply (*unfold when-def if-distrib Const-0*)
 by *auto*

lemma substitute-mult[*simp*]: $substitute \sigma (p * q) = substitute \sigma p * substitute \sigma q$

unfolding *substitute-def insertion-mult[symmetric] replace-coeff-mult ..*

lemma *replace-coeff-Var[simp]: replace-coeff Const (Var x) = Var x*
by (*metis Const-0 Const-1 Var.abs-eq Var₀-def monom.abs-eq replace-coeff-monom*)

lemma *replace-coeff-Const[simp]: replace-coeff Const (Const c) = Const (Const c)*
by (*metis Const.abs-eq Const₀-def Const-0 monom.abs-eq replace-coeff-monom*)

lemma *substitute-Var[simp]: substitute σ (Var x) = σ x*
unfolding *substitute-def* **by** (*simp add: insertion-Var*)

lemma *substitute-Const[simp]: substitute σ (Const c) = Const c*
unfolding *substitute-def* **by** (*simp add: insertion-Const*)

lemma *substitute-0[simp]: substitute σ 0 = 0*
using *substitute-Const[of σ 0, unfolded Const-0]* .

lemma *substitute-1[simp]: substitute σ 1 = 1*
using *substitute-Const[of σ 1, unfolded Const-1]* .

lemma *substitute-power[simp]: substitute σ ($p^{\wedge}e$) = (substitute σ p) $^{\wedge}e$*
by (*induct e, auto*)

lemma *substitute-monom[simp]: substitute σ (monom (monomial e x) c) = Const c * (σ x) $^{\wedge}e$*
by (*simp add: replace-coeff-monom substitute-def*)

lemma *substitute-sum-list: substitute σ (sum-list (map f xs)) = sum-list (map (substitute σ o f) xs)*
by (*induct xs, auto*)

lemma *substitute-sum: substitute σ (sum f xs) = sum (substitute σ o f) xs*
by (*induct xs rule: infinite-finite-induct, auto*)

lemma *substitute-prod: substitute σ (prod f xs) = prod (substitute σ o f) xs*
by (*induct xs rule: infinite-finite-induct, auto*)

definition *vars-list where vars-list = sorted-list-of-set o vars*

lemma *set-vars-list[simp]: set (vars-list p) = vars p*
unfolding *vars-list-def o-def* **using** *vars-finite[of p]* **by** *auto*

lift-definition *mpoly-coeff-filter :: ('a :: zero \Rightarrow bool) \Rightarrow 'a mpoly \Rightarrow 'a mpoly is*
 λ *f p. Poly-Mapping.mapp (λ m c. c when f c) p .*

lemma *mpoly-coeff-filter: coeff (mpoly-coeff-filter f p) m = (coeff p m when f (coeff p m))*
unfolding *coeff-def* **by** *transfer (simp add: in-keys-iff mapp.rep-eq)*

lemma *total-degree-add*: **assumes** *total-degree* $p \leq d$ *total-degree* $q \leq d$
shows *total-degree* $(p + q) \leq d$
using *assms*
proof *transfer*
fix d **and** $p\ q :: (\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a$
let $?exp = \lambda p. \text{Max} (\text{insert} (0 :: \text{nat}) ((\lambda m. \text{sum} (\text{lookup } m) (\text{keys } m)) \text{ `keys } p))$
assume $d: ?exp\ p \leq d\ ?exp\ q \leq d$
have $?exp\ (p + q) \leq \text{Max} (\text{insert} (0 :: \text{nat}) ((\lambda m. \text{sum} (\text{lookup } m) (\text{keys } m)) \text{ `keys } p \cup \text{keys } q))$
using *Poly-Mapping.keys-add*[*of p q*]
by (*intro Max-mono*, *auto*)
also have $\dots = \max (?exp\ p) (?exp\ q)$
by (*subst Max-Un[symmetric]*, *auto simp: image-Un*)
also have $\dots \leq d$ **using** d **by** *auto*
finally show $?exp\ (p + q) \leq d$.
qed

lemma *total-degree-Var[simp]*: *total-degree* $(\text{Var } x :: 'a :: \text{comm-semiring-1 mpoly}) = \text{Suc } 0$
by (*transfer*, *auto simp: Var₀-def*)

lemma *total-degree-Const[simp]*: *total-degree* $(\text{Const } x) = 0$
by (*transfer*, *auto simp: Const₀-def*)

lemma *total-degree-Const-mult*: **assumes** *total-degree* $p \leq d$
shows *total-degree* $(\text{Const } x * p) \leq d$
using *assms*
proof (*transfer*, *goal-cases*)
case $(1\ p\ d\ x)$
have $\text{sub: keys } (\text{Const}_0\ x * p) \subseteq \text{keys } p$
by (*rule order.trans[OF keys-mult]*, *auto simp: Const₀-def*)
show $?case$
by (*rule order.trans[OF - 1]*, *rule Max-mono*, *insert sub*, *auto*)
qed

lemma *vars-0[simp]*: *vars* $0 = \{\}$
unfolding *vars-def* **by** (*simp add: zero-mpoly.rep-eq*)

lemma *vars-1[simp]*: *vars* $1 = \{\}$
unfolding *vars-def* **by** (*simp add: one-mpoly.rep-eq*)

lemma *vars-Var[simp]*: *vars* $(\text{Var } x :: 'a :: \text{comm-semiring-1 mpoly}) = \{x\}$
unfolding *vars-def* **by** (*transfer*, *auto simp: Var₀-def*)

lemma *vars-Const[simp]*: *vars* $(\text{Const } c) = \{\}$
unfolding *vars-def* **by** (*transfer*, *auto simp: Const₀-def*)

lemma *coeff-sum-list*: $\text{coeff } (\text{sum-list } ps) \ m = (\sum p \leftarrow ps. \text{coeff } p \ m)$
by (*induct ps, auto simp: coeff-add[symmetric]*)
(metis coeff-monom monom-zero zero-when)

lemma *coeff-Const-mult*: $\text{coeff } (\text{Const } c * p) \ m = c * \text{coeff } p \ m$
by (*metis Const.abs-eq Const₀-def add-0 coeff-monom-mult monom.abs-eq*)

lemma *coeff-Const*: $\text{coeff } (\text{Const } c) \ m = (\text{if } m = 0 \text{ then } (c :: 'a :: \text{comm-semiring-1}) \text{ else } 0)$

by (*simp add: Const.rep-eq Const₀-def coeff-def lookup-single-not-eq*)

lemma *coeff-Var*: $\text{coeff } (\text{Var } x) \ m = (\text{if } m = \text{monomial } 1 \ x \text{ then } 1 :: 'a :: \text{comm-semiring-1} \text{ else } 0)$

by (*simp add: Var.rep-eq Var₀-def coeff-def lookup-single-not-eq*)

list-based representations, so that polynomials can be converted to first-order terms

lift-definition *monom-list* :: $'a :: \text{comm-semiring-1} \ \text{mpoly} \Rightarrow (\text{monom} \times 'a) \ \text{list}$
is $\lambda p. \text{map } (\lambda m. (m, \text{lookup } p \ m)) \ (\text{sorted-list-of-set } (\text{keys } p))$.

lift-definition *var-list* :: $\text{monom} \Rightarrow (\text{var} \times \text{nat}) \ \text{list}$
is $\lambda m. \text{map } (\lambda x. (x, \text{lookup } m \ x)) \ (\text{sorted-list-of-set } (\text{keys } m))$.

lemma *monom-list*: $p = (\sum (m, c) \leftarrow \text{monom-list } p. \text{monom } m \ c)$

apply *transfer*

subgoal for *p*

apply (*subst poly-mapping-sum-monomials[symmetric]*)

apply (*subst distinct-sum-list-conv-Sum*)

apply (*unfold distinct-map, simp add: inj-on-def*)

apply (*meson in-keys-iff monomial-inj*)

apply (*unfold set-map image-comp o-def split*)

apply (*subst set-sorted-list-of-set, force*)

by (*smt (verit, best) finite-keys lookup-eq-zero-in-keys-contradict monomial-inj o-def sum.cong sum.reindex-nontrivial*)

done

lemma *monom-list-coeff*: $(m, c) \in \text{set } (\text{monom-list } p) \Longrightarrow \text{coeff } p \ m = c$

unfolding *coeff-def* **by** (*transfer, auto*)

lemma *monom-list-keys*: $(m, c) \in \text{set } (\text{monom-list } p) \Longrightarrow \text{keys } m \subseteq \text{vars } p$

unfolding *vars-def* **by** (*transfer, auto*)

lemma *var-list*: $\text{monom } m \ c = \text{Const } (c :: 'a :: \text{comm-semiring-1}) * (\prod (x, e) \leftarrow \text{var-list } m. (\text{Var } x) \wedge e)$

proof *transfer*

fix *m :: monom and c :: 'a*

have *set: set (sorted-list-of-set (keys m)) = keys m*

by (*subst set-sorted-list-of-set, force+*)

```

have id:  $(\prod (x, y) \leftarrow \text{map } (\lambda x. (x, \text{lookup } m \ x)) \ (\text{sorted-list-of-set } (\text{keys } m)))$ .  $\text{Var}_0$ 
 $x \hat{=} y$ 
  =  $(\prod x \in \text{keys } m. \text{Var}_0 \ x \hat{=} \text{lookup } m \ x)$  (is ?r1 = ?r2)
  apply (unfold map-map o-def split)
  apply (subst prod.distinct-set-conv-list[symmetric])
  by auto
have monomial c m =  $\text{Const}_0 \ c * \text{monomial } 1 \ m$ 
  by (simp add: Const0-one monomial-mp)
also have monomial (1 :: 'a) m = ?r1 unfolding id
proof (induction m rule: poly-mapping-induct)
  case (single k v)
  then show ?case by (auto simp: Var0-power mult-single)
next
  case (sum f g k v)
  have id:  $\text{monomial } (1 :: 'a) \ (f + g) = \text{monomial } 1 \ f * \text{monomial } 1 \ g$ 
    by (simp add: mult-single)
  have keys:  $\text{keys } (f + g) = \text{keys } f \cup \text{keys } g$   $\text{keys } f \cap \text{keys } g = \{\}$ 
    apply (intro keys-plus-ninv-comm-monoid-add)
    using sum(3-4) by simp
  show ?case unfolding id sum(1-2) unfolding keys(1)
    apply (subst prod.union-disjoint, force, force, rule keys)
    apply (intro arg-cong2[of - - - (*)] prod.cong refl)
    apply (insert keys(2), simp add: disjoint-iff in-keys-iff lookup-add)
  by (metis add-cancel-left-left disjoint-iff-not-equal in-keys-iff plus-poly-mapping.rep-eq)
qed
finally show  $\text{monomial } c \ m = \text{Const}_0 \ c * ?r1$  .
qed

```

```

lemma var-list-keys:  $(x, e) \in \text{set } (\text{var-list } m) \implies x \in \text{keys } m$ 
  by (transfer, auto)

```

```

lemma vars-substitute: assumes  $\bigwedge x. \text{vars } (\sigma \ x) \subseteq V$ 
  shows  $\text{vars } (\text{substitute } \sigma \ p) \subseteq V$ 
proof -
  define mcs where  $mcs = \text{monom-list } p$ 
  show ?thesis unfolding monom-list[of p, folded mcs-def]
  proof (induct mcs)
  case (Cons mc mcs)
  obtain m c where  $mc = (m, c)$  by force
  define xes where  $xes = \text{var-list } m$ 
  have monom:  $\text{vars } (\text{substitute } \sigma \ (\text{monom } m \ c)) \subseteq V$  unfolding var-list[of m,
folded xes-def]
  proof (induct xes)
  case (Cons xe xes)
  obtain x e where  $xe = (x, e)$  by force
  from assms have  $\text{vars } (\sigma \ x) \subseteq V$  .
  hence  $x: \text{vars } ((\sigma \ x) \hat{=} e) \subseteq V$ 
  proof (induct e)
  case (Suc e)

```



```

    then show ?case
      by (simp, intro order.trans[OF vars-mult], auto)
    qed force
  have id: substitute  $\sigma$  (Const  $c * (\prod a \leftarrow xe \# xes. \text{case } a \text{ of } (x, a) \Rightarrow \text{Var } x \wedge a)$ )
    =  $\sigma x \wedge e * (\text{Const } c * \text{substitute } \sigma (\prod (x, y) \leftarrow xes. \text{Var } x \wedge y))$  unfolding
  xe
    by (simp add: ac-simps)
  show ?case unfolding id
    apply (rule order.trans[OF vars-mult])
    using Cons  $x$  by auto
  qed force
  show ?case unfolding mc
    apply simp
    apply (rule order.trans[OF vars-add])
    using monom Cons by auto
  qed force
qed

```

lemma insertion-monom-nonneg: **assumes** $\bigwedge x. \alpha x \geq 0$ **and** $c: (c :: 'a :: \{\text{linordered-nonzero-semiring, ordered-semiring-0}\}) \geq 0$

shows $\text{insertion } \alpha (\text{monom } m c) \geq 0$

proof –

define xes **where** $xes = \text{var-list } m$

show ?thesis **unfolding** $\text{var-list}[of m c, \text{folded } xes\text{-def}]$

proof (induct xes)

case Nil

thus ?case **using** c **by** (auto simp: insertion-Const)

next

case (Cons $xe xes$)

obtain $x e$ **where** $xe = (x, e)$ **by** force

have id: $\text{insertion } \alpha (\text{Const } c * (\prod a \leftarrow xe \# xes. \text{case } a \text{ of } (x, a) \Rightarrow \text{Var } x \wedge a))$

= $\alpha x \wedge e * \text{insertion } \alpha (\text{Const } c * (\prod a \leftarrow xes. \text{case } a \text{ of } (x, a) \Rightarrow \text{Var } x \wedge a))$

unfolding xe

by (simp add: insertion-mult insertion-power insertion-Var algebra-simps)

show ?case **unfolding id**

proof (intro mult-nonneg-nonneg Cons)

show $0 \leq \alpha x \wedge e$ **using** $\text{assms}(1)[of x]$

by (induct e , auto)

qed

qed

qed

lemma insertion-nonneg: **assumes** $\bigwedge x. \alpha x \geq 0$ ($0 :: 'a :: \text{linordered-idom}$)

and $\bigwedge m. \text{coeff } p m \geq 0$

shows $\text{insertion } \alpha p \geq 0$

proof –

```

define mcs where mcs = monom-list p
from monom-list[of p] have p: p = ( $\sum (m, c) \leftarrow$  mcs. monom m c) unfolding
mcs-def by auto
have mcs: (m,c)  $\in$  set mcs  $\implies$  c  $\geq$  0 for m c
  using monom-list-coeff assms(2) unfolding mcs-def by auto
show ?thesis using mcs unfolding p
proof (induct mcs)
  case Nil
  thus ?case by (auto simp: insertion-Const)
next
  case (Cons mc mcs)
  obtain m c where mc: mc = (m,c) by force
  with Cons have c  $\geq$  0 by auto
  from insertion-monom-nonneg[OF assms(1) this]
  have m: 0  $\leq$  insertion  $\alpha$  (monom m c) by auto
  from Cons(1)[OF Cons(2)]
  have IH: 0  $\leq$  insertion  $\alpha$  ( $\sum a \leftarrow$  mcs. case a of (a, b)  $\implies$  monom a b) by force
  show ?case unfolding mc using IH m
  by (auto simp: insertion-add)
qed
qed

```

```

lemma vars-sumlist: vars (sum-list ps)  $\subseteq$   $\bigcup$  (vars 'set ps)
by (induct ps, insert vars-add, auto)

```

```

lemma coefficients-of-linear-poly: assumes linear: total-degree (p :: 'a :: comm-semiring-1
mpoly)  $\leq$  1

```

```

  shows  $\exists$  c a vs. p = Const c + ( $\sum i \leftarrow$  vs. Const (a i) * Var i)
   $\wedge$  distinct vs  $\wedge$  set vs = vars p  $\wedge$  sorted-list-of-set (vars p) = vs  $\wedge$  ( $\forall v \in$  set
vs. a v  $\neq$  0)
   $\wedge$  ( $\forall i$ . a i = coeff p (monomial 1 i))  $\wedge$  (c = coeff p 0)

```

```

proof -

```

```

  have sum-zero: ( $\bigwedge x$ . x  $\in$  set xs  $\implies$  x = 0)  $\implies$  sum-list (xs :: 'a list) = 0 for
xs by (induct xs, auto)

```

```

  define a :: var  $\Rightarrow$  'a where a i = coeff p (monomial 1 i) for i

```

```

  define vs where vs = sorted-list-of-set (vars p)

```

```

  define c where c = coeff p 0

```

```

  define q where q = Const c + ( $\sum i \leftarrow$  vs. Const (a i) * Var i)

```

```

  show ?thesis

```

```

proof (intro exI[of - vs] exI[of - a] exI[of - c] conjI ballI vs-def[symmetric] c-def
allI a-def,

```

```

  unfold q-def[symmetric])

```

```

  show set vs = vars p and dist: distinct vs

```

```

  using sorted-list-of-set[of vars p, folded vs-def] vars-finite[of p] by auto

```

```

  show p = q

```

```

  unfolding coeff-eq[symmetric]

```

```

proof (intro ext)

```

```

  fix m

```

```

  have coeff q m = coeff (Const c) m + ( $\sum x \leftarrow$  vs. a x * coeff (Var x) m)

```

```

      unfolding q-def coeff-add[symmetric] coeff-sum-list map-map o-def coeff-Const-mult ..
    also have ... = coeff p m
    proof (cases m = 0)
      case True
      thus ?thesis by (simp add: coeff-Const coeff-Var monomial-0-iff c-def)
    next
      case False
    from False have coeff (Const (coeff p 0)) m + ( $\sum x \leftarrow vs. a x * coeff (Var x) m$ )
    = ( $\sum x \leftarrow vs. a x * coeff (Var x) m$ ) unfolding coeff-Const by simp
    also have ... = coeff p m
    proof (cases  $\exists i \in set\ vs. m = monomial\ 1\ i$ )
      case True
      then obtain i where i:  $i \in set\ vs$  and m:  $m = monomial\ 1\ i$  by auto
      from split-list[OF i] obtain bef aft where id:  $vs = bef @ i \# aft$  by auto
      from id dist have i:  $i \notin set\ bef$   $i \notin set\ aft$  by auto
      have [simp]: ( $monomial\ (Suc\ 0)\ i = monomial\ (Suc\ 0)\ j = (i = j)$ ) for i
      j :: var
      using monomial-inj by fastforce
      show ?thesis
      apply (subst id, unfold coeff-Var m, simp)
      apply (subst sum-zero, use i in force)
      apply (subst sum-zero, use i in force)
      by (simp add: a-def)
    next
      case mon: False
      hence one: ( $\sum x \leftarrow vs. a x * coeff (Var x) m = 0$ )
      by (intro sum-zero, auto simp: coeff-Var)
      have two:  $coeff\ p\ m = 0$ 
      proof (rule ccontr)
        assume n0:  $coeff\ p\ m \neq 0$ 
        show False
        proof (cases  $\exists i. m = monomial\ 1\ i$ )
          case True
          with mon obtain i where i:  $i \notin set\ vs$  and m:  $m = monomial\ 1\ i$  by
          auto
          from n0 m have  $i \in vars\ p$  unfolding vars-def coeff-def
          by (metis UN-I in-keys-iff lookup-single-eq one-neq-zero)
          with i  $\langle set\ vs = vars\ p \rangle$  show False by auto
        next
          case False
          have sum (lookup m) (keys m)  $\leq total-degree\ p$  using n0 unfolding
          coeff-def
          apply transfer
          by transfer (metis (no-types, lifting) Max-ge finite.insertI finite-imageI
          finite-keys image-eqI in-keys-iff insertCI)
          also have ...  $\leq 1$  using linear .
          finally have linear: sum (lookup m) (keys m)  $\leq 1$  by auto
        qed
      qed
    qed
  
```

```

consider (single) x where keys m = {x} | (null) keys m = {} |
  (two) x y k where keys m = {x,y} ∪ k and x ≠ y by blast
thus False
proof cases
  case null
    hence m = 0 by simp
    with  $\langle m \neq 0 \rangle$  show False by simp
  next
    case (single x)
      with linear have lookup m x ≤ 1 by auto
      moreover from single have nz: lookup m x ≠ 0
        by (metis in-keys-iff insertI1)
      ultimately have lookup m x = 1 by auto
      with single have m = monomial 1 x
by (metis Diff-cancel Diff-eq-empty-iff keys-subset-singleton-imp-monomial)
      with False show False by auto
    next
      case (two x y k)
        define k' where k' = k - {x,y}
        have keys m = insert x (insert y k') x ≠ y x ∉ k' y ∉ k' finite k'
          unfolding k'-def using two finite-keys[of m] by auto
        hence lookup m x + lookup m y ≤ sum (lookup m) (keys m) by simp
        also have  $\dots \leq 1$  by fact
        finally have lookup m x = 0 ∨ lookup m y = 0 by auto
        with two show False by blast
      qed
    qed
  qed
from one two show ?thesis by simp
qed
finally show ?thesis by (simp add: c-def)
qed
finally show coeff p m = coeff q m ..
qed

fix v
assume v: v ∈ set vs
hence v ∈ vars p using  $\langle \text{set } vs = \text{vars } p \rangle$  by auto
hence vq: v ∈ vars q unfolding  $\langle p = q \rangle$  .
from split-list[OF v] obtain bef aft where vs: vs = bef @ v # aft by auto
with dist have vba: v ∉ set bef v ∉ set aft by auto
show a v ≠ 0
proof
  assume a0: a v = 0
  have v ∈ vars p by fact
  also have p = q by fact
  also have vars q ⊆ vars (sum-list (map (λ x. Const (a x) * Var x) bef)) ∪
    vars (Const (a v) * Var v)
     $\cup$  vars (sum-list (map (λ x. Const (a x) * Var x) aft))

```

```

unfolding q-def vs apply simp
apply (rule order.trans[OF vars-add], simp)
apply (rule order.trans[OF vars-add])
by (insert vars-add, blast)
also have vars (Const (a v) * Var v) = {} unfolding a0 Const-0 by simp
finally obtain list where v: v ∈ vars (sum-list (map (λ x. Const (a x) * Var
x) list))
and not-v: v ∉ set list using vba by auto
from set-mp[OF vars-sumlist v] obtain x where x ∈ set list and v ∈ vars
(Const (a x) * Var x)
by auto
with vars-mult[of Const (a x) Var x] not-v show False by auto
qed
qed
qed

```

Introduce notion for degree of monom

definition *degree-monom :: (var ⇒₀ nat) ⇒ nat* **where**
degree-monom m = sum (lookup m) (keys m)

lemma *total-degree-alt-def: total-degree p = Max (insert 0 (degree-monom ‘ keys*
(mapping-of p)))

unfolding *degree-monom-def*
by *transfer' simp*

lemma *degree-monom-le-total-degree: assumes coeff p m ≠ 0*

shows *degree-monom m ≤ total-degree p*

using *assms* **unfolding** *total-degree-alt-def* **by** (*simp add: coeff-keys*)

lemma *degree-monom-eq-total-degree: assumes p ≠ 0*

shows $\exists m. \text{coeff } p \ m \neq 0 \wedge \text{degree-monom } m = \text{total-degree } p$

proof (*cases total-degree p = 0*)

case *False*

thus *?thesis* **unfolding** *total-degree-alt-def*

by (*metis (full-types) Max-in coeff-keys empty-not-insert finite-imageI finite-insert*
finite-keys image-iff insertE)

next

case *True*

from *assms* **obtain** *m where coeff p m ≠ 0*

using *coeff-all-0* **by** *auto*

with *degree-monom-le-total-degree[OF this]* *True* **show** *?thesis* **by** *auto*

qed

lemma *degree-add-leI: degree p x ≤ d ⇒ degree q x ≤ d ⇒ degree (p + q) x ≤*
d

apply *transfer*

subgoal for *p x d q* **using** *Poly-Mapping.keys-add[of p q]*

by (*intro Max.boundedI, auto*)

done

lemma *degree-sum-leI*: **assumes** $\bigwedge i. i \in A \implies \text{degree } (p \ i) \ x \leq d$
shows $\text{degree } (\text{sum } p \ A) \ x \leq d$
using *assms*
by (*induct A rule: infinite-finite-induct, auto intro: degree-add-leI*)

lemma *total-degree-sum-leI*: **assumes** $\bigwedge i. i \in A \implies \text{total-degree } (p \ i) \leq d$
shows $\text{total-degree } (\text{sum } p \ A) \leq d$
using *assms*
by (*induct A rule: infinite-finite-induct, auto intro: total-degree-add*)

lemma *total-degree-monom*: **assumes** $c \neq 0$
shows $\text{total-degree } (\text{monom } m \ c) = \text{degree-monom } m$
unfolding *total-degree-alt-def* **using** *assms* **by** *auto*

lemma *degree-Var[simp]*: $\text{degree } (\text{Var } x :: 'a :: \text{comm-semiring-1 mpoly}) \ x = 1$
by (*transfer, unfold Var₀-def, simp*)

lemma *Var-neq-0[simp]*: $\text{Var } x \neq (0 :: 'a :: \text{comm-semiring-1 mpoly})$
proof
assume $\text{Var } x = (0 :: 'a \ \text{mpoly})$
from *arg-cong[OF this, of $\lambda p. \text{degree } p \ x$]*
show *False* **by** *simp*
qed

lemma *degree-Const[simp]*: $\text{degree } (\text{Const } c) \ x = 0$
by *transfer (auto simp: Const₀-def)*

lemma *vars-add-subI*: $\text{vars } p \subseteq A \implies \text{vars } q \subseteq A \implies \text{vars } (p + q) \subseteq A$
by (*metis le-supI subset-trans vars-add*)

lemma *vars-mult-subI*: $\text{vars } p \subseteq A \implies \text{vars } q \subseteq A \implies \text{vars } (p * q) \subseteq A$
by (*metis le-supI subset-trans vars-mult*)

lemma *vars-eqI*: **assumes** $\text{vars } (p :: 'a :: \text{comm-ring-1 mpoly}) \subseteq V$
 $\bigwedge v. v \in V \implies \exists a \ b. \text{insertion } a \ p \neq \text{insertion } (a(v := b)) \ p$
shows $\text{vars } p = V$
proof (*rule ccontr*)
assume $\neg ?thesis$
with *assms* **obtain** v **where** $v \in V$ **and** *not: $v \notin \text{vars } p$* **by** *auto*
from *assms(2)[OF this(1)]* **obtain** $a \ b$ **where** $\text{insertion } a \ p \neq \text{insertion } (a(v := b)) \ p$ **by** *auto*
moreover **have** $\text{insertion } a \ p = \text{insertion } (a(v := b)) \ p$
by (*rule insertion-irrelevant-vars, insert not, auto*)
ultimately show *False* **by** *auto*
qed

end

2.2 Part 2 – Extensions With Importing Univariate Polynomials

```

theory Preliminaries-on-Polynomials-2
  imports
    Preliminaries-on-Polynomials-1
    Factor-Algebraic-Polynomial.Poly-Connection
begin

```

Several definitions have the same name for univariate and multivariate polynomials, so we use a prefix *m* for multi-variate.

```

hide-const (open) Symmetric-Polynomials.lead-coeff

```

```

abbreviation mdegree where mdegree  $\equiv$  MPoly-Type.degree
abbreviation mcoeff where mcoeff  $\equiv$  MPoly-Type.coeff
abbreviation mmonom where mmonom  $\equiv$  MPoly-Type.monom

```

```

lemma range-coeff-poly-to-mpoly: assumes mcoeff (poly-to-mpoly x p) m  $\neq$  0
  shows  $\exists$  d. m = monomial d x
  using assms
  unfolding coeff-def poly-to-mpoly-def MPoly-inverse[OF Set.UNIV-I] lookup-Abs-poly-mapping[OF poly-to-mpoly-finite]
  by simp (metis keys-subset-singleton-imp-monomial)

```

```

lemma degree-poly-to-mpoly[simp]: mdegree (poly-to-mpoly x p) x = degree p
proof (cases p = 0)
  case True
    thus ?thesis by (simp add: poly-to-mpoly0)
  next
    case p: False
      let ?q = poly-to-mpoly x p
      define q where q = ?q
      define dp where dp = degree p
      define dq where dq = mdegree q x
      from p have q: ?q  $\neq$  0
        by (metis poly-to-mpoly0 poly-to-mpoly-inverse)
      have pq: p = mpoly-to-poly x q unfolding q-def
        by (simp add: poly-to-mpoly-inverse)
      {
        have 0  $\neq$  coeff p dp using p by (auto simp: dp-def)
        also have coeff p dp = coeff (mpoly-to-poly x q) dp unfolding pq by simp
        also have ... = mcoeff q (monomial dp x) unfolding coeff-mpoly-to-poly by
simp
        finally have mcoeff q (monomial dp x)  $\neq$  0 by simp
      }
      hence first-part: dq  $\geq$  dp unfolding dq-def by (metis degree-geI lookup-single-eq)
      {
        from monom-of-degree-exists[OF q, folded q-def, of x] obtain m where mc:
mcoeff q m  $\neq$  0
      }

```

and *look*: *lookup* $m\ x = dq$ **by** (*auto simp*: *dq-def*)
from *range-coeff-poly-to-mpoly*[*OF mc*[*unfolded q-def*]] **obtain** d **where** m : m
 $=$ *monomial* $d\ x$ **by** *auto*
from m *look* **have** m : $m =$ *monomial* $dq\ x$ **by** *simp*
have *coeff* $p\ dq = m$ *coeff* q (*monomial* $dq\ x$)
unfolding *coeff-poly-to-mpoly*[*of x, symmetric*] *q-def dq-def* **by** *auto*
also *have* $\dots \neq 0$ **using** $m\ mc$ **by** *auto*
finally *have* $dp \geq dq$ **unfolding** *dp-def* **by** (*rule le-degree*)
}
with *first-part* **have** $dp = dq$ **by** *auto*
thus *?thesis* **unfolding** *dp-def dq-def q-def* **by** *auto*
qed

lemma *degree-mpoly-to-poly*: **assumes** $vars\ p \subseteq \{x\}$
shows *degree* (*mpoly-to-poly* $x\ p$) $=$ *mdegree* $p\ x$
proof –
define q **where** $q =$ *mpoly-to-poly* $x\ p$
from *mpoly-to-poly-inverse*[*OF assms*]
have *mdegree* $p\ x = m$ *degree* (*poly-to-mpoly* x (*mpoly-to-poly* $x\ p$)) x **by** *simp*
also *have* $\dots =$ *degree* (*mpoly-to-poly* $x\ p$) **by** *simp*
finally *show* *?thesis* **..**
qed

lemma *degree-partial-insertion-bound*: *degree* (*partial-insertion* $a\ x\ p$) \leq *MPoly-Type.degree*
 $p\ x$
using *degree-partial-insertion-le-mpoly* **by** *auto*

lemma *insertion-partial-insertion-vars*: **assumes** $\bigwedge y. y \neq x \implies y \in vars\ p \implies$
 $\beta\ y = \alpha\ y$
shows *poly* (*partial-insertion* $\beta\ x\ p$) ($\alpha\ x$) $=$ *insertion* $\alpha\ p$
proof –
let $?\alpha = (\lambda y. \text{if } y \in \text{insert } x\ (vars\ p) \text{ then } \alpha\ y \text{ else } \beta\ y)$
have *insertion* $\alpha\ p =$ *insertion* $?\alpha\ p$
by (*rule insertion-irrelevant-vars, auto*)
also *have* $\dots =$ *poly* (*partial-insertion* $\beta\ x\ p$) ($?\alpha\ x$)
by (*rule insertion-partial-insertion*[*symmetric*], *insert assms, auto*)
finally *show* *?thesis* **by** *auto*
qed

lemma *degree-mpoly-of-poly*[*simp*]: *mdegree* (*mpoly-of-poly* $x\ p$) $x =$ *degree* p
proof –
have *mdegree* (*mpoly-of-poly* $x\ p$) $x \leq$ *degree* p
by (*simp add*: *coeff-eq-0 coeff-mpoly-of-poly degree-leI*)
moreover *have* *degree* $p \leq$ *mdegree* (*mpoly-of-poly* $x\ p$) x
proof (*cases degree p = 0*)
case *True*
thus *?thesis* **by** *auto*
next
case 0 : *False*

hence $\text{coeff } p \text{ (degree } p) \neq 0$ **by** *auto*
 also have $\text{coeff } p \text{ (degree } p) = \text{MPoly-Type.coeff (mpoly-of-poly } x \text{ } p)$ (*monomial*
(degree } p) \text{ } x)
 by *simp*
 finally show *?thesis* **by** (*metis degree-geI lookup-single-eq*)
qed
 ultimately show *?thesis* **by** *auto*
qed

lemma *mpoly-extI*: **assumes** $\bigwedge \alpha. \text{insertion } \alpha \text{ } p = \text{insertion } \alpha \text{ } (q :: 'a :: \{\text{ring-char-0, idom}\} \text{mpoly})$

shows $p = q$

proof –

have *main*: $\text{finite } vs \implies \text{vars } p \subseteq vs \implies \text{vars } q \subseteq vs \implies (\bigwedge \alpha. \text{insertion } \alpha \text{ } p = \text{insertion } \alpha \text{ } q) \implies p = q$ **for** *vs*

proof (*induction vs arbitrary: p q rule: finite-induct*)

case (*insert x vs p q*)

have $p = q \iff \text{mpoly-to-mpoly-poly } x \text{ } p = \text{mpoly-to-mpoly-poly } x \text{ } q$

by (*metis poly-mpoly-to-mpoly-poly*)

also have $\dots \iff (\forall m. \text{coeff (mpoly-to-mpoly-poly } x \text{ } p) \text{ } m = \text{coeff (mpoly-to-mpoly-poly } x \text{ } q) \text{ } m)$

by (*metis poly-eqI*)

also have \dots **using** *insert*

proof (*intro allI insert.IH*)

fix *m* α

show $\text{vars (coeff (mpoly-to-mpoly-poly } x \text{ } p) \text{ } m) \subseteq vs$ **using** *insert.prem1*

by (*metis Diff-eq-empty-iff Diff-insert2 dual-order.trans vars-coeff-mpoly-to-mpoly-poly*)

show $\text{vars (coeff (mpoly-to-mpoly-poly } x \text{ } q) \text{ } m) \subseteq vs$ **using** *insert.prem2*

by (*metis Diff-eq-empty-iff Diff-insert2 dual-order.trans vars-coeff-mpoly-to-mpoly-poly*)

have *IH*: $\text{partial-insertion } \alpha \text{ } x \text{ } p = \text{partial-insertion } \alpha \text{ } x \text{ } q$

proof (*intro poly-ext*)

fix *y*

have $\text{poly (partial-insertion } \alpha \text{ } x \text{ } p) \text{ } y = \text{poly (partial-insertion } \alpha \text{ } x \text{ } q) \text{ } y \iff$
 $\text{insertion } (\alpha(x := y)) \text{ } p = \text{insertion } (\alpha(x := y)) \text{ } q$

using *insertion-partial-insertion[of x* α $\alpha(x := y)$ *]* **by** *simp*

moreover have \dots **by** (*intro insert*)

finally show $\text{poly (partial-insertion } \alpha \text{ } x \text{ } p) \text{ } y = \text{poly (partial-insertion } \alpha \text{ } x \text{ } q) \text{ } y$ **by** *blast*

qed

show $\text{insertion } \alpha \text{ } (\text{coeff (mpoly-to-mpoly-poly } x \text{ } p) \text{ } m) = \text{insertion } \alpha \text{ } (\text{coeff (mpoly-to-mpoly-poly } x \text{ } q) \text{ } m)$

using *insert.prem3* **by** (*simp add: IH*)

qed

finally show *?case* .

next

case (*empty p q*)

hence $\text{vars } p = \{\} \text{vars } q = \{\}$ **by** *auto*

from *vars-emptyE[OF vars(1)]* **obtain** *c* **where** $p = \text{Const } c$.

from *vars-emptyE[OF vars(2)]* **obtain** *d* **where** $q = \text{Const } d$.

```

    from empty(3)[of undefined, unfolded p q] have c = d by auto
    thus ?case unfolding p q by simp
qed
show ?thesis
  by (rule main[of vars p ∪ vars q], insert assms, auto simp: vars-finite)
qed

lemma vars-empty-Const: assumes vars (p :: 'a :: {ring-char-0, idom} mpoly) =
{}
shows ∃ c. p = Const c
proof -
{
  fix α
  have insertion α p = insertion (λ -. 0) p using assms
  by (intro insertion-irrelevant-vars, auto)
  also have ... = mcoeff p 0 by simp
  also have ... = insertion α (Const (mcoeff p 0)) unfolding insertion-Const
..
  finally have insertion α p = insertion α (Const (mcoeff p 0)) .
}
hence p = (Const (mcoeff p 0)) by (rule mpoly-extI)
thus ?thesis by auto
qed

```

context

```

  assumes ge1:  $\bigwedge c :: 'a :: \text{linordered-idom}. c > 0 \implies \exists x. c * x \geq 1$ 
begin

```

lemma poly-ext-bounded:

```

  fixes p q :: 'a poly
  assumes  $\bigwedge x. x \geq b \implies \text{poly } p \ x = \text{poly } q \ x$  shows p = q
proof -
  define r where r = p - q
  from assms have r:  $x \geq b \implies \text{poly } r \ x = 0$  for x by (auto simp: r-def)
  have ?thesis  $\longleftrightarrow r = 0$  unfolding r-def by simp
  also have ...
proof (cases degree r = 0)
  case True
  from degree0-coeffs[OF this] r[of b] show ?thesis by auto
next
  case dr: False
  define lc where lc = lead-coeff r
  from dr have lc:  $lc \neq 0$  by (auto simp: lc-def)
  define d where d = degree r
  define s where s = r - monom lc d
  have ds:  $\text{degree } s < d$  unfolding s-def lc-def using dr
  by (smt (verit, del-insts) Polynomial.coeff-diff Polynomial.coeff-monom
  cancel-comm-monoid-add-class.diff-cancel coeff-eq-0 d-def degree-0

```

```

diff-is-0-eq leading-coeff-0-iff linorder-neqE-nat linorder-not-le zero-diff)
{
  fix x
  have poly r x = poly (monom lc d + s) x unfolding s-def by simp
  also have ... = lc * x ^ d + poly s x by (simp add: poly-monom)
  finally have poly r x = lc * x ^ d + poly s x .
} note eq = this
have ∃ p c. (∀ x ≥ b. (c :: 'a) * x ^ d + poly p x = 0) ∧ c > 0 ∧ degree p <
d
proof (cases lc > 0)
  case True
  show ?thesis by (rule exI[of - s], rule exI[of - lc], insert True eq r ds, auto)
next
  case False
  with lc have True: - lc > 0 by auto
  show ?thesis
  proof (rule exI[of - s], rule exI[of - - lc], intro conjI allI True)
    fix x
    show b ≤ x → - lc * x ^ d + poly (- s) x = 0 using r[of x] eq[of x] by
auto
  qed (insert ds, auto)
qed
then obtain p and c :: 'a
  where c: c > 0 and dp: degree p < d and 0: ∧ x. x ≥ b ⇒ c * x ^ d +
poly p x = 0
  by auto
define m where m = Max (insert 1 ((λ i. abs (coeff p i)) ‘{..degree p}'))
define M where M = (1 + of-nat (degree p)) * m
have m1: m ≥ 1 unfolding m-def by auto
have mc: i ≤ degree p ⇒ m ≥ abs (coeff p i) for i unfolding m-def
  by (intro Max-ge, auto)
define B where B = max b 1
{
  fix x
  assume x: x ≥ B
  hence x1: x ≥ 1 unfolding B-def by auto
  have abs (poly p x) = abs (∑ i ≤ degree p. coeff p i * x ^ i)
    by (simp add: poly-altdef)
  also have ... ≤ (∑ i ≤ degree p. abs (coeff p i * x ^ i)) by blast
  also have ... ≤ (∑ i ≤ degree p. m * x ^ degree p)
  proof (intro sum-mono)
    fix i
    assume i ∈ {..degree p}
    hence i: i ≤ degree p by auto
    have |coeff p i * x ^ i| = |coeff p i| * |x ^ i| by (auto simp: abs-mult)
    also have ... ≤ m * x ^ degree p
  proof (intro mult-mono)
    show |coeff p i| ≤ m using mc i by auto
    show 0 ≤ m using m1 by auto
  end
end

```

```

    have  $|x \wedge i| = |x| \wedge i$  unfolding power-abs ..
    also have  $\dots = x \wedge i$  using x1 by simp
    also have  $\dots \leq x \wedge \text{degree } p$  using x1 i
      using power-increasing by blast
    finally show  $|x \wedge i| \leq x \wedge \text{degree } p$  by auto
  qed simp
  finally show  $|\text{coeff } p \ i * x \wedge i| \leq m * x \wedge \text{degree } p$  by simp
  qed
  also have  $\dots = M * x \wedge \text{degree } p$  by (simp add: M-def)
  finally have ineq:  $|\text{poly } p \ x| \leq M * x \wedge \text{degree } p$  .

  have  $x \geq b$  using x unfolding B-def by auto
  from  $0[OF \ \text{this}]$  have abs  $(c * x \wedge d) = \text{abs} (\text{poly } p \ x)$  by auto
  with ineq have ineq:  $c * x \wedge d \leq M * x \wedge \text{degree } p$  by auto

  define k where  $k = d - \text{Suc} (\text{degree } p)$ 
  from dp have d:  $d = \text{degree } p + \text{Suc } k$  unfolding k-def by auto
  have xp:  $x \wedge \text{degree } p \geq 1$  using x1 by simp
  have  $c * x \wedge d = (c * x \wedge k * x) * x \wedge \text{degree } p$  unfolding d
    by (simp add: algebra-simps power-add)
  from ineq[unfolded this] have ineq:  $c * x \wedge k * x \leq M$  using xp by simp
  have  $c * x \leq c * x \wedge k * x$  using c x1 by fastforce
  also have  $\dots \leq M$  by fact
  finally have  $c * x \leq M$  .
}
  hence contra:  $B \leq x \implies c * x \leq M$  for x .
  have  $\exists x. c * x \geq 1$  using c ge1 by auto
  then obtain d where cd:  $c * d \geq 1$  by auto
  with c have d:  $d > 0$ 
    by (meson less-numeral-extra(1) order-less-le-trans zero-less-mult-pos)
  have M1:  $M \geq 1$  unfolding M-def using m1
    by (simp add: order-trans)

  have  $M < M + 1$  by auto
  also have  $\dots \leq (c * d) * (M + 1)$  using cd M1 by simp
  also have  $\dots \leq c * \text{max } B \ (d * (M + 1))$  using M1 c d by auto
  also have  $\dots \leq M$  using contra[of max B (d * (M + 1))] by simp
  finally have False by simp
  thus ?thesis ..
  qed
  finally show ?thesis by simp
  qed

```

lemma *mpoly-ext-bounded*:

assumes $\bigwedge \alpha. (\bigwedge x. \alpha \ x \geq b) \implies \text{insertion } \alpha \ p = \text{insertion } \alpha \ (q :: 'a :: \text{linordered-idom } \text{mpoly})$

shows $p = q$

proof –

```

have main: finite vs  $\implies$  vars p  $\subseteq$  vs  $\implies$  vars q  $\subseteq$  vs  $\implies$  ( $\bigwedge$   $\alpha$ . ( $\bigwedge$  x.  $\alpha$  x  $\geq$  b)
 $\implies$  insertion  $\alpha$  p = insertion  $\alpha$  q)  $\implies$  p = q for vs
proof (induction vs arbitrary: p q rule: finite-induct)
  case (insert x vs p q)
    have p = q  $\iff$  mpoly-to-mpoly-poly x p = mpoly-to-mpoly-poly x q
      by (metis poly-mpoly-to-mpoly-poly)
    also have ...  $\iff$  ( $\forall$  m. coeff (mpoly-to-mpoly-poly x p) m = coeff (mpoly-to-mpoly-poly
x q) m)
      by (metis poly-eqI)
    also have ...
proof (intro allI insert.IH)
  fix m  $\alpha$ 
  show vars (coeff (mpoly-to-mpoly-poly x p) m)  $\subseteq$  vs using insert.prem1(1)
  by (metis Diff-eq-empty-iff Diff-insert2 dual-order.trans vars-coeff-mpoly-to-mpoly-poly)
  show vars (coeff (mpoly-to-mpoly-poly x q) m)  $\subseteq$  vs using insert.prem1(2)
  by (metis Diff-eq-empty-iff Diff-insert2 dual-order.trans vars-coeff-mpoly-to-mpoly-poly)
  assume alpha:  $\bigwedge$  x.  $\alpha$  (x :: nat)  $\geq$  (b :: 'a)
  have IH: partial-insertion  $\alpha$  x p = partial-insertion  $\alpha$  x q
  proof (intro poly-ext-bounded[of b])
    fix y
    assume y: y  $\geq$  (b :: 'a)
    have poly (partial-insertion  $\alpha$  x p) y = poly (partial-insertion  $\alpha$  x q) y  $\iff$ 
insertion ( $\alpha$ (x := y)) p = insertion ( $\alpha$ (x := y)) q
      using insertion-partial-insertion[of x  $\alpha$   $\alpha$ (x := y)] by simp
    moreover have ... by (intro insert, insert y alpha, auto)
    finally show poly (partial-insertion  $\alpha$  x p) y = poly (partial-insertion  $\alpha$  x
q) y by blast
  qed
  show insertion  $\alpha$  (coeff (mpoly-to-mpoly-poly x p) m) = insertion  $\alpha$  (coeff
(mpoly-to-mpoly-poly x q) m)
    using insert.prem1(3) by (simp add: IH)
  qed
finally show ?case .
next
  case (empty p q)
  hence vars: vars p = {} vars q = {} by auto
  from vars-emptyE[OF vars(1)] obtain c where p = Const c .
  from vars-emptyE[OF vars(2)] obtain d where q = Const d .
  from empty(3)[of  $\lambda$  -. b, unfolded p q] have c = d
    by (simp add: coeff-Const)
  thus ?case unfolding p q by simp
qed
show ?thesis
  by (rule main[of vars p  $\cup$  vars q], insert assms, auto simp: vars-finite)
qed
end

```

lemma mpoly-ext-bounded-int:

assumes \bigwedge α . (\bigwedge x. α x \geq b) \implies insertion α p = insertion α (q :: int mpoly)

shows $p = q$
by (*rule mpoly-ext-bounded*[of b], *insert assms*, *auto simp: exI*[of - 1])

lemma *mpoly-ext-bounded-field*:
assumes $\bigwedge \alpha. (\bigwedge x. \alpha x \geq b) \implies \text{insertion } \alpha p = \text{insertion } \alpha (q :: 'a :: \text{linordered-field mpoly})$
shows $p = q$
apply (*rule mpoly-ext-bounded*[of b])
subgoal for c **by** (*intro exI*[of - inverse c], *auto*)
subgoal using *assms* **by** *auto*
done

lemma *mpoly-of-poly-is-poly-to-mpoly*: $\text{mpoly-of-poly} = \text{poly-to-mpoly}$
unfolding *poly-to-mpoly-def*
apply *transfer'*
apply (*unfold mpoly-of-poly-aux-def*)
apply *transfer'*
apply (*unfold when-def*[*symmetric*])
by (*intro ext*, *auto*)

lemma *insertion-poly-to-mpoly* [*simp*]: $\text{insertion } f (\text{poly-to-mpoly } i p) = \text{poly } p (f i)$
unfolding *mpoly-of-poly-is-poly-to-mpoly*[*symmetric*] **by** *simp*

lemma *substitute-poly-to-mpoly*:
assumes $x: \alpha x = \text{poly-to-mpoly } y (q :: 'a :: \{\text{ring-char-0, idom}\} \text{poly})$
shows $\text{substitute } \alpha (\text{poly-to-mpoly } x p) = \text{poly-to-mpoly } y (\text{pcompose } p q)$
apply (*rule mpoly-extI*)
apply (*unfold insertion-substitute insertion-poly-to-mpoly* x)
apply (*unfold poly-pcompose*)
by *auto*

lemma *total-degree-add-Const*: $\text{total-degree } (p + \text{Const } (c :: 'a :: \text{comm-ring-1})) = \text{total-degree } p$
proof –
have $\text{total-degree } (p + \text{Const } c) \leq \text{total-degree } p$
by (*rule total-degree-add*, *auto*)
moreover have $\text{total-degree } ((p + \text{Const } c) + \text{Const } (-c)) \leq \text{total-degree } (p + \text{Const } c)$
by (*rule total-degree-add*, *auto*)
moreover have $(p + \text{Const } c) + \text{Const } (-c) = p$ **by** (*simp add: Const-add*[*symmetric*])
ultimately show *?thesis* **by** *auto*
qed

lemma *mpoly-as-sum-any*: $(p :: 'a :: \text{comm-ring-1 mpoly}) = \text{Sum-any } (\lambda m. \text{mmonom } m (\text{mcoeff } p m))$
proof (*induct p rule: mpoly-induct*)
case (*monom m a*)
thus *?case*

by transfer (smt (verit) Sum-any.cong Sum-any-when-equal' lookup-single-eq
lookup-single-not-eq single-zero when-neq-zero when-simps(1))

next

case 1: (sum p1 p2 m a)

show ?case

apply (subst 1(1), subst 1(2))

apply (unfold coeff-add monom-add)

by (smt (z3) 1(1) 1(2) MPoly-Type-monom-zero Sum-any.cong Sum-any.distrib
Sum-any.infinite add-cancel-left-left add-cancel-left-right mpoly-coeff-0)

qed

lemma mpoly-as-sum: (p :: 'a :: comm-ring-1 mpoly) = sum (λ m. mmonom m
(mcoeff p m)) {m . mcoeff p m ≠ 0}

apply (subst mpoly-as-sum-any)

by (smt (verit, ccfv-SIG) Collect-cong MPoly-Type-monom-0-iff Sum-any.expand-set)

lemma monom-as-prod: mmonom m c = Const (c :: 'a :: comm-semiring-1) *
prod (λ i. Var i ^ lookup m i) (keys m)

unfolding var-list

apply (intro arg-cong[of - - λ x. - * x])

apply transfer'

apply (subst prod.distinct-set-conv-list[symmetric])

subgoal unfolding distinct-map by (auto simp: inj-on-def)

subgoal unfolding set-map image-comp set-sorted-list-of-set[OF finite-keys]

by (smt (verit, best) case-prod-conv finite-keys o-def prod.cong prod.inject
prod.reindex-nontrivial)

done

lemma poly-to-mpoly-substitute-same: assumes poly-to-mpoly x q = substitute (λ i.
Var x) p

shows poly q a = insertion (λ x. a) p

using arg-cong[OF assms, of insertion (λ -. a), unfolded insertion-poly-to-mpoly
insertion-substitute insertion-Var]

by simp

lemma substitute-monom: fixes c :: 'a :: comm-semiring-1

shows substitute a (mmonom m c) = Const c * prod (λ i. a i ^ lookup m i) (keys
m)

by (subst monom-as-prod) (simp add: substitute-prod o-def)

lemma degree-prod: assumes prod p A ≠ (0 :: 'a :: idom mpoly)

shows mdegree (prod p A) x = sum (λ i. mdegree (p i) x) A

using assms

by (induct A rule: infinite-finite-induct) (auto simp: mpoly-degree-mult-eq)

lemma degree-prod-le: fixes p :: - ⇒ 'a :: idom mpoly

shows mdegree (prod p A) x ≤ sum (λ i. mdegree (p i) x) A

using degree-prod[of p A x] by (cases prod p A = 0; auto)

lemma *degree-power*: **assumes** $p \neq (0 :: 'a :: idom mpoly)$
shows $mdegree (p \hat{=} n) x = n * mdegree p x$
by (*induct n*) (*insert assms, auto simp: mpoly-degree-mult-eq*)

lemma *mdegree-Const-mult-le*: $mdegree (Const (c :: 'a :: idom) * p) x \leq mdegree p x$
using *mpoly-degree-mult-eq*[*of Const c p x*]
by (*cases c = 0; cases p = 0; auto*)

lemma *degree-substitute-const-same-var*: $mdegree (substitute (\lambda i. Const (c i) * Var x) (p :: 'a :: idom mpoly)) x \leq total-degree p$
proof –
{
 fix i
 let $?x = Var x :: 'a mpoly$
 assume $i: mcoeff p i \neq 0$
 have $mdegree (\prod_{ia \in keys i} (Const (c ia) * ?x) \hat{=} lookup i ia) x \leq total-degree p$
 apply (*intro order.trans*[*OF - degree-monom-le-total-degree*][*of p i, OF i*])
 apply (*intro order.trans*[*OF degree-prod-le*])
 apply (*rule order.trans*[*OF sum-mono*][*of - - lookup i*])
 apply (*unfold power-mult-distrib Const-power*[*symmetric*])
 apply (*rule order.trans*[*OF mdegree-Const-mult-le*])
 apply (*subst degree-power, force*)
 apply (*subst degree-Var*)
 by (*auto simp add: degree-monom-def*)
} **note** *main = this*
show *?thesis*
 apply (*subst* (5) *mpoly-as-sum*)
 apply (*unfold substitute-sum o-def substitute-monom substitute-mult*)
 apply (*intro degree-sum-leI*)
 apply (*rule order.trans*[*OF mdegree-Const-mult-le*])
 using *main* **by** *auto*

qed

lemma *degree-substitute-same-var*: $mdegree (substitute (\lambda i. Var x) (p :: 'a :: idom mpoly)) x \leq total-degree p$
using *degree-substitute-const-same-var*[*of \lambda -. 1, unfolded Const-1*] **by** *auto*

lemma *poly-pinfty-ge-int*: **assumes** $0 < lead-coeff (p :: int poly)$
and $degree p \neq 0$
shows $\exists n. \forall x \geq n. b \leq poly p x$
proof –
 let $?q = of-int-poly p :: real poly$
 from *assms* **have** $0 < lead-coeff ?q$ $degree ?q \neq 0$ **by** *auto*
 from *poly-pinfty-ge*[*OF this, of of-int b*] **obtain** n
 where $le: \bigwedge x. x \geq n \implies real-of-int b \leq poly ?q x$ **by** *auto*
 show *?thesis*
 proof (*intro exI*[*of - ceiling n*] *allI impI*)


```

fix x
assume x ≥ [n]
hence of-int x ≥ n by linarith
from le[OF this] show b ≤ poly p x by simp
qed
qed

```

context

```

assumes poly-pinfy-ge:  $\bigwedge p b. 0 < \text{lead-coeff } (p :: 'a :: \text{linordered-idom } \text{poly})$ 
 $\implies \text{degree } p \neq 0 \implies \exists n. \forall x \geq n. b \leq \text{poly } p x$ 

```

begin

lemma degree-mono-generic: **assumes** pos: $\text{lead-coeff } p \geq (0 :: 'a)$

and le: $\bigwedge x. x \geq c \implies \text{poly } p x \leq \text{poly } q x$

shows $\text{degree } p \leq \text{degree } q$

proof (rule ccontr)

let ?lc = lead-coeff

define r **where** $r = p - q$

assume $\neg ?thesis$

hence deg: $\text{degree } p > \text{degree } q$ **by** auto

hence deg-eq: $\text{degree } r = \text{degree } p$ **unfolding** r-def

by (metis degree-add-eq-right degree-minus uminus-add-conv-diff)

from deg **have** ?lc p $\neq 0$ **by** auto

with pos **have** pos: ?lc p > 0 **by** auto

have ?lc r = ?lc p **unfolding** r-def

using deg-eq le-degree r-def deg **by** fastforce

with pos **have** lcr: ?lc r > 0 **by** auto

from deg-eq deg **have** dr: $\text{degree } r \neq 0$ **by** auto

have $x \geq c \implies \text{poly } r x \leq 0$ **for** x **using** le[of x] **unfolding** r-def **by** auto

with poly-pinfy-ge[OF lcr dr] **show** False

by (metis dual-order.trans nle-le not-one-le-zero)

qed

lemma degree-mono'-generic: **assumes** le: $\bigwedge x. x \geq c \implies (bnd :: 'a) \leq \text{poly } p x$

$\wedge \text{poly } p x \leq \text{poly } q x$

shows $\text{degree } p \leq \text{degree } q$

proof (cases degree p = 0)

case deg: False

show ?thesis

proof (rule degree-mono-generic[of - c])

show $\bigwedge x. c \leq x \implies \text{poly } p x \leq \text{poly } q x$ **using** le **by** auto

let ?lc = lead-coeff

show $0 \leq ?lc p$

proof (rule ccontr)

assume $\neg ?thesis$

hence ?lc (- p) > 0 $\text{degree } (- p) \neq 0$ **using** deg **by** auto

from poly-pinfy-ge[OF this, of - bnd + 1, simplified]

obtain n **where** $\bigwedge x. x \geq n \implies 1 - bnd \leq - \text{poly } p x$ **by** auto

from le[of max n c] this[of max n c] **show** False **by** auto

qed

qed
qed *auto*

end

definition *nneg-poly* :: 'a :: {*linordered-semidom, semiring-no-zero-divisors*} *poly*
 \Rightarrow *bool* **where**

nneg-poly *p* = $((\forall x. x \geq 0 \longrightarrow \text{poly } p \ x \geq 0) \wedge \text{lead-coeff } p \geq 0)$

lemma *nneg-poly-*nneg**: **assumes** *nneg-poly* *p*
and $x \geq 0$

shows *poly* *p* $x \geq 0$

using *assms* **unfolding** *nneg-poly-def* **by** *auto*

lemma *nneg-poly-lead-coeff*: **assumes** *nneg-poly* *p*

shows $p \neq 0 \implies \text{lead-coeff } p > 0$

using *assms* **unfolding** *nneg-poly-def*

by (*metis antisym-conv2 leading-coeff-neq-0*)

lemma *nneg-poly-add*: **assumes** *nneg-poly* *p* *nneg-poly* *q*

shows *nneg-poly* (*p* + *q*) *degree* (*p* + *q*) = *max* (*degree* *p*) (*degree* *q*)

proof –

{

fix *p* *q* :: 'a *poly*

assume *le*: *degree* *p* \leq *degree* *q* **and** *pq*: *nneg-poly* *p* *nneg-poly* *q*

have *nneg-poly* (*p* + *q*) \wedge *degree* (*p* + *q*) = *max* (*degree* *p*) (*degree* *q*)

proof (*cases* *degree* *p* = *degree* *q*)

case *True*

show *?thesis*

proof (*cases* $p = 0 \vee q = 0$)

case *True*

thus *?thesis* **using** *pq* **by** *auto*

next

case *False*

with *nneg-poly-lead-coeff*[*of* *p*] *nneg-poly-lead-coeff*[*of* *q*] *pq*

have *lc*: *lead-coeff* *p* $>$ 0 *lead-coeff* *q* $>$ 0 **by** *auto*

have *degree* (*p* + *q*) = *degree* *q* **using** *lc* *True*

by (*smt* (*verit*, *del-insts*) *Polynomial.coeff-add add-cancel-left-left add-le-same-cancel2*

le-degree leading-coeff-0-iff linorder-not-le order-less-le)

with *lc* *pq* *True* **show** *?thesis* **unfolding** *nneg-poly-def* **by** *auto*

qed

next

case *False*

with *le* **have** *lt*: *degree* *p* $<$ *degree* *q* **by** *auto*

hence *1*: *degree* (*p* + *q*) = *degree* *q*

by (*simp* *add: degree-add-eq-right*)

with *lt* **have** *2*: *lead-coeff* (*p* + *q*) = *lead-coeff* *q*

using *lead-coeff-add-le* **by** *blast*

```

    from 1 2 pq lt show ?thesis by (auto simp: nneg-poly-def)
  qed
} note main = this
have degree p ≤ degree q ∨ degree q ≤ degree p by linarith
with main[of p q] main[of q p] assms
have nneg-poly (p + q) ∧ degree (p + q) = max (degree p) (degree q)
  by (auto simp: ac-simps)
thus nneg-poly (p + q) degree (p + q) = max (degree p) (degree q)
  by auto
qed

```

```

lemma nneg-poly-mult: assumes nneg-poly p nneg-poly q
  shows nneg-poly (p * q)
  using assms unfolding nneg-poly-def poly-mult Polynomial.lead-coeff-mult
  by (intro allI conjI mult-nonneg-nonneg impI, auto)

```

```

lemma nneg-poly-const[simp]: nneg-poly [:c:] = (c ≥ 0)
  unfolding nneg-poly-def by (auto dest: spec[of - 0] simp add: coeff-const)

```

```

lemma nneg-poly-pCons[simp]: a ≥ 0 ∧ nneg-poly p ⇒ nneg-poly (pCons a p)
  unfolding nneg-poly-def by (auto simp: coeff-pCons split: nat.splits)

```

```

lemma nneg-poly-0[simp]: nneg-poly 0
  unfolding nneg-poly-def by auto

```

```

lemma nneg-poly-pcompose: assumes nneg-poly p nneg-poly q
  shows nneg-poly (pcompose p q)
proof (cases degree q > 0)
  case True
  show ?thesis unfolding nneg-poly-def poly-pcompose lead-coeff-comp[OF True]
    using assms unfolding nneg-poly-def by auto
  next
  case False
  hence degree q = 0 by auto
  from degree0-coeffs[OF this] obtain c where q: q = [:c:] by auto
  with assms[unfolded nneg-poly-def] have c: c ≥ 0 by auto
  have pq: p ∘p q = [: poly p c :] unfolding q
    by (metis (no-types, opaque-lifting) add.right-neutral coeff-pCons-0 mult-zero-left
      pcompose-0' pcompose-assoc poly-pCons poly-pcompose)
  show ?thesis using assms(1) unfolding nneg-poly-def pq using c by auto
qed

```

```

lemma nneg-poly-degree-add-1: assumes p: nneg-poly p and a: a1 > 0 a2 > 0
  shows degree (p * [:b, a1:] + [:c, a2:]) = 1 + degree p
proof (cases degree p = 0)
  case False
  thus ?thesis

```

apply (*subst degree-add-eq-left, insert p*)
subgoal using *a*
by (*metis One-nat-def degree-mult-eq-0 degree-pCons-eq-if irreducible_a-multD less-one linear-irreducible_a linorder-neqE-nat order-less-le pCons-eq-0-iff*)
subgoal using *a*
by (*metis Suc-eq-plus1 add commute add.right-neutral degree-mult-eq degree-pCons-eq-if not-pos-poly-0 pCons-eq-0-iff pos-poly-pCons*)
done
next
case *True*
then obtain *c* **where** *p: p = [:c:]* **and** *c: c ≥ 0* **using** *p degree0-coeffs[of p]* **by** *auto*
show *?thesis unfolding p using c a* **by** (*auto simp: add-nonneg-eq-0-iff*)
qed

lemma *nneg-poly-degree-add: assumes pq: nneg-poly (p :: 'a :: linordered-idom poly) nneg-poly q and a: a3 > 0 a2 > 0 a1 > 0 shows degree ([:a3:] * q * p + ([:a2:] * q + [:a1:] * p + [:a0:])) = degree p + degree q proof –*
{
fix *p q :: 'a poly and a2 a1 :: 'a*
assume *pq: nneg-poly p nneg-poly q*
and *dq: degree q ≠ 0*
and *a: a2 > 0 a1 > 0*
have *deg0: p ≠ 0 ⇒ degree ([:a3:] * q * p) = degree p + degree q* **using** *dq ‹a3 > 0› a*
by (*metis (no-types, lifting) add commute add-cancel-left-left degree-mult-eq degree-pCons-eq-if linorder-not-le nle-le pCons-eq-0-iff*)
have *degmax: degree ([:a2:] * q + [:a1:] * p + [:a0:]) ≤ max (degree q) (degree p)*
by (*simp add: degree-add-le*)
have *deg: degree ([:a3:] * q * p + ([:a2:] * q + [:a1:] * p + [:a0:])) = degree p + degree q*
proof (*cases degree p = 0*)
case *False*
have *id: degree ([:a3:] * q * p) = degree p + degree q* **by** (*rule deg0, insert False, auto*)
moreover have *max (degree q) (degree p) < degree p + degree q* **using** *False dq* **by** *auto*
ultimately show *?thesis* **by** (*subst degree-add-eq-left, insert degmax, auto*)
next
case *True*
with *pq* **obtain** *c* **where** *p: p = [:c:]* **and** *c: c ≥ 0* **using** *degree0-coeffs[of p]*
by *auto*
define *d* **where** *d = c * a3 + a2*
from *a ‹a3 > 0› c* **have** *d0: d ≠ 0*
by (*simp add: add-nonneg-eq-0-iff d-def*)

```

    have id: [:a3:] * q * [:c:] + ([:a2:] * q + [:a1:] * [:c:] + [:a0:])
      = [:c * a1 + a0:] + [:d:] * q
    by (simp add: smult-add-left d-def)
    show ?thesis unfolding p unfolding id
    by (subst degree-add-eq-right, insert d0 dq, auto)
  qed
} note main = this
show ?thesis
proof (cases degree q = 0)
  case False
  from main[OF pq False a(2,3)] show ?thesis .
next
  case dq: True
  show ?thesis
  proof (cases degree p = 0)
    case False
    from main[OF pq(2,1) False a(3,2)] show ?thesis by (simp add: alge-
bra-simps)
  next
    case dp: True
    from degree0-coeffs[OF dp] degree0-coeffs[OF dq] show ?thesis by auto
  qed
qed
qed

```

```

lemma poly-pinfty-gt-lc:
  fixes p :: 'a :: linordered-field poly
  assumes lead-coeff p > 0
  shows  $\exists n. \forall x \geq n. \text{poly } p \ x \geq \text{lead-coeff } p$ 
  using assms
proof (induct p)
  case 0
  then show ?case by auto
next
  case (pCons a p)
  from this(1) consider a ≠ 0 p = 0 | p ≠ 0 by auto
  then show ?case
  proof cases
    case 1
    then show ?thesis by auto
  next
    case 2
    with pCons obtain n1 where gte-lcoeff:  $\forall x \geq n1. \text{lead-coeff } p \leq \text{poly } p \ x$ 
    by auto
    from pCons(3) ⟨p ≠ 0⟩ have gt-0: lead-coeff p > 0 by auto
    define n where n = max n1 (1 + |a| / lead-coeff p)
    have lead-coeff (pCons a p) ≤ poly (pCons a p) x if n ≤ x for x
    proof -

```

```

from gte-lcoeff that have lead-coeff  $p \leq \text{poly } p \ x$ 
  by (auto simp: n-def)
with gt-0 have  $|a| / \text{lead-coeff } p \geq |a| / \text{poly } p \ x$  and  $\text{poly } p \ x > 0$ 
  by (auto intro: frac-le)
with  $\langle n \leq x \rangle$  [unfolded n-def] have  $x \geq 1 + |a| / \text{poly } p \ x$ 
  by auto
with  $\langle \text{lead-coeff } p \leq \text{poly } p \ x \rangle$   $\langle \text{poly } p \ x > 0 \rangle$   $\langle p \neq 0 \rangle$ 
show lead-coeff (pCons a p)  $\leq \text{poly } (\text{pCons a } p) \ x$ 
  by (auto simp: field-simps)
qed
then show ?thesis by blast
qed
qed

```

```

lemma poly-pinfy-ge:
  fixes  $p :: 'a :: \text{linordered-field}$  poly
  assumes lead-coeff  $p > 0$  degree  $p \neq 0$ 
  shows  $\exists n. \forall x \geq n. \text{poly } p \ x \geq b$ 
proof -
  let  $?p = p - [b - \text{lead-coeff } p : ]$ 
  have id: lead-coeff  $?p = \text{lead-coeff } p$  using assms(2)
    by (cases p, auto)
  with assms(1) have lead-coeff  $?p > 0$  by auto
  from poly-pinfy-gt-lc [OF this, unfolded id] obtain  $n$ 
    where  $\bigwedge x. x \geq n \implies 0 \leq \text{poly } p \ x - b$  by auto
  thus ?thesis by auto
qed

```

```

lemma nneg-polyI: fixes  $p :: 'a :: \text{linordered-field}$  poly
  assumes  $\bigwedge x. 0 \leq x \implies 0 \leq \text{poly } p \ x$ 
  shows nneg-poly  $p$ 
  unfolding nneg-poly-def
proof (intro allI conjI impI assms)

```

```

  {
    assume lc: lead-coeff  $p < 0$ 
    hence lc0: lead-coeff  $(- p) > 0$  by auto
    from lc assms[of 0] have degree  $p \neq 0$  using degree0-coeffs[of p]
      by (cases degree p = 0; auto)
    from poly-pinfy-ge [OF lc0, of 1] this obtain  $n$  where  $\bigwedge x. x \geq n \implies \text{poly } p$ 
     $x \leq - 1$ 
      by auto
    with assms have False
      by (meson neg-0-le-iff-le nle-le not-one-le-zero order-trans)
  }
  thus lead-coeff  $p \geq 0$  by force
qed

```

```

lemma poly-bounded: fixes x :: 'a:: linordered-idom
  assumes abs x ≤ b
  shows abs (poly p x) ≤ (∑ i ≤ degree p. abs (coeff p i) * b ^ i)
  unfolding poly-altdef
  apply (intro order.trans[OF sum-abs] sum-mono)
  apply (unfold abs-mult power-abs, intro mult-left-mono power-mono assms)
  by auto

lemma poly-degree-le-large-const:
  assumes pq: degree (p :: 'a :: linordered-field poly) ≥ degree q
  and p0: ∧ x. x ≥ 0 ⇒ poly p x ≥ 0
  shows ∃ H. ∀ h ≥ H. ∀ x ≥ 0. h * poly p x + h ≥ poly q x
  proof (cases degree p = 0)
  case True
  with pq p0[of 0] obtain c d where p: p = [:c:] and q: q = [:d:] and c: c ≥ 0
  using degree0-coeffs[of p] degree0-coeffs[of q] by auto
  show ?thesis unfolding p q using c
  apply (intro exI[of - max d 0], cases d ≤ 0)
  subgoal using order-trans by fastforce
  by (simp add: add commute add-increasing2)
next
  case False
  define lc where lc = lead-coeff p
  define dp where dp = degree p
  have dp1: dp ≥ 1 using False unfolding dp-def by auto
  from p0 have lc ≥ 0 unfolding lc-def using poly-pinfty-ge[of -p 1]
  by (metis (no-types, opaque-lifting) False degree-minus lead-coeff-minus linorder-not-le
  neg-le-0-iff-le nle-le not-one-le-zero order-le-less-trans poly-minus)
  with False have lc: lc > 0 by (cases lc = 0, auto simp: lc-def)
  define d where d = inverse lc
  define dlc where dlc = d * lc
  have dlc: dlc ≥ 1 using lc by (auto simp: field-simps d-def dlc-def)
  with lc have d: d > 0 unfolding dlc-def
  by (simp add: d-def)
  define h1 where h1 = d * (1 + abs (coeff q dp))
  define r where r = smult h1 p - q
  have coeff r dp = h1 * lc - coeff q dp unfolding r-def lc-def dp-def by simp
  also have ... = dlc * (1 + abs (coeff q dp)) - coeff q dp unfolding h1-def
  dlc-def by simp
  also have - ... ≤ - ((1 + abs (coeff q dp)) - coeff q dp)
  unfolding neg-le-iff-le using dlc
  by (intro diff-right-mono)
  (simp add: abs-add-one-gt-zero)
  also have ... ≤ - 1 by simp
  finally have coeff-r: coeff r dp > 0 by auto

  have dpr: dp = degree r
  proof -
  have le: dp ≤ degree r using coeff-r

```

```

    by (simp add: le-degree)
  have degree  $r \leq dp$  unfolding  $dp\text{-def}$   $r\text{-def}$  using  $assms(1)$ 
    by (simp add: degree-diff-le)
  with  $le$  show ?thesis by auto
qed
with  $coeff\text{-}r$  have  $lcr$ :  $lead\text{-}coeff\ r > 0$  by auto
from  $dpr\ dp1$  have  $degree\ r \neq 0$  by auto
from  $poly\text{-}pinfty\text{-}ge[OF\ lcr\ this, of\ 0]$ 
obtain  $n$  where  $n: \bigwedge x. x \geq n \implies 0 \leq poly\ r\ x$  by auto
define  $M$  where  $M = max\ n\ 0$ 
from  $poly\text{-}bounded[of\ -\ M\ r]$  obtain  $h2$  where  $h2: abs\ x \leq M \implies abs\ (poly\ r\ x) \leq h2$  for  $x$  by blast
have  $h20$ :  $h2 \geq 0$  using  $h2[of\ 0]$  unfolding  $M\text{-def}$  by auto
have  $h10$ :  $h1 > 0$  using  $d$  unfolding  $h1\text{-def}$  by auto
define  $H$  where  $H = max\ h1\ h2$ 
have  $H0$ :  $H \geq 0$  using  $h10$  unfolding  $H\text{-def}$  by auto
show ?thesis
proof (intro  $exI[of\ -\ H]$   $conjI$   $allI$   $impI$ )
  fix  $h\ x :: 'a$ 
  assume  $h$ :  $h \geq H$ 
  with  $H0$  have  $h0$ :  $h \geq 0$  by auto
  assume  $x0$ :  $x \geq 0$ 
  show  $poly\ q\ x \leq h * poly\ p\ x + h$ 
  proof (cases  $x \geq M$ )
    case  $x$ :  $True$ 
    have  $h$ :  $h \geq h1$  using  $h\ H\text{-def}$  by auto
    define  $h3$  where  $h3 = h - h1$ 
    have  $h$ :  $h = h1 + h3$  and  $h2$ :  $h3 \geq 0$  using  $h$  unfolding  $h3\text{-def}$  by auto
    have  $r$ :  $0 \leq poly\ r\ x$  and  $p$ :  $0 \leq poly\ p\ x$ 
      using  $x\ n[of\ x]$   $p0[of\ x]$  unfolding  $M\text{-def}$  by auto
    have  $h * poly\ p\ x = h1 * poly\ p\ x + h3 * poly\ p\ x$  unfolding  $h$  by (simp
  add:  $algebra\text{-}simps$ )
    also have  $- \dots \leq - (h1 * poly\ p\ x)$ 
      unfolding  $neg\text{-}le\text{-}iff\text{-}le$  using  $h2\ p$  by auto
    also have  $- \dots \leq - (poly\ q\ x)$ 
      unfolding  $neg\text{-}le\text{-}iff\text{-}le$  using  $r$  unfolding  $r\text{-def}$ 
      by  $simp$ 
    finally have  $h * poly\ p\ x \geq poly\ q\ x$  by  $simp$ 
  with  $h0$  show ?thesis by auto
  next
  case  $False$ 
  with  $x0$  have  $abs\ x \leq M$  by auto
  from  $h2[OF\ this]$  have  $poly\ r\ x \geq - h2$  by auto
  from  $this[unfolded\ r\text{-def}]$ 
  have  $poly\ q\ x \leq h1 * poly\ p\ x + h2$  by  $simp$ 
  also have  $- \dots \leq h * poly\ p\ x + h$ 
    by (intro  $add\text{-}mono\ mult\text{-}right\text{-}mono\ p0\ x0$ )
    (insert  $h$ , auto  $simp$ :  $H\text{-def}$ )
  finally show ?thesis .

```


qed
 qed
 qed

lemma *degree-monom-0[simp]: degree-monom 0 = 0*
unfolding *degree-monom-def* **by** *auto*

lemma *degree-monom-monomial[simp]: degree-monom (monomial n x) = n*
unfolding *degree-monom-def* **by** *auto*

lemma *keys-add: keys (m + n :: monom) = keys m ∪ keys n*
by (*rule keys-plus-ninv-comm-monoid-add*)

lemma *degree-monom-add[simp]: degree-monom (m + n) = degree-monom m + degree-monom n*

unfolding *degree-monom-def keys-add lookup-plus-fun*

proof (*transfer, goal-cases*)

case (*1 m n*)

have *id: {k. m k ≠ 0} ∪ {k. n k ≠ 0} =*
 $\{k. m k \neq 0\} \cap \{k. n k = 0\} \cup \{k. n k \neq 0\} \cap \{k. m k = 0\}$
 $\cup \{k. m k \neq 0\} \cap \{k. n k \neq 0\}$ **by** *auto*

have *id1: sum m {k. m k ≠ 0} = sum m ({k. m k ≠ 0} ∩ {k. n k = 0} ∪ {k. m k ≠ 0} ∩ {k. n k ≠ 0})*

by (*rule sum.cong, auto*)

have *id2: sum n {k. n k ≠ 0} = sum n ({k. n k ≠ 0} ∩ {k. m k = 0} ∪ {k. m k ≠ 0} ∩ {k. n k ≠ 0})*

by (*rule sum.cong, auto*)

show *?case unfolding id*

apply (*subst sum.union-disjoint*)

subgoal using 1 **by** *auto*

subgoal using 1 **by** *auto*

subgoal **by** *auto*

apply (*subst sum.union-disjoint*)

subgoal using 1 **by** *auto*

subgoal using 1 **by** *auto*

subgoal **by** *auto*

apply (*unfold id1*)

apply (*subst sum.union-disjoint*)

subgoal using 1 **by** *auto*

subgoal using 1 **by** *auto*

subgoal **by** *auto*

apply (*unfold id2*)

apply (*subst sum.union-disjoint*)

subgoal using 1 **by** *auto*

subgoal using 1 **by** *auto*

subgoal **by** *auto*

by (*simp add: sum.distrib*)

qed

lemma *degree-monom-of-set*: $\text{finite } xs \implies \text{degree-monom } (\text{monom-of-set } xs) = \text{card } xs$
unfolding *degree-monom-def*
by (*transfer, auto*)

lemma *keys-singletonE*: **assumes** $\text{keys } m = \{x\}$
shows $\exists c. m = \text{monomial } c \ x \wedge c = \text{degree-monom } m \wedge c \neq 0$
proof –
define c **where** $c = \text{degree-monom } m$
from *assms* **have** $mc: m = \text{monomial } c \ x$ **unfolding** *c-def*
by (*metis degree-monom-monomial except-keys group-cancel.rule0 plus-except*)
have $c \neq 0$ **using** *assms* **unfolding** *mc* **by** (*simp split: if-splits*)
from *mc c-def this* **show** *?thesis* **by** *blast*
qed

lemma *degree-monom-0-iff*: $\text{degree-monom } m = 0 \longleftrightarrow m = 0$
unfolding *degree-monom-def*
by *transfer auto*

lemma *degree-0-imp-Const*: **fixes** $p :: 'a :: \text{comm-ring-1}$ *mpoly*
assumes $d0: \text{total-degree } p = 0$
shows $\exists c. p = \text{Const } c$
proof –
{
fix m
assume $m \text{coeff } p \ m \neq 0$
from *degree-monom-le-total-degree[OF this, unfolded d0]*
have $m = 0$ **by** (*auto simp: degree-monom-0-iff*)
}
hence $\{m . m \text{coeff } p \ m \neq 0\} = \{\}$ \vee $\{m . m \text{coeff } p \ m \neq 0\} = \{0\}$ **by** *auto*
thus *?thesis*
proof
assume *id*: $\{m . m \text{coeff } p \ m \neq 0\} = \{\}$
have $p = \text{sum } (\lambda m. m \text{monom } m \ (m \text{coeff } p \ m)) \ \{m . m \text{coeff } p \ m \neq 0\}$
by (*rule mpoly-as-sum*)
also **have** $\dots = 0$ **unfolding** *id* **by** *simp*
also **have** $\dots = \text{Const } 0$ **by** *simp*
finally **show** *?thesis* **by** *blast*
next
assume *id*: $\{m . m \text{coeff } p \ m \neq 0\} = \{0\}$
have $p = \text{sum } (\lambda m. m \text{monom } m \ (m \text{coeff } p \ m)) \ \{m . m \text{coeff } p \ m \neq 0\}$
by (*rule mpoly-as-sum*)
also **have** $\dots = m \text{monom } 0 \ (m \text{coeff } p \ 0)$ **unfolding** *id* **by** *simp*
also **have** $\dots = \text{Const } (m \text{coeff } p \ 0)$
using *mpoly-monom-0-eq-Const* **by** *blast*
finally **show** *?thesis* **by** *blast*
qed
qed

```

lemma binary-degree-2-poly: fixes  $p :: 'a :: \{\text{ring-char-0, idom}\}$  mpoly
  assumes td: total-degree  $p \leq 2$ 
  and vars: vars  $p = \{x, y\}$ 
  and xy:  $x \neq y$ 
shows  $\exists a b c d e f.$ 
   $p = \text{Const } a + \text{Const } b * \text{Var } x + \text{Const } c * \text{Var } y +$ 
     $\text{Const } d * \text{Var } x * \text{Var } x + \text{Const } e * \text{Var } y * \text{Var } y + \text{Const } f * \text{Var } x * \text{Var}$ 
   $y$ 
proof -
  let  $?p = \text{mcoeff } p$ 
  let  $?x = \text{monomial } 1 x$ 
  let  $?y = \text{monomial } 1 y$ 
  let  $?a = ?p 0$ 
  let  $?b = ?p ?x$ 
  let  $?c = ?p ?y$ 
  let  $?d = ?p (\text{monomial } 2 x)$ 
  let  $?e = ?p (\text{monomial } 2 y)$ 
  let  $?f = ?p (\text{monom-of-set } \{x, y\})$ 
  define XY where  $XY = \{m :: \text{nat} \Rightarrow_0 \text{nat}. \text{keys } m \subseteq \{x, y\} \wedge \text{degree-monom}$ 
   $m \leq 2\}$ 
  let  $?xy = [0, ?x, ?y, \text{monomial } 2 x, \text{monomial } 2 y, \text{monom-of-set } \{x, y\}]$ 
  have eq:  $m = n \implies \text{keys } m = \text{keys } n$  for  $m n :: \text{monom}$  by auto
  have xy: distinct  $?xy$  using xy
    by (auto dest: eq)
  have XY:  $XY = \text{set } ?xy$ 
  proof
    show  $\text{set } ?xy \subseteq XY$  unfolding XY-def by (simp add: keys-add degree-monom-of-set
  card-insert-if)
    show  $XY \subseteq \text{set } ?xy$ 
    proof
      fix  $m$ 
      assume  $m \in XY$ 
      hence keys:  $\text{keys } m \subseteq \{x, y\}$  and deg: degree-monom  $m \leq 2$  unfolding
  XY-def by auto
      define km where  $km = \text{keys } m$ 
      from keys have  $\text{keys } m \in \{\{\}, \{x\}, \{y\}, \{x, y\}\}$  unfolding km-def [symmetric]
  by auto
      then consider (e)  $\text{keys } m = \{\}$  | (x)  $\text{keys } m = \{x\}$  | (y)  $\text{keys } m = \{y\}$  | (xy)
   $\text{keys } m = \{x, y\}$  by auto
      thus  $m \in \text{set } ?xy$ 
      proof cases
        case e
          thus ?thesis by auto
        next
          case x
          from keys-singletonE [OF this]
          obtain  $c$  where  $m = \text{monomial } c x$  and  $c = \text{degree-monom } m$   $c \neq 0$ 
  by auto
          from c deg have  $c \in \{1, 2\}$  by auto

```

```

    with m show ?thesis by auto
  next
    case y
    from keys-singletonE[OF this]
    obtain c where m: m = monomial c y and c: c = degree-monom m c ≠ 0
  by auto
    from c deg have c ∈ {1,2} by auto
    with m show ?thesis by auto
  next
    case xy
    have m = monom-of-set {x, y} using xy deg ⟨x ≠ y⟩
      unfolding degree-monom-def
    proof (transfer, goal-cases)
      case (1 m x y)
      have xy: m x ≠ 0 m y ≠ 0 using 1(2) by auto
      have sum m {k. m k ≠ 0} = m x + m y + sum m ({k. m k ≠ 0} -
{x,y})
        using xy 1(1,2,4) by auto
      with 1(3) xy have xy: m x = 1 m y = 1 and
        rest: sum m ({k. m k ≠ 0} - {x,y}) = 0 by auto
      from rest have rest: z ∉ {x,y} ⇒ m z = 0 for z using 1(2) by blast
      show ?case by (intro ext, insert xy rest, auto)
    qed
  thus ?thesis by auto
qed
qed
qed
have p = (∑ m. mmonom m (mcoeff p m))
  by (rule mpoly-as-sum-any)
also have ... = (∑ m∈{a. mmonom a (mcoeff p a) ≠ 0}. mmonom m (mcoeff
p m))
  unfolding Sum-any.expand-set by simp
also have ... = (∑ m∈{a. mmonom a (mcoeff p a) ≠ 0} ∩ XY. mmonom m
(mcoeff p m))
  apply (rule sum.mono-neutral-right; (intro ballI)?)
  subgoal by auto
  subgoal by auto
  subgoal for m using vars order.trans[OF degree-monom-le-total-degree[of p m]
td] unfolding XY-def
  by simp (smt (verit, best) DiffD2 MPoly-Type-monom-zero coeff-notin-vars
mem-Collect-eq)
  done
also have ... = (∑ m∈XY. mmonom m (mcoeff p m))
  apply (rule sum.mono-neutral-left)
  subgoal unfolding XY by auto
  subgoal by auto
  subgoal by auto
  done
also have ... = (∑ m ← ?xy. mmonom m (mcoeff p m))

```

unfolding XY **using** xy **by force**
also have $\dots = \text{Const } ?a + \text{Const } ?b * \text{Var } x + \text{Const } ?c * \text{Var } y +$
 $\text{Const } ?d * \text{Var } x * \text{Var } x + \text{Const } ?e * \text{Var } y * \text{Var } y + \text{Const } ?f * \text{Var } x *$
 $\text{Var } y$
apply (*intro mpoly-extI*)
unfolding *insertion-sum-list map-map o-def insertion-add insertion-mult in-*
sertion-Const insertion-Var
sum-list.Cons list.simps insertion-single insertion-monom-of-set mpoly-monom-0-eq-Const
using xy
by (*simp add: power2-eq-square*)
finally show $?thesis$ **by blast**
qed

lemma *bounded-negative-factor*: **assumes** $\bigwedge x. c \leq (x :: 'a :: \text{linordered-field}) \implies$
 $a * x \leq b$
shows $a \leq 0$
proof (*rule ccontr*)
assume $\neg ?thesis$
hence $a > 0$ **by auto**
hence $y \geq c \implies y \geq 0 \implies y \leq b$ **for** y **using** *assms[of inverse a * y]*
by (*metis (no-types, opaque-lifting) assms dual-order.trans linorder-not-le mult.commute*
mult-imp-less-div-pos nle-le)
from *this[of 1 + max 0 (max c b)]*
show False **by linarith**
qed

end

3 Definition of Monotone Algebras and Polynomial Interpretations

theory *Polynomial-Interpretation*
imports
Preliminaries-on-Polynomials-1
First-Order-Terms.Term
First-Order-Terms.Subterm-and-Context
begin
abbreviation $PVar \equiv MPoly\text{-Type}.Var$
abbreviation $TVar \equiv Term.Var$

type-synonym $(f, 'v)rule = (f, 'v)term \times (f, 'v)term$

We fix the domain to the set of nonnegative numbers

lemma *subterm-size[termination-simp]*: $x < \text{length } ts \implies \text{size } (ts ! x) < \text{Suc}$
 $(\text{size-list size } ts)$
by (*meson Suc-n-not-le-n less-eq-Suc-le not-less-eq nth-mem size-list-estimation*)

definition *assignment* :: (*var* \Rightarrow '*a* :: {ord,zero}) \Rightarrow bool **where**
assignment $\alpha = (\forall x. \alpha x \geq 0)$

lemma *assignmentD*: **assumes** *assignment* α
shows $\alpha x \geq 0$
using *assms* **unfolding** *assignment-def* **by** *auto*

definition *monotone-fun-wrt* :: ('*a* :: {zero,ord}) \Rightarrow '*a* \Rightarrow bool) \Rightarrow nat \Rightarrow ('*a* list \Rightarrow '*a*) \Rightarrow bool **where**
monotone-fun-wrt $gt\ n\ f = (\forall v' i\ vs. length\ vs = n \longrightarrow (\forall v \in set\ vs. v \geq 0) \longrightarrow i < n \longrightarrow gt\ v' (vs\ !\ i) \longrightarrow gt\ (f\ (vs\ [i := v'])\ (f\ vs))$

definition *valid-fun* :: nat \Rightarrow ('*a* list \Rightarrow '*a* :: {zero,ord}) \Rightarrow bool **where**
valid-fun $n\ f = (\forall vs. length\ vs = n \longrightarrow (\forall v \in set\ vs. v \geq 0) \longrightarrow f\ vs \geq 0)$

definition *monotone-poly-wrt* :: ('*a* :: {comm-semiring-1,zero,ord}) \Rightarrow '*a* \Rightarrow bool) \Rightarrow var set \Rightarrow '*a* mpoly \Rightarrow bool **where**
monotone-poly-wrt $gt\ V\ p = (\forall \alpha\ x\ v. assignment\ \alpha \longrightarrow x \in V \longrightarrow gt\ v\ (\alpha\ x) \longrightarrow gt\ (insertion\ (\alpha(x := v))\ p)\ (insertion\ \alpha\ p))$

definition *valid-poly* :: '*a* :: {ord,comm-semiring-1} mpoly \Rightarrow bool **where**
valid-poly $p = (\forall \alpha. assignment\ \alpha \longrightarrow insertion\ \alpha\ p \geq 0)$

locale *term-algebra* =
fixes *F* :: ('*f* \times nat) set
and *I* :: '*f* \Rightarrow ('*a* :: {ord,zero} list) \Rightarrow '*a*
and *gt* :: '*a* \Rightarrow '*a* \Rightarrow bool
begin

abbreviation *monotone-fun* **where** *monotone-fun* \equiv *monotone-fun-wrt* *gt*

definition *valid-monotone-fun* :: ('*f* \times nat) \Rightarrow bool **where**
valid-monotone-fun $fn = (\forall f\ n\ p. fn = (f,n) \longrightarrow p = I\ f \longrightarrow valid-fun\ n\ p \wedge monotone-fun\ n\ p)$

definition *valid-monotone-inter* **where** *valid-monotone-inter* = Ball *F* *valid-monotone-fun*

definition *orient-rule* :: ('*f*,var)rule \Rightarrow bool **where**
orient-rule $rule = (case\ rule\ of\ (l,r) \Rightarrow (\forall \alpha. assignment\ \alpha \longrightarrow gt\ (I\ [l]\ \alpha)\ (I\ [r]\ \alpha)))$
end

locale *omega-term-algebra* = *term-algebra* *F* *I* (*>*) :: int \Rightarrow int \Rightarrow bool **for** *F* **and** *I* :: '*f* \Rightarrow - +
assumes *vm-inter*: *valid-monotone-inter*

```

begin
definition termination-by-interpretation :: ('f,var) rule set ⇒ bool where
  termination-by-interpretation R = (∀ (l,r) ∈ R. orient-rule (l,r) ∧ funas-term l
  ∪ funas-term r ⊆ F)
end

locale poly-inter =
  fixes F :: ('f × nat) set
  and I :: 'f ⇒ 'a :: linordered-idom mpoly
  and gt :: 'a ⇒ 'a ⇒ bool (infix <> 50)
begin

definition I' where I' f vs = insertion (λ i. if i < length vs then vs ! i else 0) (I
f)
sublocale term-algebra F I' gt .

abbreviation monotone-poly where monotone-poly ≡ monotone-poly-wrt gt

abbreviation weakly-monotone-poly where weakly-monotone-poly ≡ monotone-poly-wrt
(≥)

definition gt-poly :: 'a mpoly ⇒ 'a mpoly ⇒ bool (infix <>p 50) where
  (p >p q) = (∀ α. assignment α → insertion α p > insertion α q)

definition valid-monotone-poly :: ('f × nat) ⇒ bool where
  valid-monotone-poly fn = (∀ f n p. fn = (f,n) → p = I f
  → valid-poly p ∧ monotone-poly {..definition valid-weakly-monotone-poly :: ('f × nat) ⇒ bool where
  valid-weakly-monotone-poly fn = (∀ f n p. fn = (f,n) → p = I f
  → valid-poly p ∧ weakly-monotone-poly {..definition valid-monotone-poly-inter where valid-monotone-poly-inter = Ball F
valid-monotone-poly
definition valid-weakly-monotone-inter where valid-weakly-monotone-inter = Ball
F valid-weakly-monotone-poly

fun eval :: ('f,var)term ⇒ 'a mpoly where
  eval (TVar x) = PVar x
  | eval (Fun f ts) = substitute (λ i. if i < length ts then eval (ts ! i) else 0) (I f)

lemma I'-is-insertion-eval: I' [[t]] α = insertion α (eval t)
proof (induct t)
  case (Var x)
  then show ?case by (simp add: insertion-Var)
next
  case (Fun f ts)
  then show ?case
  apply (simp add: insertion-substitute I'-def[of f])

```

```

    apply (intro arg-cong[of - - λ α. insertion α (I f)] ext)
    subgoal for i by (cases i < length ts, auto)
  done
qed

lemma orient-rule: orient-rule (l,r) = (eval l >_p eval r)
  unfolding orient-rule-def split I'-is-insertion-eval gt-poly-def ..

lemma vars-eval: vars (eval t) ⊆ vars-term t
proof (induct t)
  case (Fun f ts)
  define V where V = vars-term (Fun f ts)
  define σ where σ = (λi. if i < length ts then eval (ts ! i) else 0)
  {
    fix i
    have IH: vars (σ i) ⊆ V
    proof (cases i < length ts)
      case False
      thus ?thesis unfolding σ-def by auto
    next
      case True
      hence ts ! i ∈ set ts by auto
      with Fun(1)[OF this] have vars (eval (ts ! i)) ⊆ V by (auto simp: V-def)
      thus ?thesis unfolding σ-def using True by auto
    qed
  }
  note σ-vars = this
  define p where p = (I f)
  show ?case unfolding eval.simps σ-def[symmetric] V-def[symmetric] p-def[symmetric]
using σ-vars
  vars-substitute[of σ] by auto
qed auto

lemma monotone-imp-weakly-monotone: assumes valid: valid-monotone-poly p
  and gt: ∧ x y. (x > y) = (x > y)
  shows valid-weakly-monotone-poly p
  unfolding valid-weakly-monotone-poly-def
proof (intro allI impI, clarify, intro conjI)
  fix f n
  assume p = (f,n)
  note * = valid[unfolded valid-monotone-poly-def, rule-format, OF this refl]
  from * show valid-poly (I f) by auto
  from * show vars (I f) ⊆ {..

```



```

qed
qed

lemma valid-imp-insertion-eval-pos: assumes valid: valid-monotone-poly-inter
  and funas-term  $t \subseteq F$ 
  and assignment  $\alpha$ 
shows insertion  $\alpha$  (eval  $t$ )  $\geq 0$ 
  using assms(2-3)
proof (induct  $t$  arbitrary: alpha)
  case (Var  $x$ )
  thus ?case by (auto simp: assignment-def insertion-Var)
next
  case (Fun  $f$   $ts$ )
  let  $?n = \text{length } ts$ 
  let  $?f = (f, ?n)$ 
  let  $?p = I f$ 
  from Fun have  $?f \in F$  by auto
  from valid[unfolded valid-monotone-poly-inter-def, rule-format, OF this, unfolded
valid-monotone-poly-def]
  have valid: valid-poly ?p and vars ?p = {.. $?n$ } by auto
  from valid[unfolded valid-poly-def]
  have ins: assignment  $\alpha \implies 0 \leq \text{insertion } \alpha (I f)$  for  $\alpha$  by auto
  {
    fix  $i$ 
    assume  $i < ?n$ 
    hence  $ts ! i \in \text{set } ts$  by auto
    with Fun(1)[OF this - Fun(3)] Fun(2) have  $0 \leq \text{insertion } \alpha (\text{eval } (ts ! i))$  by
auto
  }
  note IH = this
  show ?case
    apply (simp add: insertion-substitute)
    apply (intro ins, unfold assignment-def, intro allI)
    subgoal for  $i$  using IH[of i] by auto
  done
qed

end

locale delta-poly-inter = poly-inter  $F$   $I$  ( $\lambda x y. x \geq y + \delta$ ) for  $F :: ('f \times \text{nat}) \text{ set}$ 
and  $I$  and
   $\delta :: 'a :: \{\text{floor-ceiling, linordered-field}\} +$ 
  assumes valid: valid-monotone-poly-inter
  and  $\delta 0: \delta > 0$ 
begin
definition termination-by-delta-interpretation  $:: ('f, \text{var}) \text{ rule set} \implies \text{bool}$  where
  termination-by-delta-interpretation  $R = (\forall (l, r) \in R. \text{orient-rule } (l, r) \wedge \text{funas-term } l \cup \text{funas-term } r \subseteq F)$ 
end

```

```

locale int-poly-inter = poly-inter F I ( $>$ ) :: int  $\Rightarrow$  int  $\Rightarrow$  bool for F :: (f  $\times$  nat)
set and I +
  assumes valid: valid-monotone-poly-inter
begin

sublocale omega-term-algebra F I'
proof (unfold-locales, unfold valid-monotone-inter-def, intro ballI)
  fix fn
  assume fn  $\in$  F
  from valid[unfolded valid-monotone-poly-inter-def, rule-format, OF this]
  have valid: valid-monotone-poly fn .
  show valid-monotone-fun fn unfolding valid-monotone-fun-def
  proof (intro allI impI conjI)
    fix f n p
    assume fn: fn = (f,n) and p: p = I' f
    from valid[unfolded valid-monotone-poly-def, rule-format, OF fn refl]
    have valid: valid-poly (I f) and mono: monotone-poly  $\{..<n\}$  (I f) by auto

    show valid-fun n p unfolding valid-fun-def
    proof (intro allI impI)
      fix vs
      assume length vs = n and vs: Ball (set vs) ( $(\leq)$  (0 :: int))
      show  $0 \leq p$  vs unfolding p I'-def
      by (rule valid[unfolded valid-poly-def, rule-format], insert vs, auto simp:
assignment-def)
    qed

    show monotone-fun n p unfolding monotone-fun-wrt-def
    proof (intro allI impI)
      fix v' i vs
      assume *: length vs = n Ball (set vs) ( $(\leq)$  (0 :: int)) i < n vs ! i < v'
      show p vs < p (vs[i := v']) unfolding p I'-def
      by (rule ord-less-eq-trans[OF mono[unfolded monotone-poly-wrt-def, rule-format,
of - i v']
insertion-irrelevant-vars], insert *, auto simp: assignment-def)
    qed
  qed
qed

```

```

definition termination-by-poly-interpretation :: (f,var) rule set  $\Rightarrow$  bool where
  termination-by-poly-interpretation = termination-by-interpretation
end

locale wm-int-poly-inter = poly-inter F I ( $>$ ) :: int  $\Rightarrow$  int  $\Rightarrow$  bool for F :: (f  $\times$ 
nat) set and I +
  assumes valid: valid-weakly-monotone-inter
begin

```

definition *oriented-by-interpretation* :: ('f,var) rule set ⇒ bool **where**
oriented-by-interpretation R = (∀ (l,r) ∈ R. orient-rule (l,r) ∧ funas-term l ∪ funas-term r ⊆ F)
end

locale *linear-poly-inter* = *poly-inter* F I gt **for** F I gt +
assumes *linear*: ∧ f n. (f,n) ∈ F ⇒ total-degree (I f) ≤ 1

locale *linear-int-poly-inter* = *int-poly-inter* F I + *linear-poly-inter* F I (>)
for F :: ('f × nat) set **and** I

locale *linear-wm-int-poly-inter* = *wm-int-poly-inter* F I + *linear-poly-inter* F I (>)
for F :: ('f × nat) set **and** I

definition *termination-by-linear-int-poly-interpretation* :: ('f × nat) set ⇒ ('f,var) rule set ⇒ bool **where**
termination-by-linear-int-poly-interpretation F R = (∃ I. *linear-int-poly-inter* F I ∧ I ∧ *int-poly-inter.termination-by-poly-interpretation* F I R)

definition *omega-termination* :: ('f × nat) set ⇒ ('f,var) rule set ⇒ bool **where**
omega-termination F R = (∃ I. *omega-term-algebra* F I ∧ *omega-term-algebra.termination-by-interpretation* F I R)

definition *termination-by-int-poly-interpretation* :: ('f × nat) set ⇒ ('f,var) rule set ⇒ bool **where**
termination-by-int-poly-interpretation F R = (∃ I. *int-poly-inter* F I ∧ *int-poly-inter.termination-by-poly-interpretation* F I R)

definition *termination-by-delta-poly-interpretation* :: 'a :: {floor-ceiling,linordered-field} itself ⇒ ('f × nat) set ⇒ ('f,var) rule set ⇒ bool **where**
termination-by-delta-poly-interpretation TYPE('a) F R = (∃ I δ. *delta-poly-inter* F I (δ :: 'a) ∧ *delta-poly-inter.termination-by-delta-interpretation* F I δ R)

definition *orientation-by-linear-wm-int-poly-interpretation* :: ('f × nat) set ⇒ ('f,var) rule set ⇒ bool **where**
orientation-by-linear-wm-int-poly-interpretation F R = (∃ I. *linear-wm-int-poly-inter* F I ∧ *wm-int-poly-inter.oriented-by-interpretation* F I R)

end

4 Hilbert's 10th Problem to Linear Inequality

theory *Hilbert10-to-Inequality*
imports
Preliminaries-on-Polynomials-1

begin

definition *hilbert10-problem* :: *int mpoly* \Rightarrow *bool* **where**
hilbert10-problem *p* = ($\exists \alpha$. *insertion* α *p* = 0)

A polynomial is positive, if every coefficient is positive. Since the @{*const coeff*}-function of '*a mpoly* maps a coefficient to every monomial, this means that positiveness is expressed as $\text{coeff } p \ m \neq 0 \longrightarrow 0 < \text{coeff } p \ m$ for monomials *m*. However, this condition is equivalent to just demand $0 \leq \text{coeff } p \ m$ for all *m*.

This is the reason why *positive polynomials* are defined in the same way as one would define *non-negative polynomials*.

definition *positive-poly* :: '*a* :: *linordered-idom mpoly* \Rightarrow *bool* **where**
positive-poly *p* = ($\forall m$. *coeff* *p* *m* \geq 0)

definition *positive-interpr* :: (*var* \Rightarrow '*a* :: *linordered-idom*) \Rightarrow *bool* **where**
positive-interpr α = ($\forall x$. α *x* > 0)

definition *positive-poly-problem* :: '*a* :: *linordered-idom mpoly* \Rightarrow '*a mpoly* \Rightarrow *bool* **where**
positive-poly *p* \Longrightarrow *positive-poly* *q* \Longrightarrow *positive-poly-problem* *p* *q* =
($\exists \alpha$. *positive-interpr* α \wedge *insertion* α *p* \geq *insertion* α *q*)

datatype *flag* = *Positive* | *Negative* | *Zero*

fun *flag-of* :: '*a* :: {*ord,zero*} \Rightarrow *flag* **where**
flag-of *x* = (if *x* < 0 then *Negative* else if *x* > 0 then *Positive* else *Zero*)

definition *subst-flag* :: *var set* \Rightarrow (*var* \Rightarrow *flag*) \Rightarrow *var* \Rightarrow '*a* :: *comm-ring-1 mpoly* **where**
subst-flag *V* *flag* *x* = (if *x* \in *V* then (case *flag* *x* of
 Positive \Rightarrow *Var* *x*
 | *Negative* \Rightarrow - *Var* *x*
 | *Zero* \Rightarrow 0)
else 0)

definition *assignment-flag* :: *var set* \Rightarrow (*var* \Rightarrow *flag*) \Rightarrow (*var* \Rightarrow '*a* :: *comm-ring-1*) \Rightarrow (*var* \Rightarrow '*a*) **where**
assignment-flag *V* *flag* α *x* = (if *x* \in *V* then (case *flag* *x* of
 Positive \Rightarrow α *x*
 | *Negative* \Rightarrow - α *x*
 | *Zero* \Rightarrow 1)
else 1)

definition *correct-flags* :: *var set* \Rightarrow (*var* \Rightarrow *flag*) \Rightarrow (*var* \Rightarrow '*a* :: *ordered-comm-ring*) \Rightarrow *bool* **where**
correct-flags *V* *flag* α = ($\forall x \in V$. *flag* *x* = *flag-of* (α *x*))

lemma *correct-flag-substitutions*: **fixes** $p :: 'a :: \text{linordered-idom mpoly}$
assumes $\text{vars } p \subseteq V$
and $\text{beta}: \beta = \text{assignment-flag } V \text{ flag } \alpha$
and $\text{sigma}: \sigma = \text{subst-flag } V \text{ flag}$
and $q: q = \text{substitute } \sigma p$
and $\text{corr}: \text{correct-flags } V \text{ flag } \alpha$
shows $\text{insertion } \beta q = \text{insertion } \alpha p \text{ positive-interpr } \beta$
proof –
show $\text{insertion } \beta q = \text{insertion } \alpha p$ **unfolding** q **insertion-substitute**
proof (*rule insertion-irrelevant-vars*)
fix x
assume $x \in \text{vars } p$
with *assms* **have** $x: x \in V$ **by** *auto*
with *corr* **have** $\text{flag}: \text{flag } x = \text{flag-of } (\alpha x)$ **unfolding** *correct-flags-def* **by** *auto*

show $\text{insertion } \beta (\sigma x) = \alpha x$
unfolding *beta sigma assignment-flag-def subst-flag-def* **using** x *flag*
by (*cases flag x, auto split: if-splits simp: insertion-Var insertion-uminus*)
qed
show *positive-interpr* β **using** *corr*
unfolding *positive-interpr-def beta assignment-flag-def correct-flags-def*
by *auto*
qed

definition *hilbert-encode1* :: $\text{int mpoly} \Rightarrow \text{int mpoly list}$ **where**
 $\text{hilbert-encode1 } r = (\text{let } r2 = r^{\wedge}2;$
 $V = \text{vars-list } r2;$
 $\text{flag-lists} = \text{product-lists } (\text{map } (\lambda x. \text{map } (\lambda f. (x,f)) [\text{Positive}, \text{Negative}, \text{Zero}])$
 $V);$
 $\text{subst} = (\lambda \text{fl}. \text{subst-flag } (\text{set } V) (\lambda x. \text{case map-of fl } x \text{ of Some } f \Rightarrow f \mid \text{None}$
 $\Rightarrow \text{Zero}))$
 $\text{in map } (\lambda \text{fl}. \text{substitute } (\text{subst fl}) r2) \text{flag-lists}$)

lemma *hilbert-encode1*:
 $\text{hilbert10-problem } r \longleftrightarrow (\exists p \in \text{set } (\text{hilbert-encode1 } r). \exists \alpha. \text{positive-interpr } \alpha \wedge$
 $\text{insertion } \alpha p \leq 0)$
proof
define $r2$ **where** $r2 = r^{\wedge}2$
define V **where** $V = \text{vars-list } r2$
define *flag-list* **where** *flag-list* = $\text{product-lists } (\text{map } (\lambda x. \text{map } (\lambda f. (x,f))$
 $[\text{Positive}, \text{Negative}, \text{Zero}]) V)$
define *subst* **where** $\text{subst} = (\lambda \text{fl}. \text{subst-flag } (\text{set } V) (\lambda x. \text{case map-of fl } x \text{ of}$
 $\text{Some } f \Rightarrow f \mid \text{None} \Rightarrow \text{Zero}) :: \text{var} \Rightarrow \text{int mpoly})$
have *hilb-enc*: $\text{hilbert-encode1 } r = \text{map } (\lambda \text{fl}. \text{substitute } (\text{subst fl}) r2) \text{flag-list}$
unfolding *subst-def flag-list-def V-def r2-def Let-def hilbert-encode1-def ..*
have $\text{hilbert10-problem } r \longleftrightarrow (\exists \alpha. \text{insertion } \alpha r = 0)$ **unfolding** *hilbert10-problem-def*
by *auto*
also **have** $\dots \longleftrightarrow (\exists \alpha. (\text{insertion } \alpha r)^{\wedge}2 \leq 0)$
by (*intro ex-cong1, auto*)

also have ... $\longleftrightarrow (\exists \alpha. \text{insertion } \alpha \ r2 \leq 0)$
by (*intro ex-cong1*, *auto simp: power2-eq-square insertion-mult r2-def*)
finally have *hilb*: *hilbert10-problem* *r* = $(\exists \alpha. \text{insertion } \alpha \ r2 \leq 0)$ (**is** *?h1* = *?h2*) .
let *?r1* = $(\exists p \in \text{set } (\text{hilbert-encode1 } r). \exists \alpha. \text{positive-interpr } \alpha \wedge \text{insertion } \alpha \ p \leq 0)$
{
 assume *?r1*
 from *this*[*unfolded hilb-enc*]
 show *hilbert10-problem* *r* **unfolding** *hilb* **by** (*auto simp add: insertion-substitute*)
}
{
 assume *?h1*
 with *hilb* **obtain** α **where** *solution*: *insertion* $\alpha \ r2 \leq 0$ **by** *auto*
 define *fl* **where** *fl* = *map* $(\lambda x. (x, \text{flag-of } (\alpha \ x))) \ V$
 define *flag* **where** *flag* = $(\lambda x. \text{case map-of } fl \ x \text{ of } \text{Some } f \Rightarrow f \mid \text{None} \Rightarrow \text{Zero})$

 have *vars*: *vars* $r2 \subseteq \text{set } V$ **unfolding** *V-def* **by** *simp*
 have *fl*: *fl* $\in \text{set flag-list}$ **unfolding** *flag-list-def product-lists-set fl-def*
 apply (*simp add: list-all2-map2 list-all2-map1, intro list-all2-refl*)
 by *auto*
 have *mem*: *substitute* (*subst-flag* (*set* V) *flag*) $r2 \in \text{set } (\text{hilbert-encode1 } r)$
 unfolding *hilb-enc subst-def flag-def* **using** *fl* **by** *auto*
 have *corr*: *correct-flags* (*set* V) *flag* α **unfolding** *correct-flags-def flag-def fl-def*
 by (*auto split: option.splits dest!: map-of-SomeD simp: map-of-eq-None-iff image-comp*)
 show *?r1* **using** *solution correct-flag-substitutions[OF vars refl refl refl corr]*
 by (*intro bexI[OF - mem], auto*)
}
qed

lemma *pos-neg-split*: *mpoly-coeff-filter* $(\lambda x. (x :: 'a :: \text{linordered-idom}) > 0)$ *p* + *mpoly-coeff-filter* $(\lambda x. x < 0)$ *p* = *p* (**is** *?l* + *?r* = *p*)

proof –
{
 fix *m*
 let *?c* = *coeff* *p* *m*
 have *coeff* (*?l* + *?r*) *m* = *coeff* *?l* *m* + *coeff* *?r* *m* **by** (*simp add: coeff-add*)
 also have ... = *coeff* *p* *m* **unfolding** *mpoly-coeff-filter*
 by (*cases ?c < 0; cases ?c > 0; cases ?c = 0, auto*)
 finally have *coeff* (*?l* + *?r*) *m* = *coeff* *p* *m* .
}
thus *?thesis* **using** *coeff-eq* **by** *blast*
qed

definition *hilbert-encode2* :: *int mpoly* \Rightarrow *int mpoly* \times *int mpoly* **where**
hilbert-encode2 *p* =
 ($-$ *mpoly-coeff-filter* $(\lambda x. x < 0)$ *p*, *mpoly-coeff-filter* $(\lambda x. x > 0)$ *p*)

lemma *hilbert-encode2*: **assumes** *hilbert-encode2* $p = (r, s)$
shows *positive-poly r positive-poly s insertion α $p \leq 0 \iff insertion \alpha r \geq insertion \alpha s$*
proof –
from *assms*[*unfolded hilbert-encode2-def, simplified*]
have $s = mpoly-coeff-filter (\lambda x. x > 0) p$
and $r = - mpoly-coeff-filter (\lambda x. x < 0) p$ (**is** $- = - ?q$) **by** *auto*
have $p = s + ?q$ **unfolding** s **using** *pos-neg-split*[*of p*] **by** *simp*
also have $\dots = s - r$ **unfolding** $s r$ **by** *simp*
finally have $insertion \alpha p \leq 0 \iff insertion \alpha (s - r) \leq 0$ **by** *simp*
also have $insertion \alpha (s - r) = insertion \alpha s - insertion \alpha r$
by (*metis add-uminus-conv-diff insertion-add insertion-uminus*)
finally show $insertion \alpha p \leq 0 \iff insertion \alpha r \geq insertion \alpha s$ **by** *auto*
show *positive-poly s unfolding positive-poly-def s using mpoly-coeff-filter*[*of* ($\lambda x. x > 0$) p]
by (*auto simp: when-def*)
show *positive-poly r unfolding positive-poly-def r coeff-uminus using mpoly-coeff-filter*[*of* ($\lambda x. x < 0$) p]
by (*auto simp: when-def*)
qed

definition *hilbert-encode* :: $int\ mpoly \Rightarrow (int\ mpoly \times int\ mpoly)list$ **where**
hilbert-encode = *map hilbert-encode2 o hilbert-encode1*

Lemma 2.2 in paper

lemma *hilbert-encode-positive: hilbert10-problem p*
 $\iff (\exists (r, s) \in set (hilbert-encode\ p). positive-poly-problem\ r\ s)$
proof –
have *hilbert10-problem p* $\iff (\exists p' \in set (hilbert-encode1\ p). \exists \alpha. positive-interpr\ \alpha \wedge insertion\ \alpha\ p' \leq 0)$
using *hilbert-encode1*[*of p*] **by** *blast*
also have $\dots \iff (\exists (r, s) \in set (hilbert-encode\ p). positive-poly-problem\ r\ s)$ (**is** $?l = ?r$)
proof
assume $?l$
then obtain $p' \alpha$ **where** $mem: p' \in set (hilbert-encode1\ p)$ **and** $sol: positive-interpr\ \alpha\ insertion\ \alpha\ p' \leq 0$ **by** *blast*
obtain $r\ s$ **where** $?2: hilbert-encode2\ p' = (r, s)$ **by** *force*
from $mem\ 2$ **have** $mem: (r, s) \in set (hilbert-encode\ p)$ **unfolding** *hilbert-encode-def o-def* **by** *force*
from *hilbert-encode2*[*OF 2*] sol **have** *positive-poly-problem r s using positive-poly-problem-def*[*of r s*] **by** *force*
with mem **show** $?r$ **by** *blast*
next
assume $?r$
then obtain $r\ s$ **where** $mem: (r, s) \in set (hilbert-encode\ p)$ **and** $sol: positive-poly-problem\ r\ s$ **by** *auto*
from mem [*unfolded hilbert-encode-def o-def*] **obtain** p' **where**
 $mem: p' \in set (hilbert-encode1\ p)$

```

    and hilbert-encode2 p' = (r,s) by force
  from hilbert-encode2[OF this(2)] sol positive-poly-problem-def[of r s]
  have ( $\exists \alpha. \text{positive-interpr } \alpha \wedge \text{insertion } \alpha \ p' \leq 0$ ) by auto
  with mem hilbert-encode1[of p] show ?l by auto
qed
finally show ?thesis .
qed
end

```

5 Undecidability of Linear Polynomial Termination

```

theory Linear-Poly-Termination-Undecidable
  imports
    Hilbert10-to-Inequality
    Polynomial-Interpretation
begin

```

Definition 3.1

```

locale poly-input =
  fixes p q :: int mpoly
  assumes pq: positive-poly p positive-poly q
begin

```

```

datatype symbol = a-sym | z-sym | o-sym | f-sym | v-sym var | q-sym | h-sym |
g-sym

```

```

abbreviation a-t where a-t t1 t2  $\equiv$  Fun a-sym [t1, t2]
abbreviation z-t where z-t  $\equiv$  Fun z-sym []
abbreviation o-t where o-t  $\equiv$  Fun o-sym []
abbreviation f-t where f-t t1 t2 t3 t4  $\equiv$  Fun f-sym [t1,t2,t3,t4]
abbreviation v-t where v-t i t  $\equiv$  Fun (v-sym i) [t]

```

```

definition encode-num :: var  $\Rightarrow$  int  $\Rightarrow$  (symbol,var)term where
  encode-num x n = (( $\lambda t. a-t$  (Var x) t)  $\widetilde{\sim}$  (nat n)) z-t

```

```

definition encode-monom :: var  $\Rightarrow$  monom  $\Rightarrow$  int  $\Rightarrow$  (symbol,var)term where
  encode-monom x m c = rec-list (encode-num x c) ( $\lambda (i,e) . (\lambda t. v-t i t) \widetilde{\sim} e$ )
  (var-list m)

```

```

definition encode-poly :: var  $\Rightarrow$  int mpoly  $\Rightarrow$  (symbol,var)term where
  encode-poly x r = rec-list z-t ( $\lambda (m,c) . t. a-t$  (encode-monom x m c) t) (monom-list
  r)

```

```

lemma vars-encode-num: vars-term (encode-num x n)  $\subseteq$  {x}
proof -
  define m where m = nat n

```



```

show ?thesis
  unfolding encode-num-def m-def[symmetric]
  by (induct m, auto)
qed

```

```

lemma vars-encode-monom: vars-term (encode-monom x m c)  $\subseteq$  {x}
proof –
  define xes where xes = var-list m
  show ?thesis unfolding encode-monom-def xes-def[symmetric]
  proof (induct xes)
    case Nil
    thus ?case using vars-encode-num by auto
  next
    case (Cons ye xes)
    obtain y e where ye: ye = (y,e) by force
    have [simp]: vars-term ((v-t y  $\widehat{\sim}$  e) t) = vars-term t for t :: (symbol,var)term
    by (induct e arbitrary: t, auto)
    from Cons show ?case unfolding ye by auto
  qed
qed

```

```

lemma vars-encode-poly: vars-term (encode-poly x r)  $\subseteq$  {x}
proof –
  define mcs where mcs = monom-list r
  show ?thesis unfolding encode-poly-def mcs-def[symmetric]
  proof (induct mcs)
    case (Cons mc mcs)
    obtain m c where mc: mc = (m,c) by force
    from Cons show ?case unfolding mc using vars-encode-monom[of x m c] by
    auto
  qed auto
qed

```

```

definition V where V = vars p  $\cup$  vars q

```

```

definition y1 :: var where y1 = 0
definition y2 :: var where y2 = 1
definition y3 :: var where y3 = 2

```

```

lemma y-vars: y1  $\neq$  y2 y2  $\neq$  y3 y1  $\neq$  y3
  unfolding y1-def y2-def y3-def by auto

```

Definition 3.3

```

definition lhs-R = f-t (Var y1) (Var y2) (a-t (encode-poly y3 p) (Var y3)) o-t
definition rhs-R = f-t (a-t (Var y1) z-t) (a-t z-t (Var y2)) (a-t (encode-poly y3
q) (Var y3)) z-t

```

```

definition F where F = {(a-sym, 2), (z-sym, 0)}  $\cup$  ( $\lambda$  i. (v-sym i, 1 :: nat)) ‘
V

```

definition *F-R* **where** $F-R = \{(f\text{-sym}, 4), (o\text{-sym}, 0)\} \cup F$

definition *R* **where** $R = \{(lhs-R, rhs-R)\}$

definition *V-list* **where** $V\text{-list} = \text{sorted-list-of-set } V$

definition *contexts* $:: (symbol \times nat \times nat)$ list

where *contexts* = [
 (*a-sym*, 2, 0),
 (*a-sym*, 2, 1),
 (*f-sym*, 4, 0),
 (*f-sym*, 4, 1),
 (*f-sym*, 4, 2),
 (*f-sym*, 4, 3)] @
 map ($\lambda i. (v\text{-sym } i, 1, 0)$) *V-list*

replace *t* by $f(z, \dots, z, t, z, \dots, z)$

definition *z-context* $:: symbol \times nat \times nat \Rightarrow (symbol, var)term \Rightarrow (symbol, var)term$ **where**

z-context *c t* = (case *c* of (*f, n, i*) \Rightarrow Fun *f* (replicate *i* *z-t* @ [*t*] @ replicate (*n* - *i* - 1) *z-t*))

definition *z-contexts* **where**

z-contexts *cs* = foldr *z-context* *cs*

definition *all-symbol-pos-ctxt* $:: (symbol, var)term \Rightarrow (symbol, var)term$ **where**

all-symbol-pos-ctxt = *z-contexts contexts*

definition *lhs-R'* = *all-symbol-pos-ctxt lhs-R*

definition *rhs-R'* = *all-symbol-pos-ctxt rhs-R*

definition *R'* **where** $R' = \{(lhs-R', rhs-R')\}$

lemma *funas-encode-num*: *funas-term* (encode-num *x n*) $\subseteq F$

proof -

define *m* **where** $m = nat\ n$

show ?thesis

unfolding encode-num-def *m-def*[symmetric]

by (induct *m*, auto simp: *F-def*)

qed

lemma *funas-encode-monom*: **assumes** $keys\ m \subseteq V$

shows *funas-term* (encode-monom *x m c*) $\subseteq F$

proof -

define *xes* **where** $xes = var\text{-list } m$

show ?thesis **using** *var-list-keys*[of - - *m*] **unfolding** encode-monom-def *xes-def*[symmetric]

proof (induct *xes*)

case Nil

thus ?case **using** *funas-encode-num* **by** auto

next

case (*Cons ye xes*)
obtain $y e$ **where** $ye = (y, e)$ **by** *force*
have $sub: funas-term ((v-t y \overset{\sim}{\sim} e) t) \subseteq insert (v-sym y, 1) (funas-term t)$ **for**
 $t :: (symbol, var)term$
by (*induct e arbitrary: t, auto*)
from *Cons(2)[unfolded ye] assms* **have** $y \in V$ **by** *auto*
hence $inF: (v-sym y, 1) \in F$ **unfolding** *F-def* **by** *auto*
from *Cons sub inF* **show** *?case unfolding ye by fastforce*
qed
qed

lemma *funas-encode-poly*: **assumes** $vars r \subseteq V$ **shows** $funas-term (encode-poly x r) \subseteq F$

proof –
define mcs **where** $mcs = monom-list r$
show *?thesis using monom-list-keys[of - - r] unfolding encode-poly-def mcs-def[symmetric]*
proof (*induct mcs*)
case (*Cons mc mcs*)
obtain $m c$ **where** $mc: mc = (m, c)$ **by** *force*
have $a: (a-sym, 2) \in F$ **unfolding** *F-def* **by** *auto*
from *Cons(2)[unfolded mc] assms* **have** $keys m \subseteq V$ **by** *auto*
from *funas-encode-monom[OF this, of x c] Cons(1)[OF Cons(2)] a*
show *?case unfolding mc by (force simp: numeral-eq-Suc)*
qed (*auto simp: F-def*)
qed

lemma *funas-encode-poly-p*: $funas-term (encode-poly x p) \subseteq F$
by (*rule funas-encode-poly, auto simp: V-def*)

lemma *funas-encode-poly-q*: $funas-term (encode-poly x q) \subseteq F$
by (*rule funas-encode-poly, auto simp: V-def*)

lemma *lhs-R-F*: $funas-term lhs-R \subseteq F-R$

proof –
from *funas-encode-poly-p*
show $funas-term lhs-R \subseteq F-R$ **unfolding** *lhs-R-def* **by** (*auto simp: F-R-def F-def*)
qed

lemma *rhs-R-F*: $funas-term rhs-R \subseteq F-R$

proof –
from *funas-encode-poly-q*
show $funas-term rhs-R \subseteq F-R$ **unfolding** *rhs-R-def* **by** (*auto simp: F-R-def F-def*)
qed

lemma *finite-V*: $finite V$ **unfolding** *V-def* **using** *vars-finite* **by** *auto*

lemma *V-list*: *set V-list = V unfolding V-list-def using finite-V by auto*

lemma *contexts*: **assumes** $(f,n,i) \in \text{set contexts}$
shows $(f,n) \in F\text{-}R \ i < n$
using *assms* **unfolding** *contexts-def F-R-def F-def* **by** (*auto simp: V-list*)

lemma *z-contexts-append*: $z\text{-contexts } (cs @ ds) t = z\text{-contexts } cs (z\text{-contexts } ds t)$
unfolding *z-contexts-def* **by** (*induct cs, auto*)

lemma *z-context*: **assumes** $(f,n) \in F\text{-}R \ i < n$ **and** *funas-term* $t \subseteq F\text{-}R$
shows *funas-term* $(z\text{-context } (f,n,i) t) \subseteq F\text{-}R$
proof –
have $z: (z\text{-sym}, 0) \in F\text{-}R$ **unfolding** *F-R-def F-def* **by** *auto*
thus *?thesis* **unfolding** *z-context-def split* **using** *assms* **by** *auto*
qed

lemma *funas-all-symbol-pos-ctxt*: **assumes** *funas-term* $t \subseteq F\text{-}R$
shows *funas-term* $(\text{all-symbol-pos-ctxt } t) \subseteq F\text{-}R$
proof –
define *cs* **where** $cs = \text{contexts}$
have *sub*: $\text{set } cs \subseteq \text{set contexts}$ **unfolding** *cs-def* **by** *auto*
have *id*: $\text{all-symbol-pos-ctxt } t = \text{foldr } z\text{-context } cs \ t$ **unfolding** *cs-def all-symbol-pos-ctxt-def*
z-contexts-def
by (*auto simp: id-def*)
show *?thesis* **unfolding** *id* **using** *sub assms(1)*
proof (*induct cs arbitrary: t*)
case (*Cons c cs t*)
obtain *f n i* **where** $c = (f,n,i)$ **by** (*cases c, auto*)
from *c Cons* **have** $(f,n,i) \in \text{set contexts}$ **by** *auto*
from $z\text{-context}[OF \text{contexts}[OF \text{this}], \text{folded } c]$ *Cons*
show *?case* **by** *auto*
qed *auto*
qed

lemma *lhs-R'-F*: *funas-term* $\text{lhs-}R' \subseteq F\text{-}R$
unfolding *lhs-R'-def* **by** (*rule funas-all-symbol-pos-ctxt[OF lhs-R-F]*)

lemma *rhs-R'-F*: *funas-term* $\text{rhs-}R' \subseteq F\text{-}R$
unfolding *rhs-R'-def* **by** (*rule funas-all-symbol-pos-ctxt[OF rhs-R-F]*)
end

lemma *insertion-positive-poly*: **assumes** $\bigwedge x. \alpha \ x \geq (0 :: 'a :: \text{linordered-idom})$
and *positive-poly* *p*
shows *insertion* $\alpha \ p \geq 0$
by (*rule insertion-nonneg, insert assms[unfolded positive-poly-def], auto*)

locale *solvable-poly-problem* = *poly-input* *p q* **for** *p q* +
assumes *sol*: *positive-poly-problem* *p q*
begin

definition α **where** $\alpha = (\text{SOME } \alpha. \text{positive-interpr } \alpha \wedge \text{insertion } \alpha \ q \leq \text{insertion } \alpha \ p)$

lemma α : $\text{positive-interpr } \alpha \ \text{insertion } \alpha \ q \leq \text{insertion } \alpha \ p$
using $\text{someI-ex}[OF \ \text{sol}[\text{unfolded positive-poly-problem-def}[OF \ pq]], \text{folded } \alpha\text{-def}]$
by auto

lemma $\alpha 1$: $\alpha \ x > 0$ **using** α **unfolding** $\text{positive-interpr-def}$ **by** auto

context

fixes $I :: \text{symbol} \Rightarrow \text{int mpoly}$
assumes $\text{inter}: I \ a\text{-sym} = \text{PVar } 0 + \text{PVar } 1$
 $I \ z\text{-sym} = 0$
 $I \ o\text{-sym} = 1$
 $I \ (v\text{-sym } i) = \text{Const } (\alpha \ i) * \text{PVar } 0$

begin

lemma inter-encode-num : **assumes** $c \geq 0$
shows $\text{poly-inter.eval } I \ (\text{encode-num } x \ c) = \text{Const } c * \text{PVar } x$
proof $-$
from assms **obtain** n **where** $\text{cn}: c = \text{int } n$ **by** $(\text{metis nonneg-eq-int})$
hence $\text{nac}: \text{nat } c = n$ **by** auto
show $?thesis$ **unfolding** $\text{encode-num-def nac unfolding cn}$
by $(\text{induct } n, \text{auto simp: inter poly-inter.eval.simps Const-0 Const-1 algebra-simps Const-add})$
qed

lemma inter-v-pow-e : $\text{poly-inter.eval } I \ ((v\text{-t } x \ \widehat{\wedge} \ e) \ t) = \text{Const } ((\alpha \ x) \widehat{\wedge} \ e) * \text{poly-inter.eval } I \ t$
by $(\text{induct } e, \text{auto simp: Const-1 Const-mult inter poly-inter.eval.simps})$

lemma $\text{inter-encode-monom}$: **assumes** $c: c \geq 0$
shows $\text{poly-inter.eval } I \ (\text{encode-monom } y \ m \ c) = \text{Const } (\text{insertion } \alpha \ (\text{monom } m \ c)) * \text{PVar } y$
proof $-$
define xes **where** $\text{xes} = \text{var-list } m$
from $\text{var-list}[of \ m \ c]$
have $\text{monom}: \text{monom } m \ c = \text{Const } c * (\prod (x, e) \leftarrow \text{xes} . \text{PVar } x \ \widehat{\wedge} \ e)$ **unfolding** xes-def .
show $?thesis$ **unfolding** $\text{encode-monom-def monom xes-def[symmetric]}$
proof $(\text{induct } \text{xes})$
case Nil
show $?case$ **by** $(\text{simp add: inter-encode-num}[OF \ c] \text{insertion-Const})$
next
case $(\text{Cons } \text{xe} \ \text{xes})$
obtain $x \ e$ **where** $\text{xe}: \text{xe} = (x, e)$ **by** force
show $?case$ **by** $(\text{simp add: xe inter-v-pow-e Cons Const-power})$

```

      insertion-Const insertion-mult insertion-power insertion-Var Const-mult)
    qed
  qed

lemma inter-foldr-v-t:
  poly-inter.eval I (foldr v-t xs t) = Const (prod-list (map  $\alpha$  xs)) * poly-inter.eval
  I t
  by (induct xs arbitrary: t, auto simp: Const-1 inter poly-inter.eval.simps Const-mult)

lemma inter-encode-poly-generic: assumes positive-poly r
  shows poly-inter.eval I (encode-poly x r) = Const (insertion  $\alpha$  r) * PVar x
  proof -
    define mcs where mcs = monom-list r
    from monom-list[of r] have r: r = ( $\sum$  (m, c) $\leftarrow$  mcs. monom m c) unfolding
    mcs-def by auto
    have mcs: (m,c)  $\in$  set mcs  $\implies$  c  $\geq$  0 for m c
      using monom-list-coeff assms unfolding mcs-def positive-poly-def by auto
    note [simp] = inter poly-inter.eval.simps
    show ?thesis unfolding encode-poly-def mcs-def[symmetric] unfolding r inser-
    tion-sum-list map-map o-def
      using mcs
    proof (induct mcs)
      case (Cons mc mcs)
      obtain m c where mc: mc = (m,c) by force
      from Cons(2) mc have c: c  $\geq$  0 by auto
      note monom = inter-encode-monom[OF this, of x m]
      show ?case
        by (simp add: mc monom algebra-simps, subst Cons(1), insert Cons(2), auto
        simp: Const-add algebra-simps)
    qed simp
  qed

lemma valid-monotone-inter-F: assumes positive-interpr  $\alpha$ 
  and inF: fn  $\in$  F
  shows poly-inter.valid-monotone-poly I ( $>$ ) fn
  proof -
    obtain f n where fn: fn = (f,n) by force
    with inF have f: (f,n)  $\in$  F by auto
    show ?thesis unfolding poly-inter.valid-monotone-poly-def fn
    proof (intro allI impI, clarify, intro conjI)
      let ?valid = valid-poly
      let ?mono = poly-inter.monotone-poly ( $>$ )
      have [simp]: vars ((PVar 0 :: int mpoly) + PVar (Suc 0) + PVar 2 + PVar
      3) = {0,1,2,3}
      unfolding vars-def apply (transfer, simp add: Var0-def image-comp) by
      code-simp
      have [simp]: vars ((PVar 0 :: int mpoly) + PVar (Suc 0)) = {0,1}
      unfolding vars-def apply (transfer, simp add: Var0-def image-comp) by

```

```

code-simp
  note [simp] = inter poly-inter.eval.simps
  {
    fix i
    assume i: i ∈ V and f = v-sym i and n: n = 1
    hence I: I f = Const (α i) * PVar 0 by simp
    from assms[unfolding positive-interpr-def] have alpha: α i > 0 by auto
    have valid: ?valid (I f)
      unfolding I valid-poly-def using alpha
      by (auto simp: insertion-mult insertion-Const insertion-Var assignment-def
intro!: mult-nonneg-nonneg)
    have mono: ?mono {..<n} (I f)
      unfolding I unfolding n monotone-poly-wrt-def using alpha
      by (auto simp: insertion-Const insertion-mult insertion-Var)
    have vars (I f) ⊆ {..<n} unfolding I unfolding n
      by (rule order.trans[OF vars-mult], auto)
    moreover have 0 ∈ vars (I f)
      unfolding I unfolding n
    proof (rule ccontr)
      let ?p = Const (α i) * PVar 0
      assume not: 0 ∉ vars ?p
      define β :: var ⇒ int where β x = 0 for x
      have insertion β ?p = insertion (β(0 := 1)) ?p
        by (rule insertion-irrelevant-vars, insert not, auto)
      thus False using alpha by (simp add: β-def insertion-mult insertion-Const
insertion-Var)
    qed
    ultimately have vars (I f) = {..<n} unfolding n by auto
    note this valid mono
  } note v-sym = this
from f v-sym show vars (I f) = {..<n} unfolding F-def by auto
from f v-sym show ?valid (I f) unfolding F-def
  by (auto simp: valid-poly-def insertion-add assignment-def insertion-Var)
have x4: x < 4 ⇒ x = 0 ∨ x = Suc 0 ∨ x = 2 ∨ x = 3 for x by linarith
have x2: x < 2 ⇒ x = 0 ∨ x = Suc 0 for x by linarith
from f v-sym show ?mono {..<n} (I f) unfolding F-R-def F-def
  by (auto simp: monotone-poly-wrt-def insertion-add insertion-Var assign-
ment-def
      dest: x4 x2)
  qed
qed
end

fun I-R :: symbol ⇒ int mpolynomial where
  I-R f-sym = PVar 0 + PVar 1 + PVar 2 + PVar 3
| I-R a-sym = PVar 0 + PVar 1
| I-R z-sym = 0
| I-R o-sym = 1

```

| $I\text{-}R (v\text{-}sym\ i) = Const (\alpha\ i) * PVar\ 0$

interpretation *inter-R*: *poly-inter F-R I-R (>)* .

lemma *inter-R-encode-poly*: **assumes** *positive-poly r*
shows $inter\text{-}R.eval (encode\text{-}poly\ x\ r) = Const (insertion\ \alpha\ r) * PVar\ x$
by (*rule inter-encode-poly-generic[OF - - - assms]*, *auto*)

lemma *valid-monotone-inter-R*: *inter-R.valid-monotone-poly-inter unfolding inter-R.valid-monotone-poly-inter-def*

proof (*intro ballI*)

fix *fn*

assume $f: fn \in F\text{-}R$

show *inter-R.valid-monotone-poly fn*

proof (*cases fn \in F*)

case *True*

show *inter-R.valid-monotone-poly fn*

by (*rule valid-monotone-inter-F[OF - - - \alpha(1) True]*, *auto*)

next

case *False*

with *f* **have** $f: fn \in F\text{-}R - F$ **by** *auto*

have [*simp*]: $vars ((PVar\ 0 :: int\ mpoly) + PVar (Suc\ 0) + PVar\ 2 + PVar\ 3) = \{0,1,2,3\}$

unfolding *vars-def apply (transfer, simp add: Var₀-def image-comp)* **by** *code-simp*

show *?thesis unfolding inter-R.valid-monotone-poly-def using f*

proof (*intro ballI impI allI, clarify, intro conjI*)

fix *f n*

assume $f: (f,n) \in F\text{-}R (f,n) \notin F$

from *f* **show** $vars (I\text{-}R\ f) = \{..<n\}$ **unfolding** *F-R-def* **by** *auto*

from *f* **show** *valid-poly (I-R f) unfolding F-R-def*

by (*auto simp: valid-poly-def insertion-add assignment-def insertion-Var*)

have $x_4: x < 4 \implies x = 0 \vee x = Suc\ 0 \vee x = 2 \vee x = 3$ **for** *x* **by** *linarith*

from *f* **show** *inter-R.monotone-poly \{..<n\} (I-R f) unfolding F-R-def*

by (*auto simp: monotone-poly-wrt-def insertion-add insertion-Var assignment-def dest: x₄*)

qed

qed

qed

sublocale *inter-R*: *linear-int-poly-inter F-R I-R*

proof

show *inter-R.valid-monotone-poly-inter* **by** (*rule valid-monotone-inter-R*)

fix *f n*

assume $(f,n) \in F\text{-}R$

thus *total-degree (I-R f) \le 1* **by** (*cases f, auto simp: F-R-def F-def intro!: total-degree-add total-degree-Const-mult*)

qed

lemma *orient-R-main*: **assumes** *assignment* β
shows *insertion* β (*inter-R.eval lhs-R*) > *insertion* β (*inter-R.eval rhs-R*)
proof –
have *lhs-R*: *inter-R.eval lhs-R* = *PVar y1* + *PVar y2* + *Const* (*insertion* α *p* +
1) * *PVar y3* + 1
unfolding *lhs-R-def* **by** (*simp add: inter-R-encode-poly*[*OF pq*(1)] *algebra-simps*
Const-add Const-1)
have *rhs-R*: *inter-R.eval rhs-R* = *PVar y1* + *PVar y2* + *Const* (*insertion* α *q*
+ 1) * *PVar y3*
unfolding *rhs-R-def* **by** (*simp add: inter-R-encode-poly*[*OF pq*(2)] *algebra-simps*
Const-add Const-1)
show ?*thesis*
unfolding *lhs-R rhs-R*
apply (*simp add: insertion-add insertion-mult insertion-Var insertion-Const*)
apply (*intro mult-right-mono*)
subgoal using α (2) **by** *simp*
subgoal using *assms* **unfolding** *assignment-def* **by** *auto*
done
qed

The easy direction of Theorem 3.4

lemma *orient-R*: *inter-R.termination-by-poly-interpretation* *R*
unfolding *inter-R.termination-by-poly-interpretation-def* *inter-R.termination-by-interpretation-def*
R-def *inter-R.orient-rule*
proof (*clarify, intro conjI*)
show *inter-R.gt-poly* (*inter-R.eval lhs-R*) (*inter-R.eval rhs-R*)
unfolding *inter-R.gt-poly-def*
by (*intro allI impI orient-R-main*)
qed (*insert lhs-R-F rhs-R-F, auto*)

lemma *solution-imp-linear-termination-R*: *termination-by-linear-int-poly-interpretation*
F-R R
unfolding *termination-by-linear-int-poly-interpretation-def*
by (*intro exI, rule conjI*[*OF - orient-R*], *unfold-locales*)
end

context *poly-input*
begin

lemma *inter-z-context*:
assumes *i*: $i < n$ **and** *I*: $I f = \text{Const } c0 + (\text{sum-list } (\text{map } (\lambda j. \text{Const } (c j) * \text{PVar } j) [0..<n]))$
and *Ize*: $I z\text{-sym} = \text{Const } d0$
shows $\exists d. \forall t. \text{poly-inter.eval } I (z\text{-context } (f, n, i) t) = \text{Const } d + \text{Const } (c i) * \text{poly-inter.eval } I t$
proof –
define *d* **where** $d = c0 + (\sum x \leftarrow [0..<i]. c x * d0) + (\sum x \leftarrow [\text{Suc } i..<n]. c x * d0)$

```

show ?thesis
proof (intro exI[of - d] allI)
  fix t :: (symbol, nat) term
  define list where list = replicate i (Fun z-sym []) @ [t] @ replicate (n - i -
1) (Fun z-sym [])
  have len: length list = n
  using i unfolding list-def by auto
  have z[simp]: poly-inter.eval I (Fun z-sym []) = Const d0 unfolding poly-inter.eval.simps
using Ize by auto
  let ?xs1 = [0 ..< i]
  let ?xs2 = [Suc i ..< n]
  define ev where ev = (λ x. Const (c x) * poly-inter.eval I (list ! x))
  have poly-inter.eval I (z-context (f,n,i) t) = Const c0 +
  (∑ x←[0..<n]. ev x)
  unfolding z-context-def split list-def[symmetric]
  unfolding poly-inter.eval.simps len I ev-def
  unfolding substitute-add substitute-Const substitute-sum-list o-def substi-
tute-mult substitute-Var
  apply (rule arg-cong[of - - λ xs. (+) - (sum-list xs)])
  by (rule map-cong[OF refl], auto)
  also have [0 ..< n] = ?xs1 @ i # ?xs2 using i
  by (metis less-imp-add-positive upt-add-eq-append upt-rec zero-le)
  also have sum-list (map ev ...) = sum-list (map ev ?xs1) + sum-list (map ev
?xs2) + ev i by simp
  also have map ev ?xs1 = map (λ x. (Const (c x * d0))) ?xs1
  unfolding o-def by (intro map-cong, auto simp: ev-def list-def nth-append
Const-mult)
  also have sum-list ... = Const (sum-list (map (λ x. c x * d0) ?xs1)) unfolding
Const-sum-list o-def ..
  also have map ev ?xs2 = map (λ x. (Const (c x * d0))) ?xs2
  unfolding o-def by (intro map-cong, auto simp: ev-def list-def nth-append
Const-mult)
  also have sum-list ... = Const (sum-list (map (λ x. c x * d0) ?xs2)) unfolding
Const-sum-list o-def ..
  also have ev i = Const (c i) * poly-inter.eval I t unfolding ev-def list-def by
(auto simp: nth-append)
  finally show poly-inter.eval I (z-context (f, n, i) t) = Const d + Const (c i)
* poly-inter.eval I t
  unfolding add.assoc[symmetric] Const-add[symmetric] d-def by blast
qed
qed

```

lemma inter-z-contexts:

```

assumes cs: ∧ f n i. (f,n,i) ∈ set cs ⇒ i < n ∧ If = Const (c0 f) + (sum-list
(map (λ j. Const (c f j) * PVar j) [0..<n]))
and Ize: I z-sym = Const d0
shows ∃ d. ∀ t. poly-inter.eval I (z-contexts cs t) = Const d + Const (prod-list
(map (λ (f,n,i). c f i) cs)) * poly-inter.eval I t
proof -

```

```

define  $c'$  where  $c' = (\lambda (f, n :: \text{nat}, i). c f i)$ 
have  $c'$ :  $c f i = c' (f, n, i)$  for  $f i n$  unfolding  $c'$ -def split ..
{
  fix  $fni$ 
  assume  $mem$ :  $fni \in \text{set } cs$ 
  obtain  $f n i$  where  $fni$ :  $fni = (f, n, i)$  by (cases  $fni$ , auto)
  from  $cs$ [OF  $mem$ [unfolded  $fni$ ]]
  have  $i$ :  $i < n$  and  $I f = \text{Const } (c0 f) + (\sum_{j \leftarrow [0..<n]}. \text{Const } (c f j) * PVar$ 
j) by auto
  note  $inter\text{-}z\text{-context}$ [OF  $this$   $Ize$ , unfolded  $c'$ [of - -  $n$ ], folded  $fni$ ]
} note  $z\text{-pre}\text{-}ctxt = this$ 
define  $p$  where  $p fni d t = (fni \in \text{set } cs \longrightarrow \text{poly}\text{-}inter.\text{eval } I (z\text{-context } fni t)$ 
=  $\text{Const } d + \text{Const } (c' fni) * \text{poly}\text{-}inter.\text{eval } I t)$ 
  for  $fni d t$ 
  from  $z\text{-pre}\text{-}ctxt$ 
  have  $\forall fni. \exists d. \forall t. p fni d t$  by (auto simp:  $p\text{-def}$ )
  from  $choice$ [OF  $this$ ] obtain  $d'$  where  $\bigwedge fni t. p fni (d' fni) t$  by auto
  hence  $z\text{-ctxt}$ :  $\bigwedge fni t. fni \in \text{set } cs \implies \text{poly}\text{-}inter.\text{eval } I (z\text{-context } fni t) = \text{Const}$ 
( $d' fni$ ) +  $\text{Const } (c' fni) * \text{poly}\text{-}inter.\text{eval } I t$ 
  unfolding  $p\text{-def}$  by auto
  define  $d$  where  $d = \text{foldr } (\lambda fni c. d' fni + c' fni * c) cs 0$ 
  show  $?thesis$ 
  proof (intro  $exI$ [of -  $d$ ]  $allI$ )
    fix  $t :: (\text{symbol}, \text{var})\text{term}$ 
    show  $\text{poly}\text{-}inter.\text{eval } I (z\text{-contexts } cs t) = \text{Const } d + \text{Const } (\prod (f, n, i) \leftarrow cs. c$ 
 $f i) * \text{poly}\text{-}inter.\text{eval } I t$ 
    unfolding  $d\text{-def } z\text{-contexts}\text{-def}$  using  $z\text{-ctxt}$ 
    proof (induct  $cs$ )
      case  $Nil$ 
      show  $?case$  by (simp add:  $\text{Const}\text{-}0 \text{Const}\text{-}1$ )
    next
      case ( $Cons fni cs$ )
      from  $Cons(2)$ [of  $fni$ ]
      have  $z\text{-ctxt}$ :  $\text{poly}\text{-}inter.\text{eval } I (z\text{-context } fni t) = \text{Const } (d' fni) + \text{Const } (c'$ 
 $fni) * \text{poly}\text{-}inter.\text{eval } I t$  for  $t$  by auto
      from  $Cons(1)$ [OF  $Cons(2)$ ]
      have  $IH$ :  $\text{poly}\text{-}inter.\text{eval } I (\text{foldr } z\text{-context } cs t) =$ 
 $\text{Const } (\text{foldr } (\lambda fni c. d' fni + c' fni * c) cs 0) + \text{Const } (\prod (f, n, y) \leftarrow cs. c$ 
 $f y) * \text{poly}\text{-}inter.\text{eval } I t$ 
      by auto
      have [ $simp$ ]: (case  $fni$  of ( $f, n, xa$ )  $\implies c f xa$ ) =  $c' fni$  unfolding  $c'$ -def ..
      show  $?case$ 
      by (simp add:  $z\text{-ctxt } IH$  algebra-simps  $\text{Const}\text{-}mult$ )
        (simp add:  $\text{Const}\text{-}add$ [symmetric]  $\text{Const}\text{-}mult$ [symmetric])
    qed
  qed
qed

```

lemma $inter\text{-}all\text{-}symbol\text{-}pos\text{-}ctxt\text{-}generic$:

```

assumes  $f: I f\text{-sym} = \text{Const } fc + \text{Const } f0 * PVar\ 0 + \text{Const } f1 * PVar\ 1 +$ 
 $\text{Const } f2 * PVar\ 2 + \text{Const } f3 * PVar\ 3$ 
and  $a: I a\text{-sym} = \text{Const } ac + \text{Const } a0 * PVar\ 0 + \text{Const } a1 * PVar\ 1$ 
and  $v: \bigwedge i. i \in V \implies I (v\text{-sym } i) = \text{Const } (vc\ i) + \text{Const } (v0\ i) * PVar\ 0$ 
and  $I z\text{-sym} = \text{Const } zc$ 
shows  $\exists d. \forall t. \text{poly-inter.eval } I (\text{all-symbol-pos-ctxt } t) = \text{Const } d + \text{Const}$ 
 $(\text{prod-list } ([a0, a1, f0, f1, f2, f3] @ \text{map } v0\ V\text{-list}))$ 
 $* \text{poly-inter.eval } I\ t$ 
proof -
  define  $c$  where  $c = (\lambda f\ i. \text{case } f\ \text{of}$ 
     $a\text{-sym} \implies \text{if } i = 0 \text{ then } a0 \text{ else } a1$ 
     $| v\text{-sym } x \implies v0\ x$ 
     $| f\text{-sym} \implies \text{if } i = 0 \text{ then } f0 \text{ else if } i = \text{Suc } 0 \text{ then } f1 \text{ else if } i = 2 \text{ then } f2 \text{ else } f3)$ 
  define  $c0$  where  $c0 = (\lambda f. \text{case } f\ \text{of } a\text{-sym} \implies ac \ | f\text{-sym} \implies fc \ | v\text{-sym } x \implies vc$ 
 $x)$ 
  have  $id: [a0, a1, f0, f1, f2, f3] @ \text{map } v0\ V\text{-list} = \text{map } (\lambda (f, n, i). c\ f\ i)\ \text{contexts}$ 

  unfolding  $\text{contexts-def map-append}$ 
  by  $(\text{auto simp: } c\text{-def})$ 
  have  $\text{lists: } [0..<2] = [0, \text{Suc } 0] \ [0..<4] = [0, \text{Suc } 0, 2, 3]$  by  $\text{code-simp+}$ 
  show  $?thesis$  unfolding  $id\ \text{all-symbol-pos-ctxt-def}$ 
  proof  $(\text{rule } \text{inter-z-contexts}[of\ -\ -\ c0\ c\ zc])$ 
    show  $I\ z\text{-sym} = \text{Const } zc$  by  $\text{fact}$ 
    fix  $f\ n\ i$ 
    assume  $(f, n, i) \in \text{set } \text{contexts}$ 
    thus  $i < n \wedge I\ f = \text{Const } (c0\ f) + (\sum j \leftarrow [0..<n]. \text{Const } (c\ f\ j) * PVar\ j)$ 
    unfolding  $\text{contexts-def } c0\text{-def } c\text{-def}$  by  $(\text{auto simp: } f\ a\ v\ V\text{-list } \text{lists})$ 
  qed
qed
end

context  $\text{solvable-poly-problem}$ 
begin

lemma  $\text{inter-all-symbol-pos-ctxt}$ :
   $\exists d\ e. e \geq 1 \wedge (\forall t. \text{inter-R.eval } (\text{all-symbol-pos-ctxt } t) = \text{Const } d + \text{Const } e *$ 
 $\text{inter-R.eval } t)$ 
proof -
  from  $\text{inter-all-symbol-pos-ctxt-generic}[of\ I\text{-R } 0\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ \alpha\ 0, \text{unfolded}$ 
 $\text{Const-0 } \text{Const-1}]$ 
  obtain  $d$  where  $\text{inter: } \bigwedge t. \text{inter-R.eval } (\text{all-symbol-pos-ctxt } t) = \text{Const } d +$ 
 $\text{Const } (\text{prod-list } (\text{map } \alpha\ V\text{-list})) * \text{inter-R.eval } t$ 
  by  $\text{auto}$ 
  show  $?thesis$ 
  proof  $(\text{rule } \text{exI}[of\ -\ d], \text{rule } \text{exI}[of\ -\ \text{prod-list } (\text{map } \alpha\ V\text{-list})], \text{intro } \text{conjI } \text{allI}$ 
 $\text{inter})$ 
  define  $vs$  where  $vs = V\text{-list}$ 
  show  $1 \leq \text{prod-list } (\text{map } \alpha\ V\text{-list})$  unfolding  $vs\text{-def}[symmetric]$ 
  proof  $(\text{induct } vs)$ 

```

```

    case (Cons v vs)
  from  $\alpha(1)[\text{unfolded positive-interpr-def, rule-format, of } v]$  have  $1 \leq \alpha v$  by
  auto
  with Cons show ?case by simp (smt (verit, ccfv-threshold) mult-pos-pos)
  qed auto
  qed
  qed

```

The easy direction of Theorem 3.4 for R'

```

lemma orient-R': inter-R.termination-by-poly-interpretation R'
  unfolding inter-R.termination-by-interpretation-def inter-R.termination-by-poly-interpretation-def
  R'-def inter-R.orient-rule
proof (clarify, intro conjI)
  from inter-all-symbol-pos-ctxt obtain d e where
    e:  $e \geq 1$  and
    ctxt:  $\bigwedge t. \text{inter-R.eval (all-symbol-pos-ctxt } t) = \text{Const } d + \text{Const } e * \text{inter-R.eval } t$ 
  by auto
  let ?ctxt =  $\lambda f. \text{Const } d + \text{Const } e * f$ 
  show inter-R.gt-poly (inter-R.eval lhs-R') (inter-R.eval rhs-R')
    unfolding inter-R.gt-poly-def
  proof (intro allI impI)
    fix  $\beta :: \text{var} \Rightarrow \text{int}$ 
    assume ass: assignment  $\beta$ 
    have insertion  $\beta$  (inter-R.eval lhs-R') > insertion  $\beta$  (inter-R.eval rhs-R')
       $\longleftrightarrow$  insertion  $\beta$  (inter-R.eval lhs-R) > insertion  $\beta$  (inter-R.eval rhs-R)
    unfolding lhs-R'-def rhs-R'-def ctxt using e
    by (simp add: insertion-add insertion-mult insertion-Var insertion-Const)
    also have ... using orient-R-main[OF ass] .
    finally show insertion  $\beta$  (inter-R.eval rhs-R') < insertion  $\beta$  (inter-R.eval
  lhs-R') .
  qed
qed (insert lhs-R'-F rhs-R'-F, auto)

```

```

lemma solution-imp-linear-termination-R': termination-by-linear-int-poly-interpretation
  F-R R'
  unfolding termination-by-linear-int-poly-interpretation-def
  by (intro exI, rule conjI[OF - orient-R'], unfold-locales)
end

```

Now for the other direction of Theorem 3.4

```

lemma monotone-linear-poly-to-coeffs: fixes  $p :: \text{int}$  mpoly
  assumes linear: total-degree  $p \leq 1$ 
  and poly: valid-poly  $p$ 
  and mono: poly-inter.monotone-poly ( $>$ )  $\{..<n\}$   $p$ 
  and vars: vars  $p = \{..<n\}$ 
shows  $\exists c a. p = \text{Const } c + (\sum i \leftarrow [0..<n]. \text{Const } (a i) * \text{PVar } i)$ 
   $\wedge c \geq 0 \wedge (\forall i < n. a i > 0)$ 
proof -

```

```

have sum-zero: ( $\bigwedge x. x \in \text{set } xs \implies x = 0$ )  $\implies$  sum-list (xs :: int list) = 0 for
xs by (induct xs, auto)
interpret poly-inter undefined undefined (>) :: int  $\Rightarrow$  - .
from coefficients-of-linear-poly[OF linear] obtain c a vs
  where p: p = Const c + ( $\sum i \leftarrow vs. \text{Const } (a\ i) * PVar\ i$ )
  and vsd: distinct vs set vs = vars p sorted-list-of-set (vars p) = vs
  and nz:  $\bigwedge v. v \in \text{set } vs \implies a\ v \neq 0$ 
  and c: c = coeff p 0
  and a:  $\bigwedge i. a\ i = \text{coeff } p\ (\text{monomial } 1\ i)$  by blast
have vs: vs = [0..<n] unfolding vsd(3)[symmetric] unfolding vars
by (simp add: lessThan-atLeast0)
show ?thesis unfolding p vs
proof (intro exI conjI allI impI, rule refl)
  show c: c  $\geq$  0 using poly[unfolded valid-poly-def, rule-format, of  $\lambda -.$  0,
unfolded p]
  by (auto simp: coeff-add[symmetric] coeff-Const coeff-sum-list o-def co-
eff-Const-mult
coeff-Var monomial-0-iff assignment-def)
  fix i
  assume i < n
  hence i: i  $\in$  set vs unfolding vs by auto
  from nz[OF this] have a0: a i  $\neq$  0 by auto
  from split-list[OF i] obtain bef aft where vsi: vs = bef @ [i] @ aft by auto
  with vsd(1) have i: i  $\notin$  set (bef @ aft) by auto
  define  $\alpha$  where  $\alpha = (\lambda x. \text{if } x = i \text{ then } c + 1 \text{ else } 0)$ 
  have assignment  $\alpha$  unfolding assignment-def  $\alpha$ -def using c by auto
  from poly[unfolded valid-poly-def, rule-format, OF this, unfolded p]
  have  $0 \leq c + (\sum x \leftarrow bef @ aft. a\ x * \alpha\ x) + (a\ i * \alpha\ i)$ 
  unfolding insertion-add vsi map-append sum-list-append insertion-Const
insertion-sum-list
map-map o-def insertion-mult insertion-Var
by (simp add: algebra-simps)
  also have ( $\sum x \leftarrow bef @ aft. a\ x * \alpha\ x$ ) = 0 by (rule sum-zero, insert i, auto
simp:  $\alpha$ -def)
  also have  $\alpha\ i = (c + 1)$  unfolding  $\alpha$ -def by auto
  finally have le:  $0 \leq c * (a\ i + 1) + a\ i$  by (simp add: algebra-simps)
  with c have a i  $\geq$  0
  by (smt (verit, best) mult-le-0-iff)
  with a0 show a i > 0 by simp
qed
qed

locale poly-input-to-solution-common = poly-input p q +
poly-inter F' I (>) :: int  $\Rightarrow$  int  $\Rightarrow$  bool for p q I and F' :: (poly-input.symbol  $\times$ 
nat) set and argsL argsR +
assumes orient:
orient-rule (Fun f-sym ([Var y1, Var y2, a-t (encode-poly y3 p) (Var y3)] @
argsL),
Fun f-sym ([a-t (Var y1) z-t, a-t z-t (Var y2), a-t (encode-poly y3 q) (Var y3)]

```

@ argsR))
and len-args: length argsL = length argsR
and y123: {y1,y2,y3} ∩ (∪ (vars-term ‘ set (argsL @ argsR))) = {}
and FF': insert (f-sym, 3 + length argsR) F ⊆ F'
and linear-mono-interpretation: (g,n) ∈ insert (f-sym, 3 + length argsR) F ⇒

$$\exists c a. I g = \text{Const } c + (\sum_{i \leftarrow [0..<n]}. \text{Const } (a \ i) * \text{PVar } i)$$

$$\wedge c \geq 0 \wedge (\forall i < n. a \ i > 0)$$

begin

abbreviation ff where ff ≡ (f-sym, 3 + length argsR)

abbreviation args where args ≡ [3..<length argsR + 3]

lemma extract-a-poly: ∃ a0 a1 a2. I a-sym = Const a0 + Const a1 * PVar 0 + Const a2 * PVar 1

$\wedge a0 \geq 0 \wedge a1 > 0 \wedge a2 > 0$

proof –

have [simp]: [0 ..<2] = [0,1] **by** code-simp

have [simp]: (∀ i < 2. P i) = (P 0 ∧ P (1 :: nat)) **for** P **by** (auto simp add: numeral-eq-Suc less-Suc-eq)

have (a-sym,2) ∈ insert ff F **unfolding** F-def **by** auto

from linear-mono-interpretation[OF this]

show ?thesis **by** force

qed

lemma extract-f-poly: ∃ f0 f1 f2 f3 f4. I f-sym = Const f0 + Const f1 * PVar 0 + Const f2 * PVar 1

+ Const f3 * PVar 2 + (∑ i ← args. Const (f4 i) * PVar i)

$\wedge f0 \geq 0 \wedge f1 > 0 \wedge f2 > 0 \wedge f3 > 0$

proof –

have id: [0..<3 + length argsR] = [0,1,2] @ args

by (simp add: numeral-3-eq-3 upt-rec)

have ff ∈ insert ff F **by** auto

from linear-mono-interpretation[OF this] **obtain** c a

where Iff: I f-sym = Const c + (∑ i ← [0..<3 + length argsR]. Const (a i) * PVar i)

and c: 0 ≤ c **and** a: ∧ i. i < 3 + length argsR ⇒ 0 < a i **by** blast

show ?thesis

apply (rule exI[of - c])

apply (rule exI[of - a 0])

apply (rule exI[of - a 1])

apply (rule exI[of - a 2])

apply (rule exI[of - a])

using c a[of 0] a[of 1] a [of 2] Iff id **by** auto

qed

lemma extract-z-poly: ∃ ze0. I z-sym = Const ze0 ∧ ze0 ≥ 0

proof –

have (z-sym,0) ∈ insert ff F **unfolding** F-def **by** auto

from *linear-mono-interpretation*[*OF this*] **show** *?thesis* **by** *auto*
qed

lemma *solution: positive-poly-problem p q*

proof –

from *extract-a-poly* **obtain** *a0 a1 a2* **where**

*Ia: I a-sym = Const a0 + Const a1 * PVar 0 + Const a2 * PVar 1*

and *a: 0 ≤ a0 0 < a1 0 < a2*

by *auto*

from *extract-f-poly* **obtain** *f0 f1 f2 f3 f4* **where**

*If: I f-sym = Const f0 + Const f1 * PVar 0 + Const f2 * PVar 1 + Const f3*
** PVar 2 + (∑ i←args. Const (f4 i) * PVar i)*

and *f: 0 ≤ f0 0 < f1 0 < f2 0 < f3*

by *auto*

from *extract-z-poly* **obtain** *ze0* **where**

Iz: I z-sym = Const ze0

and *z: 0 ≤ ze0*

by *auto*

{

fix *x*

assume *x ∈ V*

hence *(v-sym x, 1) ∈ insert ff F unfolding F-def* **by** *auto*

from *linear-mono-interpretation*[*OF this*]

have $\exists c a. I (v\text{-sym } x) = \text{Const } c + \text{Const } a * \text{PVar } 0 \wedge 0 < a$ **by** *auto*

}

hence $\forall x. \exists c a. x \in V \longrightarrow I (v\text{-sym } x) = \text{Const } c + \text{Const } a * \text{PVar } 0 \wedge 0 < a$ **by** *auto*

from *choice*[*OF this*] **obtain** *v0* **where** $\forall x. \exists a. x \in V \longrightarrow I (v\text{-sym } x) = \text{Const } (v0 x) + \text{Const } a * \text{PVar } 0 \wedge 0 < a$ **by** *auto*

from *choice*[*OF this*] **obtain** *v1* **where**

*Iv: $\bigwedge x. x \in V \implies I (v\text{-sym } x) = \text{Const } (v0 x) + \text{Const } (v1 x) * \text{PVar } 0$* **and**

v: $\bigwedge x. x \in V \implies 0 < v1 x$ **by** *auto*

let *?lhs = Fun f-sym ([TVar y1, TVar y2, Fun a-sym [encode-poly y3 p, TVar y3]] @ argsL)*

let *?rhs = Fun f-sym*

([Fun a-sym [TVar y1, Fun z-sym []], Fun a-sym [Fun z-sym [], TVar y2],

Fun a-sym [encode-poly y3 q, TVar y3]] @

argsR)

from *orient*[*unfolded orient-rule*]

have *gt: gt-poly (eval ?lhs) (eval ?rhs)* **by** *auto*

have [*simp*]: *Suc (Suc (Suc (Suc 0))) = 4* **by** *simp*

have [*simp*]: *Suc (Suc 0) = 2* **by** *simp*

define *restL* **where** *restL = substitute*

(λi. if i < length argsR + 3

then eval ((TVar y1 # TVar y2 # Fun a-sym [encode-poly y3 p, TVar y3]

argsL) ! i) else 0)

*(∑ i←local.args. PVar i * Const (f4 i))*


```

define b0 where b0 = f3 * a0 + f0
define b1 where b1 = f3 * a0 + f0 + f1 * a0 + f1 * a2 * ze0 + f2 * a0 +
f2 * a1 * ze0
define b2 where b2 = f3 * a1
define b3 where b3 = f3 * a2
have b23: b2 > 0 b3 > 0 unfolding b2-def b3-def using a f by auto
let ?pt = encode-poly y3 p
let ?qt = encode-poly y3 q
from vars-encode-poly[of y3]
have vars: vars-term ?pt  $\cup$  vars-term ?qt  $\subseteq$  {y3} by auto
from vars-eval vars
have vars: vars (eval ?pt)  $\cup$  vars (eval ?qt)  $\subseteq$  {y3} by auto
have [simp]: Suc (Suc (Suc (length argsR))) = length argsR + 3
by presburger

have lhs: eval ?lhs = Const b0 +
  Const f1 * PVar y1 +
  Const f2 * PVar y2 +
  Const b2 * eval ?pt + Const b3 * PVar y3 + restL
using If Ia len-args by (simp add: algebra-simps Const-add Const-mult b0-def
b2-def b3-def restL-def)
define  $\beta$  where  $\beta$  z1 z2 z3 = ((( $\lambda$  x. 0 :: int) (y1 := z1)) (y2 := z2)) (y3 :=
z3) for z1 z2 z3
have args: args = map ( $\lambda$  z. z + 3) [0.. $\text{length}$  argsR]
using map-add-upt by presburger
define rl where rl = insertion ( $\beta$  0 0 0) restL
{
  have insRestL: insertion ( $\beta$  z1 z2 z3) restL = ( $\sum$  x $\leftarrow$ [0.. $\text{length}$ 
argsR]. (insertion ( $\beta$  z1 z2 z3) (eval (argsL ! x)) * (f4 (x + 3)))) for
z1 z2 z3
  unfolding restL-def insertion-substitute insertion-sum-list map-map o-def
if-distrib args insertion-mult insertion-Var insertion-Const
  apply (rule arg-cong[of - - sum-list])
  apply (rule map-cong[OF refl]) by auto
  have insRestL: insertion ( $\beta$  z1 z2 z3) restL = rl for z1 z2 z3
  unfolding insRestL rl-def
  apply (rule arg-cong[of - - sum-list])
  apply (rule map-cong[OF refl])
  apply (rule arg-cong[of - -  $\lambda$  x. x * -])
  apply (rule insertion-irrelevant-vars)
  subgoal for v i unfolding len-args[symmetric] using y123 vars-eval[of argsL
! v]
  by (auto simp:  $\beta$ -def)
  done
} note ins-restL = this

define restR where restR = substitute
  ( $\lambda$ i. if i < length argsR + 3
  then eval

```

```

      ((Fun a-sym [TVar y1, Fun z-sym []] #
        Fun a-sym [Fun z-sym [], TVar y2] # Fun a-sym [encode-poly y3 q,
TVar y3] # argsR) !
      i)
    else 0)
  (∑ i←args. PVar i * Const (f4 i))
have rhs: eval ?rhs = Const b1 +
  Const (f1 * a1) * PVar y1 +
  Const (f2 * a2) * PVar y2 +
  Const b2 * eval ?qt + Const b3 * PVar y3 + restR
  unfolding restR-def using If Ia Iz by (simp add: algebra-simps Const-add
Const-mult b1-def b2-def b3-def)
  define rr where rr = insertion (β 0 0 0) restR
  {
    have insRestR: insertion (β z1 z2 z3) restR = (∑ x←[0..<length
argsR]. (insertion (β z1 z2 z3) (eval (argsR ! x)) * (f4 (x + 3)))) for
z1 z2 z3
    unfolding restR-def insertion-substitute insertion-sum-list map-map o-def
if-distrib args insertion-mult insertion-Var insertion-Const
    apply (rule arg-cong[of - - sum-list])
    apply (rule map-cong[OF refl]) by auto
    have insRestR: insertion (β z1 z2 z3) restR = rr for z1 z2 z3
    unfolding insRestR rr-def
    apply (rule arg-cong[of - - sum-list])
    apply (rule map-cong[OF refl])
    apply (rule arg-cong[of - - λ x. x * -])
    apply (rule insertion-irrelevant-vars)
    subgoal for v i using y123 vars-eval[of argsR ! v]
    by (auto simp: β-def)
    done
  } note ins-restR = this

  have [simp]: β z1 z2 z3 y1 = z1 for z1 z2 z3 unfolding β-def using y-vars by
auto
  have [simp]: β z1 z2 z3 y2 = z2 for z1 z2 z3 unfolding β-def using y-vars by
auto
  have [simp]: β z1 z2 z3 y3 = z3 for z1 z2 z3 unfolding β-def using y-vars by
auto
  have β: z1 ≥ 0 ⇒ z2 ≥ 0 ⇒ z3 ≥ 0 ⇒ assignment (β z1 z2 z3) for z1 z2
z3
  unfolding assignment-def β-def by auto
  define l1 where l1 = insertion (β 0 0 0) (eval ?lhs)
  have ins-lhs: insertion (β z1 z2 0) (eval ?lhs) = f1 * z1 + f2 * z2 + l1 for z1
z2
  unfolding lhs l1-def
  apply (simp add: insertion-add insertion-mult insertion-Const insertion-Var
ins-restL)
  apply (rule disjI2)
  apply (rule insertion-irrelevant-vars)

```

```

using vars by auto

define l2 where l2 = insertion (β 0 0 0) (eval ?rhs)
have ins-rhs: insertion (β z1 z2 0) (eval ?rhs) = f1 * a1 * z1 + f2 * a2 * z2
+ l2 for z1 z2
  unfolding rhs l2-def
  apply (simp add: insertion-add insertion-mult insertion-Const insertion-Var
ins-restR)
  apply (rule disjI2)
  apply (rule insertion-irrelevant-vars)
  using vars by auto
define l where l = l2 - l1
have gt-inst: 0 ≤ z1 ⇒ 0 ≤ z2 ⇒ f1 * a1 * z1 + f2 * a2 * z2 + l < f1 *
z1 + f2 * z2 for z1 z2
  using gt[unfolded gt-poly-def, rule-format, OF β, of z1 z2 0, unfolded ins-lhs
ins-rhs]
  by (auto simp: l-def)
{
  define a1' where a1' = a1 - 1
  define z where z = f1 * a1'
  have a1: a1 = 1 + a1' unfolding a1'-def by auto
  have a1': a1' ≥ 0 using a unfolding a1 by auto
  from gt-inst[of abs l 0, unfolded a1]
  have z * |l| + l < 0
    by (simp add: algebra-simps z-def)
  hence z ≤ 0
    by (smt (verit) mult-le-cancel-right1)
  with ⟨0 < f1⟩ have a1' ≤ 0 unfolding z-def
    by (simp add: mult-le-0-iff)
  with a1' a1 have a1 = 1 by auto
} note a1 = this
{
  define a2' where a2' = a2 - 1
  define z where z = f2 * a2'
  have a2: a2 = 1 + a2' unfolding a2'-def by auto
  have a2': a2' ≥ 0 using a unfolding a2 by auto
  from gt-inst[of 0 abs l, unfolded a2]
  have z * |l| + l < 0
    by (simp add: algebra-simps z-def)
  hence z ≤ 0
    by (smt (verit) mult-le-cancel-right1)
  with ⟨0 < f2⟩ have a2' ≤ 0 unfolding z-def
    by (simp add: mult-le-0-iff)
  with a2' a2 have a2 = 1 by auto
} note a2 = this

have Ia: I a-sym = Const a0 + PVar 0 + PVar 1
  unfolding Ia a1 a2 Const-1 by simp

```

```

{
  fix c :: int
  assume c ≥ 0
  then obtain n where cn: c = int n by (metis nonneg-eq-int)
  hence natc: nat c = n by auto
  have ∃ d. eval (encode-num y3 c) = Const d + Const c * PVar y3
    unfolding encode-num-def natc unfolding cn
    by (induct n, auto simp: Iz Ia Const-0 Const-1 algebra-simps Const-add, auto
simp: Const-add[symmetric])
  } note encode-num = this

{
  fix x e f t
  assume x: x ∈ V and eval: ∃ c. eval t = Const c + Const f * PVar y3
  have ∃ d. eval ((v-t x ^^ e) t) = Const d + Const ((v1 x) ^ e * f) * PVar y3
  proof (induct e)
    case 0
    show ?case using eval by auto
  next
    case (Suc e)
    then obtain d where IH: eval ((v-t x ^^ e) t) = Const d + Const (v1 x ^
e * f) * PVar y3 by auto
    show ?case by (simp add: IH Iv[OF x] algebra-simps Const-mult)
      (auto simp: Const-mult[symmetric] Const-add[symmetric])
  qed
  } note v-pow-e = this

{
  fix c :: int and m
  assume c: c ≥ 0
  define base where base = encode-num y3 c
  define xes where xes = var-list m
  assume keys: keys m ⊆ V
  from encode-num[OF c] obtain d where base: eval base = Const d + Const
c * PVar y3
  by (auto simp: base-def)
  from var-list[of m c]
  have monom: monom m c = Const c * (∏ (x, e)← xes . PVar x ^ e) unfolding
xes-def .
  have ∃ d. eval (encode-monom y3 m c) = Const d + Const (insertion v1
(monom m c)) * PVar y3
  using var-list-keys[of - - m]
  unfolding encode-monom-def monom xes-def[symmetric] base-def[symmetric]
  proof (induct xes)
    case Nil
    show ?case by (auto simp: base insertion-Const)
  next
    case (Cons xe xes)

```

```

obtain  $x e$  where  $xe: xe = (x, e)$  by force
with Cons keys have  $x: x \in V$  by auto
from Cons
have  $\exists d. \text{eval} (\text{rec-list base } (\lambda (i, e) \cdot v\text{-}t\ i \ \widehat{\sim} \ e) \ xes) =$ 
   $\text{Const } d + \text{Const} (c * \text{insertion } v1 (\prod (x, y) \leftarrow xes. \text{PVar } x \ \widehat{\sim} \ y)) * \text{PVar } y^3$ 
by (auto simp: insertion-mult insertion-Const)
from  $v\text{-pow-}e[\text{OF } x \ \text{this}, \text{ of } e]$  obtain  $d$  where
   $\text{id: eval } ((v\text{-}t\ x \ \widehat{\sim} \ e) (\text{rec-list base } (\lambda (i, e) \cdot v\text{-}t\ i \ \widehat{\sim} \ e) \ xes)) =$ 
   $\text{Const } d + \text{Const} (v1\ x \ \widehat{\sim} \ e * (c * \text{insertion } v1 (\prod (x, y) \leftarrow xes. \text{PVar } x \ \widehat{\sim} \ y))) * \text{PVar } y^3$ 
by auto
show  $?case$  by (intro exI[of - d], simp add: xe id,
  auto simp: Const-power Const-mult insertion-mult insertion-Const
insertion-power insertion-Var)
qed
} note encode-monom = this

{
  fix  $r :: \text{int mpoly}$ 
  assume vars: vars  $r \subseteq V$  and pos: positive-poly  $r$ 
  define mcs where  $mcs = \text{monom-list } r$ 
  from monom-list[of r] have  $r: r = (\sum (m, c) \leftarrow mcs. \text{monom } m\ c)$  unfolding
mcs-def by auto
  have mcs-pos: (m,c) \in set mcs \implies c \ge 0 for  $m\ c$ 
    using monom-list-coeff pos unfolding mcs-def positive-poly-def by auto
  from monom-list-keys[of - - r, folded mcs-def] vars
  have mcs-V: (m,c) \in set mcs \implies keys m \subseteq V for  $m\ c$  by auto
  have  $\exists d. \text{eval} (\text{encode-poly } y^3\ r) = \text{Const } d + \text{Const} (\text{insertion } v1\ r) * \text{PVar } y^3$ 
mcs-V
    unfolding encode-poly-def mcs-def[symmetric] unfolding  $r$  using mcs-pos
mcs-V
    unfolding insertion-sum-list map-map o-def
proof (induct mcs)
  case Nil
    show  $?case$  by (auto simp add: Iz Const-0)
  next
    case (Cons mc mcs)
    obtain  $m\ c$  where  $mc: mc = (m, c)$  by force
    from Cons(2) mc have  $c: c \ge 0$  by auto
    from Cons(3) mc have  $\text{keys } m \subseteq V$  by auto
    from encode-monom[OF c this]
    obtain  $d1$  where  $m: \text{eval} (\text{encode-monom } y^3\ m\ c) = \text{Const } d1 + \text{Const}$ 
(insertion v1 (monom m c)) * PVar y^3 by auto
    from Cons(1)[OF Cons(2-3)]
    obtain  $d2$  where  $\text{IH: eval } (\text{rec-list } z\text{-}t\ (\lambda (m, c) \cdot a\text{-}t\ (\text{encode-monom } y^3\ m\ c))\ mcs) =$ 
   $\text{Const } d2 + \text{Const} (\sum mc \leftarrow mcs. \text{insertion } v1 (\text{case } mc \text{ of } (m, c) \Rightarrow \text{monom } m\ c)) * \text{PVar } y^3$ 
by force

```

```

show ?case unfolding mc
  apply (simp add: Ia m IH)
  apply (simp add: Const-add algebra-simps)
  by (auto simp flip: Const-add)
qed
} note encode-poly = this

from encode-poly[OF - pq(1)] V-def
obtain d1 where p: eval (encode-poly y3 p) = Const d1 + Const (insertion v1 p) * PVar y3 by auto

from encode-poly[OF - pq(2)] V-def
obtain d2 where q: eval (encode-poly y3 q) = Const d2 + Const (insertion v1 q) * PVar y3 by auto

define d3 where d3 = b0 + b2 * d1 + rl
have ins-lhs: insertion (β 0 0 z3) (eval ?lhs) = d3 + (b3 + b2 * insertion v1 p) * z3 for z3
  unfolding p d3-def lhs
  by (simp add: insertion-add insertion-mult insertion-Const insertion-Var algebra-simps ins-restL)

define d4 where d4 = b1 + b2 * d2 + rr
have ins-rhs: insertion (β 0 0 z3) (eval ?rhs) = d4 + (b3 + b2 * insertion v1 q) * z3 for z3
  unfolding q d4-def rhs
  by (simp add: insertion-add insertion-mult insertion-Const insertion-Var algebra-simps ins-restR)

define d5 where d5 = d4 - d3

define left where left = b3 + b2 * insertion v1 p
define right where right = b3 + b2 * insertion v1 q
define diff where diff = left - right

have gt-inst: z3 ≥ 0 ⇒ diff * z3 > d5 for z3
  using gt[unfolded gt-poly-def, rule-format, OF β, of 0 0 z3, unfolded ins-lhs ins-rhs]
  by (auto simp: d5-def left-def right-def diff-def algebra-simps)
from this[of abs d5]
have diff ≥ 0
  by (smt (verit) Groups.mult-ac(2) mult-le-cancel-right1 mult-minus-right)
from this[unfolded diff-def left-def right-def]
have b2 * insertion v1 p ≥ b2 * insertion v1 q by auto
with <b2 > 0> have solution: insertion v1 p ≥ insertion v1 q by simp

define α where α x = (if x ∈ V then v1 x else 1) for x
from v have α: positive-interpret α unfolding positive-interpret-def α-def by auto
have insertion α q = insertion v1 q

```

by (rule insertion-irrelevant-vars, auto simp: α -def V-def)
 also have $\dots \leq$ insertion v1 p by fact
 also have $\dots =$ insertion α p
 by (rule insertion-irrelevant-vars, auto simp: α -def V-def)
 finally show positive-poly-problem p q
 unfolding positive-poly-problem-def[OF pq] using α by auto
 qed
 end

locale solution-poly-input-R = poly-input p q + poly-inter F-R I ($>$) :: int \Rightarrow -
for p q I +
 assumes orient: orient-rule (lhs-R,rhs-R)
 and linear-mono-interpretation: $(g,n) \in F-R \implies$
 $\exists c a. I g = \text{Const } c + (\sum i \leftarrow [0..<n]. \text{Const } (a i) * PVar i)$
 $\wedge c \geq 0 \wedge (\forall i < n. a i > 0)$
begin

lemma solution: positive-poly-problem p q
apply (rule poly-input-to-solution-common.solution[of - - I F-R [o-t] [z-t]])
apply (unfold-locales)
subgoal using orient **unfolding** lhs-R-def rhs-R-def **by** simp
subgoal **by** simp
subgoal **by** simp
subgoal **unfolding** F-R-def **by** auto
subgoal **for** g n **using** linear-mono-interpretation[of g n] **unfolding** F-R-def **by**
 auto
done
end

locale lin-term-poly-input = poly-input p q **for** p q +
 assumes lin-term: termination-by-linear-int-poly-interpretation F-R R
begin

definition I **where** $I = (\text{SOME } I. \text{linear-int-poly-inter } F-R I \wedge \text{int-poly-inter.termination-by-poly-interpretation } F-R I R)$

lemma I: linear-int-poly-inter F-R I int-poly-inter.termination-by-poly-interpretation
 F-R I R
using someI-ex[OF lin-term[unfolded termination-by-linear-int-poly-interpretation-def],
 folded I-def] **by** auto

sublocale linear-int-poly-inter F-R I **by** (rule I(1))

lemma orient: orient-rule (lhs-R,rhs-R)
using I(2)[unfolded termination-by-interpretation-def termination-by-poly-interpretation-def]
unfolding R-def **by** auto

lemma extract-linear-poly: **assumes** g: $(g,n) \in F-R$
shows $\exists c a. I g = \text{Const } c + (\sum i \leftarrow [0..<n]. \text{Const } (a i) * PVar i)$

$\wedge c \geq 0 \wedge (\forall i < n. a_i > 0)$

proof –

define p **where** $p = I\ g$

have $sum-zero: (\bigwedge x. x \in set\ xs \implies x = 0) \implies sum-list\ (xs :: int\ list) = 0$ **for**

xs **by** (*induct* xs , *auto*)

from $valid[unfolded\ valid-monotone-poly-inter-def, rule-format, OF\ g]$

have $poly: valid-poly\ p$

and $mono: monotone-poly\ \{..<n\}\ p$

and $vars: vars\ p = \{..<n\}$

by (*auto simp: valid-monotone-poly-def p-def*)

from $linear[OF\ g]\ p-def$

have $linear: total-degree\ p \leq 1$ **by** *auto*

show $?thesis\ unfolding\ p-def[symmetric]$

by (*rule monotone-linear-poly-to-coeffs[OF linear poly mono vars]*)

qed

lemma $solution: positive-poly-problem\ p\ q$

apply (*rule solution-poly-input-R.solution[of - - I]*)

apply (*unfold-locales*)

apply (*rule orient*)

apply (*rule extract-linear-poly*)

by *auto*

end

locale $wm-lin-orient-poly-input = poly-input\ p\ q$ **for** $p\ q +$

assumes $wm-orient: orientation-by-linear-wm-int-poly-interpretation\ F-R\ R'$

begin

definition I **where** $I = (SOME\ I. linear-wm-int-poly-inter\ F-R\ I \wedge wm-int-poly-inter.oriented-by-interpretation\ F-R\ I\ R')$

lemma $I: linear-wm-int-poly-inter\ F-R\ I\ wm-int-poly-inter.oriented-by-interpretation\ F-R\ I\ R'$

using $someI-ex[OF\ wm-orient[unfolded\ orientation-by-linear-wm-int-poly-interpretation-def], folded\ I-def]$ **by** *auto*

sublocale $linear-wm-int-poly-inter\ F-R\ I$ **by** (*rule I(1)*)

lemma $orient-R': orient-rule\ (lhs-R', rhs-R')$

using $I(2)[unfolded\ oriented-by-interpretation-def]$ **unfolding** $R'-def$ **by** *auto*

lemma $extract-linear-poly: assumes\ g: (g, n) \in F-R$

shows $\exists c\ a. I\ g = Const\ c + (\sum i \leftarrow [0..<n]. Const\ (a\ i) * PVar\ i)$

$\wedge c \geq 0 \wedge (\forall i < n. a_i \geq 0)$

proof –

define p **where** $p = I\ g$

have $sum-zero: (\bigwedge x. x \in set\ xs \implies x = 0) \implies sum-list\ (xs :: int\ list) = 0$ **for**

xs **by** (*induct* xs , *auto*)

from $valid[unfolded\ valid-weakly-monotone-inter-def\ valid-weakly-monotone-poly-def,$


```

rule-format, OF g refl p-def]
have poly: valid-poly p
  and mono: weakly-monotone-poly {.. $n$ } p
  and vars: vars p  $\subseteq$  {.. $n$ }
  by (auto simp: valid-monotone-poly-def p-def)
from linear[OF g] p-def
have linear: total-degree p  $\leq$  1 by auto
from coefficients-of-linear-poly[OF linear] obtain c b vs
  where p: p = Const c + ( $\sum$   $i \leftarrow$  vs. Const (b i) * PVar i)
  and vsd: distinct vs set vs = vars p sorted-list-of-set (vars p) = vs
  and nz:  $\bigwedge$  v. v  $\in$  set vs  $\implies$  b v  $\neq$  0
  and c: c = coeff p 0
  and b:  $\bigwedge$  i. b i = coeff p (monomial 1 i) by blast
define a where a x = (if x  $\in$  vars p then b x else 0) for x
have p = Const c + ( $\sum$   $i \leftarrow$  vs. Const (b i) * PVar i) by fact
also have ( $\sum$   $i \leftarrow$  vs. Const (b i) * PVar i) = ( $\sum$  i  $\in$  set vs. Const (b i) * PVar
i) using vsd(1)
  by (rule sum-list-distinct-conv-sum-set)
also have ... = ( $\sum$  i  $\in$  set vs. Const (a i) * PVar i) + 0 by (subst sum.cong,
auto simp: a-def vsd)
also have 0 = ( $\sum$  i  $\in$  {.. $n$ } - set vs. Const (a i) * PVar i)
  by (subst sum.neutral, auto simp: a-def vsd)
also have ( $\sum$  i  $\in$  set vs. Const (a i) * PVar i) + ... = ( $\sum$  i  $\in$  set vs  $\cup$  {.. $n$ }
- set vs). Const (a i) * PVar i)
  by (subst sum.union-inter[symmetric], auto)
also have set vs  $\cup$  {.. $n$ } - set vs = set [0.. $n$ ] using vars vsd by auto
finally have pca: p = Const c + ( $\sum$  i  $\leftarrow$  [0.. $n$ ]. Const (a i) * PVar i)
  by (subst sum-list-distinct-conv-sum-set, auto)

show ?thesis unfolding p-def[symmetric] pca
proof (intro exI conjI allI impI, rule refl)
  show c: c  $\geq$  0 using poly[unfolded valid-poly-def, rule-format, of  $\lambda$  -. 0,
unfolded p]
  by (auto simp: coeff-add[symmetric] coeff-Const coeff-sum-list o-def co-
eff-Const-mult
coeff-Var monomial-0-iff assignment-def)
fix i
assume i < n
show a i  $\geq$  0
proof (cases i  $\in$  set vs)
  case False
  thus ?thesis unfolding a-def using vsd by auto
next
  case i: True
  from nz[OF this] have a0: a i  $\neq$  0 b i = a i using i by (auto simp: a-def
vsd)
  from split-list[OF i] obtain bef aft where vsi: vs = bef @ [i] @ aft by auto
  with vsd(1) have i: i  $\notin$  set (bef @ aft) by auto
  define  $\alpha$  where  $\alpha$  = ( $\lambda$  x. if x = i then c + 1 else 0)

```

have *assignment* α **unfolding** *assignment-def* α -def **using** c **by** *auto*
from *poly*[*unfolded valid-poly-def*, *rule-format*, *OF this*, *unfolded p*]
have $0 \leq c + (\sum x \leftarrow \text{bef} \ @ \ \text{aft. } b \ x \ * \ \alpha \ x) + (b \ i \ * \ \alpha \ i)$
unfolding *insertion-add vsi map-append sum-list-append insertion-Const*
insertion-sum-list
map-map o-def insertion-mult insertion-Var
by (*simp add: algebra-simps*)
also have $(\sum x \leftarrow \text{bef} \ @ \ \text{aft. } b \ x \ * \ \alpha \ x) = 0$ **by** (*rule sum-zero, insert i, auto*
simp: α -def)
also have $\alpha \ i = (c + 1)$ **unfolding** α -def **by** *auto*
finally have $le: 0 \leq c * (a \ i + 1) + a \ i$ **using** $a0$ **by** (*simp add: algebra-simps*)
with c **show** $a \ i \geq 0$
by (*smt (verit, best) mult-le-0-iff*)
qed
qed
qed

lemma *extract-a-poly*: $\exists a0 \ a1 \ a2. I \ a\text{-sym} = \text{Const } a0 + \text{Const } a1 * \text{PVar } 0 +$
 $\text{Const } a2 * \text{PVar } 1$
 $\wedge a0 \geq 0 \wedge a1 \geq 0 \wedge a2 \geq 0$
proof –
have [*simp*]: $[0 \ ..<2] = [0,1]$ **by** *code-simp*
have [*simp*]: $(\forall i < 2. P \ i) = (P \ 0 \ \wedge \ P \ (1 \ :: \ \text{nat}))$ **for** P **by** (*auto simp add:*
numeral-eq-Suc less-Suc-eq)
have $(a\text{-sym}, 2) \in F\text{-R}$ **unfolding** *F-R-def F-def* **by** *auto*
from *extract-linear-poly*[*OF this*]
show *?thesis* **by** *force*
qed

lemma *extract-f-poly*: $\exists f0 \ f1 \ f2 \ f3 \ f4. I \ f\text{-sym} = \text{Const } f0 + \text{Const } f1 * \text{PVar } 0$
 $+ \text{Const } f2 * \text{PVar } 1$
 $+ \text{Const } f3 * \text{PVar } 2 + \text{Const } f4 * \text{PVar } 3$
 $\wedge f0 \geq 0 \wedge f1 \geq 0 \wedge f2 \geq 0 \wedge f3 \geq 0 \wedge f4 \geq 0$
proof –
have [*simp*]: $[0 \ ..<4] = [0,1,2,3]$ **by** *code-simp*
have [*simp*]: $(\forall i < 4. P \ i) = (P \ 0 \ \wedge \ P \ (1 \ :: \ \text{nat}) \ \wedge \ P \ 2 \ \wedge \ P \ 3)$ **for** P
by (*auto simp add: numeral-eq-Suc less-Suc-eq*)
have $(f\text{-sym}, 4) \in F\text{-R}$ **unfolding** *F-R-def* **by** *auto*
from *extract-linear-poly*[*OF this*] **obtain** $c \ f$ **where**
 $\text{main: } I \ f\text{-sym} = \text{Const } c + (\sum i \leftarrow [0..<4]. \text{Const } (f \ i) * \text{PVar } i) \wedge 0 \leq c \wedge$
 $(\forall i < 4. 0 \leq f \ i)$ **by** *auto*
show *?thesis*
apply (*rule exI[of - c]*)
apply (*rule exI[of - f 0]*)
apply (*rule exI[of - f 1]*)
apply (*rule exI[of - f 2]*)
apply (*rule exI[of - f 3]*)
by (*insert main, auto*)
qed

lemma *solution: positive-poly-problem p q*
proof –
from *extract-f-poly* **obtain** $f0\ f1\ f2\ f3\ f4$ **where**
If: I f-sym =
 $Const\ f0 + Const\ f1 * PVar\ 0 + Const\ f2 * PVar\ 1 + Const\ f3 * PVar\ 2$
 $+ Const\ f4 * PVar\ 3$
and *fpos: $0 \leq f0\ 0 \leq f1\ 0 \leq f2\ 0 \leq f3\ 0 \leq f4$* **by** *auto*
from *extract-a-poly* **obtain** $a0\ a1\ a2$ **where**
*Ia: I a-sym = Const a0 + Const a1 * PVar 0 + Const a2 * PVar 1*
and *apos: $0 \leq a0\ 0 \leq a1\ 0 \leq a2$* **by** *auto*
{
fix i
assume $i \in V$
hence $v: (v\text{-sym}\ i, 1) \in F\text{-R}$ **unfolding** *F-R-def F-def* **by** *auto*
from *extract-linear-poly[OF v]* **have** $\exists\ v0\ v1. I\ (v\text{-sym}\ i) = Const\ v0 + Const$
 $v1 * PVar\ 0 \wedge v0 \geq 0 \wedge v1 \geq 0$
by *auto*
}
hence $\forall\ i. \exists\ v0\ v1. i \in V \longrightarrow I\ (v\text{-sym}\ i) = Const\ v0 + Const\ v1 * PVar\ 0$
 $\wedge v0 \geq 0 \wedge v1 \geq 0$ **by** *auto*
from *choice[OF this]* **obtain** $v0$ **where** $\forall\ i. \exists\ v1. i \in V \longrightarrow I\ (v\text{-sym}\ i) =$
 $Const\ (v0\ i) + Const\ v1 * PVar\ 0 \wedge v0\ i \geq 0 \wedge v1 \geq 0$ **by** *auto*
from *choice[OF this]* **obtain** $v1$ **where** $Iv: \bigwedge\ i. i \in V \implies I\ (v\text{-sym}\ i) = Const$
 $(v0\ i) + Const\ (v1\ i) * PVar\ 0$
and $vpos: \bigwedge\ i. i \in V \implies v0\ i \geq 0 \wedge v1\ i \geq 0$ **by** *auto*

have $(z\text{-sym}, 0) \in F\text{-R}$ **unfolding** *F-R-def F-def* **by** *auto*
from *extract-linear-poly[OF this]* **obtain** $z0$ **where**
Iz: I z-sym = Const z0
and $zpos: z0 \geq 0$ **by** *auto*

have $(o\text{-sym}, 0) \in F\text{-R}$ **unfolding** *F-R-def F-def* **by** *auto*
from *extract-linear-poly[OF this]* **obtain** $o0$ **where**
Io: I o-sym = Const o0
and $opos: o0 \geq 0$ **by** *auto*

have $prod\text{-ge}: (\bigwedge\ x. x \in set\ xs \implies x \geq 0) \implies prod\text{-list}\ xs \geq 0$ **for** $xs :: int\ list$
by *(induct xs, auto)*
define $d1$ **where** $d1 = prod\text{-list}\ ([a1, a2, f1, f2, f3, f4] @ map\ v1\ V\text{-list})$
have $d1: d1 \geq 0$ **unfolding** *d1-def* **using** *apos fpos vpos*
by *(intro prod-ge, auto simp: V-list)*
from *inter-all-symbol-pos-ctxt-generic[of I, OF If Ia Iv Iz]*
obtain d **where** $ctxt: \bigwedge\ t. eval\ (all\text{-symbol}\text{-pos}\text{-ctxt}\ t) =$
 $Const\ d + Const\ d1 * eval\ t$ **by** *(auto simp: d1-def)*

{
fix $\beta :: var \Rightarrow int$

```

assume assignment  $\beta$ 
from orient-R'[unfolded orient-rule split gt-poly-def, rule-format, OF this]
have insertion  $\beta$  (eval lhs-R') > insertion  $\beta$  (eval rhs-R') (is ?A) by auto
also have ?A  $\longleftrightarrow$  d1 * insertion  $\beta$  (eval lhs-R) > d1 * insertion  $\beta$  (eval rhs-R)

  unfolding lhs-R'-def rhs-R'-def ctxt
  insertion-add insertion-mult insertion-Const by auto
  also have ...  $\longleftrightarrow$  (d1 > 0  $\wedge$  insertion  $\beta$  (eval lhs-R) > insertion  $\beta$  (eval
rhs-R))
  using d1 by (simp add: mult-less-cancel-left-disj)
  finally have d1 > 0 insertion  $\beta$  (eval lhs-R) > insertion  $\beta$  (eval rhs-R) by
auto
}
from this(2) this(1)[of  $\lambda$  -. 0]
have d1: d1 > 0 and gt: gt-poly (eval lhs-R) (eval rhs-R)
  unfolding gt-poly-def by (auto simp: assignment-def)

hence orient-R: orient-rule (lhs-R, rhs-R) unfolding orient-rule by auto

from d1 have d1  $\neq$  0 by auto
from this[unfolded d1-def, simplified] apos fpos
have apos: a0  $\geq$  0 a1 > 0 a2 > 0
  and fpos: f0  $\geq$  0 f1 > 0 f2 > 0 f3 > 0 f4 > 0
  and prod: prod-list (map v1 V-list)  $\neq$  0 by auto
from prod have vpos1:  $i \in V \implies v0\ i \geq 0 \wedge v1\ i > 0$  for i using vpos[of i]
  unfolding prod-list-zero-iff set-map V-list by auto

{
  fix g n
  assume (g,n)  $\in$  F-R
  then consider (f) (g,n) = (f-sym,4) | (a) (g,n) = (a-sym,2) | (z) (g,n) =
(z-sym,0)
  | (o) (g,n) = (o-sym,0) | (v) i where (g,n) = (v-sym i, Suc 0)  $i \in V$ 
  unfolding F-R-def F-def by auto
  hence  $\exists c\ a.$   $I\ g = \text{Const } c + (\sum i \leftarrow [0..<n]. \text{Const } (a\ i) * \text{PVar } i) \wedge 0 \leq c \wedge$ 
( $\forall i < n. 0 < a\ i$ )
  proof cases
  case *: a
  have [simp]:  $[0..<2] = [0,1]$  by code-simp
  thus ?thesis using * apos Ia
  by (intro exI[of - a0] exI[of -  $\lambda\ i.$  if  $i = 0$  then a1 else a2], auto)
next
  case *: f
  have [simp]:  $[0..<4] = [0,1,2,3]$  by code-simp
  thus ?thesis using * If fpos
  by (intro exI[of - f0]
  exI[of -  $\lambda\ i.$  if  $i = 0$  then f1 else if  $i = 1$  then f2 else if  $i = 2$  then f3 else
f4], auto)
next

```

```

    case *: z
    show ?thesis using * Iz zpos by auto
next
    case *: o
    show ?thesis using * Io opos by auto
next
    case *: (v i)
    show ?thesis using * Iv[OF *(2)] vpos1[OF *(2)]
      by (intro exI[of - v0 i] exI[of - λ -. v1 i], auto)
    qed
} note main = this

show ?thesis
  apply (rule solution-poly-input-R.solution[of - - I])
  apply unfold-locales
  using orient-R main by auto
qed
end

```

```

context poly-input
begin

```

Theorem 3.4 in paper

theorem *linear-polynomial-termination-with-natural-numbers-undecidable:*
positive-poly-problem p q \longleftrightarrow *termination-by-linear-int-poly-interpretation F-R*
R

proof

```

  assume positive-poly-problem p q
  interpret solvable-poly-problem
    by (unfold-locales, fact)
  from solution-imp-linear-termination-R
  show termination-by-linear-int-poly-interpretation F-R R .
next
  assume termination-by-linear-int-poly-interpretation F-R R
  interpret lin-term-poly-input
    by (unfold-locales, fact)
  from solution show positive-poly-problem p q .
qed

```

Theorem 3.9

theorem *orientation-by-linear-wm-int-poly-interpretation-undecidable:*
positive-poly-problem p q \longleftrightarrow *orientation-by-linear-wm-int-poly-interpretation F-R*
R'

proof

```

  assume positive-poly-problem p q
  interpret solvable-poly-problem
    by (unfold-locales, fact)
  from solution-imp-linear-termination-R'
  have termination-by-linear-int-poly-interpretation F-R R' .

```

```

from this[unfolding termination-by-linear-int-poly-interpretation-def] obtain I
  where lin: linear-int-poly-inter F-R I and
    R': int-poly-inter.termination-by-poly-interpretation F-R I R'
  by auto
interpret linear-int-poly-inter F-R I by fact
show orientation-by-linear-wm-int-poly-interpretation F-R R'
  unfolding orientation-by-linear-wm-int-poly-interpretation-def
proof (intro exI conjI)
  show linear-wm-int-poly-inter F-R I
  proof
  show valid-weakly-monotone-inter unfolding valid-weakly-monotone-inter-def
  proof
  fix f
  assume f ∈ F-R
  from valid[unfolding valid-monotone-poly-inter-def, rule-format, OF this]
  have valid-monotone-poly f by auto
  thus valid-weakly-monotone-poly f
    by (rule monotone-imp-weakly-monotone, auto)
  qed
qed
interpret linear-wm-int-poly-inter F-R I by fact
show oriented-by-interpretation R' unfolding oriented-by-interpretation-def
using R' unfolding termination-by-poly-interpretation-def termination-by-interpretation-def
.
qed
next
  assume orientation-by-linear-wm-int-poly-interpretation F-R R'
  interpret wm-lin-orient-poly-input
    by (unfold-locales, fact)
  from solution show positive-poly-problem p q .
qed
end

```

Separate locale to define another interpretation, i.e., the one of Lemma 3.6

```

locale poly-input-non-lin-solution = poly-input
begin

```

Non-linear interpretation of Lemma 3.6

```

fun I :: symbol ⇒ int mpoly where
  I f-sym = PVar 2 * PVar 3 + PVar 0 + PVar 1 + PVar 2 + PVar 3
| I a-sym = PVar 0 + PVar 1
| I z-sym = 0
| I o-sym = Const (1 + insertion (λ -. 1) q)
| I (v-sym i) = PVar 0

```

```

sublocale inter-R: poly-inter F-R I (>) .

```

```

lemma inter-encode-num: assumes c ≥ 0

```

shows $inter-R.eval (encode-num x c) = Const c * PVar x$
proof –
from *assms* **obtain** n **where** $cn: c = int n$ **by** (*metis nonneg-eq-int*)
hence $natc: nat c = n$ **by** *auto*
show *?thesis* **unfolding** *encode-num-def natc* **unfolding** cn
by (*induct n, auto simp: Const-0 Const-1 algebra-simps Const-add*)
qed

lemma *inter-v-pow-e*: $inter-R.eval ((v-t x \hat{~} e) t) = inter-R.eval t$
by (*induct e, auto*)

lemma *inter-encode-monom*: **assumes** $c: c \geq 0$
shows $inter-R.eval (encode-monom y m c) = Const (insertion (\lambda -.1) (monom m c)) * PVar y$
proof –
define xes **where** $xes = var-list m$
from *var-list*[*of m c*]
have $monom: monom m c = Const c * (\prod (x, e) \leftarrow xes . PVar x \hat{~} e)$ **unfolding** *xes-def* .
show *?thesis* **unfolding** *encode-monom-def monom xes-def*[*symmetric*]
proof (*induct xes*)
case *Nil*
show *?case* **by** (*simp add: inter-encode-num[OF c] insertion-Const*)
next
case (*Cons xe xes*)
obtain $x e$ **where** $xe: xe = (x, e)$ **by** *force*
show *?case* **by** (*simp add: xe inter-v-pow-e Cons Const-power insertion-Const insertion-mult insertion-power insertion-Var Const-mult*)
qed
qed

lemma *inter-encode-poly*: **assumes** *positive-poly r*
shows $inter-R.eval (encode-poly x r) = Const (insertion (\lambda -.1) r) * PVar x$
proof –
define mcs **where** $mcs = monom-list r$
from *monom-list*[*of r*] **have** $r: r = (\sum (m, c) \leftarrow mcs. monom m c)$ **unfolding** *mcs-def* **by** *auto*
have $mcs: (m, c) \in set mcs \implies c \geq 0$ **for** $m c$
using *monom-list-coeff assms* **unfolding** *mcs-def positive-poly-def* **by** *auto*
show *?thesis* **unfolding** *encode-poly-def mcs-def*[*symmetric*] **unfolding** r *insertion-sum-list map-map o-def*
using mcs
proof (*induct mcs*)
case (*Cons mc mcs*)
obtain $m c$ **where** $mc: mc = (m, c)$ **by** *force*
from *Cons(2) mc* **have** $c: c \geq 0$ **by** *auto*
note $monom = inter-encode-monom$ [*OF this, of x m*]
show *?case*
by (*simp add: mc monom algebra-simps, subst Cons(1), insert Cons(2), auto*)

```

simp: Const-add algebra-simps)
qed simp
qed

lemma valid-monotone-inter: inter-R.valid-monotone-poly-inter
  unfolding inter-R.valid-monotone-poly-inter-def
proof (intro ballI, unfold inter-R.valid-monotone-poly-def, clarify, intro conjI)
  fix f n
  assume f: (f,n) ∈ F-R
  have [simp]: vars (PVar 2 * PVar 3 + (PVar 0 :: int mpoly) + PVar (Suc 0)
+ PVar 2 + PVar 3) = {0,1,2,3}
    unfolding vars-def apply (transfer, simp add: Var0-def image-comp) by
code-simp
  have [simp]: vars ((PVar 0 :: int mpoly) + PVar (Suc 0)) = {0,1}
    unfolding vars-def apply (transfer, simp add: Var0-def image-comp) by
code-simp
  from f show vars (I f) = {.. $n$ } unfolding F-R-def F-def by auto
  have insertion (λ -. 1)  $q \geq 0$ 
    by (rule insertion-positive-poly[OF - pq(2)], auto)
  with f show valid-poly (I f) unfolding F-R-def F-def
    by (auto simp: valid-poly-def insertion-add assignment-def insertion-Var inser-
tion-mult insertion-Const)
  have x4:  $x < 4 \implies x = 0 \vee x = \text{Suc } 0 \vee x = 2 \vee x = 3$  for x by linarith
  have x2:  $x < 2 \implies x = 0 \vee x = \text{Suc } 0$  for x by linarith
  have tedious-case: inter-R.monotone-poly {.. $4$ } (I f-sym) unfolding
    monotone-poly-wrt-def I.simps
  proof (intro allI impI, goal-cases)
    case (1  $\alpha$  x v)
    have manual:  $(\alpha(x := v)) 2 * (\alpha(x := v)) 3 \geq \alpha 2 * \alpha 3$ 
      by (intro mult-mono, insert 1, auto simp: assignment-def dest: spec[of - 2])
    thus ?case unfolding insertion-add insertion-mult insertion-Var using 1 x4
  by auto
qed
with f show inter-R.monotone-poly {.. $n$ } (I f) unfolding F-R-def F-def
  by (auto simp: monotone-poly-wrt-def insertion-add insertion-mult insertion-Var
assignment-def
    dest: x4 x2)
qed

```

Lemma 3.6 in the paper

```

lemma orient-R-main: assumes assignment  $\beta$ 
  shows insertion  $\beta$  (inter-R.eval lhs-R) > insertion  $\beta$  (inter-R.eval rhs-R)
proof -
  let ? $\alpha$  = λ -. 1
  have reason: insertion ? $\alpha$   $q + \beta y3 + insertion ?\alpha p * insertion ?\alpha q * \beta y3 +$ 
insertion ? $\alpha p * 2 * \beta y3 \geq 0$ 
    by (intro add-nonneg-nonneg mult-nonneg-nonneg insertion-positive-poly pq,
    insert assms, auto simp: assignment-def)
  show insertion  $\beta$  (inter-R.eval lhs-R) > insertion  $\beta$  (inter-R.eval rhs-R)

```



```

unfolding lhs-R-def rhs-R-def
using reason
by (simp add: inter-encode-poly[OF pq(1)] inter-encode-poly[OF pq(2)]
      insertion-add insertion-mult insertion-Const insertion-Var algebra-simps)
qed

lemma polynomial-termination-R: termination-by-int-poly-interpretation F-R R
unfolding termination-by-int-poly-interpretation-def
proof (intro exI conjI)
interpret int-poly-inter F-R I
by (unfold-locales, rule valid-monotone-inter)
show int-poly-inter F-R I ..
show termination-by-poly-interpretation R
unfolding termination-by-interpretation-def termination-by-poly-interpretation-def
R-def
proof (clarify, intro conjI)
show inter-R.orient-rule (lhs-R,rhs-R)
unfolding inter-R.gt-poly-def inter-R.orient-rule
by (intro allI impI orient-R-main)
qed (insert lhs-R-F rhs-R-F, auto)
qed

lemma polynomial-termination-R': termination-by-int-poly-interpretation F-R R'
unfolding termination-by-int-poly-interpretation-def
proof (intro exI conjI)
interpret int-poly-inter F-R I
by (unfold-locales, rule valid-monotone-inter)
show int-poly-inter F-R I ..
show termination-by-poly-interpretation R'
unfolding termination-by-poly-interpretation-def termination-by-interpretation-def
R'-def
proof (clarify, intro conjI)
show inter-R.orient-rule (lhs-R',rhs-R')
unfolding inter-R.gt-poly-def inter-R.orient-rule
proof (intro allI impI)
fix  $\beta :: \text{var} \Rightarrow \text{int}$ 
assume ass: assignment  $\beta$ 
define zctxt where zctxt vs = z-contexts (map ( $\lambda i. (v\text{-sym } i, 1, 0)$ ) vs) for
vs
have zctxt: inter-R.eval (zctxt vs t) = inter-R.eval t for vs t
unfolding zctxt-def z-contexts-def z-context-def by (induct vs, auto)
have (insertion  $\beta$  (inter-R.eval lhs-R') > insertion  $\beta$  (inter-R.eval rhs-R'))
 $\longleftrightarrow$  insertion  $\beta$  (inter-R.eval (zctxt V-list lhs-R)) > insertion  $\beta$  (inter-R.eval
(zctxt V-list rhs-R))
unfolding lhs-R'-def rhs-R'-def
unfolding all-symbol-pos-ctxt-def contexts-def
unfolding z-contexts-append zctxt-def[symmetric]
by (simp add: z-contexts-def z-context-def nth-append)
also have ...  $\longleftrightarrow$  insertion  $\beta$  (inter-R.eval lhs-R) > insertion  $\beta$  (inter-R.eval

```

```

rhs-R)
  unfolding zctx ..
  also have ... by (rule orient-R-main[OF ass])
  finally show insertion  $\beta$  (inter-R.eval lhs-R') > insertion  $\beta$  (inter-R.eval
rhs-R') .
  qed
qed (insert lhs-R'-F rhs-R'-F, auto)
qed

end
end

```

6 Undecidability of KBO with Subterm Coefficients

```

theory KBO-Subterm-Coefficients-Undecidable

```

```

  imports

```

```

    Hilbert10-to-Inequality

```

```

    Knuth-Bendix-Order.KBO

```

```

    Linear-Poly-Termination-Undecidable

```

```

begin

```

```

lemma count-sum-list: count (sum-list ms) x = sum-list (map ( $\lambda$  m. count m x)
ms)

```

```

  by (induct ms, auto)

```

```

lemma sum-list-scf-list-prod: sum-list (map f (scf-list scf as)) = sum-list (map ( $\lambda$ 
i. scf i * f (as ! i)) [0.. $\text{length}$  as])

```

```

  unfolding scf-list-def

```

```

  unfolding map-concat

```

```

  unfolding sum-list-concat map-map o-def

```

```

  apply (subst zip-nth-conv, force)

```

```

  unfolding map-map o-def split

```

```

  apply (rule arg-cong[of - - sum-list])

```

```

  by (intro nth-equalityI, auto simp: sum-list-replicate)

```

```

lemma count-vars-term-different-var: assumes x:  $x \notin \text{vars-term } t$ 

```

```

  shows count (vars-term-ms (scf-term scf t)) x = 0

```

```

proof -

```

```

  from assms have  $x \notin \text{vars-term (scf-term scf t)}$ 

```

```

  using vars-term-scf-subset by fastforce

```

```

  thus ?thesis

```

```

  by (simp add: count-eq-zero-iff)

```

```

qed

```

```

context kbo

```

```

begin

```

```

definition kbo-orientation :: ( $f, v$ )rule set  $\Rightarrow$  bool where

```

```

  kbo-orientation R = ( $\forall$  (l,r)  $\in$  R. fst (kbo l r))

```

end

definition *kbo-with-sc-termination* :: (*f, v*)rule set \Rightarrow bool **where**

kbo-with-sc-termination $R = (\exists w w0\ sc\ least\ pr\text{-}strict\ pr\text{-}weak.\ admissible\text{-}kbo\ w\ w0\ pr\text{-}strict\ pr\text{-}weak\ least\ sc$

$\wedge\ kbo.kbo\text{-}orientation\ w\ w0\ sc\ least\ pr\text{-}strict\ pr\text{-}weak\ R)$

context *poly-input*

begin

context

fixes *sc*

assumes *sc*: $sc\ (a\text{-}sym,\ Suc\ (Suc\ 0))\ 0 = (1 :: nat)$

$sc\ (a\text{-}sym,\ Suc\ (Suc\ 0))\ (Suc\ 0) = 1$

begin

lemma *count-vars-term-encode-num-nat*:

$count\ (vars\text{-}term\text{-}ms\ (scf\text{-}term\ sc\ (encode\text{-}num\ x\ (int\ n))))\ x = n$

unfolding *encode-num-def nat-int*

by (*induct n, auto simp add: scf-list-def sc*)

lemma *count-vars-term-encode-num*:

$c \geq 0 \implies int\ (count\ (vars\text{-}term\text{-}ms\ (scf\text{-}term\ sc\ (encode\text{-}num\ x\ c)))\ x) = c$

using *count-vars-term-encode-num-nat[of x nat c]* **by** *auto*

lemma *count-vars-term-v-pow-e*:

$count\ (vars\text{-}term\text{-}ms\ (scf\text{-}term\ sc\ ((v\text{-}t\ x\ \widehat{\wedge}\ e)\ t)))\ y$

$= (sc\ (v\text{-}sym\ x, 1)\ 0)\ \widehat{\wedge}\ e * count\ (vars\text{-}term\text{-}ms\ (scf\text{-}term\ sc\ t))\ y$

proof (*induct e*)

case (*Suc e*)

thus *?case*

by (*simp split: if-splits add: scf-list-def sum-mset-sum-list sum-list-replicate count-sum-list sc*

flip: mset-replicate)

qed *force*

lemma *count-vars-term-encode-monom*: **assumes** *c*: $c \geq 0$

shows $int\ (count\ (vars\text{-}term\text{-}ms\ (scf\text{-}term\ sc\ (encode\text{-}monom\ x\ m\ c)))\ x)$

$= insertion\ (\lambda\ v.\ int\ (sc\ (v\text{-}sym\ v, 1)\ 0))\ (monom\ m\ c)$

proof –

define *xes* **where** *xes* = *var-list m*

from *var-list[of m c]*

have *monom*: $monom\ m\ c = Const\ c * (\prod (x, e) \leftarrow xes.\ PVar\ x\ \widehat{\wedge}\ e)$ **unfolding** *xes-def* .

show *?thesis* **unfolding** *encode-monom-def monom xes-def[symmetric]*

proof (*induct xes*)

case *Nil*

show *?case* **by** (*simp add: count-vars-term-encode-num[OF c] insertion-Const sc*)

next

```

case (Cons xe xes)
obtain x e where xe: xe = (x,e) by force
show ?case
  by (simp add: xe count-vars-term-v-pow-e Cons
      insertion-Const insertion-mult insertion-power insertion-Var when-def)
qed
qed

```

Lemma 4.5

```

lemma count-vars-term-encode-poly-generic: assumes positive-poly r
shows int (count (vars-term-ms (scf-term sc (encode-poly x r))) x) =
  insertion (λ v. int (sc (v-sym v,1) 0)) r
proof –
  define mcs where mcs = monom-list r
  from monom-list[of r] have r: r = (∑ (m, c)← mcs. monom m c) unfolding
  mcs-def by auto
  have mcs: (m,c) ∈ set mcs ⇒ c ≥ 0 for m c
  using monom-list-coeff assms unfolding mcs-def positive-poly-def by auto
  show ?thesis unfolding encode-poly-def mcs-def[symmetric] unfolding r inser-
  tion-sum-list map-map o-def
  using mcs
  proof (induct mcs)
  case (Cons mc mcs)
  obtain m c where mc: mc = (m,c) by force
  from Cons(2) mc have c: c ≥ 0 by auto
  note monom = count-vars-term-encode-monom[OF this, of x m]
  show ?case
  apply (simp add: mc monom scf-list-def sc)
  apply (subst Cons(1))
  using Cons(2) by (auto simp: when-def)
qed simp
qed
end

```

Theorem 4.6

```

theorem kbo-sc-termination-R-imp-solution:
assumes kbo-with-sc-termination R
shows positive-poly-problem p q
proof –
  from assms[unfolded kbo-with-sc-termination-def] obtain w w0 sc least pr-strict
  pr-weak
  where
    admissible-kbo w w0 pr-strict pr-weak least sc
  and orient: kbo.kbo-orientation w w0 sc least pr-strict pr-weak R
  by blast
interpret admissible-kbo w w0 pr-strict pr-weak least sc by fact
define l where l i = args lhs-R ! i for i
define r where r i = args rhs-R ! i for i
define as :: nat list where as = [0,1,2,3]

```

```

have upt-as: [0..<length as] = as unfolding as-def by auto
have lhs: lhs-R = Fun f-sym (map l as) unfolding lhs-R-def l-def as-def by simp
have rhs: rhs-R = Fun f-sym (map r as) unfolding rhs-R-def r-def as-def by
simp
from orient[unfolded kbo-orientation-def R-def]
have fst (kbo lhs-R rhs-R) by auto
from this[unfolded kbo.simps[of lhs-R]]
have vars-term-ms (SCF rhs-R) ⊆# vars-term-ms (SCF lhs-R) by (auto split:
if-splits)
hence count (vars-term-ms (SCF rhs-R)) x ≤ count (vars-term-ms (SCF
lhs-R)) x for x
by (rule mset-subset-eq-count)
let ?f = (f-sym, length as)
{
  fix i
  assume i: i ∈ set as
  from i have vl: vars-term (l i) ⊆ {i} unfolding l-def lhs-R-def as-def y1-def
y2-def y3-def
  using vars-encode-poly[of i p] by auto
  from count-vars-term-different-var[of - l i sc] vl
  have count-l-diff: i ≠ j ⇒ count (vars-term-ms (SCF (l i))) j = 0 for j by
auto
  from i have vr: vars-term (r i) ⊆ {i} unfolding r-def rhs-R-def as-def y1-def
y2-def y3-def
  using vars-encode-poly[of i q] by auto
  from count-vars-term-different-var[of - r i sc] vr
  have count-r-diff: i ≠ j ⇒ count (vars-term-ms (SCF (r i))) j = 0 for j by
auto
  {
    fix x
    have count (vars-term-ms (SCF rhs-R)) x
      = sum-list (map (λ i. count (vars-term-ms (SCF (r i))) x) (scf-list (sc ?f)
as)) unfolding rhs
    apply (simp add: o-def)
    apply (unfold mset-map[symmetric] sum-mset-sum-list)
    apply (unfold count-sum-list map-map o-def)
    by simp
    also have ... = ( $\sum i \leftarrow as. sc \ ?f \ i * count (vars-term-ms (SCF (r (as \ i))))$ )
x)
    unfolding sum-list-scf-list-prod upt-as ..
    finally have count (vars-term-ms (SCF rhs-R)) x = ( $\sum i \leftarrow as. sc \ ?f \ i * count$ 
(vars-term-ms (SCF (r (as \ i)))) x) .
  } note count-rhs = this
  {
    fix x
    have count (vars-term-ms (SCF lhs-R)) x
      = sum-list (map (λ i. count (vars-term-ms (SCF (l i))) x) (scf-list (sc ?f)
as)) unfolding lhs
    apply (simp add: o-def)
  }

```

```

apply (unfold mset-map[symmetric] sum-mset-sum-list)
apply (unfold count-sum-list map-map o-def)
by simp
also have ... = ( $\sum i \leftarrow as. sc \ ?f \ i * count (vars-term-ms (SCF (l (as ! i))))$ )
x)
  unfolding sum-list-scf-list-prod upt-as ..
  finally have count (vars-term-ms (SCF lhs-R)) x = ( $\sum i \leftarrow as. sc \ ?f \ i * count$ 
(vars-term-ms (SCF (l (as ! i)))) x) .
  } note count-lhs = this
  note count-lhs count-rhs count-l-diff count-r-diff
  } note cf = this[unfolded as-def]
  let ?f = (f-sym, Suc (Suc (Suc (Suc 0))))

{
  fix i :: nat
  assume i: i ∈ {0,1,2,3}
  have sc ?f i * count (vars-term-ms (SCF (r i))) i = count (vars-term-ms (SCF
rhs-R)) i
  by (subst cf(2), insert i, auto simp add: cf)
  also have ... ≤ count (vars-term-ms (SCF lhs-R)) i by fact
  also have ... = sc ?f i * count (vars-term-ms (SCF (l i))) i
  by (subst cf(1), insert i, auto simp add: cf)
  finally have count (vars-term-ms (SCF (r i))) i ≤ count (vars-term-ms (SCF
(l i))) i
  using scf[of i Suc (Suc (Suc 0)) f-sym] i by auto
  } note count-le = this

from count-le[of 0, unfolded r-def l-def rhs-R-def lhs-R-def y1-def]
have sc (a-sym, Suc (Suc 0)) 0 ≤ 1
  apply simp
  apply (unfold mset-map[symmetric] sum-mset-sum-list)
  by (simp add: count-sum-list sum-list-scf-list-prod)
with scf[of 0 Suc (Suc 0) a-sym]
have a20: sc (a-sym, Suc (Suc 0)) 0 = 1 by auto

from count-le[of 1, unfolded r-def l-def rhs-R-def lhs-R-def y2-def]
have sc (a-sym, Suc (Suc 0)) 1 ≤ 1
  apply simp
  apply (unfold mset-map[symmetric] sum-mset-sum-list)
  by (simp add: count-sum-list sum-list-scf-list-prod)
with scf[of 1 Suc (Suc 0) a-sym]
have a21: sc (a-sym, Suc (Suc 0)) (Suc 0) = 1 by auto

note encode = count-vars-term-encode-poly-generic[of sc, OF a20 a21]

have Suc (count (vars-term-ms (SCF (encode-poly y3 q))) y3) = count (vars-term-ms
(SCF (r 2))) 2
  by (simp add: r-def rhs-R-def scf-list-def a20 a21 y3-def)
  also have ... ≤ count (vars-term-ms (SCF (l 2))) 2 using count-le[of 2] by

```

```

simp
  also have ... = Suc (count (vars-term-ms (SCF (encode-poly y3 p))) y3)
    by (simp add: l-def lhs-R-def scf-list-def a20 a21 y3-def)
  finally have int (count (vars-term-ms (SCF (encode-poly y3 q))) y3) ≤ int
    (count (vars-term-ms (SCF (encode-poly y3 p))) y3)
    by auto
  from this[unfolded encode[OF pq(1)] encode[OF pq(2)]]
  show ?thesis
    unfolding positive-poly-problem-def[OF pq]
    by (intro exI[of - λv. int (sc (v-sym v, 1) 0)], auto simp: positive-interpr-def
scf)
qed
end

```

```

context solvable-poly-problem
begin

```

```

definition w0 :: nat where w0 = 1

```

```

fun sc :: symbol × nat ⇒ nat ⇒ nat where
  sc (v-sym i, Suc 0) - = nat (α i)
| sc - - = 1

```

```

context fixes wr :: nat
begin

```

```

fun w-R :: symbol × nat ⇒ nat where
  w-R (f-sym, n) = (if n = 4 then 0 else 1)
| w-R (a-sym, n) = (if n = 2 then 0 else 1)
| w-R (o-sym, 0) = wr
| w-R - = 1
end

```

```

definition w-rhs where w-rhs = weight-fun.weight (w-R 1) w0 sc rhs-R

```

```

abbreviation w where w ≡ w-R w-rhs

```

```

definition least where least f = (w (f, 0) = w0 ∧ (∀ g. w (g, 0) = w0 → (g,
0 :: nat) = (f, 0)))

```

```

lemma α0: α x > 0 using α(1) unfolding positive-interpr-def by auto

```

```

sublocale admissible-kbo w w0 (λ - -. False) (=) least sc
  apply (unfold-locales)
  subgoal for f unfolding w0-def
    by (cases f, auto simp add: weight-fun.weight.simps w-rhs-def rhs-R-def)
  subgoal by (simp add: w0-def)
  subgoal for f g n by (cases f, auto)
  subgoal for f unfolding least-def by auto
  subgoal for i n f by (cases f; cases n; cases n - 1; auto intro: α0)

```

by *auto*

lemma *insertion-pos*: *positive-poly* $r \implies \text{insertion } \alpha r \geq 0$
unfolding *positive-poly-def* **by** (*smt* (*verit*) $\alpha 0$ *insertion-nonneg*)

lemma *count-vars-term-encode-poly*: **assumes** *positive-poly* r
shows $\text{count } (\text{vars-term-ms } (\text{SCF } (\text{encode-poly } x r))) y = (\text{nat } (\text{insertion } \alpha r))$
when $x = y$
proof (*cases* $y = x$)
case *False*
with *count-vars-term-different-var*[*of* y *encode-poly* $x r$ *sc*] *vars-encode-poly*[*of* x r]
show *?thesis* **by** (*auto simp: when-def*)
next
case y : *True*
from *count-vars-term-encode-poly-generic*[*of* $sc - x$, *OF - - assms*]
have $\text{int } (\text{count } (\text{vars-term-ms } (\text{SCF } (\text{encode-poly } x r))) x)$
 $= \text{insertion } (\lambda v. \text{int } (\text{sc } (\text{v-sym } v, 1) 0)) r$ **by** *auto*
also have $(\lambda v. \text{int } (\text{sc } (\text{v-sym } v, 1) 0)) = \alpha$
by (*intro ext, insert* $\alpha 0$, *auto simp: order.order-iff-strict*)
finally show *?thesis* **unfolding** y
using *insertion-pos*[*OF assms*] **by** *auto*
qed

Theorem 4.7 in context

theorem *kbo-with-sc-termination*: *kbo-with-sc-termination* R
unfolding *kbo-with-sc-termination-def*
proof (*intro exI conjI*)
show *admissible-kbo* $w w0$ ($\lambda - -. \text{False}$) (=) *least sc ..*
show *kbo-orientation* R **unfolding** *R-def kbo-orientation-def*
proof (*clarify*)
 $\{$
fix $t :: (\text{symbol}, \text{var}) \text{term}$
assume $(o\text{-sym}, 0) \notin \text{funas-term } t$
hence $\text{weight-fun.weight } (w\text{-R } (\text{Suc } 0)) w0 \text{ sc } t = \text{weight } t$ (**is** *?id* t)
proof (*induct* t)
case (*Var* x)
show *?case* **by** (*auto simp: weight-fun.weight.simps*)
next
case (*Fun* f ts)
hence $t \in \text{set } ts \implies \text{?id } t$ **for** t **by** *auto*
hence *IH*: $\text{map2 } (\lambda ti i. \text{weight-fun.weight } (w\text{-R } (\text{Suc } 0)) w0 \text{ sc } ti * \text{sc } (f,$
length $ts) i) ts$
 $[0..<\text{length } ts] =$
 $\text{map2 } (\lambda ti i. \text{weight } ti * \text{sc } (f, \text{length } ts) i) ts [0..<\text{length } ts]$
by (*intro nth-equalityI, auto*)
have $\text{id}: w\text{-R } (\text{Suc } 0) (f, \text{length } ts) = w (f, \text{length } ts)$
using *Fun(2)* **by** (*cases* f ; *cases* ts , *auto*)
show *?case* **by** (*auto simp: id weight-fun.weight.simps Let-def IH*)
 $\}$


```

    qed
  } note weight-switch = this

from funas-encode-poly-q[of y3]
have o-q: (o-sym,0)  $\notin$  funas-term (encode-poly y3 q) by (auto simp: F-def)
have weight rhs-R = 3 + 3 * w0 + weight (encode-poly y3 q)
  unfolding rhs-R-def by (simp add: scf-list-def)
also have ... = w-rhs unfolding weight-switch[OF o-q, symmetric]
  unfolding w-rhs-def rhs-R-def by (simp add: weight-fun.weight.simps)
also have ... < w0 + w-rhs using w0 by auto
also have ...  $\leq$  weight lhs-R unfolding lhs-R-def
  by (simp add: scf-list-def)
finally have weight: weight rhs-R < weight lhs-R .
from  $\alpha(2)$  insertion-pos[OF pq(1)] insertion-pos[OF pq(2)]
have sol: nat (insertion  $\alpha$  q)  $\leq$  nat (insertion  $\alpha$  p) by auto
have vars: vars-term-ms (SCF rhs-R)  $\subseteq\#$  vars-term-ms (SCF lhs-R)
proof (intro mset-subset-eqI)
  fix x
  show count (vars-term-ms (SCF rhs-R)) x  $\leq$  count (vars-term-ms (SCF
lhs-R)) x
  unfolding rhs-R-def lhs-R-def using y-vars sol
  by (simp add: scf-list-def count-vars-term-encode-poly[OF pq(1)] count-vars-term-encode-poly[OF
pq(2)])
qed
from weight vars show fst (kbo lhs-R rhs-R)
  unfolding kbo.simps[of lhs-R rhs-R] by auto
qed
qed
end

Theorem 4.7 outside solvable-context
context poly-input
begin
theorem solvable-imp-kbo-with-sc-termination:
  assumes positive-poly-problem p q
  shows kbo-with-sc-termination R
  by (rule solvable-poly-problem.kbo-with-sc-termination, unfold-locales, fact)
end

```

Theorem 4.7 outside solvable-context

```

context poly-input
begin
theorem solvable-imp-kbo-with-sc-termination:
  assumes positive-poly-problem p q
  shows kbo-with-sc-termination R
  by (rule solvable-poly-problem.kbo-with-sc-termination, unfold-locales, fact)
end

```

Combining 4.6 and 4.7

```

corollary solvable-iff-kbo-with-sc-termination:
  positive-poly-problem p q  $\longleftrightarrow$  kbo-with-sc-termination R
  using solvable-imp-kbo-with-sc-termination kbo-sc-termination-R-imp-solution by
blast
end
end

```

7 Undecidability of Polynomial Termination over Integers

```

theory Poly-Termination-Undecidable
  imports
    Linear-Poly-Termination-Undecidable
    Preliminaries-on-Polynomials-2
begin

context poly-input
begin

definition  $y_4 :: \text{var}$  where  $y_4 = 3$ 
definition  $y_5 :: \text{var}$  where  $y_5 = 4$ 
definition  $y_6 :: \text{var}$  where  $y_6 = 5$ 
definition  $y_7 :: \text{var}$  where  $y_7 = 6$ 

abbreviation  $q\text{-}t$  where  $q\text{-}t\ t \equiv \text{Fun } q\text{-}sym\ [t]$ 
abbreviation  $h\text{-}t$  where  $h\text{-}t\ t \equiv \text{Fun } h\text{-}sym\ [t]$ 
abbreviation  $g\text{-}t$  where  $g\text{-}t\ t1\ t2 \equiv \text{Fun } g\text{-}sym\ [t1, t2]$ 

```

Definition 5.1

```

definition  $lhs\text{-}S = \text{Fun } f\text{-}sym\ [$ 
   $\text{Var } y1,$ 
   $\text{Var } y2,$ 
   $a\text{-}t\ (\text{encode-poly } y3\ p)\ (\text{Var } y3),$ 
   $q\text{-}t\ (h\text{-}t\ (\text{Var } y4)),$ 
   $h\text{-}t\ (\text{Var } y5),$ 
   $h\text{-}t\ (\text{Var } y6),$ 
   $g\text{-}t\ (\text{Var } y7)\ o\text{-}t]$ 

definition  $rhs\text{-}S = \text{Fun } f\text{-}sym\ [$ 
   $a\text{-}t\ (\text{Var } y1)\ z\text{-}t,$ 
   $a\text{-}t\ z\text{-}t\ (\text{Var } y2),$ 
   $a\text{-}t\ (\text{encode-poly } y3\ q)\ (\text{Var } y3),$ 
   $h\text{-}t\ (h\text{-}t\ (q\text{-}t\ (\text{Var } y4))),$ 
   $\text{foldr } v\text{-}t\ V\text{-}list\ (a\text{-}t\ (\text{Var } y5)\ (\text{Var } y5)),$ 
   $\text{Fun } f\text{-}sym\ (\text{replicate } 7\ (\text{Var } y6)),$ 
   $g\text{-}t\ (\text{Var } y7)\ z\text{-}t]$ 

definition  $S$  where  $S = \{(lhs\text{-}S, rhs\text{-}S)\}$ 

definition  $F\text{-}S$  where  $F\text{-}S = \{(f\text{-}sym, 7), (h\text{-}sym, 1), (g\text{-}sym, 2), (o\text{-}sym, 0), (q\text{-}sym, 1)\}$ 
 $\cup F$ 

lemma  $lhs\text{-}S\text{-}F$ :  $\text{funas-term } lhs\text{-}S \subseteq F\text{-}S$ 
proof –
  from  $\text{funas-encode-poly-p}$ 
  show  $\text{funas-term } lhs\text{-}S \subseteq F\text{-}S$  unfolding  $lhs\text{-}S\text{-}def$  by ( $\text{auto simp: } F\text{-}S\text{-}def\ F\text{-}def$ )

```

qed

lemma *funas-fold-vs[simp]*: *funas-term* (foldr *v-t* *V-list* *t*) = (λ *i*. (*v-sym* *i*,1)) ‘ *V*
 \cup *funas-term* *t*

proof –

have *id*: *funas-term* (foldr *v-t* *xs* *t*) = (λ *i*. (*v-sym* *i*,1)) ‘ *set* *xs* \cup *funas-term* *t*
for *xs*

by (*induct* *xs*, *auto*)

show ?*thesis* **unfolding** *id*

by (*auto simp: V-list*)

qed

lemma *vars-fold-vs[simp]*: *vars-term* (foldr *v-t* *vs* *t*) = *vars-term* *t*

by (*induct* *vs*, *auto*)

lemma *funas-term-r5*: *funas-term* (foldr *v-t* *V-list* (*a-t* (*Var* *y5*) (*Var* *y5*))) \subseteq *F-S*

by (*auto simp: F-S-def F-def*)

lemma *rhs-S-F*: *funas-term* *rhs-S* \subseteq *F-S*

proof –

from *funas-encode-poly-q funas-term-r5*

show *funas-term* *rhs-S* \subseteq *F-S* **unfolding** *rhs-S-def* **by** (*auto simp: F-S-def F-def*)

qed

end

lemma *poly-inter-eval-cong*: **assumes** \bigwedge *f a*. (*f*,*a*) \in *funas-term* *t* \implies *I f* = *I' f*

shows *poly-inter.eval* *I* *t* = *poly-inter.eval* *I'* *t*

using *assms*

proof (*induct* *t*)

case (*Var* *x*)

show ?*case* **by** (*simp add: poly-inter.eval.simps*)

next

case (*Fun* *f* *ts*)

 {

fix *i*

assume *i* < *length* *ts*

hence *ts* ! *i* \in *set* *ts*

by *auto*

with *Fun*(1)[*OF* *this* *Fun*(2)]

have *poly-inter.eval* *I* (*ts* ! *i*) = *poly-inter.eval* *I'* (*ts* ! *i*) **by** *force*

 } **note** *IH* = *this*

from *Fun*(2) **have** *I f* = *I' f* **by** *auto*

thus ?*case* **using** *IH*

by (*auto simp: poly-inter.eval.simps insertion-substitute intro!: mpolynomial-irrelevant-vars*)

qed

The easy direction of Theorem 5.4

context *solvable-poly-problem*
begin

definition *c-S* **where** $c-S = \max \gamma (2 * \text{prod-list} (\text{map } \alpha V\text{-list}))$

lemma *c-S*: $c-S > 0$ **unfolding** *c-S-def* **by** *auto*

fun *I-S* :: *symbol* \Rightarrow *int mpoly* **where**

I-S f-sym = *PVar 0* + *PVar 1* + *PVar 2* + *PVar 3* + *PVar 4* + *PVar 5* +
PVar 6
| *I-S a-sym* = *PVar 0* + *PVar 1*
| *I-S z-sym* = 0
| *I-S o-sym* = 1
| *I-S (v-sym i)* = *Const* (αi) * *PVar 0*
| *I-S q-sym* = *mmonom* (*monomial 2 0*) *c-S* — $c * (PVar 0)^2$
| *I-S g-sym* = *PVar 0* + *PVar 1*
| *I-S h-sym* = *mmonom* (*monomial 1 0*) *c-S* — $c * PVar 0$

declare *single-numeral*[*simp del*]

declare *insertion-monom*[*simp del*]

interpretation *inter-S*: *poly-inter F-S I-S* ($>$) .

lemma *inter-S-encode-poly*: **assumes** *positive-poly r*

shows $\text{inter-S.eval} (\text{encode-poly } x r) = \text{Const} (\text{insertion } \alpha r) * PVar x$
by (*rule inter-encode-poly-generic*[*OF - - - assms*], *auto*)

lemma *valid-monotone-inter-S*: *inter-S.valid-monotone-poly-inter*

unfolding *inter-S.valid-monotone-poly-inter-def*

proof (*intro ballI*)

fix *fn*

assume $f: fn \in F-S$

show *inter-S.valid-monotone-poly fn*

proof (*cases fn* $\in F$)

case *True*

show *inter-S.valid-monotone-poly fn*

by (*rule valid-monotone-inter-F*[*OF - - - $\alpha(1)$ True*], *auto*)

next

case *False*

with *f* **have** $f: fn \in F-S - F$ **by** *auto*

have [*simp*]: $\text{vars} ((PVar 0 :: \text{int mpoly}) + PVar (Suc 0) + PVar 2 + PVar 3$
 $+ PVar 4 + PVar 5 + PVar 6) = \{0,1,2,3,4,5,6\}$

unfolding *vars-def* **apply** (*transfer'*, *simp add: Var₀-def image-comp*) **by**
code-simp

have [*simp*]: $\text{vars} ((PVar 0 :: \text{int mpoly}) + PVar (Suc 0)) = \{0,1\}$

unfolding *vars-def* **apply** (*transfer'*, *simp add: Var₀-def image-comp*) **by**
code-simp

show *?thesis* **unfolding** *inter-S.valid-monotone-poly-def* **using** *f*

proof (*intro ballI impI allI, clarify, intro conjI*)

```

fix f n
assume f: (f,n) ∈ F-S (f,n) ∉ F
from f show vars (I-S f) = {..<n} unfolding F-S-def using c-S
  by (auto simp: vars-monom-single-cases)
from f c-S show valid-poly (I-S f) unfolding F-S-def
  by (auto simp: valid-poly-def insertion-add assignment-def)
have x2: x < 2 ⇒ x = 0 ∨ x = Suc 0 for x by linarith
have x7: x < 7 ⇒ x = 0 ∨ x = Suc 0 ∨ x = 2 ∨ x = 3 ∨ x = 4 ∨ x = 5
∨ x = 6 for x by linarith
from f c-S show inter-S.monotone-poly {..<n} (I-S f) unfolding F-S-def
by (auto simp: monotone-poly-wrt-def insertion-add assignment-def power-strict-mono
dest: x2 x7)
qed
qed
qed

```

```

interpretation inter-S: int-poly-inter F-S I-S
proof
show inter-S.valid-monotone-poly-inter by (rule valid-monotone-inter-S)
qed

```

```

lemma orient-trs: inter-S.termination-by-poly-interpretation S
unfolding inter-S.termination-by-poly-interpretation-def
inter-S.termination-by-interpretation-def S-def inter-S.orient-rule
proof (clarify, intro conjI)
have lhs-S: inter-S.eval lhs-S =
(PVar y1 +
PVar y2 +
(Const (insertion α p) + 1) * PVar y3 +
(Const c-S)^3 * (PVar y4)^2 +
Const c-S * PVar y5 +
Const c-S * PVar y6 +
PVar y7) +
1
unfolding lhs-S-def by (simp add: inter-S.encode-poly[OF pq(1)]
power2-eq-square power3-eq-cube algebra-simps)
have foldr: inter-S.eval (foldr (λi t. Fun (v-sym i) [t]) V-list (Fun a-sym [TVar
y5, TVar y5])) =
Const (prod-list (map α V-list)) * 2 * PVar y5
by (subst inter-foldr-v-t, auto)
have rhs-S: inter-S.eval rhs-S =
(PVar y1 +
PVar y2 +
(Const (insertion α q) + 1) * PVar y3 +
(Const c-S)^3 * (PVar y4)^2 +
Const (prod-list (map α V-list)) * 2 * PVar y5 +
7 * PVar y6 +
PVar y7) +
0

```

```

unfolding rhs-S-def by (simp add: inter-S-encode-poly[OF pq(2)] Const-add
  power2-eq-square power3-eq-cube algebra-simps foldr)
show inter-S.gt-poly (inter-S.eval lhs-S) (inter-S.eval rhs-S)
unfolding inter-S.gt-poly-def
proof (intro allI impI)
  fix  $\beta :: \text{var} \Rightarrow \text{int}$ 
  assume ass: assignment  $\beta$ 
  hence  $\beta: \bigwedge x. \beta x \geq 0$  unfolding assignment-def by auto
  have  $\alpha 0: \alpha x \geq 0$  for  $x$  using  $\alpha(1)$ [unfolded positive-interpret-def, rule-format,
of  $x$ ] by auto
  from c-S have c0:  $c-S \geq 0$  by simp
  have  $\gamma: \gamma = (\text{Const } \gamma :: \text{int mpoly})$  by code-simp
  have  $2: 2 = (\text{Const } 2 :: \text{int mpoly})$  by code-simp
  have ins7: insertion  $\beta \gamma = (\gamma :: \text{int})$  unfolding  $\gamma$  insertion-Const by simp
  have ins2: insertion  $\beta 2 = (2 :: \text{int})$  unfolding  $2$  insertion-Const by simp
  show insertion  $\beta$  (inter-S.eval lhs-S) > insertion  $\beta$  (inter-S.eval rhs-S)
    unfolding lhs-S rhs-S insertion-add ins7 ins2 insertion-mult insertion-Var
insertion-Const insertion-Const insertion-power
  proof (intro add-le-less-mono add-mono mult-mono add-nonneg-nonneg zero-le-power
 $\alpha(2) \beta c0$ )
    show  $0 \leq \text{insertion } \alpha p$  by (intro insertion-positive-poly[OF  $\alpha 0 pq(1)$ ])
    show  $\gamma \leq c-S$  unfolding c-S-def by auto
    show prod-list (map  $\alpha V\text{-list}$ ) *  $2 \leq c-S$  unfolding c-S-def by simp
  qed (force+)
qed
qed (insert lhs-S-F rhs-S-F, auto)

```

lemma solution-imp-poly-termination: termination-by-int-poly-interpretation $F-S$
 S

```

unfolding termination-by-int-poly-interpretation-def
by (intro exI, rule conjI[OF - orient-trs], unfold-locales)

```

end

Towards Lemma 5.2

lemma (in int-poly-inter) monotone-imp-weakly-monotone: **assumes** monotone-poly
 $xs p$

```

shows weakly-monotone-poly  $xs p$ 
unfolding monotone-poly-wrt-def

```

proof (intro allI impI)

```

fix  $\alpha :: \text{var} \Rightarrow \text{int}$  and  $x v$ 

```

```

assume assignment  $\alpha x \in xs \alpha x \leq v$ 

```

from *assms*[unfolded monotone-poly-wrt-def, rule-format, OF *this(1-2)*, of v]
this(3)

```

show insertion  $\alpha p \leq \text{insertion } (\alpha(x := v)) p$ 

```

```

by (cases  $\alpha x < v$ , auto)

```

qed

context

```

fixes gt :: 'a :: linordered-idom ⇒ 'a ⇒ bool
assumes trans-gt: transp gt
and gt-imp-ge:  $\bigwedge x y. gt\ x\ y \implies x \geq y$ 
begin

lemma monotone-poly-wrt-insertion-main: assumes monotone-poly-wrt gt xs p
and a: assignment (a :: var ⇒ 'a :: linordered-idom)
and b:  $\bigwedge x. x \in xs \implies gt^{==} (b\ x) (a\ x)$ 
 $\bigwedge x. x \notin xs \implies a\ x = b\ x$ 
shows  $gt^{==} (insertion\ b\ p) (insertion\ a\ p)$ 
proof –
from sorted-list-of-set(1)[OF vars-finite[of p]] sorted-list-of-set[of vars p] obtain
ys where
ysp: set ys = vars p and dist: distinct ys by auto
define c where c ys = ( $\lambda x. if\ x \in set\ ys\ then\ a\ x\ else\ b\ x$ ) for ys
have ass: assignment (c ys) for ys unfolding assignment-def
proof
fix x
show  $0 \leq c\ ys\ x$  using b[of x] a[unfolded assignment-def, rule-format, of x]
gt-imp-ge[of b x a x]
unfolding c-def by auto linarith
qed
have id:  $insertion\ a\ p = insertion\ (c\ ys)\ p$  unfolding c-def ysp
by (rule insertion-irrelevant-vars, auto)
also have  $gt^{\wedge} == (insertion\ b\ p) (insertion\ (c\ ys)\ p)$  using dist
proof (induct ys)
case Nil
show ?case unfolding c-def by auto
next
case (Cons x ys)
show ?case
proof (cases x ∈ xs)
case False
from b(2)[OF this] have  $c\ (Cons\ x\ ys) = c\ ys$ 
unfolding c-def by auto
thus ?thesis using Cons by auto
next
case True
from b(1)[OF this] have  $ab: gt^{\wedge} == (b\ x) (a\ x)$  by auto
let ?c =  $c\ (Cons\ x\ ys)$ 
have id1:  $c\ ys = ?c(x := b\ x)$ 
using Cons(2) unfolding c-def by auto
have id2:  $c\ (x \# ys)\ x = a\ x$  using True unfolding c-def by auto
have IH:  $gt^{\wedge} == (insertion\ b\ p) (insertion\ (c\ ys)\ p)$  using Cons by auto
have  $gt^{\wedge} == (insertion\ (?c(x := b\ x))\ p) (insertion\ ?c\ p)$ 
proof (cases b x = a x)
case True
hence  $?c(x := b\ x) = ?c$  using id1 id2
by (intro ext, auto)

```

```

      thus ?thesis by simp
    next
      case False
      with ab have ab: gt (b x) (a x) by auto
      have gt(insertion (?c(x := b x)) p) (insertion ?c p)
      proof (rule assms(1)[unfolded monotone-poly-wrt-def, rule-format, OF ass
True])
        show gt (b x) (c (x # ys) x) unfolding id2 by fact
      qed
      thus ?thesis by auto
    qed
  also have insertion (?c(x := b x)) p = insertion (c ys) p unfolding id1 ..
  finally have gt^== (insertion (c ys) p) (insertion (c (x # ys)) p) .
  from transpE[OF trans-gt] IH this
  show ?thesis by auto
qed
qed
finally show ?thesis .
qed

```

```

lemma monotone-poly-wrt-insertion: assumes monotone-poly-wrt gt (vars p) p
  and a: assignment (a :: var ⇒ 'a :: linordered-idom)
  and b:  $\bigwedge x. x \in \text{vars } p \implies \text{gt}^{\text{==}} (b x) (a x)$ 
shows  $\text{gt}^{\text{==}} (\text{insertion } b p) (\text{insertion } a p)$ 
proof -
  define b' where b' x = (if x ∈ vars p then b x else a x) for x
  have gt^== (insertion b' p) (insertion a p)
  by (rule monotone-poly-wrt-insertion-main[OF assms(1-2)], insert b, auto
simp: b'-def)
  also have insertion b' p = insertion b p
  by (rule insertion-irrelevant-vars, auto simp: b'-def)
  finally show ?thesis .
qed

```

```

lemma partial-insertion-mono-wrt: assumes mono: monotone-poly-wrt gt (vars
p) p
  and a: assignment a
  and b:  $\bigwedge y. y \neq x \implies \text{gt}^{\text{==}} (b y) (a y)$ 
  and d:  $\bigwedge y. y \geq d \implies \text{gt}^{\text{==}} y 0$ 
shows  $\exists c. \forall y. y \geq d \implies c \leq \text{poly} (\text{partial-insertion } a x p) y$ 
 $\wedge \text{poly} (\text{partial-insertion } a x p) y \leq \text{poly} (\text{partial-insertion } b x p) y$ 
proof -
  define pa where pa = partial-insertion a x p
  define pb where pb = partial-insertion b x p
  define c where c = insertion (a(x := 0)) p
  {
    fix y :: 'a
    assume y: y ≥ d
    with d have gty: gt== y 0 by auto
  }

```



```

from  $a$  have  $ass: assignment (a(x := 0))$  unfolding  $assignment-def$  by  $auto$ 
from  $monotone-poly-wrt-insertion[OF mono ass, of a(x := y)]$ 
have  $gt^{==} (insertion (a(x := y)) p) (insertion (a(x := 0)) p)$  using  $gty$  by
 $auto$ 
from  $this[folded c-def] gt-imp-ge[of - c]$ 
have  $c \leq insertion (a(x := y)) p$  by  $auto$ 
} note  $le-c = this$ 
{
  fix  $y :: 'a$ 
  assume  $y: y \geq d$ 
  with  $d$  have  $gty: gt^{==} y 0$  by  $auto$ 
  from  $y a gty gt-imp-ge[of y]$  have  $ass: assignment (a(x := y))$  unfolding
 $assignment-def$  by  $auto$ 
  from  $monotone-poly-wrt-insertion[OF mono this, of b(x := y)]$ 
  have  $gt^{==} (insertion (b(x := y)) p) (insertion (a(x := y)) p)$ 
  using  $b$  by  $auto$ 
  with  $gt-imp-ge$ 
  have  $insertion (a(x := y)) p \leq insertion (b(x := y)) p$  by  $auto$ 
} note  $le-ab = this$ 
have  $id: poly (partial-insertion a x p) y = insertion (a(x := y)) p$  for  $a y$ 
using  $insertion-partial-insertion[of x a a(x := y) p]$  by  $auto$ 
{
  fix  $y :: 'a$ 
  assume  $y: y \geq d$ 
  from  $le-ab[OF y, folded id, folded pa-def pb-def]$ 
  have  $poly pa y \leq poly pb y$  by  $auto$ 
} note  $le1 = this$ 
show  $?thesis$ 
proof ( $intro exI[of - c], intro allI impI conjI le1[unfolded pa-def pb-def]$ )
  fix  $y :: 'a$ 
  assume  $y: y \geq d$ 
  show  $c \leq poly (partial-insertion a x p) y$  using  $le-c[OF y]$  unfolding  $id$  .
qed
qed

```

context

assumes $poly-pinfty-ge: \bigwedge p b. 0 < lead-coeff (p :: 'a poly) \implies degree p \neq 0$
 $\implies \exists n. \forall x \geq n. b \leq poly p x$

begin

context

fixes $p d$

assumes $mono: monotone-poly-wrt gt (vars p) p$

and $d: \bigwedge y. y \geq d \implies gt^{==} y 0$

begin

lemma $degree-partial-insertion-mono-generic: assumes$

$a: assignment a$

and $b: \bigwedge y. y \neq x \implies gt^{==} (b y) (a y)$

shows $\text{degree}(\text{partial-insertion } a \ x \ p) \leq \text{degree}(\text{partial-insertion } b \ x \ p)$
proof –
define qa **where** $qa = \text{partial-insertion } a \ x \ p$
define qb **where** $qb = \text{partial-insertion } b \ x \ p$
from $\text{partial-insertion-mono-wrt}[OF \ \text{mono } a \ b \ d, \ \text{of } x \ d]$
obtain c **where** $c: \bigwedge y. y \geq d \implies c \leq \text{poly } qa \ y$
and $ab: \bigwedge y. y \geq d \implies \text{poly } qa \ y \leq \text{poly } qb \ y$
by $(\text{auto simp: } qa\text{-def } qb\text{-def})$
show $?thesis$
proof $(\text{cases } \text{degree } qa = 0)$
case $True$
thus $?thesis$ **unfolding** $qa\text{-def}$ **by** auto
next
case $False$
let $?lc = \text{lead-coeff}$
have $lc\text{-pos}: ?lc \ qa > 0$
proof $(\text{rule } c\text{contr})$
assume $\neg ?thesis$
with $False$ **have** $?lc \ qa < 0$ **using** $\text{leading-coeff-neq-0}$ **by** force
hence $?lc \ (-qa) > 0$ **by** simp
from $\text{poly-pinfty-ge}[OF \ \text{this}, \ \text{of } -c + 2]$ $False$
obtain n **where** $le: \bigwedge x. x \geq n \implies -c + 2 \leq -\text{poly } qa \ x$ **by** auto
from $le[\text{of } \text{max } n \ d]$ $c[\text{of } \text{max } n \ d]$ **show** $False$ **by** auto
qed
from $\text{this } ab$ **have** $\text{degree } qa \leq \text{degree } qb$ **by** $(\text{intro } \text{degree-mono-generic}[OF \ \text{poly-pinfty-ge}], \ \text{auto})$
thus $?thesis$ **unfolding** $qa\text{-def } qb\text{-def}$ **by** auto
qed
qed

lemma $\text{degree-partial-insertion-stays-constant-generic}$:
 $\exists a. \text{assignment } a \wedge$
 $(\forall b. (\forall y. \text{gt}^{\text{==}}(b \ y) \ (a \ y)) \implies \text{degree}(\text{partial-insertion } a \ x \ p) = \text{degree}(\text{partial-insertion } b \ x \ p))$
proof –
define n **where** $n = \text{mdegree } p \ x$
define pi **where** $pi \ a = \text{partial-insertion } a \ x \ p$ **for** a
have $n: \text{assignment } a \implies \text{degree}(\text{pi } a) \leq n$ **for** a **unfolding** $n\text{-def } pi\text{-def}$
by $(\text{rule } \text{degree-partial-insertion-bound})$
thus $?thesis$ **unfolding** $pi\text{-def}[\text{symmetric}]$
proof $(\text{induct } n \ \text{rule: } \text{less-induct})$
case $(\text{less } n)$
show $?case$
proof $(\text{cases } \exists a. \text{assignment } a \wedge \text{degree}(\text{pi } a) = n)$
case $True$
then obtain a **where** $a: \text{assignment } a$ **and** $\text{deg}: \text{degree}(\text{pi } a) = n$ **by** auto
show $?thesis$
proof $(\text{intro } \text{exI}[\text{of } - \ a] \ \text{conjI } a \ \text{allI } \text{impI})$
fix b

```

assume ge:  $\forall y. gt^{==} (b y) (a y)$ 
with a gt-imp-ge[of b y a y for y] have b: assignment b unfolding assignment-def
  using order-trans[of 0 a y for y] by fastforce
  have degree (pi a)  $\leq$  degree (pi b)
  by (rule degree-partial-insertion-mono-generic[OF a, of x b, folded pi-def],
insert ge, auto)
  with less(2)[of b] deg b
  show degree (pi a) = degree (pi b) by simp
qed
next
  case False
  with less(2) have deg: assignment b  $\implies$  degree (pi b) < n for b by fastforce
  have ass: assignment ( $\lambda \cdot 0 :: 'a$ ) unfolding assignment-def by auto
  define m where m = n - 1
  from deg[OF ass] have mn: m < n and less-id: x < n  $\longleftrightarrow$  x  $\leq$  m for x
unfolding m-def by auto
  from less(1)[OF mn deg[unfolded less-id]] show ?thesis by auto
qed
qed
qed
end

```

lemma *monotone-poly-partial-insertion-generic*:

```

assumes delta-order:  $\bigwedge x y. gt y x \longleftrightarrow y \geq x + \delta$ 
  and delta:  $\delta > 0$ 
  and eps-delta:  $\varepsilon * \delta \geq 1$ 
  and ceil-nat:  $\bigwedge x :: 'a. of-nat (ceil-nat x) \geq x$ 
assumes x: x  $\in$  xs
and mono: monotone-poly-wrt gt xs p
and ass: assignment a
shows 0 < degree (partial-insertion a x p)
  lead-coeff (partial-insertion a x p) > 0
  valid-poly p  $\implies$  poly (partial-insertion a x p) ( $\delta * of-nat y$ )  $\geq$   $\delta * of-nat y$ 
proof -
  define q where q = partial-insertion a x p
  {
    fix w1 w2 :: 'a
    assume w: 0  $\leq$  w1 gt w2 w1
    from gt-imp-ge[OF w(2)] w have w2: w2  $\geq$  0 by auto
    have assw: assignment (a (x := w1)) using ass w(1) w2 unfolding assignment-def by auto
    note main = insertion-partial-insertion[of x - - p, symmetric]
    have gt (insertion (a(x := w2)) p) (insertion (a(x := w1)) p)
      using mono[unfolded monotone-poly-wrt-def, rule-format, OF assw x, of w2]
  by (auto simp: w)
    also have insertion (a(x := w2)) p = poly (partial-insertion a x p) w2 using
      main[of a (x := w2)] by auto
    also have insertion (a(x := w1)) p = poly (partial-insertion a x p) w1 using

```

```

main[of a a(x := w1)] by auto
  finally have gt (poly q w2) (poly q w1) by (auto simp: q-def)
} note gt = this
have 0 ≤ a x using ass unfolding assignment-def by auto
from gt[OF this, of a x + δ] have poly q (a x) ≠ poly q (a x + δ) unfolding
delta-order using delta by auto
hence deg: degree q > 0
  using degree0-coeffs[of q] by force
show 0 < degree (partial-insertion a x p) unfolding q-def[symmetric] by fact

have unbounded: poly q (δ * of-nat n) ≥ poly q 0 + δ * of-nat n for n
proof (induct n)
  case (Suc n)
  have poly q 0 + δ * of-nat (Suc n) = (poly q 0 + δ * of-nat n) + δ by (simp
add: algebra-simps)
  also have ... ≤ poly q (δ * of-nat n) + δ using Suc by simp
  also have ... ≤ poly q (δ * of-nat n + δ)
  by (rule gt[unfolded delta-order], insert delta, auto)
  finally show ?case by (simp add: algebra-simps)
qed force
let ?lc = lead-coeff
have ?lc q > 0
proof (rule ccontr)
  define d where d = poly q 0
  assume ¬ ?thesis
  hence ?lc q ≤ 0 by auto
  moreover have ?lc q ≠ 0 using deg by auto
  ultimately have ?lc q < 0 by auto
  hence ?lc (-q) > 0 by auto
  from poly-pinfty-ge[OF this, of -d] deg obtain n where le: ∧ x. x ≥ n ⇒
- d ≤ - poly q x by auto
  have d: x ≥ n ⇒ d ≥ poly q x for x using le[of x] by linarith
  define m where m = ε * (max n 0 + 1)
  from eps-delta delta have eps: ε > 0
  by (metis mult.commute order-less-le-trans zero-less-mult-pos zero-less-one)
  hence m: m > 0 unfolding m-def by auto
  from ceil-nat[of m] m have cm: ceil-nat m > 0
  using linorder-not-less by force
  have poly q (δ * of-nat (ceil-nat m)) ≤ d
  proof (rule d)
    have n ≤ max n 0 * 1 by simp
    also have ... ≤ max n 0 * (ε * δ) using eps-delta
    by (simp add: max-def)
    also have ... = δ * m - δ * ε unfolding m-def by (simp add: field-simps)
    also have ... ≤ δ * m using eps-delta by (auto simp: ac-simps)
    also have ... ≤ δ * of-nat (ceil-nat m)
    by (rule mult-left-mono[OF ceil-nat], insert delta, auto)
    finally show n ≤ δ * of-nat (ceil-nat m) .
  qed
qed

```

also have $\dots < \text{poly } q \ 0 + \delta * \text{of-nat } (\text{ceil-nat } m)$ **unfolding** $d\text{-def}$ **using**
delta cm by auto
also have $\dots \leq \text{poly } q (\delta * \text{of-nat } (\text{ceil-nat } m))$ **by** *(rule unbounded)*
finally show False **by** *simp*
qed
thus $\text{lead-coeff } q > 0$ **unfolding** $q\text{-def}$.

assume $\text{valid: valid-poly } p$
{
fix $y :: \text{nat}$
let $?y = \delta * \text{of-nat } y$
from $\text{unbounded}[of\ y]$
have $\text{poly } q\ ?y \geq \text{poly } q\ 0 + ?y$.
moreover have $\text{poly } q\ 0 = \text{insertion } (a(x := 0))\ p$ **unfolding** $q\text{-def}$
using $\text{insertion-partial-insertion}[of\ x\ a\ a(x := 0)\ p]$ **by** *auto*
moreover have $\dots \geq 0$
by *(intro valid[unfolded valid-poly-def, rule-format], insert ass, auto simp: assignment-def)*
ultimately have $\text{poly } q\ ?y \geq ?y$ **by** *auto*
thus $\text{poly } (\text{partial-insertion } a\ x\ p)\ ?y \geq ?y$ **unfolding** $q\text{-def}$.
} **note** $ge = \text{this}$
qed
end
end

context poly-inter
begin

lemma $\text{monotone-poly-eval-generic}$:
assumes $\text{valid: valid-monotone-poly-inter}$
and $\text{trans-gt: transp } (\succ)$
and $\text{gt-imp-ge: } \bigwedge x\ y. x \succ y \implies y \leq x$
and $\text{gt-exists: } \bigwedge x. x \geq 0 \implies \exists y. y \succ x$
and $\text{gt-irrefl: } \bigwedge x. \neg (x \succ x)$
and $tF: \text{funas-term } t \subseteq F$
shows $\text{monotone-poly } (\text{vars-term } t)\ (\text{eval } t)\ \text{vars } (\text{eval } t) = \text{vars-term } t$
proof –
have $\text{monotone-poly } (\text{vars-term } t)\ (\text{eval } t) \wedge \text{vars } (\text{eval } t) = \text{vars-term } t$ **using**
 tF
proof *(induct t)*
case $(\text{Var } x)$
show $?case$ **by** *(auto simp: monotone-poly-wrt-def)*
next
case $(\text{Fun } f\ ts)$
{
fix t
assume $t \in \text{set } ts$
with $\text{Fun}(1)[OF\ \text{this}]\ \text{Fun}(2)$
have $\text{monotone-poly } (\text{vars-term } t)\ (\text{eval } t)$

```

      vars (eval t) = vars-term t
    by auto
  } note IH = this
  let ?n = length ts
  let ?f = (f, ?n)
  define p where p = I f
  from Fun have ?f ∈ F by auto
  from valid[unfolded valid-monotone-poly-inter-def, rule-format, OF this, un-
folded valid-monotone-poly-def]
  have valid: valid-poly p and mono: monotone-poly (vars p) p and vars: vars p
= {..<?n}
  unfolding p-def by auto
  have wm: assignment b ⇒ (∧x. x ∈ vars p ⇒ (⋗)== (a x) (b x)) ⇒ (⋗)==
(insertion a p) (insertion b p)
  for b a using monotone-poly-wrt-insertion[OF trans-gt gt-imp-ge mono] by
auto
  have id: eval (Fun f ts) = substitute (λi. if i < length ts then eval (ts ! i) else
0) p
  unfolding eval.simps p-def[symmetric] id by simp

have mono: monotone-poly (vars-term (Fun f ts)) (eval (Fun f ts))
  unfolding monotone-poly-wrt-def
proof (intro allI impI)
  fix α :: - ⇒ 'a and x v
  assume α: assignment α
  and x: x ∈ vars-term (Fun f ts)
  and v: v ⋗ α x
  define β where β = α(x := v)
  define α' where α' = (λ i. if i < ?n then insertion α (eval (ts ! i)) else 0)
  define β' where β' = (λ i. if i < ?n then insertion β (eval (ts ! i)) else 0)
  {
    fix i
    assume n: i < ?n
    hence tsi: ts ! i ∈ set ts by auto
    {
      assume x ∈ vars-term (ts ! i)
      from IH(1)[OF tsi, unfolded monotone-poly-wrt-def, rule-format, OF α
this v]
      have ins: β' i ⋗ α' i unfolding β-def α'-def β'-def using n by auto
    } note gt = this
    {
      assume x ∉ vars-term (ts ! i)
      with IH(2)[OF tsi] have x: x ∉ vars (eval (ts ! i)) by auto
      hence α' i = β' i unfolding α'-def β'-def using n
      by (auto simp: β-def intro: insertion-irrelevant-vars)
    }
  } with gt have gt^== (β' i) (α' i) by fastforce
  note gt this
} note gt-le = this

```

```

have  $\alpha'$ : assignment  $\alpha'$  unfolding  $\alpha'$ -def assignment-def using Fun(2)
  by (force intro!: valid-imp-insertion-eval-pos[OF assms(1)] -  $\alpha$ ] set-conv-nth)

define  $\gamma$  where  $\gamma$   $n$   $i$  = (if  $i < n$  then  $\beta'$   $i$  else  $\alpha'$   $i$ ) for  $n$   $i$ 
have  $\gamma$ :  $n < ?n \implies$  assignment ( $\gamma$   $n$ ) for  $n$  unfolding  $\gamma$ -def using gt-le(2)
 $\alpha'$  gt-imp-ge
  unfolding assignment-def using order.trans[of  $0$   $\alpha$   $x$   $\beta$   $x$  for  $x$ ]
  by (smt (verit, best) dual-order.strict-trans dual-order.trans sup2E)

from  $x$  obtain  $i$  where  $x$ :  $x \in$  vars-term ( $ts$  !  $i$ ) and  $i$ :  $i < ?n$  by (auto simp: set-conv-nth)
from  $i$  vars have  $iv$ :  $i \in$  vars  $p$  by auto
have  $\gamma i$ : ( $\gamma$  (Suc  $i$ )) = ( $\gamma$   $i$ )(  $i := \beta' i$ ) unfolding  $\gamma$ -def using  $i$  by (intro ext, auto)
have  $1$ :  $gt \hat{=} (insertion (\gamma i) p) (insertion \alpha' p)$ 
  by (rule monotone-poly-wrt-insertion[OF trans-gt gt-imp-ge mono  $\alpha'$ ], insert gt-le i, auto simp:  $\gamma$ -def)
have  $2$ :  $gt (insertion (\gamma (Suc i)) p) (insertion (\gamma i) p)$ 
  using mono[unfolded monotone-poly-wrt-def, rule-format, OF  $\gamma$ [OF i] iv, of  $\beta' i$ ] gt-le(1)[OF i x]
  unfolding  $\gamma i$  by (auto simp:  $\gamma$ -def)
have  $3$ :  $gt \hat{=} (insertion (\gamma ?n) p) (insertion (\gamma (Suc i)) p)$ 
proof (cases  $Suc i < ?n$ )
  case True
  show ?thesis
  by (rule monotone-poly-wrt-insertion[OF trans-gt gt-imp-ge mono  $\gamma$ [OF True]], insert gt-le True, auto simp:  $\gamma$ -def)
  next
  case False
  with  $i$  have  $Suc i = ?n$  by auto
  thus ?thesis by simp
qed
have  $4$ :  $insertion \beta' p = (insertion (\gamma ?n) p)$ 
  unfolding  $\gamma$ -def by (rule insertion-irrelevant-vars, insert vars, auto)
from  $1$   $2$   $3$ 
have  $gt (insertion \beta' p) (insertion \alpha' p)$  using trans-gt unfolding  $4$ 
  by (metis (full-types) sup2E transp-def)
moreover have  $insertion \alpha' p = insertion \alpha (eval (Fun f ts)) \wedge$ 
   $insertion \beta' p = insertion (\alpha(x := v)) (eval (Fun f ts))$ 
  unfolding id insertion-substitute
  unfolding  $\beta'$ -def  $\alpha'$ -def if-distrib  $\beta$ -def[symmetric]
  by (auto intro: insertion-irrelevant-vars)
ultimately show  $gt (insertion (\alpha(x := v)) (eval (Fun f ts))) (insertion \alpha (eval (Fun f ts)))$  by auto
qed
define  $t'$  where  $t' = Fun f ts$ 
define  $\alpha$  where  $\alpha = (\lambda - :: nat. 0 :: 'a)$ 
have ass: assignment  $\alpha$  by (auto simp: assignment-def  $\alpha$ -def)

```

```

show ?case
proof (intro conjI mono, unfold t'-def[symmetric])
  have vars (eval t')  $\subseteq$  vars-term t' by (rule vars-eval)
  moreover have vars-term t'  $\subseteq$  vars (eval t')
  proof (rule ccontr)
    assume  $\neg$  ?thesis
    then obtain x where xt: x  $\in$  vars-term t' and x: x  $\notin$  vars (eval t') by
auto
    from gt-exists[of  $\alpha$  x] obtain l where l: l  $\succ$   $\alpha$  x unfolding  $\alpha$ -def by auto

    from mono[folded t'-def, unfolded monotone-poly-wrt-def, rule-format, OF
ass xt l]
    have insertion ( $\alpha$ (x := l)) (eval t')  $\succ$  insertion  $\alpha$  (eval t') by auto
    also have insertion ( $\alpha$ (x := l)) (eval t') = insertion  $\alpha$  (eval t')
    by (rule insertion-irrelevant-vars, insert x, auto)
    finally show False using gt-irrefl by auto
  qed
  ultimately show vars (eval t') = vars-term t' by auto
qed
qed
thus monotone-poly (vars-term t) (eval t) vars (eval t) = vars-term t by auto
qed
end

```

```

context int-poly-inter
begin

```

```

lemma degree-mono: assumes pos: lead-coeff p  $\geq$  (0 :: int)
  and le:  $\bigwedge$  x. x  $\geq$  c  $\implies$  poly p x  $\leq$  poly q x
shows degree p  $\leq$  degree q
  by (rule degree-mono-generic[OF poly-pinfty-ge-int assms])

```

```

lemma degree-mono': assumes  $\bigwedge$  x. x  $\geq$  c  $\implies$  (bnd :: int)  $\leq$  poly p x  $\wedge$  poly p x
 $\leq$  poly q x
  shows degree p  $\leq$  degree q
  by (rule degree-mono'-generic[OF poly-pinfty-ge-int assms])

```

```

lemma weakly-monotone-insertion: assumes weakly-monotone-poly (vars p) p
  and assignment (a :: -  $\Rightarrow$  int)
  and  $\bigwedge$  x. x  $\in$  vars p  $\implies$  a x  $\leq$  b x
shows insertion a p  $\leq$  insertion b p
proof -
  from monotone-poly-wrt-insertion[OF - - assms(1,2), of b] assms(3)
  show ?thesis by auto
qed

```

Lemma 5.2

lemma *degree-partial-insertion-stays-constant*: **assumes** *mono*: *monotone-poly* (vars p) p
shows $\exists a$. *assignment* ($a :: - \Rightarrow \text{int}$) \wedge
 $(\forall b. (\forall y. a\ y \leq b\ y) \longrightarrow \text{degree} (\text{partial-insertion } a\ x\ p) = \text{degree} (\text{partial-insertion } b\ x\ p))$
using *degree-partial-insertion-stays-constant-generic*[*OF* - - *poly-pinfy-ge-int mono*,
of 0 x]
by (*simp*, *metis le-less*)

lemma *degree-partial-insertion-stays-constant-wm*: **assumes** *wm*: *weakly-monotone-poly* (vars p) p
shows $\exists a$. *assignment* ($a :: - \Rightarrow \text{int}$) \wedge
 $(\forall b. (\forall y. a\ y \leq b\ y) \longrightarrow \text{degree} (\text{partial-insertion } a\ x\ p) = \text{degree} (\text{partial-insertion } b\ x\ p))$
using *degree-partial-insertion-stays-constant-generic*[*OF* - - *poly-pinfy-ge-int wm*,
of 0 x]
by *auto*

Lemma 5.3

lemma *subst-same-var-weakly-monotone-imp-same-degree*:
assumes *wm*: *weakly-monotone-poly* (vars p) ($p :: \text{int mpoly}$)
and *qp*: *poly-to-mpoly* $x\ q = \text{substitute } (\lambda i. \text{PVar } x)\ p$
shows *total-degree* $p = \text{degree } q$
proof (*cases total-degree p = 0*)
case *False*
from *False* **have** $p0$: $p \neq 0$ **by** *auto*
obtain d **where** dq : *degree* $q = d$ **by** *blast*
let $?mc = (\lambda m. \text{mmonom } m (\text{mcoeff } p\ m))$
let $?cfs = \{m . \text{mcoeff } p\ m \neq 0\}$
let $?lc = \text{lead-coeff}$
note $fin = \text{finite-coeff-support}[of\ p]$
define M **where** $M = \text{total-degree } p$
from *degree-monom-eq-total-degree*[*OF* $p0$]
obtain mM **where** mM : $\text{mcoeff } p\ mM \neq 0$ *degree-monom* $mM = M$ **unfolding**
 $M\text{-def}$ **by** *blast*
from *degree-substitute-same-var*[*of* $x\ p$, *folded* $M\text{-def } qp$]
have dM : $d \leq M$ **unfolding** dq *degree-poly-to-mpoly* .
from *False* $M\text{-def}$ **have** $M1$: $M \geq 1$ **by** *auto*
define $p1$ **where** $p1 = \text{sum } ?mc\ (?cfs \cap \{m. \text{degree-monom } m = M\})$
define $p2$ **where** $p2 = \text{sum } ?mc\ (?cfs \cap \{m. \text{degree-monom } m < M\})$
have $p = \text{sum } ?mc\ ?cfs$
by (*rule mpoly-as-sum*)
also **have** $?cfs = ?cfs \cap \{m. \text{degree-monom } m = M\}$
 $\cup ?cfs \cap \{m. \text{degree-monom } m \neq M\}$ **by** *auto*
also **have** $?cfs \cap \{m. \text{degree-monom } m \neq M\} = ?cfs \cap \{m. \text{degree-monom } m <$
 $M\}$
using *degree-monom-le-total-degree*[*of* p , *folded* $M\text{-def}$] **by** *force*
also **have** $\text{sum } ?mc\ (?cfs \cap \{m. \text{degree-monom } m = M\} \cup \dots) = p1 + p2$
unfolding $p1\text{-def } p2\text{-def}$

using *fin* **by** (*intro sum.union-disjoint*, *auto*)
finally have *p-split*: $p = p1 + p2$.
have *total-degree p2* $\leq M - 1$ **unfolding** *p2-def*
by (*intro total-degree-sum-leI*, *subst total-degree-monom*, *auto*)
also have $\dots < M$ **using** *M1* **by** *auto*
finally have *deg-p'*: *total-degree p2* $< M$ **by** *auto*
have $p1 \neq 0$
proof
assume $p1 = 0$
hence $p = p2$ **unfolding** *p-split* **by** *auto*
hence $M = \text{total-degree } p2$ **unfolding** *M-def* **by** *simp*
with *deg-p'* **show** *False* **by** *auto*
qed
with *mpoly-ext-bounded-int*[*of 0 p1 0*] **obtain** *b*
where $b: \bigwedge v. b \ v \geq 0$ **and** *bpm0*: *insertion b p1* $\neq 0$ **by** *auto*
define *B* **where** $B = \text{Max } (\text{insert } 1 \ (b \ \text{vars } p))$
define *X* **where** $X = (0 :: \text{nat})$
define *pb* **where** $pb \ p = \text{mpoly-to-poly } X \ (\text{substitute } (\lambda v. \text{Const } (b \ v) * \text{PVar } X) \ p)$ **for** *p*
have *varsX*: $\text{vars } (\text{substitute } (\lambda v. \text{Const } (b \ v) * \text{PVar } X) \ p) \subseteq \{X\}$ **for** *p*
by (*intro vars-substitute order.trans*[*OF vars-mult*], *auto*)
have *pb*: $\text{substitute } (\lambda v. \text{Const } (b \ v) * \text{PVar } X) \ p = \text{poly-to-mpoly } X \ (pb \ p)$ **for** *p*
unfolding *pb-def*
by (*rule mpoly-to-poly-inverse*[*symmetric*, *OF varsX*])
have *poly-pb*: $\text{poly } (pb \ p) \ x = \text{insertion } (\lambda v. b \ v * x) \ p$ **for** $x \ p$
using *arg-cong*[*OF pb*, *of insertion* ($\lambda \cdot. x$)],
unfolded insertion-poly-to-mpoly
by (*auto simp: insertion-substitute insertion-mult*)
define *lb* **where** $lb = \text{insertion } (\lambda \cdot. 0) \ p$
{
fix *x*
have $\text{poly } (pb \ p) \ x = \text{insertion } (\lambda v. b \ v * x) \ p$ **by** *fact*
also have $\dots = \text{insertion } (\lambda v. b \ v * x) \ p1 + \text{insertion } (\lambda v. b \ v * x) \ p2$
unfolding *p-split*
by (*simp add: insertion-add*)
also have $\text{insertion } (\lambda v. b \ v * x) \ p1 = \text{insertion } b \ p1 * x^M$
unfolding *p1-def insertion-sum insertion-mult insertion-monom sum-distrib-right*

power-mult-distrib
proof (*intro sum.cong*[*OF refl*], *goal-cases*)
case (*1 m*)
from *1* **have** *M*: $M = \text{degree-monom } m$ **by** *auto*
have $\{v. \text{lookup } m \ v \neq 0\} \subseteq \text{keys } m$
by (*simp add: keys.rep-eq*)
from *finite-subset*[*OF this*] **have** *fin*: *finite* $\{v. \text{lookup } m \ v \neq 0\}$ **by** *auto*
have $(\prod v. b \ v \wedge \text{lookup } m \ v * x \wedge \text{lookup } m \ v)$
 $= (\prod v. b \ v \wedge \text{lookup } m \ v) * (\prod v. x \wedge \text{lookup } m \ v)$
by (*subst* (*1 2 3*) *Prod-any.expand-superset*[*OF fin*])

```

      (insert zero-less-iff-neq-zero, force simp: prod.distrib)+
    also have ( $\prod v. x \hat{\text{lookup}} m v = x \hat{M}$  unfolding M degree-monom-def
      by (smt (verit) Prod-any.conditionalize Prod-any.cong finite-keys in-keys-iff
        power-0 power-sum)
      finally show ?case by simp
    qed
    also have insertion ( $\lambda v. b v * x$ ) p2 = poly (pb p2) x unfolding poly-pb ..
    finally have poly (pb p) x = poly (monom (insertion b p1) M + pb p2) x by
      (simp add: poly-monom)
  }
  hence pbp-split: pb p = monom (insertion b p1) M + pb p2 by blast
  have degree (pb p2)  $\leq$  total-degree p2 unfolding pb-def
  apply (subst degree-mpoly-to-poly)
  apply (simp add: varsX)
  by (rule degree-substitute-const-same-var)
  also have ... < M by fact
  finally have deg-pbp2: degree (pb p2) < M .
  have degree (monom (insertion b p1) M) = M using bpm0 by (rule de-
    gree-monom-eq)
  with deg-pbp2 pbp-split have deg-pbp: degree (pb p) = M unfolding pbp-split
  by (subst degree-add-eq-left, auto)
  have ?lc (pb p) = insertion b p1 unfolding pbp-split
  using deg-pbp2 bpm0 coeff-eq-0 deg-pbp pbp-split by auto
  define bnd where bnd = insertion ( $\lambda \cdot. 0$ ) p

{
  fix x :: int
  assume x: x  $\geq$  0
  have ass: assignment ( $\lambda v. b v * x$ ) unfolding assignment-def using x b by
    auto
  have poly (pb p) x = insertion ( $\lambda v. b v * x$ ) p by fact
  also have insertion ( $\lambda v. b v * x$ ) p  $\leq$  insertion ( $\lambda v. B * x$ ) p
  proof (rule weakly-monotone-insertion[OF wm ass])
  fix v
  show v  $\in$  vars p  $\implies$  b v * x  $\leq$  B * x using b[of v] x unfolding B-def
  by (intro mult-right-mono, auto intro!: Max-ge vars-finite)
  qed
  also have ... = poly q (B * x) unfolding poly-to-mpoly-substitute-same[OF
    qp] ..
  also have ... = poly (q  $\circ_p$  [:0, B:]) x by (simp add: poly-pcompose ac-simps)
  finally have ineq: poly (pb p) x  $\leq$  poly (q  $\circ_p$  [:0, B:]) x .
  have bnd  $\leq$  insertion ( $\lambda v. b v * x$ ) p unfolding bnd-def
  by (intro weakly-monotone-insertion[OF wm], insert b x, auto simp: assign-
    ment-def)
  also have ... = poly (pb p) x using poly-pb by auto
  finally have bnd  $\leq$  poly (pb p) x by auto
  note this ineq
} note pb-approx = this
have M = degree (pb p) unfolding deg-pbp ..

```

```

also have ...  $\leq$  degree ( $q \circ_p [ :0, B:]$ )
  by (intro degree-mono[of 0 bnd], insert pb-approx, auto)
also have ...  $\leq d$  by (simp add: dq)
finally have deg-pbp:  $M \leq d$  .
with dM have  $M = d$  by auto
thus ?thesis unfolding M-def dq .
next
  case True
  then obtain c where  $p = \text{Const } c$  using degree-0-imp-Const by blast
  with qp have poly-to-mpoly  $x \ q = p$  by auto
  thus ?thesis
    by (metis True degree-Const degree-poly-to-mpoly p)
qed

lemma monotone-poly-partial-insertion:
  assumes  $x: x \in xs$ 
  and mono: monotone-poly  $xs \ p$ 
  and ass: assignment  $a$ 
shows  $0 < \text{degree} (\text{partial-insertion } a \ x \ p)$ 
  lead-coeff (partial-insertion  $a \ x \ p$ )  $> 0$ 
  valid-poly  $p \implies y \geq 0 \implies \text{poly} (\text{partial-insertion } a \ x \ p) \ y \geq y$ 
  valid-poly  $p \implies \text{insertion } a \ p \geq a \ x$ 
proof -
  have  $0: \text{transp } ((>) :: \text{int} \Rightarrow -)$  by auto
  have  $1: (x < y) = (x + 1 \leq y)$  for  $x \ y :: \text{int}$  by auto
  have  $2: x \leq \text{int } (\text{nat } x)$  for  $x$  by auto
  note main = monotone-poly-partial-insertion-generic[of (>) 1 1 nat, OF 0 -
poly-pinfy-ge-int 1 - - 2 x mono ass, simplified]
  show  $0 < \text{degree} (\text{partial-insertion } a \ x \ p) \ 0 < \text{lead-coeff} (\text{partial-insertion } a \ x \ p)$ 

    using main by auto
  assume valid: valid-poly  $p$ 
  {
    fix  $y :: \text{int}$ 
    assume  $y \geq 0$ 
    then obtain  $n$  where  $y = \text{int } n$ 
      by (metis int-nat-eq)
    from main( $\exists$ )[OF valid, of n, folded y]
    show  $y \leq \text{poly} (\text{partial-insertion } a \ x \ p) \ y$  by auto
  } note estimation = this
  from ass have  $a \ x \geq 0$  unfolding assignment-def by auto
  from estimation[OF this] show insertion  $a \ p \geq a \ x$ 
  using insertion-partial-insertion[of x a a p] by auto
qed

end

context int-poly-inter
begin

```

```

lemma insertion-eval-pos: assumes funas-term  $t \subseteq F$ 
  and assignment  $\alpha$ 
shows insertion  $\alpha$  (eval  $t$ )  $\geq 0$ 
  by (rule valid-imp-insertion-eval-pos[OF valid assms])

lemma monotone-poly-eval: assumes funas-term  $t \subseteq F$ 
  shows monotone-poly (vars-term  $t$ ) (eval  $t$ ) vars (eval  $t$ ) = vars-term  $t$ 
proof –
  have  $\exists y. x < y$  for  $x :: int$  by (intro exI[of - x + 1], auto)
  from monotone-poly-eval-generic[OF valid - - this - assms]
  show monotone-poly (vars-term  $t$ ) (eval  $t$ ) vars (eval  $t$ ) = vars-term  $t$  by auto
qed
end

locale term-poly-input = poly-input  $p$   $q$  for  $p$   $q$  +
  assumes terminating-poly: termination-by-int-poly-interpretation  $F$ - $S$   $S$ 
begin

definition  $I$  where  $I = (SOME\ I. int-poly-inter\ F\ S\ I \wedge int-poly-inter.termination-by-poly-interpretation\ F\ S\ I\ S)$ 

lemma  $I$ : int-poly-inter  $F$ - $S$   $I$  int-poly-inter.termination-by-poly-interpretation  $F$ - $S$ 
 $I$   $S$ 
  using someI-ex[OF terminating-poly[unfolded termination-by-int-poly-interpretation-def],
folded I-def] by auto

sublocale int-poly-inter  $F$ - $S$   $I$  by (rule I(1))

lemma orient: orient-rule (lhs-S,rhs-S)
  using  $I$ (2)[unfolded termination-by-interpretation-def termination-by-poly-interpretation-def]
unfolding  $S$ -def by auto

lemma solution: positive-poly-problem  $p$   $q$ 
proof –
  from orient[unfolded orient-rule]
  have gt: gt-poly (eval lhs-S) (eval rhs-S) by auto
  from valid[unfolded valid-monotone-poly-inter-def]
  have valid:  $\bigwedge f. f \in F\ S \implies valid-monotone-poly\ f$  by auto
  let  $?lc = lead-coeff$ 
  let  $?f = (f-sym, \gamma)$ 
  have  $?f \in F\ S$  unfolding  $F$ - $S$ -def by auto
  from valid[OF this, unfolded valid-monotone-poly-def] obtain  $f$  where
    If:  $I\ f-sym = f$  and  $f$ : valid-poly  $f$  monotone-poly (vars  $f$ )  $f$  vars  $f = \{.. < \gamma\}$ 
by auto
  from  $f$ (2) have wmf: weakly-monotone-poly (vars  $f$ )  $f$  by (rule monotone-imp-weakly-monotone)
  define  $l$  where  $l\ i = args\ (lhs-S)\ !\ i$  for  $i$ 
  define  $r$  where  $r\ i = args\ (rhs-S)\ !\ i$  for  $i$ 

```

```

have list: [0..<7] = [0,1,2,3,4,5,6 :: nat] by code-simp
have lhs-S: lhs-S = Fun f-sym (map l [0..<7]) unfolding lhs-S-def l-def by
(auto simp: list)
have rhs-S: rhs-S = Fun f-sym (map r [0..<7]) unfolding rhs-S-def r-def by
(auto simp: list)
{
  fix i :: var
  define vs where vs = V-list
  assume i < 7
  hence choice: i = 0 ∨ i = 1 ∨ i = 2 ∨ i = 3 ∨ i = 4 ∨ i = 5 ∨ i = 6 by
linarith
  have set: {0..<7 :: nat} = {0,1,2,3,4,5,6} by code-simp
  from choice have vars: vars-term (l i) = {i} vars-term (r i) = {i} unfolding
l-def lhs-S-def r-def rhs-S-def
  using vars-encode-poly[of 2 p] vars-encode-poly[of 2 q]
  by (auto simp: y1-def y2-def y3-def y4-def y5-def y6-def y7-def vs-def[symmetric])
  from choice set have funs: funas-term (l i) ∪ funas-term (r i) ⊆ F-S using
rhs-S-F lhs-S-F unfolding lhs-S rhs-S
  by auto
  have lr ∈ {l,r} ⇒ vars-term (lr i) = {i} lr ∈ {l,r} ⇒ funas-term (lr i) ⊆
F-S for lr
  by (insert vars funs, force)+
} note signature-l-r = this
{
  fix i :: var and lr
  assume i: i < 7 and lr: lr ∈ {l,r}
  from signature-l-r[OF i lr] monotone-poly-eval[of lr i]
  have vars: vars (eval (lr i)) = {i}
  and mono: monotone-poly {i} (eval (lr i)) by auto
} note eval-l-r = this

define upoly where upoly l-or-r i = mpoly-to-poly i (eval (l-or-r i)) for l-or-r ::
var ⇒ (-,-)term and i

{
  fix lr and i :: nat and a :: - ⇒ int
  assume a: assignment a and i: i < 7 and lr: lr ∈ {l,r}
  with eval-l-r[OF i] signature-l-r[OF i]
  have vars: vars (eval (lr i)) = {i} and mono: monotone-poly {i} (eval (lr i))
  and funs: funas-term (lr i) ⊆ F-S by auto
  from insertion-eval-pos[OF funs]
  have valid: valid-poly (eval (lr i)) unfolding valid-poly-def by auto
  from monotone-poly-partial-insertion[OF - mono a, of i] valid
  have deg: degree (partial-insertion a i (eval (lr i))) > 0
  and lc: ?lc (partial-insertion a i (eval (lr i))) > 0
  and ineq: insertion a (eval (lr i)) ≥ a i by auto
  moreover have partial-insertion a i (eval (lr i)) = upoly lr i unfolding
upoly-def
  using vars eval-l-r[OF i, of r, simplified]

```

```

    by (intro poly-ext)
      (metis i insertion-partial-insertion-vars poly-eq-insertion poly-inter.vars-eval
signature-l-r(1)[of - r, simplified] singletonD)
  ultimately
  have degree (upoly lr i) > 0 ?lc (upoly lr i) > 0
    insertion a (eval (lr i)) ≥ a i by auto
} note upoly-pos-subterm = this

{
  fix i :: var
  assume i: i < 7
  from degree-partial-insertion-stays-constant[OF f(2), of i] obtain a where
    a: assignment a and
    deg-a:  $\bigwedge b. (\bigwedge y. a y \leq b y) \implies \text{degree (partial-insertion a i f)} = \text{degree (partial-insertion b i f)}$ 
  by auto
  define c where c j = (if j < 7 then insertion a (eval (l j)) else a j) for j
  define e where e j = (if j < 7 then insertion a (eval (r j)) else a j) for j
  {
    fix x :: int
    assume x: x ≥ 0
    have ass: assignment (a (i := x)) using x a unfolding assignment-def by
auto
    from gt[unfolded gt-poly-def, rule-format, OF ass, unfolded rhs-S lhs-S]
    have insertion (a(i := x)) (eval (Fun f-sym (map r [0..<7])))
      < insertion (a(i := x)) (eval (Fun f-sym (map l [0..<7]))) by simp
    also have insertion (a(i := x)) (eval (Fun f-sym (map r [0..<7]))) =
      insertion (λj. insertion (a(i := x)) (eval (r j))) f
    by (simp add: If insertion-substitute, intro insertion-irrelevant-vars, auto
simp: f)
    also have ... = poly (partial-insertion e i f) (poly (upoly r i) x)
  proof -
    let ?α = (λj. insertion (a(i := x)) (eval (r j)))
    have insi: poly (upoly r i) x = insertion (a(i := x)) (eval (r i))
      unfolding upoly-def using eval-l-r(1)[OF i, of r]
      by (subst poly-eq-insertion, force)
      (intro insertion-irrelevant-vars, auto)
    show ?thesis unfolding insi
  proof (rule insertion-partial-insertion-vars[of i f e ?α, symmetric])
    fix j
    show j ≠ i  $\implies j \in \text{vars } f \implies e j = \text{insertion (a(i := x)) (eval (r j))}$ 
      unfolding e-def f using eval-l-r[of j] f by (auto intro!: inser-
tion-irrelevant-vars)
  qed
qed
  also have insertion (a(i := x)) (eval (Fun f-sym (map l [0..<7]))) =
    insertion (λj. insertion (a(i := x)) (eval (l j))) f
  by (simp add: If insertion-substitute, intro insertion-irrelevant-vars, auto

```

```

simp: f)
  also have ... = poly (partial-insertion c i f) (poly (upoly l i) x)
  proof -
    let ?α = (λj. insertion (a(i := x)) (eval (l j)))
    have insi: poly (upoly l i) x = insertion (a(i := x)) (eval (l i))
      unfolding upoly-def using eval-l-r[OF i]
      by (subst poly-eq-insertion, force)
        (intro insertion-irrelevant-vars, auto)
    show ?thesis unfolding insi
    proof (rule insertion-partial-insertion-vars[of i f c ?α, symmetric])
      fix j
      show j ≠ i ⇒ j ∈ vars f ⇒ c j = insertion (a(i := x)) (eval (l j))
        unfolding c-def f using eval-l-r[of j] f by (auto intro!: inser-
tion-irrelevant-vars)
      qed
    qed

  finally have poly (partial-insertion c i f) (poly (upoly l i) x)
    > poly (partial-insertion e i f) (poly (upoly r i) x) .
} note 1 = this

define er where er = partial-insertion e i f ∘p upoly r i
define cl where cl = partial-insertion c i f ∘p upoly l i
define d where d = degree (partial-insertion e i f)
{
  fix x
  have a x ≤ c x ∧ a x ≤ e x
  proof (cases x ∈ vars f)
    case False
    thus ?thesis unfolding c-def e-def f by auto
  next
    case True
    hence id: (x < 7) = True and x: x < 7 unfolding f by auto
    show ?thesis unfolding c-def e-def id if-True using upoly-pos-subterm(3)[OF
a x] by auto
  qed
  hence a x ≤ c x a x ≤ e x by auto
} note a-ce = this

have d-eq: d = degree (partial-insertion c i f) unfolding d-def
  by (subst (1 2) deg-a[symmetric], insert a-ce, auto)

have e: assignment e using a-ce(2) unfolding assignment-def
  by (smt (verit, del-insts))

have d-pos: d > 0 unfolding d-def
  by (intro monotone-poly-partial-insertion[OF - f(2) e], insert f i, auto)
have lc-e-pos: ?lc (partial-insertion e i f) > 0
  by (intro monotone-poly-partial-insertion[OF - f(2) e], insert f i, auto)

```



```

have lc-r-pos: ?lc (upoly r i) > 0 by (intro upoly-pos-subterm[OF a i], auto)
have deg-r: 0 < degree (upoly r i) by (intro upoly-pos-subterm[OF a i], auto)
have lc-er-pos: ?lc er > 0 unfolding er-def
  by (subst lead-coeff-comp[OF deg-r], insert lc-e-pos deg-r lc-r-pos, auto)

from 1 [folded poly-pcompose, folded er-def cl-def]
have er-cl-poly: 0 ≤ x ⇒ poly er x < poly cl x for x by auto
have degree er ≤ degree cl
proof (intro degree-mono[of - 0])
  show 0 ≤ ?lc er using lc-er-pos by auto
  show 0 ≤ x ⇒ poly er x ≤ poly cl x for x using er-cl-poly[of x] by auto
qed
also have degree er = d * degree (upoly r i)
  unfolding er-def d-def by simp
also have degree cl = d * degree (upoly l i)
  unfolding cl-def d-eq by simp
finally have degree (upoly l i) ≥ degree (upoly r i) using d-pos by auto
} note deg-inequality = this

{
  fix p :: int mpoly and x
  assume p: monotone-poly {x} p vars p = {x}
  define q where q = mpoly-to-poly x p
  from mpoly-to-poly-inverse[of p x]
  have pq: p = poly-to-mpoly x q using p unfolding q-def by auto
  from pq p(2) have deg: degree q > 0
    by (simp add: degree-mpoly-to-poly degree-pos-iff q-def)
  from deg pq have ∃ q. p = poly-to-mpoly x q ∧ degree q > 0 unfolding q-def
by auto
} note mono-unary-poly = this

{
  fix f
  assume f ∈ {q-sym, h-sym} ∪ v-sym ‘ V
  hence (f, 1) ∈ F-S unfolding F-S-def F-def by auto
  from valid[OF this, unfolded valid-monotone-poly-def] obtain p
    where p: p = I f monotone-poly {..<1} p vars p = {0} by auto
  have id: {..<(1 :: nat)} = {0} by auto
  have ∃ q. I f = poly-to-mpoly 0 q ∧ degree q > 0 unfolding p(1)[symmetric]
    by (intro mono-unary-poly, insert p(2-3)[unfolded id], auto)
} note unary-symbol = this

{
  fix f and n :: nat and x :: var
  assume f ∈ {f-sym, a-sym} f = f-sym ⇒ n = 7 f = a-sym ⇒ n = 2
  hence n: n > 1 and f: (f, n) ∈ F-S unfolding F-def F-S-def by force+
  define p where p = I f
  from valid[OF f, unfolded valid-monotone-poly-def, rule-format, OF refl p-def]

```

have *mono*: *monotone-poly* (*vars p*) *p* **and** *vars*: *vars p* = {..*n*} **and** *valid*:
valid-poly p **by** *auto*
let *?t* = *Fun f* (*replicate n* (*TVar x*))
have *t-F*: *funas-term ?t* \subseteq *F-S* **using** *f* **by** *auto*
have *vt*: *vars-term ?t* = {*x*} **using** *n* **by** *auto*
define *q* **where** *q* = *eval ?t*
from *monotone-poly-eval*[*OF t-F*, *unfolded vt*, *folded q-def*]
have *monotone-poly* {*x*} *q* *vars q* = {*x*} **by** *auto*
from *mono-unary-poly*[*OF this*] **obtain** *q'* **where**
qq': *q* = *poly-to-mpoly x q'* **and** *dq'*: *degree q' > 0* **by** *auto*
have *q't*: *poly-to-mpoly x q' = eval ?t* **unfolding** *qq'*[*symmetric*] *q-def* **by** *simp*
also have ... = *substitute* (λi . *if i < n* then *eval* (*replicate n* (*TVar x*) ! *i*) *else*
0) *p*
by (*simp add*: *p-def*[*symmetric*])
also have (λi . *if i < n* then *eval* (*replicate n* (*TVar x*) ! *i*) *else 0*) = (λi . *if i*
< n then *PVar x* *else 0*)
by (*intro ext*, *auto*)
also have *substitute* ... *p* = *substitute* (λi . *PVar x*) *p* **using** *vars*
unfolding *substitute-def* **using** *vars-replace-coeff*[*of Const*, *OF Const-0*]
by (*intro insertion-irrelevant-vars*, *auto*)
finally have *eq*: *poly-to-mpoly x q' = substitute* (λi . *PVar x*) *p* .
have $\exists p q$. *I f* = *p* \wedge *eval ?t = poly-to-mpoly x q* \wedge *poly-to-mpoly x q =*
substitute (λi . *PVar x*) *p* \wedge *degree q > 0*
 \wedge *vars p* = {..*n*} \wedge *monotone-poly* (*vars p*) *p*
by (*intro exI*[*of - p*] *exI*[*of - q'*] *conjI* *valid eq dq' p-def*[*symmetric*] *q't*[*symmetric*]
mono vars)
} note *f-a-sym = this*

from *unary-symbol*[*of q-sym*] **obtain** *q* **where** *Iq*: *I q-sym = poly-to-mpoly 0 q*
and *dq*: *degree q > 0* **by** *auto*
from *unary-symbol*[*of h-sym*] **obtain** *h* **where** *Ih*: *I h-sym = poly-to-mpoly 0 h*
and *dh*: *degree h > 0* **by** *auto*

from *unary-symbol*[*of v-sym i for i*] **have** $\forall i$. $\exists q$. $i \in V \longrightarrow I$ (*v-sym i*) =
poly-to-mpoly 0 q \wedge $0 < \text{degree } q$ **by** *auto*
from *choice*[*OF this*] **obtain** *v* **where**
Iv: $i \in V \implies I$ (*v-sym i*) = *poly-to-mpoly 0* (*v i*) **and**
dv: $i \in V \implies \text{degree } (v i) > 0$
for *i* **by** *auto*

have *eval-pm-Var*: *eval* (*TVar y*) = *poly-to-mpoly y* [:0,1:] **for** *y*
unfolding *eval.simps* *mpoly-of-poly-is-poly-to-mpoly*[*symmetric*] **by** *simp*
have *id*: (*if 0 = (0 :: nat)* then *eval* (*[t] ! 0*) *else 0*) = *eval t* **for** *t* **by** *simp*
{
have *y*: *eval* (*TVar y4*) = *poly-to-mpoly y4* [:0,1:] (**is - = poly-to-mpoly -**
?poly1) **by** *fact*
have *hy*: *eval* (*Fun h-sym* [*TVar y4*]) = *poly-to-mpoly y4 h* **using** *Ih*
apply (*simp*)
apply (*subst substitute-poly-to-mpoly*[*of - - y4 ?poly1*])

```

    apply (unfold id, intro y)
  by simp
  have qhy: eval (Fun q-sym [Fun h-sym [TVar y4]]) = poly-to-mpoly y4 (pcompose
q h) using Iq
    apply (simp)
    apply (subst substitute-poly-to-mpoly[of - - y4 h])
    apply (unfold id, intro hy)
    by simp
  hence l3: eval (l 3) = poly-to-mpoly y4 (pcompose q h) unfolding l-def lhs-S-def
by simp

  have qy: eval (Fun q-sym [TVar y4]) = poly-to-mpoly y4 q using Iq
    apply (simp)
    apply (subst substitute-poly-to-mpoly[of - - y4 ?poly1])
    apply (unfold id, intro y)
    by simp
  have hqy: eval (Fun h-sym [Fun q-sym [TVar y4]]) = poly-to-mpoly y4 (pcompose
h q) using Ih
    apply (simp)
    apply (subst substitute-poly-to-mpoly[of - - y4 q])
    apply (unfold id, intro qy)
    by simp
  have hhqy: eval (Fun h-sym [Fun h-sym [Fun q-sym [TVar y4]]]) = poly-to-mpoly
y4 (pcompose h (pcompose h q)) using Ih
    apply (simp)
    apply (subst substitute-poly-to-mpoly[of - - y4 pcompose h q])
    apply (unfold id, intro hqy)
    by simp
  hence r3: eval (r 3) = poly-to-mpoly y4 (pcompose h (pcompose h q)) unfolding
r-def rhs-S-def by simp

  from deg-inequality[of 3] have deg: degree (upoly r 3) ≤ degree (upoly l 3) by
simp
  hence degree h * (degree h * degree q) ≤ degree q * degree h
    unfolding upoly-def l3 r3 y4-def poly-to-mpoly-inverse by simp
  with dq have degree h * degree h ≤ degree h by simp
  with dh have degree h = 1 by auto
} note dh = this

define tayy where tayy = Fun a-sym (replicate 2 (TVar y5))
from f-a-sym[of a-sym 2 y5, folded tayy-def] obtain a ayy where
  Ia: I a-sym = a
  and eval-ayy: eval tayy = poly-to-mpoly y5 ayy
  and dayy: degree ayy > 0 and payy: poly-to-mpoly y5 ayy = substitute (λi.
PVar y5) a
  and monoa: monotone-poly (vars a) a and varsa: vars a = {..<2} by blast

{
  define vs where vs = V-list

```

have vs : set $vs \subseteq V$ **unfolding** vs -def V -list **by** *auto*
have $r\ 4 = \text{foldr } (\lambda i\ t.\ \text{Fun } (v\text{-sym } i) [t])\ vs\ \text{tayy}$ **unfolding** tayy -def r -def
 rhs-S-def sub-def vs -def
by (*simp add: numeral-eq-Suc*)
also have $\exists q.\ \text{eval } \dots = \text{poly-to-mpoly } y5\ q \wedge \text{degree } q = \text{prod-list } (\text{map } (\lambda i.\ \text{degree } (v\ i))\ vs) * \text{degree } \text{ayy}$
using vs
proof (*induct vs*)
case *Nil*
show $?case$ **using** eval-ayy **by** *auto*
next
case (*Cons x vs*)
from *Cons* **obtain** q **where** $IH1$: $\text{eval } (\text{foldr } (\lambda i\ t.\ \text{Fun } (v\text{-sym } i) [t])\ vs\ \text{tayy})$
 $= \text{poly-to-mpoly } y5\ q$
and $IH2$: $\text{degree } q = (\prod i \leftarrow vs.\ \text{degree } (v\ i)) * \text{degree } \text{ayy}$ **by** *auto*
from *Cons* **have** x : $x \in V$ **by** *auto*
have $\text{eval}:$ $\text{eval } (\text{foldr } (\lambda i\ t.\ \text{Fun } (v\text{-sym } i) [t])\ (x \# vs)\ \text{tayy}) = \text{poly-to-mpoly}$
 $y5\ (v\ x\ \circ_p\ q)$ **using** $Iv[OF\ x]$
apply *simp*
apply (*subst substitute-poly-to-mpoly[of - - y5 q]*)
apply (*unfold id, intro IH1*)
by *simp*
show $?case$ **unfolding** eval **by** (*intro exI[of - v x \circ_p q], auto simp: IH2*)
qed
finally obtain q **where**
 $r4$: $\text{eval } (r\ 4) = \text{poly-to-mpoly } y5\ q$ **and**
 q : $\text{degree } q = \text{prod-list } (\text{map } (\lambda i.\ \text{degree } (v\ i))\ vs) * \text{degree } \text{ayy}$
by *auto*

have y : $\text{eval } (TVar\ y5) = \text{poly-to-mpoly } y5\ [:0,1:]$ (**is** $- = \text{poly-to-mpoly } -$
 $?poly1$) **by** *fact*
have hy : $\text{eval } (\text{Fun } h\text{-sym } [TVar\ y5]) = \text{poly-to-mpoly } y5\ h$ **using** Ih
apply (*simp*)
apply (*subst substitute-poly-to-mpoly[of - - y5 ?poly1]*)
apply (*unfold id, intro y*)
by *simp*

hence $l4$: $\text{eval } (l\ 4) = \text{poly-to-mpoly } y5\ h$ **unfolding** l -def lhs-S-def **by** *simp*

from $\text{deg-inequality}[of\ 4]$ **have** deg : $\text{degree } (\text{upoly } r\ 4) \leq \text{degree } (\text{upoly } l\ 4)$ **by**
simp
hence $\text{degree } q \leq \text{degree } h$
unfolding $\text{upoly-def } l4\ r4\ y5$ -def $\text{poly-to-mpoly-inverse}$ **by** *simp*
hence $\text{deg}q$: $\text{degree } q \leq 1$ **unfolding** dh **by** *simp*
hence ($\forall x \in \text{set } vs.\ \text{degree } (v\ x) = 1$) $\wedge \text{degree } \text{ayy} = 1 \wedge \text{degree } q = 1$ **using**
 vs **unfolding** q
proof (*induct vs*)
case *Nil*
thus $?case$ **using** dayy **by** *auto*

```

next
case (Cons x vs)
define rec where rec = (∏ i←vs. degree (v i)) * degree ayy
have id: (∏ i←x # vs. degree (v i)) * degree ayy = degree (v x) * rec
  unfolding rec-def by auto
from Cons(2)[unfolded id] have prems: degree (v x) * rec ≤ 1 by auto
from Cons(3) have x: x ∈ V and sub: set vs ⊆ V by auto
from dv[OF x] have dv: degree (v x) ≥ 1 by auto
from dv prems have rec ≤ 1
  by (metis dual-order.trans mult.commute mult.right-neutral mult-le-mono2)
from Cons(1)[folded rec-def, OF this sub]
have IH: (∀ x∈set vs. degree (v x) = 1) degree ayy = 1 rec = 1 by auto
from IH(3) dv prems have dvx: degree (v x) = 1 by simp
show ?case unfolding id using dvx IH by auto
qed
from this[unfolded vs-def V-list]
have dv: ∧ x. x ∈ V ⇒ degree (v x) = 1 and dayy: degree ayy = 1 by auto
}
hence dv: ∧ x. x ∈ V ⇒ degree (v x) = 1 and dayy: degree ayy = 1 by auto

define tfyy where tfyy = Fun f-sym (replicate 7 (TVar y6))
from f-a-sym[of f-sym 7 y6, folded tfyy-def] obtain f fyy where
  If: I f-sym = f
  and eval-fyy: eval tfyy = poly-to-mpoly y6 fyy
  and dfyy: degree fyy > 0 and pfyy: poly-to-mpoly y6 fyy = substitute (λi. PVar
y6) f
  and monof: monotone-poly (vars f) f and varsf: vars f = {..<7} by blast

{
  have y: eval (TVar y6) = poly-to-mpoly y6 [:0,1:] (is - = poly-to-mpoly -
?poly1) by fact
  have hy: eval (Fun h-sym [TVar y6]) = poly-to-mpoly y6 h using Ih
  apply (simp)
  apply (subst substitute-poly-to-mpoly[of - - y6 ?poly1])
  apply (unfold id, intro y)
  by simp

  hence l5: eval (l 5) = poly-to-mpoly y6 h unfolding l-def lhs-S-def by simp
  have r 5 = tfyy unfolding tfyy-def r-def rhs-S-def by simp
  hence r5: eval (r 5) = poly-to-mpoly y6 fyy using eval-fyy by simp

  from deg-inequality[of 5] have deg: degree (upoly r 5) ≤ degree (upoly l 5) by
simp
  from this[unfolded upoly-def l5 r5 y6-def poly-to-mpoly-inverse dh]
  have degree fyy ≤ 1 .
}
with dfyy
have dfyy: degree fyy = 1 by auto

```

```

note lemma-5-3 = subst-same-var-weakly-monotone-imp-same-degree[OF mono-
tone-imp-weakly-monotone]
from lemma-5-3[OF monof] dfyy pfy have df: total-degree f = 1 by auto
from lemma-5-3[OF monoa] dayy payy have da: total-degree a = 1 by auto

let ?argsL = [q-t (h-t (Var y4)),
  h-t (Var y5),
  h-t (Var y6),
  g-t (Var y7) o-t]
let ?argsR = [h-t (h-t (q-t (Var y4))),
  foldr v-t V-list (a-t (Var y5) (Var y5)),
  Fun f-sym (replicate 7 (Var y6)),
  g-t (Var y7) z-t]

show ?thesis
apply (rule poly-input-to-solution-common.solution[of - - I F-S ?argsL ?argsR])
apply (unfold-locales)
subgoal using orient unfolding lhs-S-def rhs-S-def by simp
subgoal by simp
subgoal using signature-l-r(1)[of 4 r]
  by (auto simp: y1-def y2-def y3-def y4-def y5-def y6-def y7-def r-def rhs-S-def)
subgoal unfolding F-S-def by auto
subgoal for g n
proof (goal-cases)
  case 1
  hence ch: (g,n) = (f-sym,7)  $\vee$  (g,n)  $\in$  F by auto
  hence (g,n)  $\in$  F-S unfolding F-S-def by auto
from valid[rule-format, OF this, unfolded valid-monotone-poly-def, rule-format,
OF refl refl]
  have *: valid-poly (I g) monotone-poly {.. $n$ } (I g) vars (I g) = {.. $n$ }
  by auto
  show ?case
proof (intro monotone-linear-poly-to-coeffs *)
  show total-degree (I g)  $\leq$  1
  proof (rule ccontr)
  assume not:  $\neg$  ?thesis
  with ch df da If Ia have (g,n)  $\in$  F - {(a-sym,2)} by auto
  then consider (V) i where i  $\in$  V g = v-sym i n = 1 | (z) g = z-sym n
= 0
  unfolding F-def by auto
  thus False
proof cases
  case V
  have total-degree (I g) = 1 unfolding dv[OF V(1), symmetric]
proof (rule lemma-5-3[OF *(2)[folded *(3)])]
  show poly-to-mpoly 0 (v i) = substitute ( $\lambda$ i. PVar 0) (I g)
  unfolding V Iv[OF V(1)]
  by (intro mpoly-extI, auto simp: insertion-substitute)
qed

```

```

    with not show False by auto
  next
  case z
  with * have vars (I g) = {} by auto
  from vars-empty-Const[OF this] obtain c where I g = Const c by auto
  hence total-degree (I g) = 0 by simp
  with not show False by auto
  qed
  qed
  qed
  done
qed
end

```

```

context poly-input
begin

```

Theorem 5.4 in paper

theorem *polynomial-termination-with-natural-numbers-undecidable:*
positive-poly-problem p q \longleftrightarrow *termination-by-int-poly-interpretation F-S S*

proof

```

  assume positive-poly-problem p q
  interpret solvable-poly-problem
  by (unfold-locales, fact)
  from solution-imp-poly-termination
  show termination-by-int-poly-interpretation F-S S .
next
  assume termination-by-int-poly-interpretation F-S S
  interpret term-poly-input
  by (unfold-locales, fact)
  from solution show positive-poly-problem p q .
qed

```

end

Now head for Lemma 5.6

```

locale poly-input-omega-solution = poly-input
begin

```

```

fun I :: symbol  $\Rightarrow$  int list  $\Rightarrow$  int where
  I o-sym xs = insertion ( $\lambda$  -. 1) q
| I z-sym xs = 0
| I a-sym xs = xs ! 0 + xs ! 1
| I g-sym xs = (xs ! 1 + 1) * xs ! 0 + xs ! 1
| I h-sym xs = (xs ! 0)2 + 7 * (xs ! 0) + 4
| I f-sym xs = xs ! 2 * xs ! 6 + sum-list xs
| I q-sym xs = 5(nat (xs ! 0))
| I (v-sym i) xs = xs ! 0

```

lemma *I-encode-num*: **assumes** $c \geq 0$
shows $I[\text{encode-num } x \ c]\alpha = c * \alpha \ x$
proof –
from *assms* **obtain** n **where** $cn: c = \text{int } n$ **by** (*metis nonneg-eq-int*)
hence *natc*: $\text{nat } c = n$ **by** *auto*
show *?thesis* **unfolding** *encode-num-def natc* **unfolding** *cn*
by (*induct n, auto simp: algebra-simps*)
qed

lemma *I-v-pow-e*: $I[(v-t \ x \ \hat{\sim} \ e) \ t]\alpha = I[t]\alpha$
by (*induct e, auto*)

lemma *I-encode-monom*: **assumes** $c: c \geq 0$
shows $I[\text{encode-monom } x \ m \ c]\alpha = c * \alpha \ x$
proof –
define *xes* **where** $xes = \text{var-list } m$
from *var-list*[*of m c*]
have *monom*: $mmonom \ m \ c = \text{Const } c * (\prod (x, e) \leftarrow xes . PVar \ x \ \hat{\sim} \ e)$ **unfolding**
xes-def .
show *?thesis* **unfolding** *encode-monom-def monom xes-def*[*symmetric*]
by (*induct xes, auto simp: I-encode-num[OF c] I-v-pow-e*)
qed

lemma *I-encode-poly*: **assumes** *positive-poly r*
shows $I[\text{encode-poly } x \ r]\alpha = \text{insertion } (\lambda \ . \ 1) \ r * \alpha \ x$
proof –
define *mcs* **where** $mcs = \text{monom-list } r$
from *monom-list*[*of r*] **have** $r: r = (\sum (m, c) \leftarrow mcs. mmonom \ m \ c)$ **unfolding**
mcs-def **by** *auto*
have *mcs*: $(m, c) \in \text{set } mcs \implies c \geq 0$ **for** $m \ c$
using *monom-list-coeff assms* **unfolding** *mcs-def positive-poly-def* **by** *auto*
show *?thesis* **unfolding** *encode-poly-def mcs-def*[*symmetric*] **unfolding** r *insertion-sum-list map-map o-def*
using *mcs*
proof (*induct mcs*)
case (*Cons mc mcs*)
obtain $m \ c$ **where** $mc: mc = (m, c)$ **by** *force*
from *Cons(2) mc* **have** $c: c \geq 0$ **by** *auto*
note *monom = I-encode-monom[OF this, of x m]*
show *?case*
by (*simp add: mc monom algebra-simps, subst Cons(1), insert Cons(2), auto simp: Const-add algebra-simps*)
qed *simp*
qed
end

lemma *length2-cases*: $\text{length } xs = 2 \implies \exists \ x \ y. \ xs = [x, y]$
by (*cases xs; cases tl xs, auto*)


```

lemma length7-cases:  $\text{length } xs = 7 \implies \exists x1\ x2\ x3\ x4\ x5\ x6\ x7. xs = [x1, x2, x3, x4, x5, x6, x7]$ 
  apply (cases xs, force)
  apply (cases drop 1 xs, force)
  apply (cases drop 2 xs, force)
  apply (cases drop 3 xs, force)
  apply (cases drop 4 xs, force)
  apply (cases drop 5 xs, force)
  by (cases drop 6 xs, force+)

lemma length1-cases:  $\text{length } xs = \text{Suc } 0 \implies \exists x. xs = [x]$ 
  by (cases xs; auto)

lemma less2-cases:  $i < 2 \implies i = 0 \vee (i :: \text{nat}) = 1$ 
  by auto

lemma less7-cases:  $i < 7 \implies i = 0 \vee (i :: \text{nat}) = 1 \vee i = 2 \vee i = 3 \vee i = 4$ 
 $\vee i = 5 \vee i = 6$ 
  by auto

context poly-input-omega-solution
begin

sublocale inter-S: term-algebra F-S I (>) .
sublocale inter-S: omega-term-algebra F-S I
proof (unfold-locales, unfold inter-S.valid-monotone-inter-def, intro ballI)
  fix fn
  assume  $fn \in F-S$ 
  note  $F = \text{this}[\text{unfolded } F-S\text{-def } F\text{-def}]$ 
  show inter-S.valid-monotone-fun fn
    unfolding inter-S.valid-monotone-fun-def
  proof (intro allI impI, clarify)
    fix f n
    assume  $fn: fn = (f, n)$ 
    note  $\text{defs} = \text{valid-fun-def monotone-fun-wrt-def}$ 
    show  $\text{valid-fun } n (I f) \wedge \text{inter-S.monotone-fun } n (I f)$ 
    proof (cases f)
      case f: a-sym
      with  $F\ fn$  have  $n: n = 2$  by auto
      show ?thesis unfolding f n
      by (auto simp: defs dest!: length2-cases less2-cases)
    next
      case f: g-sym
      with  $F\ fn$  have  $n: n = 2$  by auto
      show ?thesis unfolding f n
      by (auto simp: defs dest!: length2-cases less2-cases)
      (smt (verit, ccfv-SIG) mult-mono')
    next
      case f: z-sym

```

```

with F fn have n: n = 0 by auto
show ?thesis unfolding f n
  by (auto simp: defs)
next
case f: o-sym
with F fn have n: n = 0 by auto
show ?thesis unfolding f n
  by (auto simp: defs intro!: insertion-positive-poly pq)
next
case f: f-sym
with F fn have n: n = 7 by auto
show ?thesis unfolding f n
  by (auto simp: defs intro!: add-le-less-mono mult-mono
    dest!: length7-cases less7-cases)
next
case f: (v-sym i)
with F fn have n: n = 1 by auto
show ?thesis unfolding f n
  by (auto simp: defs)
next
case f: q-sym
with F fn have n: n = 1 by auto
show ?thesis unfolding f n
  by (auto simp: defs dest: length1-cases)
next
case f: h-sym
with F fn have n: n = 1 by auto
show ?thesis unfolding f n
  by (auto simp: defs power2-eq-square dest!: length1-cases
    (insert mult-strict-mono', fastforce))
qed
qed
qed

```

Lemma 5.6

```

lemma S-is-omega-terminating: omega-termination F-S S
  unfolding omega-termination-def
proof (intro exI[of -] conjI)
  show omega-term-algebra F-S I ..
  show inter-S.termination-by-interpretation S
    unfolding inter-S.termination-by-interpretation-def S-def
  proof (clarify, intro conjI)
    show funas-term lhs-S  $\cup$  funas-term rhs-S  $\subseteq$  F-S using lhs-S-F rhs-S-F by
  auto
  show inter-S.orient-rule (lhs-S, rhs-S) unfolding inter-S.orient-rule-def split
  proof (intro allI impI)
    fix  $\alpha :: \text{var} \Rightarrow \text{int}$ 
    assume assignment  $\alpha$ 
    hence  $\alpha: \alpha x \geq 0$  for  $x$  unfolding assignment-def by auto

```

```

from  $\alpha$ [of  $y4$ ] obtain  $n4$  where  $n4: \alpha y4 = \text{int } n4$ 
  using nonneg-int-cases by blast
define  $q1$  where  $q1 = \text{insertion } (\lambda-. 1) q$ 
have  $q1: q1 \geq 0$  unfolding q1-def using pq(2)
  by (simp add: insertion-positive-poly)
define  $p1$  where  $p1 = \text{insertion } (\lambda-. 1) p$ 
have  $p1: p1 \geq 0$  unfolding p1-def using pq(1)
  by (simp add: insertion-positive-poly)
have [simp]:  $I[\text{foldr } (\lambda i t. \text{Fun } (v\text{-sym } i) [t]) xs t]\alpha = I[t]\alpha$  for  $xs t$ 
  by (induct xs, auto)
define  $l$  where  $l i = \text{args } (lhs\text{-}S) ! i$  for  $i$ 
define  $r$  where  $r i = \text{args } (rhs\text{-}S) ! i$  for  $i$ 
note defs = l-def r-def lhs-S-def rhs-S-def
have 1:  $I[l 0]\alpha \geq I[r 0]\alpha$  unfolding defs by auto
have 2:  $I[l 1]\alpha \geq I[r 1]\alpha$  unfolding defs by auto
have 5:  $I[l 4]\alpha \geq I[r 4]\alpha$  unfolding defs using  $\alpha$ [of  $y5$ ] by auto
  have 6:  $I[l 5]\alpha > I[r 5]\alpha$  unfolding defs using  $\alpha$ [of  $y6$ ] by (auto simp:
power2-eq-square)
  have 7:  $I[l 6]\alpha \geq I[r 6]\alpha$  unfolding defs using  $\alpha$ [of  $y7$ ]  $q1$ 
    by (auto simp: q1-def[symmetric] field-simps)

have  $n44: n4 * 4 = n4 + n4 + n4 + n4$  by simp
have  $r3: I[r 3]\alpha = 1 * 5^{(4 * n4)} + 14 * 5^{(3 * n4)} + 64 * 5^{(2 * n4)}$ 
+  $105 * 5^{n4} + 48 * 5^0$ 
  unfolding defs by (simp add: n4 field-simps power-mult power2-eq-square)
  (simp flip: power-add power-mult add: field-simps n44)
let  $?large = 125 * 5^{(n4^2 + 7 * n4)}$ 
have  $l3: I[l 3]\alpha = ?large + ?large + ?large + ?large + ?large$ 
unfolding defs by (simp add: n4 power2-eq-square nat-add-distrib nat-mult-distrib
power-add)
have 4:  $I[l 3]\alpha \geq I[r 3]\alpha$  unfolding  $l3 r3$ 
  by (intro add-mono mult-mono power-increasing, auto)

have  $I[r 2]\alpha * I[r 6]\alpha + I[r 2]\alpha$ 
=  $((q1 + 1) * \alpha y7 + q1 + 1) * \alpha y3$ 
  unfolding defs by (simp add: I-encode-poly[OF pq(2)] q1-def field-simps)
also have  $\dots \leq ((q1 + 1) * \alpha y7 + q1 + 1) * ((p1 + 1) * \alpha y3)$ 
  by (rule mult-left-mono, insert p1 q1  $\alpha$ , auto simp: field-simps)
also have  $\dots = I[l 2]\alpha * I[l 6]\alpha + I[l 2]\alpha$ 
  unfolding defs by (simp add: I-encode-poly[OF pq(1)] q1-def p1-def
field-simps)
finally have 37:  $I[l 2]\alpha * I[l 6]\alpha + I[l 2]\alpha \geq I[r 2]\alpha * I[r 6]\alpha + I[r 2]\alpha$ 
.

have lhs: lhs-S = Fun f-sym (map l [0,1,2,3,4,5,6]) unfolding lhs-S-def l-def
by simp
  have rhs: rhs-S = Fun f-sym (map r [0,1,2,3,4,5,6]) unfolding rhs-S-def
r-def by simp
  have  $I[rhs\text{-}S]\alpha = (I[r 2]\alpha * I[r 6]\alpha + I[r 2]\alpha) +$ 

```

```

      (I[[r 0]]α + I[[r 1]]α + I[[r 3]]α + I[[r 4]]α + I[[r 6]]α) + I[[r 5]]α
    unfolding rhs by simp
  also have ... < (I[[l 2]]α * I[[l 6]]α + I[[l 2]]α) +
    (I[[l 0]]α + I[[l 1]]α + I[[l 3]]α + I[[l 4]]α + I[[l 6]]α) + I[[l 5]]α
  apply (rule add-le-less-mono[OF - 6])
  apply (rule add-mono[OF 37])
  by (intro add-mono 1 2 4 5 7)
  also have ... = I[[lhs-S]]α unfolding lhs by simp
  finally show I[[lhs-S]]α > I[[rhs-S]]α .
qed
qed
qed
end

end

```

8 Undecidability of Polynomial Termination using δ -Orders

theory *Delta-Poly-Termination-Undecidable*

imports

Poly-Termination-Undecidable

begin

context *poly-input*

begin

definition *y8* :: *var* **where** *y8* = 7

definition *y9* :: *var* **where** *y9* = 8

Definition 6.3

definition *lhs-Q* = *Fun f-sym* [

q-t (*h-t* (*Var y1*)),
h-t (*Var y2*),
h-t (*Var y3*),
g-t (*q-t* (*Var y4*)) (*h-t* (*h-t* (*h-t* (*Var y4*)))),
q-t (*Var y5*),
a-t (*Var y6*) (*Var y6*),
Var y7,
Var y8,
h-t (*a-t* (*encode-poly y9 p*) (*Var y9*))]

fun *g-list* :: - \Rightarrow (*symbol, var*)*term* **where**

g-list [] = *z-t*

| *g-list* ((*f, n*) # *fs*) = *g-t* (*Fun f* (*replicate n z-t*)) (*g-list fs*)

definition *symbol-list* **where** *symbol-list* = [(*f-sym, 9*), (*q-sym, 1*), (*h-sym, 1*), (*a-sym, 2*)]

@ *map* (λ *i. (v-sym i, 1)*) *V-list*

definition $t-t :: (\text{symbol}, \text{var})\text{term}$ **where** $t-t = (g\text{-list } ((z\text{-sym}, 0) \# \text{symbol-list}))$

definition $\text{rhs-}Q = \text{Fun } f\text{-sym } [$
 $h-t (h-t (q-t (\text{Var } y1))),$
 $g-t (\text{Var } y2) (\text{Var } y2),$
 $\text{Fun } f\text{-sym } (\text{replicate } 9 (\text{Var } y3)),$
 $q-t (g-t (\text{Var } y4) t-t),$
 $a-t (\text{Var } y5) (\text{Var } y5),$
 $q-t (\text{Var } y6),$
 $a-t z-t (\text{Var } y7),$
 $a-t (\text{Var } y8) z-t,$
 $a-t (\text{encode-poly } y9 q) (\text{Var } y9)]$

definition Q **where** $Q = \{(\text{lhs-}Q, \text{rhs-}Q)\}$

definition $F-Q$ **where** $F-Q = \{(f\text{-sym}, 9), (h\text{-sym}, 1), (g\text{-sym}, 2), (q\text{-sym}, 1)\} \cup F$

lemma $\text{lhs-}Q\text{-}F$: $\text{funas-term } \text{lhs-}Q \subseteq F-Q$

proof –

from $\text{funas-encode-poly-p}$

show $\text{funas-term } \text{lhs-}Q \subseteq F-Q$ **unfolding** $\text{lhs-}Q\text{-def}$ **by** $(\text{auto simp: } F-Q\text{-def } F\text{-def})$

qed

lemma $g\text{-list-}F$: $\text{set } zs \subseteq F-Q \implies \text{funas-term } (g\text{-list } zs) \subseteq F-Q$

proof $(\text{induct } zs)$

case Nil

thus $?case$ **by** $(\text{auto simp: } F-Q\text{-def } F\text{-def})$

next

case $(\text{Cons } fa \ ts)$

then obtain $f \ a$ **where** $fa: fa = (f, a)$ **and** $\text{in}F: (f, a) \in F-Q$ **by** $(\text{cases } fa, \text{auto})$

have $\{(g\text{-sym}, \text{Suc } (\text{Suc } 0)), (z\text{-sym}, 0)\} \subseteq F-Q$ **by** $(\text{auto simp: } F-Q\text{-def } F\text{-def})$

with $\text{Cons } fa \ \text{in}F$ **show** $?case$ **by** auto

qed

lemma symbol-list : $\text{set } \text{symbol-list} \subseteq F-Q$ **unfolding** symbol-list-def $F-Q\text{-def}$ $F\text{-def}$ **using** $V\text{-list}$ **by** auto

lemma $t\text{-}F$: $\text{funas-term } t-t \subseteq F-Q$

unfolding $t-t\text{-def}$ **using** $g\text{-list-}F[OF \ \text{symbol-list}]$

by $(\text{auto simp: } F-Q\text{-def } F\text{-def})$

lemma $\text{vars-}g\text{-list}[simp]$: $\text{vars-term } (g\text{-list } zs) = \{\}$

by $(\text{induct } zs, \text{auto})$

lemma $\text{vars-}t$: $\text{vars-term } t-t = \{\}$

unfolding $t-t\text{-def}$ **by** simp

```

lemma rhs-Q-F: funas-term rhs-Q  $\subseteq$  F-Q
proof –
  from funas-encode-poly-q
  show funas-term rhs-Q  $\subseteq$  F-Q unfolding rhs-Q-def using t-F by (auto simp:
F-Q-def F-def)
qed

context
  fixes I :: symbol  $\Rightarrow$  'a :: linordered-field mpoly and  $\delta$  :: 'a and a3 a2 a1 a0 z0 v
  assumes I: I a-sym = Const a3 * PVar 0 * PVar 1 + Const a2 * PVar 0 +
Const a1 * PVar 1 + Const a0
  I z-sym = Const z0
  I (v-sym i) = mpoly-of-poly 0 (v i)
  and a: a3 > 0 a2 > 0 a1 > 0 a0  $\geq$  0
  and z: z0  $\geq$  0
  and v: nneg-poly (v i) degree (v i) > 0
begin

lemma nneg-combination: assumes nneg-poly r
  shows nneg-poly ([:a1, a3:] * r + [:a0, a2:])
  by (intro nneg-poly-add nneg-poly-mult assms, insert a, auto)

lemma degree-combination: assumes nneg-poly r
  shows degree ([:a1, a3:] * r + [:a0, a2:]) = Suc (degree r)
  using nneg-poly-degree-add-1[OF assms, OF a(1) a(2)] by auto

lemma degree-eval-encode-num: assumes c: c  $\geq$  0
  shows  $\exists$  p. mpoly-of-poly x p = poly-inter.eval I (encode-num x c)  $\wedge$  nneg-poly
p  $\wedge$  int (degree p) = c
proof –
  interpret poly-inter UNIV I .
  from assms obtain n where cn: c = int n by (metis nonneg-eq-int)
  hence natc: nat c = n by auto
  note [simp] = I
  show ?thesis unfolding encode-num-def natc unfolding cn int-int-eq
proof (induct n)
  case 0
  show ?case using z by (auto simp: intro!: exI[of - [:z0:]])
next
  case (Suc n)
  define t where t = ((( $\lambda$ t. Fun a-sym [TVar x, t])  $\widehat{\sim}$  n) (Fun z-sym []))
  from Suc obtain p where mp: mpoly-of-poly x p = eval t
  and deg: degree p = n and p: nneg-poly p by (auto simp: t-def)
  show ?case apply (simp add: t-def[symmetric])
  apply (unfold deg[symmetric])
  apply (intro exI[of - [: a1, a3:] * p + [:a0, a2:]] conjI mpoly-extI de-
gree-combination p nneg-combination)
  by (simp add: mp insertion-add insertion-mult field-simps)

```

qed
qed

lemma *degree-eval-encode-monom*: **assumes** $c: c > 0$
and $\alpha: \alpha = (\lambda i. \text{int } (\text{degree } (v \ i)))$
shows $\exists p. \text{mpoly-of-poly } y \ p = \text{poly-inter.eval } I \ (\text{encode-monom } y \ m \ c) \wedge \text{nneg-poly } p \wedge$
 $\text{int } (\text{degree } p) = \text{insertion } \alpha \ (\text{mmonom } m \ c) \wedge \text{degree } p > 0$
proof –
interpret *poly-inter UNIV I* .
define *xes* **where** $xes = \text{var-list } m$
from *var-list*[of $m \ c$]
have *monom*: $\text{mmonom } m \ c = \text{Const } c * (\prod (x, e) \leftarrow xes . \text{PVar } x \hat{\ } e)$ **unfolding**
xes-def .
show *?thesis* **unfolding** *encode-monom-def monom xes-def*[*symmetric*]
proof (*induct xes*)
case *Nil*
show *?case* **using** *degree-eval-encode-num*[of $c \ y$] c **by** *auto*
next
case (*Cons xe xes*)
obtain $x \ e$ **where** $xe = (x, e)$ **by** *force*
define *expr* **where** $\text{expr} = \text{rec-list } (\text{encode-num } y \ c) \ (\lambda a. \text{case } a \ \text{of } (i, e) \Rightarrow$
 $\lambda t. (\lambda t. \text{Fun } (v\text{-sym } i) [t]) \hat{\ } e)$
define *exes* **where** $\text{exes} = \text{expr } xes$
define *ixes* **where** $\text{ixes} = \text{insertion } \alpha \ (\text{Const } c * (\prod a \leftarrow xes. \text{case } a \ \text{of } (x, a)$
 $\Rightarrow \text{PVar } x \hat{\ } a))$
have *step*: $\text{expr } (xe \ \# \ xes) = ((\lambda t. \text{Fun } (v\text{-sym } x) [t]) \hat{\ } e) \ (\text{exes})$
unfolding *xe expr-def exes-def* **by** *auto*
have *step'*: $\text{insertion } \alpha \ (\text{Const } c * (\prod a \leftarrow xe \ \# \ xes. \text{case } a \ \text{of } (x, a) \Rightarrow \text{PVar } x$
 $\hat{\ } a))$
 $= (\alpha \ x) \hat{\ } e * \text{ixes}$
unfolding *xe ixes-def* **by** (*simp add: insertion-mult insertion-power*)
from *Cons(1)*[*folded expr-def exes-def ixes-def*] **obtain** p **where**
 $IH: \text{mpoly-of-poly } y \ p = \text{eval } \text{exes} \ \text{nneg-poly } p$
 $\text{int } (\text{degree } p) = \text{ixes } \text{degree } p > 0$
by *auto*
show *?case*
unfolding *expr-def*[*symmetric*]
unfolding *step step'*
proof (*induct e*)
case 0
thus *?case* **using** *IH* **by** *auto*
next
case (*Suc e*)
define *rec* **where** $\text{rec} = ((\lambda t. \text{Fun } (v\text{-sym } x) [t]) \hat{\ } e) \ \text{exes}$
from *Suc*[*folded rec-def*] **obtain** p **where**
 $IH: \text{mpoly-of-poly } y \ p = \text{eval } \text{rec} \ \text{nneg-poly } p \ \text{int } (\text{degree } p) = \alpha \ x \hat{\ } e * \text{ixes}$
 $\text{degree } p > 0$ **by** *auto*
have $((\lambda t. \text{Fun } (v\text{-sym } x) [t]) \hat{\ } \text{Suc } e) \ \text{exes} = \text{Fun } (v\text{-sym } x) [rec]$

unfolding *rec-def* **by** *simp*
also have $eval \dots = substitute (\lambda i. \text{if } i = 0 \text{ then } eval ([rec] ! i) \text{ else } 0)$
(poly-to-mpoly 0 (v x))
by *(simp add: I mpoly-of-poly-is-poly-to-mpoly)*
also have $\dots = poly\text{-to-mpoly } y (v x \circ_p p)$
by *(rule substitute-poly-to-mpoly, auto simp: IH(1)[symmetric] mpoly-of-poly-is-poly-to-mpoly)*
finally have $id: eval (((\lambda t. Fun (v\text{-sym } x) [t]) \widehat{\sim} Suc e) \text{ exes}) = poly\text{-to-mpoly}$
 $y (v x \circ_p p)$.
show *?case* **unfolding** *id mpoly-of-poly-is-poly-to-mpoly*
proof *(intro exI[of - v x \circ_p p] conjI refl)*
show $int (degree (v x \circ_p p)) = \alpha x \widehat{\sim} Suc e * ixes$
unfolding *degree-pcompose* **using** *IH(3)* **by** *(auto simp: \alpha)*
show *nneg-poly (v x \circ_p p)* **using** *IH(2) v[of x]*
by *(intro nneg-poly-pcompose, insert IH, auto)*
show $0 < degree (v x \circ_p p)$ **unfolding** *degree-pcompose* **using** *IH(4) v[of*
x] **by** *auto*
qed
qed
qed
qed

Lemma 6.2

lemma *degree-eval-encode-poly-generic*: **assumes** *positive-poly r*
and $\alpha: \alpha = (\lambda i. int (degree (v i)))$
shows $\exists p. poly\text{-to-mpoly } x p = poly\text{-inter.eval } I (encode\text{-poly } x r) \wedge nneg\text{-poly } p$
 \wedge
 $int (degree p) = insertion \alpha r$
proof –
interpret *poly-inter UNIV I* .
define *mcs* **where** $mcs = monom\text{-list } r$
from *monom-list[of r]* **have** $r: r = (\sum (m, c) \leftarrow mcs. mmonom m c)$ **unfolding**
mcs-def **by** *auto*
{
fix $m c$
assume $mc: (m, c) \in set mcs$
hence $c \geq 0$
using *monom-list-coeff assms* **unfolding** *mcs-def positive-poly-def* **by** *auto*
moreover from *mc* **have** $c \neq 0$ **unfolding** *mcs-def*
by *(transfer, auto)*
ultimately have $c > 0$ **by** *auto*
} **note** $mcs = this$
note $[simp] = I$
show *?thesis* **unfolding** *encode-poly-def mcs-def[symmetric]* **unfolding** *r inser-*
tion-sum-list map-map o-def
unfolding *mpoly-of-poly-is-poly-to-mpoly[symmetric]*
using *mcs*
proof *(induct mcs)*
case Nil
show *?case* **by** *(rule exI[of - [:z0:]], insert z, auto)*


```

next
  case (Cons mc mcs)
  define trm where trm = rec-list (Fun z-sym []) (λa. case a of (m, c) ⇒ λ- t.
Fun a-sym [encode-monom x m c, t])
  define expr where expr mcs = (∑ x←mcs. insertion α (case x of (x, xa) ⇒
mmonom x xa)) for mcs
  obtain m c where mc: mc = (m, c) by force
  from Cons(2) mc have c: c > 0 by auto
  from degree-eval-encode-monom[OF this α, of x m]
  obtain q where monom: mpoly-of-poly x q = eval (encode-monom x m c)
    nneg-poly q int (degree q) = insertion α (mmonom m c)
    and dq: degree q > 0 by auto
  from Cons(1)[folded trm-def expr-def, OF Cons(2)]
  obtain p where IH: mpoly-of-poly x p = eval (trm mcs) nneg-poly p int (degree
p) = expr mcs by force
  have step: trm (mc # mcs) = Fun a-sym [encode-monom x m c, trm mcs]
    unfolding mc trm-def by simp
  have step': expr (mc # mcs) = insertion α (mmonom m c) + expr mcs
  unfolding mc expr-def by simp
  have deg: degree ([:a3:] * q * p + ([:a2:] * q + [:a1:] * p + [:a0:])) = degree p
+ degree q
  by (rule nneg-poly-degree-add, insert a IH monom, auto)
  show ?case unfolding expr-def[symmetric] trm-def[symmetric]
    unfolding step step'
    unfolding IH(3)[symmetric] monom(3)[symmetric]
    apply (intro exI[of - [:a3:] * q * p + [:a2:] * q + [:a1:] * p + [:a0:]] conjI)
    subgoal by (intro mpoly-extI, simp add: IH(1)[symmetric] monom(1)[symmetric]
insertion-mult insertion-add)
    subgoal by (intro nneg-poly-mult nneg-poly-add IH monom, insert a, auto)
    subgoal using deg by (auto simp: ac-simps)
  done
qed
qed
end
end

```

```

context delta-poly-inter
begin

```

```

lemma transp-gt-delta: transp (λ x y. x ≥ y + δ) using δ0
  by (auto simp: transp-def)

```

```

lemma gt-delta-imp-ge: y + δ ≤ x ⇒ y ≤ x using δ0 by auto

```

```

lemma weakly-monotone-insertion: assumes mono: monotone-poly (vars p) p
  and a: assignment (a :: - ⇒ 'a)
  and gt: ∧ x. x ∈ vars p ⇒ a x + δ ≤ b x
shows insertion a p ≤ insertion b p
  using monotone-poly-wrt-insertion[OF transp-gt-delta gt-delta-imp-ge mono a, of

```

b] *gt* $\delta 0$ **by** *auto*

Lemma 6.5

lemma *degree-partial-insertion-stays-constant*: **assumes** *mono*: *monotone-poly* (*vars* *p*) *p*

shows $\exists a. \text{assignment } a \wedge (\forall b. (\forall y. a \ y + \delta \leq b \ y) \longrightarrow \text{degree } (\text{partial-insertion } a \ x \ p) = \text{degree } (\text{partial-insertion } b \ x \ p))$

using *degree-partial-insertion-stays-constant-generic*

[*OF transp-gt-delta gt-delta-imp-ge poly-pinfty-ge mono, of δ x, simplified*]

by *metis*

lemma *degree-mono*: **assumes** *pos*: *lead-coeff* *p* $\geq (0 :: 'a)$

and *le*: $\bigwedge x. x \geq c \implies \text{poly } p \ x \leq \text{poly } q \ x$

shows *degree* *p* \leq *degree* *q*

by (*rule degree-mono-generic*[*OF poly-pinfty-ge assms*])

lemma *degree-mono'*: **assumes** $\bigwedge x. x \geq c \implies (\text{bnd } :: 'a) \leq \text{poly } p \ x \wedge \text{poly } p \ x \leq \text{poly } q \ x$

shows *degree* *p* \leq *degree* *q*

by (*rule degree-mono'-generic*[*OF poly-pinfty-ge assms*])

Lemma 6.6

lemma *subst-same-var-monotone-imp-same-degree*:

assumes *mono*: *monotone-poly* (*vars* *p*) (*p* $:: 'a \ \text{mpoly}$)

and *qp*: *poly-to-mpoly* *x* *q* = *substitute* ($\lambda i. \text{PVar } x$) *p*

shows *total-degree* *p* = *degree* *q*

proof (*cases total-degree* *p* = 0)

case *False*

from *False* **have** *p0*: *p* $\neq 0$ **by** *auto*

obtain *d* **where** *dq*: *degree* *q* = *d* **by** *blast*

let *?mc* = ($\lambda m. \text{mmonom } m \ (\text{mcoeff } p \ m)$)

let *?cfs* = $\{m \ . \ \text{mcoeff } p \ m \neq 0\}$

let *?lc* = *lead-coeff*

note *fin* = *finite-coeff-support*[*of* *p*]

define *M* **where** *M* = *total-degree* *p*

with *False* **have** *M1*: *M* ≥ 1 **by** *auto*

from *degree-monom-eq-total-degree*[*OF* *p0*]

obtain *mM* **where** *mM*: *mcoeff* *p* *mM* $\neq 0$ *degree-monom* *mM* = *M* **unfolding**

M-def **by** *blast*

from *degree-substitute-same-var*[*of* *x* *p*, *folded* *M-def* *qp*]

have *dM*: *d* \leq *M* **unfolding** *dq* *degree-poly-to-mpoly* .

define *p1* **where** *p1* = *sum* *?mc* (*?cfs* \cap $\{m. \text{degree-monom } m = M\}$)

define *p2* **where** *p2* = *sum* *?mc* (*?cfs* \cap $\{m. \text{degree-monom } m < M\}$)

have *p* = *sum* *?mc* *?cfs*

by (*rule mpoly-as-sum*)

also have *?cfs* = *?cfs* \cap $\{m. \text{degree-monom } m = M\}$

\cup *?cfs* \cap $\{m. \text{degree-monom } m \neq M\}$ **by** *auto*

also have *?cfs* \cap $\{m. \text{degree-monom } m \neq M\}$ = *?cfs* \cap $\{m. \text{degree-monom } m <$

$M\}$
using *degree-mono-le-total-degree*[of p , folded M -def] **by force**
also have $\text{sum } ?mc \ (\?cfs \cap \{m. \text{degree-monom } m = M\} \cup \dots) = p1 + p2$
unfolding $p1$ -def $p2$ -def
using *fin* **by** (*intro sum.union-disjoint*, *auto*)
finally have p -split: $p = p1 + p2$.
have *total-degree* $p2 \leq M - 1$ **unfolding** $p2$ -def
by (*intro total-degree-sum-leI*, *subst total-degree-monom*, *auto*)
also have $\dots < M$ **using** $M1$ **by** *auto*
finally have $\text{deg-}p'$: *total-degree* $p2 < M$ **by** *auto*
have $p1 \neq 0$
proof
assume $p1 = 0$
hence $p = p2$ **unfolding** p -split **by** *auto*
hence $M = \text{total-degree } p2$ **unfolding** M -def **by** *simp*
with $\text{deg-}p'$ **show** *False* **by** *auto*
qed
with *mpoly-ext-bounded-field*[of $\max 1 \delta p1 0$] **obtain** b
where $b: \bigwedge v. b \ v \geq \max 1 \delta$ **and** $\text{bpm0}: \text{insertion } b \ p1 \neq 0$ **by** *auto*
from b **have** $b1: \bigwedge v. b \ v \geq 1$ **and** $b\delta: \bigwedge v. b \ v \geq \delta$ **by** *auto*
define c **where** $c = \text{Max } (\text{insert } 1 \ (b \ \text{'vars } p)) + \delta$
define X **where** $X = (0 :: \text{nat})$
define pb **where** $pb \ p = \text{mpoly-to-poly } X \ (\text{substitute } (\lambda v. \text{Const } (b \ v) * \text{PVar } X) \ p)$ **for** p
have $c1: c \geq 1$ **unfolding** c -def **using** *vars-finite*[of p] $\delta 0$ *Max-ge*[of $- 1 :: 'a$]
by (*meson add-increasing2 finite.insertI finite-imageI insertI1 nless-le*)
have $\text{vars}X: \text{vars } (\text{substitute } (\lambda v. \text{Const } (b \ v) * \text{PVar } X) \ p) \subseteq \{X\}$ **for** p
by (*intro vars-substitute order.trans[OF vars-mult]*, *auto*)
have $pb: \text{substitute } (\lambda v. \text{Const } (b \ v) * \text{PVar } X) \ p = \text{poly-to-mpoly } X \ (pb \ p)$ **for**
 p
unfolding pb -def
by (*rule mpoly-to-poly-inverse[symmetric, OF varsX]*)
have $\text{poly-pb}: \text{poly } (pb \ p) \ x = \text{insertion } (\lambda v. b \ v * x) \ p$ **for** $x \ p$
using *arg-cong*[OF pb , of *insertion* $(\lambda -. x)$,
unfolded insertion-poly-to-mpoly]
by (*auto simp: insertion-substitute insertion-mult*)
define lb **where** $lb = \text{insertion } (\lambda -. 0) \ p$
{
fix x
have $\text{poly } (pb \ p) \ x = \text{insertion } (\lambda v. b \ v * x) \ p$ **by** *fact*
also have $\dots = \text{insertion } (\lambda v. b \ v * x) \ p1 + \text{insertion } (\lambda v. b \ v * x) \ p2$
unfolding p -split
by (*simp add: insertion-add*)
also have $\text{insertion } (\lambda v. b \ v * x) \ p1 = \text{insertion } b \ p1 * x^{\wedge}M$
unfolding $p1$ -def *insertion-sum insertion-mult insertion-monom sum-distrib-right*

power-mult-distrib
proof (*intro sum.cong[OF refl]*, *goal-cases*)
case $(1 \ m)$

```

from 1 have M: M = degree-monom m by auto
have { v. lookup m v ≠ 0 } ⊆ keys m
  by (simp add: keys.rep-eq)
from finite-subset[OF this] have fin: finite { v. lookup m v ≠ 0 } by auto
have (∏ v. b v ^ lookup m v * x ^ lookup m v)
  = (∏ v. b v ^ lookup m v) * (∏ v. x ^ lookup m v)
  by (subst (1 2 3) Prod-any.expand-superset[OF fin])
  (insert zero-less-iff-neq-zero, force simp: prod.distrib)+
also have (∏ v. x ^ lookup m v) = x ^ M unfolding M degree-monom-def
  by (smt (verit) Prod-any.conditionalize Prod-any.cong finite-keys in-keys-iff
power-0 power-sum)
  finally show ?case by simp
qed
also have insertion (λv. b v * x) p2 = poly (pb p2) x unfolding poly-pb ..
finally have poly (pb p) x = poly (monom (insertion b p1) M + pb p2) x by
(simp add: poly-monom)
}
hence pbp-split: pb p = monom (insertion b p1) M + pb p2 by blast
have degree (pb p2) ≤ total-degree p2 unfolding pb-def
  apply (subst degree-mpoly-to-poly)
  apply (simp add: varsX)
  by (rule degree-substitute-const-same-var)
also have ... < M by fact
finally have deg-pbp2: degree (pb p2) < M .
  have degree (monom (insertion b p1) M) = M using bpm0 by (rule de-
gree-monom-eq)
with deg-pbp2 pbp-split have deg-pbp: degree (pb p) = M unfolding pbp-split
  by (subst degree-add-eq-left, auto)
have ?lc (pb p) = insertion b p1 unfolding pbp-split
  using deg-pbp2 bpm0 coeff-eq-0 deg-pbp pbp-split by auto
define bnd where bnd = insertion (λ -. 0) p

{
  fix x :: 'a
  assume x1: x ≥ 1
  hence x: x ≥ 0 by simp
  have ass: assignment (λ v. b v * x) unfolding assignment-def using x b1
    by (meson linorder-not-le mult-le-cancel-right1 order-trans)
  have poly (pb p) x = insertion (λv. b v * x) p by fact
  also have insertion (λ v. b v * x) p ≤ insertion (λ v. c * x) p
  proof (rule weakly-monotone-insertion[OF mono ass])
    fix v
    assume v: v ∈ vars p
    have b v + δ ≤ c unfolding c-def using vars-finite[of p] v Max-ge[of - b v]
by auto
    thus b v * x + δ ≤ c * x using b[of v] x1 c1 δ 0
    by (smt (verit) c-def add-le-imp-le-right add-mono comm-semiring-class.distrib
mult.commute mult-le-cancel-right1 mult-right-mono order.asym x)
  qed
}

```

```

also have ... = poly q (c * x) unfolding poly-to-mpoly-substitute-same[OF qp]
..
also have ... = poly (q ∘p [:0, c:]) x by (simp add: poly-pcompose ac-simps)
finally have ineq: poly (pb p) x ≤ poly (q ∘p [:0, c:]) x .
have bnd ≤ insertion (λv. b v * x) p unfolding bnd-def
  apply (intro weakly-monotone-insertion[OF mono])
  subgoal by (simp add: assignment-def)
  subgoal for v using bδ[of v] x1 δ0
    by simp (metis dual-order.trans less-le-not-le mult-le-cancel-left1)
  done
also have ... = poly (pb p) x using poly-pb by auto
finally have bnd ≤ poly (pb p) x by auto
note this ineq
} note pb-approx = this
have M = degree (pb p) unfolding deg-pbp ..
also have ... ≤ degree (q ∘p [:0, c:])
  by (intro degree-mono'[of 1 bnd], insert pb-approx, auto)
also have ... ≤ d by (simp add: dq)
finally have deg-pbp: M ≤ d .
with dM have M = d by auto
thus ?thesis unfolding M-def dq .
next
case True
then obtain c where p: p = Const c using degree-0-imp-Const by blast
with qp have poly-to-mpoly x q = p by auto
thus ?thesis
  by (metis True degree-Const degree-poly-to-mpoly p)
qed

lemma monotone-poly-partial-insertion:
  assumes x: x ∈ xs
  and mono: monotone-poly xs p
  and ass: assignment a
shows 0 < degree (partial-insertion a x p)
  lead-coeff (partial-insertion a x p) > 0
  valid-poly p ⇒ y ≥ 0 ⇒ poly (partial-insertion a x p) y ≥ y - δ
  valid-poly p ⇒ insertion a p ≥ a x - δ
proof -
  have 0: 1 ≤ inverse δ * δ using δ0 by auto
  define ceil-nat :: 'a ⇒ nat where ceil-nat x = nat (ceiling x) for x
  have 1: x ≤ of-nat (ceil-nat x) for x unfolding ceil-nat-def
    by (simp add: of-nat-ceiling)
  note main = monotone-poly-partial-insertion-generic[OF transp-gt-delta gt-delta-imp-ge
poly-pinfy-ge refl δ0 0 1 x mono ass, simplified]
  show 0 < degree (partial-insertion a x p) 0 < lead-coeff (partial-insertion a x p)

  using main by auto
  assume valid: valid-poly p
  from main(3)[OF this] have estimation: δ * of-nat y ≤ poly (partial-insertion a

```

```

x p) ( $\delta * \text{of-nat } y$ ) for y by auto
{
  fix y :: 'a
  assume y:  $y \geq 0$ 
  with ass have ass': assignment (a(x := y)) unfolding assignment-def by auto
  from valid[unfolded valid-poly-def, rule-format, OF ass]
  have ge0: insertion (a(x := y))  $p \geq 0$  by auto
  have id: poly (partial-insertion a x p) y = insertion (a(x := y)) p
    using insertion-partial-insertion[of x a a(x:=y) p] by auto
  show  $y - \delta \leq \text{poly}$  (partial-insertion a x p) y
  proof (cases  $y \geq \delta$ )
    case False
    with ge0[folded id] y show ?thesis by auto
  next
  case True
  define z where  $z = y - \delta$ 
  from True have z0:  $z \geq 0$  unfolding z-def by auto
  define n where  $n = \text{nat}$  (floor (z * inverse  $\delta$ ))
  have  $\delta * \text{of-nat } n \leq z$  unfolding n-def using  $\delta 0 z 0$ 
    by (metis field-class.field-divide-inverse mult-of-nat-commute mult-zero-left
of-nat-floor pos-le-divide-eq)
  hence  $gt: \delta * \text{of-nat } n + \delta \leq y$  unfolding z-def by auto

  define b where  $b = a(x := \delta * \text{of-nat } n)$ 
  have ass-b: assignment b using  $\delta 0$  ass unfolding b-def assignment-def by
auto
  from mono[unfolded monotone-poly-wrt-def, rule-format, OF ass-b x, of y] gt
  have  $gt: \text{insertion } b p \leq \text{insertion } (b(x := y)) p - \delta$  by (auto simp: b-def)

  have  $\delta * \text{of-nat } n + \delta \geq z$  unfolding n-def using  $\delta 0 z 0$ 
  by (smt (verit, del-insts) comm-semiring-class.distrib field-class.field-divide-inverse
floor-divide-upper inverse-nonnegative-iff-nonnegative mult.commute mult-cancel-left2
mult-nonneg-nonneg of-nat-nat order-less-le z-def z-def z-def zero-le-floor)
  hence  $y - 2 * \delta \leq \delta * \text{of-nat } n$  unfolding z-def by auto
  also have  $\delta * \text{of-nat } n \leq \text{poly}$  (partial-insertion a x p) ( $\delta * \text{of-nat } n$ )
    by fact
  also have  $\dots = \text{insertion } b p$  using insertion-partial-insertion[of x a b p]
    by (auto simp: b-def)
  also have  $\dots \leq \text{insertion } (b(x := y)) p - \delta$  by fact
  also have insertion (b(x := y)) p = poly (partial-insertion a x p) y
    using insertion-partial-insertion[of x a b(x := y) p]
    by (auto simp: b-def)
  finally show ?thesis by simp
  qed
} note estimation = this
from ass have  $a x \geq 0$  unfolding assignment-def by auto
from estimation[OF this] show insertion a p  $\geq a x - \delta$ 
  using insertion-partial-insertion[of x a a p] by auto
qed

```

end

context *solvable-poly-problem*
begin

context
 assumes *SORT-CONSTRAINT('a :: floor-ceiling)*
begin

context
 fixes *h :: 'a*
begin

fun *IQ :: symbol* \Rightarrow *'a mpoly* **where**
 IQ f-sym = *PVar 0 + PVar 1 + PVar 2 + PVar 3 + PVar 4 + PVar 5 + PVar 6 + PVar 7 + PVar 8*
 | *IQ a-sym* = *PVar 0 * PVar 1 + PVar 0 + PVar 1*
 | *IQ z-sym* = *0*
 | *IQ (v-sym i)* = *PVar 0 ^ (nat (α i))*
 | *IQ q-sym* = *PVar 0 * PVar 0 + Const 2 * PVar 0*
 | *IQ g-sym* = *PVar 0 + PVar 1*
 | *IQ h-sym* = *Const h * PVar 0 + Const h*
 | *IQ o-sym* = *0*

interpretation *interQ*: *poly-inter F-Q IQ* ($\lambda x y. x \geq y + (1 :: 'a)$) .

Lemma 6.2 specialized for this interpretation

lemma *degree-eval-encode-poly*: **assumes** *positive-poly r*
 shows $\exists p. \text{poly-to-mpoly } y9 p = \text{interQ.eval (encode-poly } y9 r) \wedge \text{nneg-poly } p \wedge \text{int (degree } p) = \text{insertion } \alpha r$

proof –

define *v* **where** *v i* = (*monom 1 (nat (α i)) :: 'a poly*) **for** *i*
 define γ **where** $\gamma = (\lambda i. \text{int (degree (v i))})$
 have *nneg-v*: *nneg-poly (v i) 0 < degree (v i)* **for** *i* **unfolding** *v-def* **using** $\alpha 1$ [*of i*]
 by (*auto simp: nneg-poly-def degree-monom-eq poly-monom*)
 have *id*: *int (Polynomial.degree (v i)) = α i* **for** *i* **unfolding** *v-def*
 using $\alpha 1$ [*of i*] **by** (*auto simp: nneg-poly-def degree-monom-eq*)
 have *IQ (v-sym i) = mpoly-of-poly 0 (v i)* **for** *i*
 unfolding *v-def* **by** (*intro mpoly-extI, simp add: insertion-power poly-monom*)
 from *degree-eval-encode-poly-generic*[*of IQ 1 1 1 0 0 v - γ , OF - - this, simplified, OF nneg-v assms γ -def, unfolded id*]
 show *?thesis* **by** *auto*
qed

definition *pp* **where** *pp* = (*SOME pp. poly-to-mpoly y9 pp = interQ.eval (encode-poly y9 p) \wedge nneg-poly pp \wedge int (degree pp) = insertion α p*)

lemma *pp*: $\text{inter}Q.\text{eval} (\text{encode-poly } y9\ p) = \text{poly-to-mpoly } y9\ pp$
 $\text{nneg-poly } pp\ \text{int} (\text{degree } pp) = \text{insertion } \alpha\ p$
using *someI-ex*[*OF degree-eval-encode-poly*[*OF pq(1)*], *folded pp-def*] **by** *auto*

definition *qq* **where** $qq = (\text{SOME } qq.\ \text{poly-to-mpoly } y9\ qq = \text{inter}Q.\text{eval} (\text{encode-poly } y9\ q) \wedge \text{nneg-poly } qq \wedge \text{int} (\text{degree } qq) = \text{insertion } \alpha\ q)$

lemma *qq*: $\text{inter}Q.\text{eval} (\text{encode-poly } y9\ q) = \text{poly-to-mpoly } y9\ qq$
 $\text{nneg-poly } qq\ \text{int} (\text{degree } qq) = \text{insertion } \alpha\ q$
using *someI-ex*[*OF degree-eval-encode-poly*[*OF pq(2)*], *folded qq-def*] **by** *auto*

definition *ppp* = $pp * [:1,1:] + [:0,1:]$
definition *qqq* = $qq * [:1,1:] + [:0,1:]$

lemma *degree-ppp*: $\text{int} (\text{degree } ppp) = 1 + \text{insertion } \alpha\ p$
unfolding *ppp-def* *pp(3)*[*symmetric*]
using *nneg-poly-degree-add-1*[*OF pp(2)*, *of 1 1 1 0*] **by** *simp*

lemma *degree-qqq*: $\text{int} (\text{degree } qqq) = 1 + \text{insertion } \alpha\ q$
unfolding *qqq-def* *qq(3)*[*symmetric*]
using *nneg-poly-degree-add-1*[*OF qq(2)*, *of 1 1 1 0*] **by** *simp*

lemma *ppp-qqq*: $\text{degree } ppp \geq \text{degree } qqq$
using *degree-ppp degree-qqq* $\alpha(2)$ **by** *auto*

lemma *nneg-ppp*: $\text{nneg-poly } ppp$
unfolding *ppp-def*
by (*intro nneg-poly-add nneg-poly-mult pp, auto*)

definition *H* **where** $H = (\text{SOME } H.\ \forall h \geq H.\ \forall x \geq 0.\ \text{poly } qqq\ x \leq h * \text{poly } ppp\ x + h)$

lemma *H*: $h \geq H \implies x \geq 0 \implies \text{poly } qqq\ x \leq h * \text{poly } ppp\ x + h$
proof –
from *poly-degree-le-large-const*[*OF ppp-qqq nneg-poly- nneg* [*OF nneg-ppp*]]
have $\exists H.\ \forall h \geq H.\ \forall x \geq 0.\ \text{poly } qqq\ x \leq h * \text{poly } ppp\ x + h$ **by** *auto*
from *someI-ex*[*OF this, folded H-def*]
show $h \geq H \implies x \geq 0 \implies \text{poly } qqq\ x \leq h * \text{poly } ppp\ x + h$ **by** *auto*
qed
end

definition *h* **where** $h = \max\ 9\ (H\ 1)$

lemma *h*: $h \geq 1$ **unfolding** *h-def* **by** *auto*

abbreviation *I-Q* **where** $I-Q \equiv IQ\ h$

interpretation *inter-Q*: $\text{poly-inter } F-Q\ I-Q\ (\lambda x\ y.\ x \geq y + (1 :: 'a))$.

Well-definedness of Interpretation in Theorem 6.4


```

lemma valid-monotone-inter-Q:
  inter-Q.valid-monotone-poly-inter
  unfolding inter-Q.valid-monotone-poly-inter-def
proof (intro ballI)
  note [simp] = insertion-add insertion-mult
  fix fn
  assume f: fn ∈ F-Q
  then consider
    (a) fn = (a-sym,2)
    | (g) fn = (g-sym,2)
    | (h) fn = (h-sym,1)
    | (q) fn = (q-sym,1)
    | (f) fn = (f-sym,9)
    | (z) fn = (z-sym,0)
    | (v) i where fn = (v-sym i, 1) i ∈ V
  unfolding F-Q-def F-def by auto
thus inter-Q.valid-monotone-poly fn
proof cases
  case *: a
  have vars: vars (PVar 0 * PVar 1 + PVar 0 + PVar 1 :: 'a mpoly) = {0,1}
  apply (intro vars-eqI)
  subgoal by (intro vars-mult-subI vars-add-subI, auto)
  subgoal for v by (intro exI[of - λ -. 1] exI[of - 0], auto)
  done
  show ?thesis unfolding inter-Q.valid-monotone-poly-def *
  apply (intro allI impI, clarify, unfold IQ.simps vars valid-poly-def
    monotone-poly-wrt-def
    insertion-mult insertion-add insertion-Var,
    intro conjI allI impI)
  subgoal for  $\alpha$  unfolding assignment-def by simp
  subgoal for - - -  $\alpha$  x v
  proof goal-cases
  case 1
  from assignmentD[OF 1(1)] have 0:  $\alpha$  0 ≥ 0  $\alpha$  1 ≥ 0 by auto
  from 1 have x = 0 ∨ x = 1 by auto
  thus ?case using 0 1(3) mult-right-mono[OF 1(3), of  $\alpha$  (x - 1)]
  by (auto simp: field-simps)
  (smt (verit, ccfv-threshold) 1(3) add.assoc add commute add-increasing
add-le-imp-le-right add-right-mono diff-ge-0-iff-ge le-add-diff-inverse2 mult-right-mono
zero-less-one-class.zero-le-one)
  qed
  subgoal by auto
  done
next
  case *: f
  have vars: vars (PVar 0 + PVar 1 + PVar 2 + PVar 3 + PVar 4 + PVar 5
+ PVar 6 + PVar 7 + PVar 8 :: 'a mpoly) = {0,1,2,3,4,5,6,7,8}
  apply (intro vars-eqI)
  subgoal by (intro vars-mult-subI vars-add-subI, auto)

```

```

    subgoal for v by (intro exI[of - λ -. 1] exI[of - 0], auto)
  done
show ?thesis unfolding inter-Q.valid-monotone-poly-def *
apply (intro allI impI, clarify, unfold IQ.simps vars valid-poly-def
      monotone-poly-wrt-def
      insertion-mult insertion-add insertion-Var,
      intro conjI allI impI)
subgoal for α unfolding assignment-def by simp
subgoal for - - - α x v
proof goal-cases
  case 1
  hence x ∈ {0,1,2,3,4,5,6,7,8} by auto
  thus ?case using 1(3) by auto
qed
subgoal by auto
done
next
case *: h
have vars: vars (Const h * PVar 0 + Const h :: 'a mpoly) = {0}
  apply (intro vars-eqI)
subgoal by (intro vars-mult-subI vars-add-subI, auto)
subgoal for v using h by (intro exI[of - λ -. 1] exI[of - 0], auto)
done
show ?thesis unfolding inter-Q.valid-monotone-poly-def *
apply (intro allI impI, clarify, unfold IQ.simps vars valid-poly-def
      monotone-poly-wrt-def
      insertion-mult insertion-add insertion-Var,
      intro conjI allI impI)
subgoal for α using h unfolding assignment-def by simp
subgoal for - - - α x v
proof goal-cases
  case 1
  from assignmentD[OF 1(1), of 0]
  show ?case using 1 h
    by (auto simp: field-simps)
      (smt (verit, ccfv-threshold) add commute add-le-cancel-left distrib-left
linordered-nonzero-semiring-class.zero-le-one mult commute mult-cancel-left1 mult-left-mono
nle-le order-trans)
  qed
subgoal by auto
done
next
case z
  thus ?thesis by (auto simp: inter-Q.valid-monotone-poly-def valid-poly-def
monotone-poly-wrt-def)
next
case *: g
have vars: vars (PVar 0 + PVar 1 :: 'a mpoly) = {0,1}
  apply (intro vars-eqI)

```

```

    subgoal by (intro vars-mult-subI vars-add-subI, auto)
    subgoal for v by (intro exI[of - λ -. 1] exI[of - 0], auto)
    done
  show ?thesis unfolding inter-Q.valid-monotone-poly-def *
  apply (intro allI impI, clarify, unfold IQ.simps vars valid-poly-def
    monotone-poly-wrt-def
    insertion-mult insertion-add insertion-Var,
    intro conjI allI impI)
  subgoal for α unfolding assignment-def by simp
  subgoal for - - - α x v
  proof goal-cases
    case 1
    hence x ∈ {0,1} by auto
    thus ?case using 1(3) by auto
  qed
  subgoal by auto
  done
next
case *: q
have vars: vars (PVar 0 * PVar 0 + Const 2 * PVar 0 :: 'a mpoly) = {0}
  apply (intro vars-eqI)
  subgoal by (intro vars-mult-subI vars-add-subI, auto)
  subgoal for v by (intro exI[of - λ -. 1] exI[of - 2], auto)
  done
show ?thesis unfolding inter-Q.valid-monotone-poly-def *
  apply (intro allI impI, clarify, unfold IQ.simps vars valid-poly-def
    monotone-poly-wrt-def
    insertion-mult insertion-add insertion-Var,
    intro conjI allI impI)
  subgoal for α unfolding assignment-def by simp
  subgoal for - - - α x v
  proof goal-cases
    case 1
    hence [simp]: x = 0 by auto
    from 1(1) have α 0 ≥ 0 unfolding assignment-def by simp
    thus ?case using 1(3)
      by auto
      (metis (no-types, opaque-lifting) add.assoc add-mono le-add-same-cancel1
        mult-2 mult-mono order-trans zero-less-one-class.zero-le-one)
  qed
  subgoal by auto
  done
next
case *: (v i)
from α[unfolded positive-interpr-def] have pos: α i > 0 by auto
have vars: vars ((PVar 0) ^ (nat (α i)) :: 'a mpoly) = {0}
  apply (intro vars-eqI)
  subgoal by (metis Preliminaries-on-Polynomials-1.vars-Var vars-power)
  subgoal for v using pos apply (intro exI[of - λ -. 2] exI[of - 1])

```

```

    by (auto simp: insertion-power)
      (metis less-numeral-extra(4) one-less-numeral-iff one-less-power semiring-norm(76) zero-less-nat-eq)
  done
show ?thesis unfolding inter-Q.valid-monotone-poly-def *
apply (intro allI impI, clarify, unfold IQ.simps vars valid-poly-def
      monotone-poly-wrt-def
      insertion-Var insertion-power,
      intro conjI allI impI)
subgoal for - - -  $\beta$  using pos unfolding assignment-def by simp
subgoal for - - -  $\beta$   $x$   $v$ 
proof goal-cases
  case 1
  hence [simp]:  $x = 0$  by auto
  from 1(1) have  $b0: \beta \ 0 \geq 0$  unfolding assignment-def by simp
  from pos obtain  $k$  where  $nik: \text{nat } (\alpha \ i) = \text{Suc } k$ 
    using  $gr0\text{-implies-Suc}$  zero-less-nat-eq by presburger
  define  $b0$  where  $b0 = \beta \ 0$ 
  have  $\beta \ 0 \wedge \text{nat } (\alpha \ i) + 1 \leq (\beta \ 0 + 1) \wedge \text{nat } (\alpha \ i)$  using  $b0$  unfolding
     $nik$   $b0\text{-def}$ [symmetric]
  proof (induct  $k$ )
    case (Suc  $k$ )
    define  $sk$  where  $sk = \text{Suc } k$ 
    from Suc show ?case unfolding  $sk\text{-def}$ [symmetric]
    by (auto simp: field-simps add-mono ordered-comm-semiring-class.comm-mult-left-mono)
  qed auto
  also have  $\dots \leq v \wedge \text{nat } (\alpha \ i)$  using 1(3) by (simp add:  $b0$  power-mono)
  finally show ?case by simp
qed
subgoal by auto
done
qed
qed

```

lemma $I\text{-}Q\text{-}\delta\text{-poly-inter}$: $\delta\text{-poly-inter } F\text{-}Q \ I\text{-}Q \ (1 :: 'a)$
 by (unfold-locales, rule valid-monotone-inter-Q, auto)

interpretation $inter\text{-}Q$: $\delta\text{-poly-inter } F\text{-}Q \ I\text{-}Q \ 1 :: 'a$ by (rule $I\text{-}Q\text{-}\delta\text{-poly-inter}$)

Orientation part of Theorem 6.4

lemma $orient\text{-}Q$: $inter\text{-}Q.orient\text{-rule} \ (lhs\text{-}Q, rhs\text{-}Q)$
 unfolding $inter\text{-}Q.orient\text{-rule-def}$ split $inter\text{-}Q.I'\text{-is-insertion-eval}$
proof (intro allI impI)
 fix $x :: - \Rightarrow 'a$
 assume $assignment \ x$
 hence $x: x \ i \geq 0$ for i unfolding $assignment\text{-def}$ by auto
 have $h9: h \geq 9$ unfolding $h\text{-def}$ by auto
 define l where $l \ i = \text{args } (lhs\text{-}Q) ! i$ for i
 define r where $r \ i = \text{args } (rhs\text{-}Q) ! i$ for i

```

let ?e = inter-Q.eval
let ?poly = λ t. insertion x (?e t)
note defs = l-def r-def lhs-Q-def rhs-Q-def
let ?nums = [0,1,2,3,4,5,6,7,8] :: nat list
note [simp] = insertion-add insertion-mult y1-def y2-def y3-def y4-def y5-def
y6-def y7-def y8-def y9-def

have e-lhs: ?e lhs-Q = sum-list (map (λ i. (?e (l i))) ?nums)
unfolding defs by simp
have e-rhs: ?e rhs-Q = sum-list (map (λ i. (?e (r i))) ?nums)
unfolding defs by simp

have [simp]: 2 = (Const (2 :: 'a))
by (metis mpoly-Const-1 mpoly-Const-add one-add-one)

have ?poly (r 0) = h2 * ((x 0)2 + 2 * x 0) + h2 + h
by (simp add: field-simps power2-eq-square defs)
also have ... ≤ (h * x 0 + h)2 + 2 * (h * x 0 + h) using h x[of 0]
by (simp add: field-simps power2-eq-square)
also have ... = ?poly (l 0)
by (simp add: field-simps power2-eq-square defs)
finally have 1: ?poly (l 0) ≥ ?poly (r 0) .

from h9 have h2: h ≥ 2 by auto
have ?poly (r 1) = 2 * x 1
by (simp add: field-simps defs)
also have ... ≤ h * x 1 + h using mult-right-mono[OF h2 x[of 1]] h
by auto
also have ... = ?poly (l 1)
by (simp add: field-simps power2-eq-square defs)
finally have 2: ?poly (l 1) ≥ ?poly (r 1) .

have ?poly (r 2) + 1 = 9 * x 2 + 1 unfolding defs by simp
also have ... ≤ h * x 2 + h
by (intro add-mono h mult-right-mono h9 x)
also have ... = ?poly (l 2) unfolding defs by simp
finally have 3: ?poly (l 2) ≥ ?poly (r 2) + 1 .

have eval-vs: insertion x (inter-Q.eval (g-list (map (λi. (v-sym i, Suc 0)) xs)))
= 0
for xs by (induct xs, auto simp: insertion-power α1)
have [simp]: insertion x (inter-Q.eval t-t) = h unfolding t-t-def symbol-list-def
by (simp add: eval-vs)
have ?poly (r 3) = (x 3 + h)2 + 2 * (x 3 + h)
by (simp add: field-simps power2-eq-square defs)
also have ... ≤ (x 3)2 + 2 * x 3 + h3*x 3 + h3 + h2 + h (is ?l ≤ ?r)
proof -
have 2 * 1 ≤ h * h
by (intro mult-mono, insert h2, auto)

```

hence $hh: h * h \geq 2$ **by** *auto*
 have $?l \leq ?r \iff 1 * h + (2 * h) * x \leq (h * h) * h + ((h * h) * h) * x$
 by (*auto simp: field-simps power2-eq-square defs power3-eq-cube*)
 also have ...
 by (*intro add-mono mult-right-mono x, insert h hh, auto*)
 finally show *?thesis* .
qed
 also have ... = *?poly (l 3)*
 by (*simp add: field-simps power2-eq-square defs power3-eq-cube*)
 finally have 4: *?poly (l 3) \geq ?poly (r 3)* .

 have *?poly (r 4) = ((x 4)^2 + 2 * x 4)*
 by (*simp add: field-simps power2-eq-square defs*)
 also have ... = *?poly (l 4)*
 by (*simp add: field-simps power2-eq-square defs*)
 finally have 5: *?poly (l 4) \geq ?poly (r 4)* **by** *simp*

 have *?poly (r 5) = (x 5)^2 + 2 * x 5*
 by (*simp add: field-simps power2-eq-square defs*)
 also have ... = *?poly (l 5)*
 by (*simp add: field-simps power2-eq-square defs*)
 finally have 6: *?poly (l 5) \geq ?poly (r 5)* **by** *simp*

 have 7: *?poly (l 6) \geq ?poly (r 6)* **unfolding** *defs using h x[of 6]*
 by (*simp add: add-increasing2 linorder-not-le mult-le-cancel-right1*)
 have 8: *?poly (l 7) \geq ?poly (r 7)* **unfolding** *defs using h x[of 7]*
 by (*simp add: add-increasing2 linorder-not-le mult-le-cancel-right1*)

 have 9: *?poly (l 8) \geq ?poly (r 8)*
proof –
 have *r: ?e (r 8) = poly-to-mpoly 8 (qqq h)*
 unfolding *defs qqq-def*
 by (*simp add: qq[unfolded y9-def] algebra-simps smult-conv-mult-Const Const-mult*
flip: mpoly-of-poly-is-poly-to-mpoly)
 have *l: ?e (l 8) = poly-to-mpoly 8 ([:h:] * (ppp h) + [:h:])*
 unfolding *defs ppp-def*
 by (*simp add: pp[unfolded y9-def] algebra-simps smult-conv-mult-Const Const-mult*
flip: mpoly-of-poly-is-poly-to-mpoly)
 {
 fix *r*
 assume *r: r \in \{p,q\}*
 with *funas-encode-poly-p funas-encode-poly-q*
 have *funas: funas-term (encode-poly y9 r) \subseteq F* **by** *auto*
 have *poly-inter.eval (IQ 1) (encode-poly y9 r) = inter-Q.eval (encode-poly y9*
r)
 by (*rule poly-inter-eval-cong, insert funas, auto simp: F-def*)
 } **note** *encode-eq = this*
 have *pp-eq: pp h = pp 1* **unfolding** *pp-def using encode-eq[of p]* **by** *auto*
 have *qq-eq: qq h = qq 1* **unfolding** *qq-def using encode-eq[of q]* **by** *auto*

```

have ppp-eq: ppp h = ppp 1 unfolding ppp-def pp-eq ..
have qq-q-eq: qq-q h = qq-q 1 unfolding qq-q-def qq-eq ..
have H h = H 1 unfolding H-def ppp-eq qq-q-eq ..
also have ... ≤ h unfolding h-def by auto
finally have h: h ≥ H h .
show ?thesis unfolding l r using H[OF h x[of 8]] by simp
qed

have ?poly rhs-Q + 1 =
  ?poly (r 0) + ?poly (r 1) + (?poly (r 2) + 1) + ?poly (r 3) + ?poly (r 4) +
  ?poly (r 5) + ?poly (r 6) + ?poly (r 7) + ?poly (r 8)
  unfolding e-rhs by simp
also have ... ≤ ?poly (l 0) + ?poly (l 1) + ?poly (l 2) + ?poly (l 3) + ?poly (l
  4) + ?poly (l 5) + ?poly (l 6) + ?poly (l 7) + ?poly (l 8)
  by (intro add-mono 1 2 3 4 5 6 7 8 9)
also have ... = ?poly lhs-Q
  unfolding e-lhs by simp

finally show ?poly rhs-Q + 1 ≤ ?poly lhs-Q by auto
qed
end
end

context poly-input
begin

Theorem 6.4

theorem solution-impl-delta-termination-of-Q:
  assumes positive-poly-problem p q
  shows termination-by-delta-poly-interpretation (TYPE('a :: floor-ceiling)) F-Q
  Q
proof –
  interpret solvable-poly-problem
  by (unfold-locales, fact)
  interpret I: delta-poly-inter F-Q I-Q (1 :: 'a) by (rule I-Q-delta-poly-inter)
  show ?thesis
  unfolding termination-by-delta-poly-interpretation-def
  proof (intro exI[of - 1 :: 'a] exI[of - I-Q] conjI I-Q-delta-poly-inter)
  show I.termination-by-delta-interpretation Q
  unfolding I.termination-by-delta-interpretation-def Q-def
  proof (clarify, intro conjI)
  show funas-term lhs-Q ∪ funas-term rhs-Q ⊆ F-Q using lhs-Q-F rhs-Q-F
by auto
  show I.orient-rule (lhs-Q, rhs-Q) using orient-Q by simp
  qed
qed
qed

end

```

context *delta-poly-inter*

begin

lemma *insertion-eval-pos*: **assumes** *funas-term* $t \subseteq F$

and *assignment* α

shows *insertion* α (*eval* t) ≥ 0

by (*rule valid-imp-insertion-eval-pos*[*OF valid assms*])

lemma *monotone-poly-eval*: **assumes** *funas-term* $t \subseteq F$

shows *monotone-poly* (*vars-term* t) (*eval* t) *vars* (*eval* t) = *vars-term* t

proof –

have $\exists y. x + \delta \leq y$ **for** $x :: 'a$ **by** (*intro exI*[*of - x + δ*], *auto*)

from *monotone-poly-eval-generic*[*OF valid transp-gt-delta gt-delta-imp-ge this - assms*] $\delta 0$

show *monotone-poly* (*vars-term* t) (*eval* t) *vars* (*eval* t) = *vars-term* t **by** *auto*

qed

lemma *monotone-linear-poly-to-coeffs*: **fixes** $p :: 'a$ *mpoly*

assumes *linear*: *total-degree* $p \leq 1$

and *poly*: *valid-poly* p

and *mono*: *monotone-poly* $\{..<n\}$ p

and *vars*: *vars* $p = \{..<n\}$

shows $\exists c a. p = \text{Const } c + (\sum i \leftarrow [0..<n]. \text{Const } (a \ i) * \text{PVar } i)$

$\wedge c \geq 0 \wedge (\forall i < n. a \ i \geq 1)$

proof –

have *sum-zero*: $(\bigwedge x. x \in \text{set } xs \implies x = 0) \implies \text{sum-list } (xs :: \text{int list}) = 0$ **for** xs **by** (*induct* xs , *auto*)

from *coefficients-of-linear-poly*[*OF linear*] **obtain** $c a \ vs$

where $p: p = \text{Const } c + (\sum i \leftarrow vs. \text{Const } (a \ i) * \text{PVar } i)$

and *vsd*: *distinct* vs *set* $vs = \text{vars } p$ *sorted-list-of-set* (*vars* p) = vs

and *nz*: $\bigwedge v. v \in \text{set } vs \implies a \ v \neq 0$

and $c: c = \text{mcoeff } p \ 0$

and $a: \bigwedge i. a \ i = \text{mcoeff } p$ (*monomial* $1 \ i$) **by** *blast*

have $vs: vs = [0..<n]$ **unfolding** *vsd*(3)[*symmetric*] **unfolding** *vars*

by (*simp add: lessThan-atLeast0*)

show *?thesis* **unfolding** $p \ vs$

proof (*intro exI conjI allI impI*, *rule refl*)

show $c: c \geq 0$ **using** *poly*[*unfolded valid-poly-def, rule-format, of $\lambda \ . \ 0$, unfolded* p]

by (*auto simp: coeff-add*[*symmetric*] *coeff-Const* *coeff-sum-list* *o-def* *coeff-Const-mult*

coeff-Var *monomial-0-iff* *assignment-def*)

fix i

assume $i < n$

hence $i: i \in \text{set } vs$ **unfolding** vs **by** *auto*

from *nz*[*OF* i] **have** $a0: a \ i \neq 0$ **by** *auto*

from *split-list*[*OF* i] **obtain** $\text{bef } \text{aft}$ **where** $\text{vs}i: vs = \text{bef } @ [i] @ \text{aft}$ **by** *auto*

with *vsd*(1) **have** $i: i \notin \text{set } (\text{bef } @ \text{aft})$ **by** *auto*


```

define  $\alpha$  where  $\alpha = (\lambda x :: \text{var. } 0 :: 'a)$ 
have assignment  $\alpha$  unfolding assignment-def  $\alpha$ -def using  $c$  by auto
from mono[unfolded monotone-poly-wrt-def, rule-format, OF this, of i  $\delta$ ]  $\langle i <$ 
 $n \rangle$ 
have insertion  $\alpha$   $p + \delta \leq \text{insertion } (\alpha(i := \delta))$   $p$  by (auto simp:  $\alpha$ -def)
from this[unfolded p vsi insertion-add insertion-sum-list insertion-Const map-map
o-def insertion-mult insertion-Var]
have  $(\sum x \leftarrow \text{bef } @ \text{aft. } a x * \alpha x) + \delta \leq (\sum x \leftarrow \text{bef } @ \text{aft. } a x * (\alpha(i := \delta))$ 
 $x) + a i * \delta$ 
by (auto simp:  $\alpha$ -def)
also have  $(\sum x \leftarrow \text{bef } @ \text{aft. } a x * (\alpha(i := \delta)) x) = (\sum x \leftarrow \text{bef } @ \text{aft. } a x * \alpha$ 
 $x)$ 
by (subst map-cong[OF refl, of - -  $\lambda x. a x * \alpha x$ ], insert i, auto simp:  $\alpha$ -def)
finally have  $\delta \leq a i * \delta$  by auto
with  $\delta 0$  show  $a i \geq 1$  by simp
qed
qed

```

end

Lemma 6.7

```

lemma criterion-for-degree-2: assumes  $qq$ -def:  $qq = q \circ_p [:c, a:] - \text{smult } a q$ 
and  $dq$ : degree  $q \geq 2$ 
and  $ineq$ :  $\bigwedge x :: 'a :: \text{linordered-field. } x \geq 0 \implies \text{poly } qq x \leq \text{poly } p x$ 
and  $dp$ : degree  $p \leq 1$ 
and  $a1$ :  $a \geq 1$ 
and  $lq0$ : lead-coeff  $q > 0$ 
and  $c$ :  $c > 0$ 
shows degree  $q = 2 a = 1$ 
proof -
have  $deg$ : degree  $(q \circ_p [:c, a:]) = \text{degree } q$ 
unfolding degree-pcompose using  $a1$  by simp
have  $coeff$ -dq: coeff  $qq$  (degree  $q$ ) = lead-coeff  $q * (a \wedge \text{degree } q - a)$ 
apply (simp add: qq-def)
apply (subst deg[symmetric])
apply (subst lead-coeff-comp)
subgoal using  $a1$  by simp
subgoal using  $a1$  by (simp add: field-simps)
done
have  $deg$ -qq: degree  $qq \leq \text{degree } q$  using  $deg$ 
by (simp add: degree-diff-le qq-def)
{
assume  $a \neq 1$ 
with  $a1$  have  $a1$ :  $a > 1$  by auto
hence  $a \wedge \text{degree } q > a \wedge 1$  using  $dq$ 
by (metis add-strict-increasing linorder-not-less one-add-one power-le-imp-le-exp
zero-less-one)

```

```

hence coeff: coeff qq (degree q) > 0
  unfolding coeff-dq using dq by (auto intro!: mult-pos-pos lq0)
hence degree qq ≥ degree q
  by (simp add: le-degree)
with deg-qq have eq: degree qq = degree q by auto
from coeff[folded eq] have lcqq: lead-coeff qq > 0 by auto
from dq[folded eq] have 2 ≤ degree qq by auto
also have degree qq ≤ degree p using ineq lcqq
  by (metis Preliminaries-on-Polynomials-2.poly-pinfty-ge degree-mono-generic
linorder-le-less-linear order-less-not-sym)
  also have ... ≤ 1 by fact
  finally have False by simp
}
thus a1: a = 1 by auto
hence qq: qq = q ∘p [:c, 1:] - q unfolding qq-def by auto
from coeff-dq[unfolded a1] have coeff qq (degree q) = 0 by simp
with deg-qq dq have deg-qq: degree qq < degree q
  using degree-less-if-less-eq1 by fastforce
define m where m = degree q
define m1 where m1 = m - 1
from dq have mm1: m = Suc m1 unfolding m1-def m-def by auto
define qi where qi = coeff q
define cf where cf k i = (qi k * of-nat (k choose i) * c ^ (k - i)) for i k
define inner where inner k = (∑ i < k. monom (cf k i) i) for k
define rem where rem = (∑ i < m1. monom (cf m i) i) + sum inner {..<m}
{
  fix x
  define e where e i k = of-nat (k choose i) * x ^ i * c ^ (k - i) for k i
  have poly qq x = poly (q ∘p [:c, 1:]) x - poly q x unfolding qq by simp
  also have ... = (∑ k ≤ m. qi k * (x + c) ^ k) - (∑ k ≤ m. qi k * x ^ k)
unfolding qi-def
  by (subst (1 2) poly-as-sum-of-monoms[of q, symmetric, folded m-def])
  (simp add: poly-sum poly-pcompose poly-monom ac-simps)
  also have ... = (∑ k ≤ m. qi k * (∑ i ≤ k. e i k)) - (∑ k ≤ m. qi k * x ^ k)
  by (subst binomial-ring, auto simp: e-def)
  also have ... = (∑ k ≤ m. qi k * (e k k + (∑ i < k. e i k))) - (∑ k ≤ m. qi k *
x ^ k)
  by (intro arg-cong[of - - λ x. x - -] sum.cong refl arg-cong2[of - - - - (*)])
  (metis add.commute lessThan-Suc-atMost sum.lessThan-Suc)
  also have ... = (∑ k ≤ m. qi k * e k k) + (∑ k ≤ m. qi k * (∑ i < k. e i k)) -
(∑ k ≤ m. qi k * x ^ k)
  by (simp add: field-simps sum.distrib)
  also have ... = (∑ k ≤ m. qi k * (∑ i < k. e i k))
  unfolding e-def by simp
  also have ... = poly (∑ k ≤ m. inner k) x unfolding e-def inner-def cf-def
  by (simp add: poly-sum poly-monom ac-simps sum-distrib-left)
  finally have poly qq x = poly (sum inner {..m}) x .
}
hence qq = sum inner {..m} by (intro poly-ext, auto)

```

```

also have ... = inner m + sum inner {.. $m$ }
  by (metis add.commute lessThan-Suc-atMost sum.lessThan-Suc)
also have inner m = monom (cf m m1) m1 + ( $\sum$   $i < m1$ . monom (cf m i) i)
  unfolding inner-def mm1 by simp
finally have qq: qq = monom (cf m m1) m1 + rem by (simp add: rem-def)
have cf-mm1: cf m m1 > 0 unfolding cf-def
proof (intro mult-pos-pos)
  show 0 < qi m unfolding qi-def m-def by fact
  show 0 < (of-nat (m choose m1) :: 'a) unfolding mm1
    by (simp add: add-strict-increasing)
  show 0 < c ^ (m - m1) using c by simp
qed
{
  fix k
  assume k: k  $\geq$  m1
  have coeff rem k = ( $\sum$   $i < m$ . coeff (inner i) k) using k
    by (simp add: rem-def Polynomial.coeff-sum)
  also have ... = 0
  proof (intro sum.neutral ballI)
    fix i
    show  $i \in \{.. $m$ \} \implies$  coeff (inner i) k = 0
      unfolding inner-def Polynomial.coeff-sum using k mm1
      by auto
  qed
  finally have coeff rem k = 0 .
} note zero = this
from cf-mm1 zero[of m1]
have qq-m1: coeff qq m1 > 0 unfolding qq by auto
{
  fix k
  assume k > m1
  with zero[of k] have coeff qq k = 0 unfolding qq by auto
}
with qq-m1 have deg-qq: degree qq = m1
  by (metis coeff-0 le-degree leading-coeff-0-iff order-less-le)
with qq-m1 have lc-qq: lead-coeff qq > 0 by auto

from ineq lc-qq have degree qq  $\leq$  degree p
  by (metis Preliminaries-on-Polynomials-2.poly-pinfy-ge degree-mono-generic
linorder-le-less-linear order-less-not-sym)
also have ...  $\leq$  1 by fact
finally have m1  $\leq$  1 unfolding deg-qq by simp
with mm1 have m  $\leq$  2 by auto
hence degree q  $\leq$  2 unfolding m-def by auto
with dq show degree q = 2 by auto
qed

```

locale *term-delta-poly-input* = *poly-input* *p q* **for** *p q* +
fixes *type-of-field* :: 'a :: *floor-ceiling itself*
assumes *terminating-delta-poly*: *termination-by-delta-poly-interpretation TYPE('a)*
F-Q Q
begin

definition *I* **where** $I = (\text{SOME } I. \exists \delta. \text{delta-poly-inter } F\text{-}Q\text{ } I (\delta :: 'a) \wedge$
delta-poly-inter.termination-by-delta-interpretation F-Q I δ Q)

definition δ **where** $\delta = (\text{SOME } \delta. \text{delta-poly-inter } F\text{-}Q\text{ } I (\delta :: 'a) \wedge$
delta-poly-inter.termination-by-delta-interpretation F-Q I δ Q)

lemma *I*: *delta-poly-inter F-Q I δ delta-poly-inter.termination-by-delta-interpretation*
F-Q I δ Q

using *someI-ex*[*OF someI-ex*[*OF terminating-delta-poly*[*unfolded termination-by-delta-poly-interpretation-def*
folded I-def], *folded δ -def*]
by *auto*

sublocale *delta-poly-inter F-Q I δ* **by** (*rule I(1)*)

lemma *orient*: *orient-rule (lhs-Q,rhs-Q)*

using *I(2)*[*unfolded termination-by-delta-interpretation-def*] **unfolding** *Q-def*
by *auto*

lemma *eval-t-t-gt-0*: **assumes** *Ig*: $I\ g\text{-sym} = \text{Const } g0 + \text{Const } g1 * \text{PVar } 0 +$
 $\text{Const } g2 * \text{PVar } 1$

and *Iz*: $I\ z\text{-sym} = \text{Const } z0$

and *z0*: $z0 \geq 0$

and *g0*: $g0 \geq 0$

and *g1?*: $g1 > 0\ g2 > 0$

shows *insertion β (eval t-t) > 0*

proof –

define α **where** $\alpha = (\lambda - :: \text{var. } 0 :: 'a)$

have α : *assignment α* **by** (*auto simp: assignment-def α -def*)

have *id*: *insertion β (eval t-t) = insertion α (eval t-t)*

by (*rule insertion-irrelevant-vars, insert vars-t vars-eval, auto*)

note *pos* = *insertion-eval-pos*[*OF - α*]

show *?thesis*

proof (*rule ccontr*)

assume $\langle \neg ?thesis \rangle$

from *this*[*unfolded id*] **have** *insertion α (eval t-t) ≤ 0* **by** *auto*

with *pos*[*OF t-F*] **have** 0 : *insertion α (eval t-t) = 0* **by** *auto*

note [*simp*] = *insertion-add insertion-mult insertion-substitute*

define *IA* **where** *IA t* = *insertion α (eval t)* **for** *t*

note *pos* = *pos*[*folded IA-def*]

let *?zz* = *g-list symbol-list*

from *pos*[*OF g-list-F*[*OF symbol-list*]]

```

have zz: 0 ≤ IA ?zz by auto
have 0: 0 = IA t-t using 0 by (auto simp: IA-def)
also have ... = g0 + g1 * z0 + g2 * IA ?zz unfolding t-t-def by (simp add:
Ig IA-def Iz)
finally have g0: g0 = 0 and g1 * z0 = 0 g2 * IA ?zz = 0
  using g0 z0 g12 zz mult-nonneg-nonneg[of g1 z0] mult-nonneg-nonneg[of g2
IA ?zz]
  by linarith+
with g12 have z0: z0 = 0 and 0: IA ?zz = 0 by auto
from Ig g0 have Ig: I g-sym = Const g1 * PVar 0 + Const g2 * PVar 1 by
simp
from z0 Iz have Iz: I z-sym = 0 by auto

{
fix fs f a
assume set fs ⊆ F-Q and IA (g-list fs) = 0
  and (f,a) ∈ set fs
hence mcoeff (I f) 0 = 0
proof (induct fs)
case (Cons kb fs)
obtain k b where kb: kb = (k,b) by force
let ?t = Fun k (replicate b z-t) :: (symbol,var)term
from Cons(3)[unfolded kb]
have 0: g1 * IA ?t + g2 * IA (g-list fs) = 0
  by (simp add: IA-def Ig)
from Cons(2)[unfolded kb] have (k,b) ∈ F-Q by auto
hence funas-term ?t ⊆ F-Q by (force simp: F-Q-def F-def)
from pos[OF this] have pos1: 0 ≤ IA ?t by auto
from Cons(2) have fs: set fs ⊆ F-Q by auto
from pos[OF g-list-F[OF this]] have pos2: 0 ≤ IA (g-list fs) by auto
from 0 g12 pos1 pos2 mult-nonneg-nonneg[of g1 IA ?t]
  mult-nonneg-nonneg[of g2 IA (g-list fs)]
have g1 * IA ?t = 0 g2 * IA (g-list fs) = 0
  by linarith+
with g12 have t: IA ?t = 0 and 0: IA (g-list fs) = 0 by auto
from Cons(1)[OF fs 0] have IH: (f, a) ∈ set fs ⇒ mcoeff (I f) 0 = 0 by
auto

show ?case
proof (cases (f,a) = (k,b))
case False
  with IH Cons(4) kb show ?thesis by auto
next
case True
  have 0 = IA ?t using t by simp
  also have ... = insertion α (I k)
  apply (simp add: IA-def)
  apply (rule insertion-irrelevant-vars)
  subgoal for v by (auto simp: Iz α-def)
  done
}

```

```

    also have ... = mcoeff (I k) 0 unfolding  $\alpha$ -def by simp
    finally show ?thesis using True by simp
  qed
qed auto
} note main = this

{
  fix k ka
  assume (k,ka)  $\in$  F-Q
  then consider (z) (k,ka) = (z-sym,0) | (g) (k,ka) = (g-sym,2) | (zl) (k,ka)
 $\in$  set symbol-list
    unfolding symbol-list-def F-Q-def F-def using V-list by auto
  hence mcoeff (I k) 0 = 0
  proof cases
    case (zl)
    from main[OF symbol-list 0 zl] show ?thesis .
  next
    case z
    thus ?thesis using Iz by simp
  next
    case g
    thus ?thesis using Ig by (simp add: coeff-Const-mult coeff-Var)
  qed
} note coeff-0 = this

```

```

have ins-0: funas-term  $t \subseteq$  F-Q  $\implies$  insertion  $\alpha$  (eval  $t$ ) = 0 for  $t$ 
proof (induct t)
  case (Var x)
  show ?case by (auto simp:  $\alpha$ -def coeff-Var)
next
case (Fun f ts)
{
  fix i
  assume  $i <$  length ts
  hence  $ts ! i \in$  set ts by auto
  from Fun(1)[OF this] Fun(2) this
  have insertion  $\alpha$  (eval (ts ! i)) = 0 by auto
} note IH = this
have insertion  $\alpha$  (eval (Fun f ts)) = insertion  $\alpha$  (I f)
  apply (simp)
  apply (intro insertion-irrelevant-vars)
  subgoal for v using IH[of v] by (auto simp:  $\alpha$ -def)
  done
also have ... = mcoeff (I f) 0 unfolding  $\alpha$ -def by simp
also have ... = 0 using Fun(2) coeff-0 by auto
finally show ?case by simp
qed

```

```

from orient[unfolded orient-rule gt-poly-def, rule-format, OF  $\alpha$ ] ins-0[OF
lhs-Q-F] ins-0[OF rhs-Q-F]
  show False using  $\delta 0$  by auto
qed
qed

```

Theorem 6.8

```

theorem solution: positive-poly-problem p q
proof –
  let ?q = q
  from orient[unfolded orient-rule]
  have gt: gt-poly (eval lhs-Q) (eval rhs-Q) by auto
  from valid[unfolded valid-monotone-poly-inter-def]
  have valid:  $\bigwedge f. f \in F-Q \implies$  valid-monotone-poly f by auto
  let ?lc = lead-coeff
  let ?f = (f-sym,9)
  have ?f  $\in F-Q$  unfolding F-Q-def by auto
  from valid[OF this, unfolded valid-monotone-poly-def] obtain f where
    If: I f-sym = f and f: valid-poly f monotone-poly (vars f) f vars f = {.. $9$ }
  by auto
  note mono = f(2)
  define l where l i = args (lhs-Q) ! i for i
  define r where r i = args (rhs-Q) ! i for i
  have list: [0.. $9$ ] = [0,1,2,3,4,5,6,7,8 :: nat] by code-simp
  have lhs-Q: lhs-Q = Fun f-sym (map l [0.. $9$ ]) unfolding lhs-Q-def l-def by
(auto simp: list)
  have rhs-Q: rhs-Q = Fun f-sym (map r [0.. $9$ ]) unfolding rhs-Q-def r-def by
(auto simp: list)
  {
    fix i :: var
    define vs where vs = V-list
    assume i < 9
    hence choice: i = 0  $\vee$  i = 1  $\vee$  i = 2  $\vee$  i = 3  $\vee$  i = 4  $\vee$  i = 5  $\vee$  i = 6  $\vee$  i
= 7  $\vee$  i = 8 by linarith
    have set: {0.. $9$  :: nat} = {0,1,2,3,4,5,6,7,8} by code-simp
    from choice have vars: vars-term (l i) = {i} vars-term (r i) = {i} unfolding
l-def lhs-Q-def r-def rhs-Q-def
    using vars-encode-poly[of 8 p] vars-encode-poly[of 8 q] vars-t
    by (auto simp: y1-def y2-def y3-def y4-def y5-def y6-def y7-def y8-def y9-def
vs-def[symmetric])
    from choice set have funs: funas-term (l i)  $\cup$  funas-term (r i)  $\subseteq F-Q$  using
rhs-Q-F lhs-Q-F unfolding lhs-Q rhs-Q
    by auto
    have lr  $\in \{l,r\} \implies$  vars-term (lr i) = {i} lr  $\in \{l,r\} \implies$  funas-term (lr i)  $\subseteq$ 
F-Q for lr
    by (insert vars funs, force)+
  } note signature-l-r = this
  {
    fix i :: var and lr

```

```

assume  $i: i < 9$  and  $lr: lr \in \{l,r\}$ 
from  $signature\text{-}l\text{-}r[OF\ i\ lr]$   $monotone\text{-}poly\text{-}eval[of\ lr\ i]$ 
have  $vars: vars\ (eval\ (lr\ i)) = \{i\}$ 
and  $mono: monotone\text{-}poly\ \{i\}\ (eval\ (lr\ i))$  by auto
} note  $eval\text{-}l\text{-}r = this$ 

define  $upoly$  where  $upoly\ l\text{-}or\text{-}r\ i = mpoly\text{-}to\text{-}poly\ i\ (eval\ (l\text{-}or\text{-}r\ i))$  for  $l\text{-}or\text{-}r ::$ 
 $var \Rightarrow (-,-)term$  and  $i$ 

{
fix  $lr$  and  $i :: nat$  and  $a :: - \Rightarrow 'a$ 
assume  $a: assignment\ a$  and  $i: i < 9$  and  $lr: lr \in \{l,r\}$ 
with  $eval\text{-}l\text{-}r[OF\ i]$   $signature\text{-}l\text{-}r[OF\ i]$ 
have  $vars: vars\ (eval\ (lr\ i)) = \{i\}$  and  $mono: monotone\text{-}poly\ \{i\}\ (eval\ (lr\ i))$ 
and  $funs: funas\text{-}term\ (lr\ i) \subseteq F\text{-}Q$  by auto
from  $insertion\text{-}eval\text{-}pos[OF\ funs]$ 
have  $valid: valid\text{-}poly\ (eval\ (lr\ i))$  unfolding  $valid\text{-}poly\text{-}def$  by auto
from  $monotone\text{-}poly\text{-}partial\text{-}insertion[OF\ \text{-}\ mono\ a,\ of\ i]$   $valid$ 
have  $deg: degree\ (partial\text{-}insertion\ a\ i\ (eval\ (lr\ i))) > 0$ 
and  $lc: ?lc\ (partial\text{-}insertion\ a\ i\ (eval\ (lr\ i))) > 0$ 
and  $ineq: insertion\ a\ (eval\ (lr\ i)) \geq a\ i - \delta$  by auto
moreover have  $partial\text{-}insertion\ a\ i\ (eval\ (lr\ i)) = upoly\ lr\ i$  unfolding
 $upoly\text{-}def$ 
using  $vars\ eval\text{-}l\text{-}r[OF\ i,\ of\ r,\ simplified]$ 
by  $(intro\ poly\text{-}ext)$ 
 $(metis\ i\ insertion\text{-}partial\text{-}insertion\text{-}vars\ poly\text{-}eq\text{-}insertion\ poly\text{-}inter.\ vars\text{-}eval$ 
 $signature\text{-}l\text{-}r(1)[of\ \text{-}\ r,\ simplified]\ singletonD)$ 
ultimately
have  $degree\ (upoly\ lr\ i) > 0$   $?lc\ (upoly\ lr\ i) > 0$ 
 $insertion\ a\ (eval\ (lr\ i)) \geq a\ i - \delta$  by auto
} note  $upoly\text{-}pos\text{-}subterm = this$ 

{
fix  $i :: var$ 
assume  $i: i < 9$ 
from  $degree\text{-}partial\text{-}insertion\text{-}stays\text{-}constant[OF\ f(2),\ of\ i]$  obtain  $a'$  where
 $a': assignment\ a'$  and
 $deg\text{-}a': \bigwedge b. (\bigwedge y. a'\ y + \delta \leq b\ y) \implies degree\ (partial\text{-}insertion\ a'\ i\ f) =$ 
 $degree\ (partial\text{-}insertion\ b\ i\ f)$ 
by auto
define  $a$  where  $a\ j = a'\ j + 2 * \delta$  for  $j$ 
from  $a'$  have  $a: assignment\ a$  unfolding  $assignment\text{-}def\ a\text{-}def$  using  $\delta 0$  by
 $auto$ 
{
fix  $b$ 
assume  $le: \bigwedge y. a\ y - \delta \leq b\ y$ 
have  $a'\ y + \delta \leq b\ y$  for  $y$  using  $le[of\ y]$  unfolding  $a\text{-}def$  by auto
from  $deg\text{-}a'[OF\ this]$ 

```



```

have 1: degree (partial-insertion a' i f) = degree (partial-insertion b i f) by
auto
have  $a' y + \delta \leq a y$  for  $y$  unfolding a-def using  $\delta 0$  by auto
from deg-a'[OF this] 1
have degree (partial-insertion a i f) = degree (partial-insertion b i f) by auto
} note deg-a = this

define  $c$  where  $c j = (if j < 9 then insertion a (eval (l j)) else a j)$  for  $j$ 
define  $e$  where  $e j = (if j < 9 then insertion a (eval (r j)) else a j)$  for  $j$ 
{
  fix  $x :: 'a$ 
  assume  $x: x \geq 0$ 
  have ass: assignment (a (i := x)) using  $x$  a unfolding assignment-def by
auto
from gt[unfolded gt-poly-def, rule-format, OF ass, unfolded rhs-Q lhs-Q]
have insertion (a(i := x)) (eval (Fun f-sym (map r [0..<9]))) +  $\delta$ 
   $\leq$  insertion (a(i := x)) (eval (Fun f-sym (map l [0..<9]))) by simp
also have insertion (a(i := x)) (eval (Fun f-sym (map r [0..<9]))) =
  insertion ( $\lambda j. insertion (a(i := x)) (eval (r j))) f$ 
  by (simp add: If insertion-substitute, intro insertion-irrelevant-vars, auto
simp: f)
also have  $\dots = poly (partial-insertion e i f) (poly (upoly r i) x)$ 
proof -
  let  $?\alpha = (\lambda j. insertion (a(i := x)) (eval (r j)))$ 
  have insi: poly (upoly r i) x = insertion (a(i := x)) (eval (r i))
  unfolding upoly-def using eval-l-r(1)[OF i, of r]
  by (subst poly-eq-insertion, force)
  (intro insertion-irrelevant-vars, auto)
show ?thesis unfolding insi
proof (rule insertion-partial-insertion-vars[of i f e ? $\alpha$ , symmetric])
  fix  $j$ 
  show  $j \neq i \implies j \in vars f \implies e j = insertion (a(i := x)) (eval (r j))$ 
  unfolding e-def f using eval-l-r[of j] f by (auto intro!: inser-
tion-irrelevant-vars)
  qed
qed
also have insertion (a(i := x)) (eval (Fun f-sym (map l [0..<9]))) =
  insertion ( $\lambda j. insertion (a(i := x)) (eval (l j))) f$ 
  by (simp add: If insertion-substitute, intro insertion-irrelevant-vars, auto
simp: f)
also have  $\dots = poly (partial-insertion c i f) (poly (upoly l i) x)$ 
proof -
  let  $?\alpha = (\lambda j. insertion (a(i := x)) (eval (l j)))$ 
  have insi: poly (upoly l i) x = insertion (a(i := x)) (eval (l i))
  unfolding upoly-def using eval-l-r[OF i]
  by (subst poly-eq-insertion, force)
  (intro insertion-irrelevant-vars, auto)
show ?thesis unfolding insi
proof (rule insertion-partial-insertion-vars[of i f c ? $\alpha$ , symmetric])

```

```

    fix j
    show  $j \neq i \implies j \in \text{vars } f \implies c j = \text{insertion } (a(i := x)) \text{ (eval } (l j))$ 
      unfolding c-def f using eval-l-r[of j] f by (auto intro!: inser-
tion-irrelevant-vars)
    qed
  qed

  finally have poly (partial-insertion c i f) (poly (upoly l i) x)
     $\geq \text{poly } (partial-insertion e i f) \text{ (poly } (upoly r i) x) + \delta .$ 
} note 1 = this

define er where er = partial-insertion e i f  $\circ_p$  upoly r i
define cl where cl = partial-insertion c i f  $\circ_p$  upoly l i
define d where d = degree (partial-insertion e i f)
{
  fix x
  have  $a x - \delta \leq c x \wedge a x - \delta \leq e x$ 
  proof (cases  $x \in \text{vars } f$ )
    case False
      thus ?thesis unfolding c-def e-def f using  $\delta 0$  by auto
    next
      case True
        hence id:  $(x < 9) = \text{True}$  and  $x: x < 9$  unfolding f by auto
        show ?thesis unfolding c-def e-def id if-True using upoly-pos-subterm(3)[OF
a x]
          by auto
        qed
        hence  $a x - \delta \leq c x \wedge a x - \delta \leq e x$  by auto
  } note a-ce = this

have d-eq:  $d = \text{degree } (partial-insertion c i f)$  unfolding d-def
  by (subst (1 2) deg-a[symmetric], insert a-ce, auto)

have e: assignment e using a' a-ce(2)  $\delta 0$  unfolding assignment-def a-def
  by (metis (no-types, lifting) diff-ge-0-iff-ge gt-delta-imp-ge le-add-same-cancel2
linorder-not-less mult-2 order-le-less-trans)

have d-pos:  $d > 0$  unfolding d-def
  by (intro monotone-poly-partial-insertion[OF - f(2) e], insert f i, auto)
have lc-e-pos: ?lc (partial-insertion e i f)  $> 0$ 
  by (intro monotone-poly-partial-insertion[OF - f(2) e], insert f i, auto)

have lc-r-pos: ?lc (upoly r i)  $> 0$  by (intro upoly-pos-subterm[OF a i], auto)
have deg-r:  $0 < \text{degree } (upoly r i)$  by (intro upoly-pos-subterm[OF a i], auto)
have lc-er-pos: ?lc er  $> 0$  unfolding er-def
  by (subst lead-coeff-comp[OF deg-r], insert lc-e-pos deg-r lc-r-pos, auto)

from 1[folded poly-pcompose, folded er-def cl-def]
have er-cl-poly:  $0 \leq x \implies \text{poly } er x + \delta \leq \text{poly } cl x$  for x by auto

```

```

have degree er ≤ degree cl
proof (intro degree-mono[of - 0])
  show 0 ≤ ?lc er using lc-er-pos by auto
  show 0 ≤ x ⇒ poly er x ≤ poly cl x for x using er-cl-poly[of x] δ0 by auto
qed
also have degree er = d * degree (upoly r i)
  unfolding er-def d-def by simp
also have degree cl = d * degree (upoly l i)
  unfolding cl-def d-eq by simp
finally have degree (upoly l i) ≥ degree (upoly r i) using d-pos by auto
} note deg-inequality = this

{
  fix p :: 'a mpoly and x
  assume p: monotone-poly {x} p vars p = {x}
  define q where q = mpoly-to-poly x p
  from mpoly-to-poly-inverse[of p x]
  have pq: p = poly-to-mpoly x q using p unfolding q-def by auto
  from pq p(2) have deg: degree q > 0
    by (simp add: degree-mpoly-to-poly degree-pos-iff q-def)
  from deg pq have ∃ q. p = poly-to-mpoly x q ∧ degree q > 0 unfolding q-def
by auto
} note mono-unary-poly = this

{
  fix f
  assume f ∈ {q-sym, h-sym} ∪ v-sym ' V
  hence (f, 1) ∈ F-Q unfolding F-Q-def F-def by auto
  from valid[OF this, unfolded valid-monotone-poly-def] obtain p
    where p: p = I f monotone-poly {..<1} p vars p = {0} by auto
  have id: {..<(1 :: nat)} = {0} by auto
  have ∃ q. I f = poly-to-mpoly 0 q ∧ degree q > 0 unfolding p(1)[symmetric]
    by (intro mono-unary-poly, insert p(2-3)[unfolded id], auto)
} note unary-symbol = this

{
  fix f and n :: nat and x :: var
  assume f ∈ {g-sym, f-sym, a-sym} f = f-sym ⇒ n = 9 f ∈ {a-sym, g-sym}
⇒ n = 2
  hence n: n > 1 and f: (f, n) ∈ F-Q unfolding F-def F-Q-def by force+
  define p where p = I f
  from valid[OF f, unfolded valid-monotone-poly-def, rule-format, OF refl p-def]
  have mono: monotone-poly (vars p) p and vars: vars p = {..<n} and valid:
valid-poly p by auto
  let ?t = Fun f (replicate n (TVar x))
  have t-F: funas-term ?t ⊆ F-Q using f by auto
  have vt: vars-term ?t = {x} using n by auto
  define q where q = eval ?t
  from monotone-poly-eval[OF t-F, unfolded vt, folded q-def]

```

have *monotone-poly* $\{x\}$ *q vars* $q = \{x\}$ **by** *auto*
from *mono-unary-poly*[*OF this*] **obtain** q' **where**
 qq' : $q = \text{poly-to-mpoly } x \ q'$ **and** dq' : *degree* $q' > 0$ **by** *auto*
have $q't$: *poly-to-mpoly* $x \ q' = \text{eval } ?t$ **unfolding** qq' [*symmetric*] *q-def* **by** *simp*
also have $\dots = \text{substitute } (\lambda i. \text{if } i < n \text{ then eval } (\text{replicate } n \ (TVar \ x) \ ! \ i) \ \text{else } 0)$
 $0) \ p$
by (*simp add: p-def*[*symmetric*])
also have $(\lambda i. \text{if } i < n \text{ then eval } (\text{replicate } n \ (TVar \ x) \ ! \ i) \ \text{else } 0) = (\lambda i. \text{if } i < n \text{ then } PVar \ x \ \text{else } 0)$
by (*intro ext, auto*)
also have $\text{substitute } \dots \ p = \text{substitute } (\lambda i. \ PVar \ x) \ p$ **using** *vars*
unfolding *substitute-def* **using** *vars-replace-coeff*[*of Const, OF Const-0*]
by (*intro insertion-irrelevant-vars, auto*)
finally have eq : *poly-to-mpoly* $x \ q' = \text{substitute } (\lambda i. \ PVar \ x) \ p$.
have $\exists \ p \ q. \ I \ f = p \wedge \text{eval } ?t = \text{poly-to-mpoly } x \ q \wedge \text{poly-to-mpoly } x \ q =$
 $\text{substitute } (\lambda i. \ PVar \ x) \ p \wedge \text{degree } q > 0$
 $\wedge \text{vars } p = \{..<n\} \wedge \text{monotone-poly } (\text{vars } p) \ p \wedge \text{valid-poly } p$
by (*intro exI*[*of - p*] *exI*[*of - q'*] *conjI* *valid eq dq' p-def*[*symmetric*] $q't$ [*symmetric*]
mono vars)
} note *g-f-a-sym = this*

from *unary-symbol*[*of q-sym*] **obtain** q **where** Iq : $I \ q\text{-sym} = \text{poly-to-mpoly } 0 \ q$
and dq : *degree* $q > 0$ **by** *auto*
from *unary-symbol*[*of h-sym*] **obtain** h **where** Ih : $I \ h\text{-sym} = \text{poly-to-mpoly } 0 \ h$
and dh : *degree* $h > 0$ **by** *auto*

from *g-f-a-sym*[*of f-sym 9, of y3*] **obtain** $f \ fu$ **where**
 I_f : $I \ f\text{-sym} = f$
and $eval\text{-fyy}$: $\text{eval } (\text{Fun } f\text{-sym } (\text{replicate } 9 \ (TVar \ y3))) = \text{poly-to-mpoly } y3 \ fu$
and $poly\text{-f}$: *poly-to-mpoly* $y3 \ fu = \text{substitute } (\lambda i. \ PVar \ y3) \ f$
and df : $0 < \text{degree } fu$
and $vars\text{-f}$: $\text{vars } f = \{..<9\}$
and $mono\text{-f}$: *monotone-poly* $(\text{vars } f) \ f$
and $valid\text{-f}$: *valid-poly* f **by** *auto*

from *g-f-a-sym*[*of a-sym 2, of y5*] **obtain** $a \ au$ **where**
 I_a : $I \ a\text{-sym} = a$
and $eval\text{-ayy}$: $\text{eval } (\text{Fun } a\text{-sym } (\text{replicate } 2 \ (TVar \ y5))) = \text{poly-to-mpoly } y5 \ au$
and $poly\text{-a}$: *poly-to-mpoly* $y5 \ au = \text{substitute } (\lambda i. \ PVar \ y5) \ a$
and da : $0 < \text{degree } au$
and $vars\text{-a}$: $\text{vars } a = \{..<2\}$
and $valid\text{-a}$: *valid-poly* a
and $mono\text{-a}$: *monotone-poly* $(\text{vars } a) \ a$ **by** *auto*

with *g-f-a-sym*[*of a-sym 2, of y6*] **obtain** au' **where**
 $eval\text{-ayy}'$: $\text{eval } (\text{Fun } a\text{-sym } (\text{replicate } 2 \ (TVar \ y6))) = \text{poly-to-mpoly } y6 \ au'$
and $poly\text{-a}'$: *poly-to-mpoly* $y6 \ au' = \text{substitute } (\lambda i. \ PVar \ y6) \ a$
and da' : $0 < \text{degree } au'$
by *auto*

```

from g-f-a-sym[of g-sym 2, of y2] obtain g gu where
  Ig:  $I\ g\text{-sym} = g$ 
  and eval-gyy:  $\text{eval } (\text{Fun } g\text{-sym } (\text{replicate } 2\ (\text{TVar } y2))) = \text{poly-to-mpoly } y2\ gu$ 
  and poly-g:  $\text{poly-to-mpoly } y2\ gu = \text{substitute } (\lambda i. \text{PVar } y2)\ g$ 
  and dg:  $0 < \text{degree } gu$ 
  and vars-g:  $\text{vars } g = \{..<2\}$ 
  and valid-g:  $\text{valid-poly } g$ 
  and mono-g:  $\text{monotone-poly } (\text{vars } g)\ g$  by auto

from unary-symbol[of v-sym i for i] have  $\forall i. \exists q. i \in V \longrightarrow I\ (v\text{-sym } i) =$ 
poly-to-mpoly  $0\ q \wedge 0 < \text{degree } q$  by auto
from choice[OF this] obtain v where
  Iv:  $i \in V \implies I\ (v\text{-sym } i) = \text{poly-to-mpoly } 0\ (v\ i)$  and
  dv:  $i \in V \implies \text{degree } (v\ i) > 0$ 
for i by auto

have eval-pm-Var:  $\text{eval } (\text{TVar } y) = \text{poly-to-mpoly } y\ [:0,1:]$  for y
  unfolding eval.simps mpoly-of-poly-is-poly-to-mpoly[symmetric] by simp
have id:  $(\text{if } 0 = (0 :: \text{nat}) \text{ then } \text{eval } ([t]! 0) \text{ else } 0) = \text{eval } t$  for t by simp

{
  fix y
  have y:  $\text{eval } (\text{TVar } y) = \text{poly-to-mpoly } y\ [:0,1:]$  (is  $- = \text{poly-to-mpoly } -\ ?\text{poly1}$ )
by fact
  have hy:  $\text{eval } (\text{Fun } h\text{-sym } [\text{TVar } y]) = \text{poly-to-mpoly } y\ h$  using Ih
    apply (simp)
    apply (subst substitute-poly-to-mpoly[of - - y ?poly1])
    apply (unfold id, intro y)
    by simp
  have qy:  $\text{eval } (\text{Fun } q\text{-sym } [\text{TVar } y]) = \text{poly-to-mpoly } y\ q$  using Iq
    apply (simp)
    apply (subst substitute-poly-to-mpoly[of - - y ?poly1])
    apply (unfold id, intro y)
    by simp
  have qhy:  $\text{eval } (\text{Fun } q\text{-sym } [\text{Fun } h\text{-sym } [\text{TVar } y]]) = \text{poly-to-mpoly } y\ (\text{pcompose } q\ h)$  using Iq
    apply (simp)
    apply (subst substitute-poly-to-mpoly[of - - y h])
    apply (unfold id, intro hy)
    by simp
  have hqy:  $\text{eval } (\text{Fun } h\text{-sym } [\text{Fun } q\text{-sym } [\text{TVar } y]]) = \text{poly-to-mpoly } y\ (\text{pcompose } h\ q)$  using Ih
    apply (simp)
    apply (subst substitute-poly-to-mpoly[of - - y q])
    apply (unfold id, intro qy)
    by simp
  have hhqy:  $\text{eval } (\text{Fun } h\text{-sym } [\text{Fun } h\text{-sym } [\text{Fun } q\text{-sym } [\text{TVar } y]]]) = \text{poly-to-mpoly } y\ (\text{pcompose } h\ (\text{pcompose } h\ q))$ 

```

```

apply (simp)
apply (subst Ih)
apply (subst substitute-poly-to-mpoly[of - - y pcompose h q])
  apply (unfold id, intro hqy)
  by simp
{
  assume y: y = 0
  have l: eval (l 0) = poly-to-mpoly 0 (pcompose q h) unfolding
    l-def lhs-Q-def using y qhy by (simp add: Ih y1-def)
  have r: eval (r 0) = poly-to-mpoly 0 (pcompose h (pcompose h q)) unfolding
    r-def rhs-Q-def using y hqy by (simp add: Ih y1-def)
  from deg-inequality[of 0, unfolded upoly-def l r poly-to-mpoly-inverse]
  have dh: degree h = 1 using dq and dh by auto
} note hy qy this
}
hence dh: degree h = 1
and hy:  $\bigwedge y. eval (Fun h-sym [TVar y]) = poly-to-mpoly y h$ 
and qy:  $\bigwedge y. eval (Fun q-sym [TVar y]) = poly-to-mpoly y q$ 
by auto

{
  have l: eval (l 1) = poly-to-mpoly 1 h unfolding
    l-def lhs-Q-def using hy by (simp add: Ih y2-def)
  have eval (r 1) = eval (Fun g-sym (replicate 2 (TVar y2))) unfolding r-def
rhs-Q-def
  apply (simp)
  apply (intro arg-cong[of - -  $\lambda x. substitute\ x\ -$ ] ext)
  subgoal for i by (cases i; cases i - 1; auto)
  done
  also have  $\dots = poly-to-mpoly\ y2\ gu$  by fact
  finally have r: eval (r 1) = poly-to-mpoly 1 gu by (auto simp: y2-def)
  from deg-inequality[of 1, unfolded upoly-def l r poly-to-mpoly-inverse] dh dg
  have degree gu = 1 by auto
  with subst-same-var-monotone-imp-same-degree[OF mono-g poly-g]
  have total-degree g = 1 by auto
}
hence dg: total-degree g = 1 by auto

{
  have l: eval (l 2) = poly-to-mpoly 2 h unfolding
    l-def lhs-Q-def using hy by (simp add: Ih y3-def)
  have eval (r 2) = eval (Fun f-sym (replicate 9 (TVar y3))) unfolding r-def
rhs-Q-def
  by simp
  also have  $\dots = poly-to-mpoly\ y3\ fu$  by fact
  finally have r: eval (r 2) = poly-to-mpoly 2 fu by (auto simp: y3-def)
  from deg-inequality[of 2, unfolded upoly-def l r poly-to-mpoly-inverse] df dh
  have degree fu = 1 by auto
}

```

```

with subst-same-var-monotone-imp-same-degree[OF mono-f poly-f]
have total-degree f = 1 by auto
}
hence df: total-degree f = 1 by auto

{
  fix gs g
  assume gs: (gs,1) ∈ F-Q and I: I gs = poly-to-mpoly 0 g and dg: degree g =
1
from valid[OF gs, unfolded valid-monotone-poly-def, rule-format, OF refl I[symmetric]]
have valid: valid-poly (poly-to-mpoly 0 g) monotone-poly {..<1} (poly-to-mpoly
0 g)
  vars (poly-to-mpoly 0 g) = {..<1}
  by auto
hence mono: monotone-poly (vars (I gs)) (I gs) unfolding I by auto
have total-degree (I gs) = 1 unfolding dg[symmetric]
proof (rule subst-same-var-monotone-imp-same-degree[OF mono, of 0])
  show poly-to-mpoly 0 g = substitute (λi. PVar 0) (I gs) unfolding I
  by (intro mpoly-extI, auto simp: insertion-substitute)
qed
hence total-degree (I gs) ≤ 1 by auto
from monotone-linear-poly-to-coeffs[OF this valid[folded I]]
obtain c a where I': I gs = Const c + Const a * PVar 0 and pos: 0 ≤ c 1
≤ a
  by auto
from I' have I gs = poly-to-mpoly 0 [:c, a:]
  unfolding mpoly-of-poly-is-poly-to-mpoly[symmetric] by simp
from arg-cong[OF this[unfolded I], of mpoly-to-poly 0]
have g = [:c,a:] by (simp add: poly-to-mpoly-inverse)
with I' pos have  $\exists c a. I gs = Const c + Const a * PVar 0 \wedge 0 \leq c \wedge 1 \leq$ 
a  $\wedge g = [:c,a:]$  by auto
} note unary-linear = this[unfolded F-Q-def F-def]

from unary-linear[OF - Ih dh] obtain h0 h1 where
  Ih': I h-sym = Const h0 + Const h1 * PVar 0
  and h0: 0 ≤ h0
  and h1: 1 ≤ h1
  and h: h = [:h0,h1:]
  by auto

from df have total-degree f ≤ 1 by auto
from monotone-linear-poly-to-coeffs[OF this valid-f mono-f[unfolded vars-f] vars-f]

obtain f0 fi where f: f = Const f0 + (∑ i←[0..<9]. Const (fi i) * PVar i)
and f0: 0 ≤ f0 and fi: ∧ i. i < 9 ⇒ 1 ≤ fi i
by auto

```

```

from dg have total-degree  $g \leq 1$  by auto
from monotone-linear-poly-to-coeffs[OF this valid-g mono-g[unfolded vars-g] vars-g]

obtain g0 gi where  $g: g = \text{Const } g0 + (\sum i \leftarrow [0..<2]. \text{Const } (gi\ i) * \text{PVar } i)$ 
  and  $g0: 0 \leq g0$  and  $gi: \bigwedge i. i < 2 \implies 1 \leq gi\ i$ 
  by auto
define g1 where  $g1 = gi\ 0$ 
define g2 where  $g2 = gi\ 1$ 
have id2:  $[0..<2] = [0,1 :: \text{nat}]$  by code-simp
from gi[of 0] gi[of 1] have g1:  $g1 \geq 1$  and g2:  $g2 \geq 1$  by (auto simp: g1-def
g2-def)
have  $g: g = \text{Const } g0 + \text{Const } g1 * \text{PVar } 0 + \text{Const } g2 * \text{PVar } 1$ 
  unfolding g g1-def g2-def by (auto simp: id2)

define  $\alpha$  where  $\alpha = (\lambda x :: \text{var}. 0 :: 'a)$ 
have  $\alpha: \text{assignment } \alpha$  unfolding  $\alpha$ -def assignment-def by auto
{
  fix  $i :: \text{nat}$ 
  assume  $i: i < 9$ 
  from i have  $i \in \text{set } [0..<9]$  by auto
  from split-list[OF this] obtain bef aft where  $id: [0..<9] = \text{bef } @ [i] @ \text{aft}$  by
auto
  define ba where  $ba = \text{bef } @ \text{aft}$ 
  have distinct  $[0..<9]$  by simp
  from this[unfolded id]
  have  $i \notin \text{set } (\text{bef } @ \text{aft})$  by auto
  with id have iba:  $\text{set } ba = \{0..<9\} - \{i\}$  unfolding ba-def
  by (metis Diff-insert-absorb Un-insert-right append-Cons append-Nil list.simps(15))
set-append set-upt)
  have len:  $\text{length } [0..<9] = 9$  by simp
  define diff where  $\text{diff} = (\sum x \leftarrow ba. fi\ x * \text{insertion } \alpha (eval\ (r\ x))) - (\sum x \leftarrow ba. fi\ x * \text{insertion } \alpha (eval\ (l\ x))) + \delta$ 
  {
    fix  $x :: 'a$ 
    assume  $x: x \geq 0$ 
    define a where  $a = \alpha(i := x)$ 
    have a: assignment a using  $\alpha$  unfolding a-def assignment-def using x by
auto
    from gt[unfolded gt-poly-def, rule-format, OF this]
    have  $\text{insertion } a (eval\ \text{rhs-}Q) + \delta \leq \text{insertion } a (eval\ \text{lhs-}Q)$  by auto
    also have  $\text{insertion } a (eval\ \text{lhs-}Q) = f0 + (\sum x \leftarrow [0..<9]. fi\ x * \text{insertion } a (eval\ (l\ x)))$ 
    unfolding lhs-Q eval.simps If f length-map len insertion-substitute inser-
tion-add insertion-Const
      insertion-sum-list insertion-mult map-map o-def insertion-Var
    by (intro arg-cong[of - -  $\lambda x. (+) - (\text{sum-list } x)$ ] map-cong refl arg-cong[of -
- (*) -], simp)
    also have  $(\sum x \leftarrow [0..<9]. fi\ x * \text{insertion } a (eval\ (l\ x))) =$ 
       $(\sum x \leftarrow ba. fi\ x * \text{insertion } a (eval\ (l\ x))) + fi\ i * \text{insertion } a (eval\ (l\ i))$ 

```


unfolding *id ba-def by simp*
also have $(\sum x \leftarrow \text{ba}. \text{fi } x * \text{insertion } a (\text{eval } (l \ x))) = (\sum x \leftarrow \text{ba}. \text{fi } x * \text{insertion } \alpha (\text{eval } (l \ x)))$
apply (*intro arg-cong[of - - sum-list] map-cong refl arg-cong[of - - (*) -] insertion-irrelevant-vars*)
subgoal for *v j unfolding iba using eval-l-r[of v l] by (auto simp: a-def)*

done
also have $\text{insertion } a (\text{eval } \text{rhs-}Q) = f0 + (\sum x \leftarrow [0..<9]. \text{fi } x * \text{insertion } a (\text{eval } (r \ x)))$
unfolding *rhs-Q eval.simps If f length-map len insertion-substitute insertion-add insertion-Const*
insertion-sum-list insertion-mult map-map o-def insertion-Var
by (*intro arg-cong[of - - $\lambda x. (+) - (\text{sum-list } x)$] map-cong refl arg-cong[of - - (*) -], simp*)
also have $(\sum x \leftarrow [0..<9]. \text{fi } x * \text{insertion } a (\text{eval } (r \ x))) = (\sum x \leftarrow \text{ba}. \text{fi } x * \text{insertion } a (\text{eval } (r \ x))) + \text{fi } i * \text{insertion } a (\text{eval } (r \ i))$
unfolding *id ba-def by simp*
also have $(\sum x \leftarrow \text{ba}. \text{fi } x * \text{insertion } a (\text{eval } (r \ x))) = (\sum x \leftarrow \text{ba}. \text{fi } x * \text{insertion } \alpha (\text{eval } (r \ x)))$
apply (*intro arg-cong[of - - sum-list] map-cong refl arg-cong[of - - (*) -] insertion-irrelevant-vars*)
subgoal for *v j unfolding iba using eval-l-r[of v r] by (auto simp: a-def)*

done
finally have *ineq: fi i * insertion a (eval (r i)) ≤ fi i * insertion a (eval (l i)) - diff*
unfolding *diff-def by (simp add: algebra-simps)*
from *fi[OF i] have fi: fi i ≠ 0 and inv: inverse (fi i) ≥ 0 by auto*
from *mult-left-mono[OF ineq inv]*
have $\text{insertion } a (\text{eval } (r \ i)) \leq \text{insertion } a (\text{eval } (l \ i)) + (- \text{inverse } (\text{fi } i) * \text{diff})$
using *fi by (simp add: field-simps)*
}
hence $\exists \text{diff}. \forall x \geq 0. \text{insertion } (\alpha(i := x)) (\text{eval } (r \ i)) \leq \text{insertion } (\alpha(i := x)) (\text{eval } (l \ i)) + \text{diff}$
by *blast*
}
hence $\forall i. \exists \text{diff}. i < 9 \longrightarrow (\forall x \geq 0. \text{insertion } (\alpha(i := x)) (\text{eval } (r \ i)) \leq \text{insertion } (\alpha(i := x)) (\text{eval } (l \ i)) + \text{diff})$
by *auto*
from *choice[OF this]*

Inequality (2) in paper

obtain *diff where inequality2: $\bigwedge i \ x. i < 9 \implies x \geq 0 \implies \text{insertion } (\alpha(i := x)) (\text{eval } (r \ i)) \leq \text{insertion } (\alpha(i := x)) (\text{eval } (l \ i)) + \text{diff } i$*
by *auto*

note *[simp] = insertion-mult insertion-add insertion-substitute*

```

define delt2 where delt2 = h0 + diff 1 - g0
{
  fix x
  assume  $x \geq (0 :: 'a)$ 
  from inequality2[of 1, OF - this]
  have insertion ( $\alpha(1 := x)$ ) (eval (r 1))  $\leq$  insertion ( $\alpha(1 := x)$ ) (eval (l 1)) +
diff 1 by auto
  also have insertion ( $\alpha(1 := x)$ ) (eval (r 1)) = g0 + g1 * x + g2 * x
    by (simp add: r-def rhs-Q-def Ig g y2-def)
  also have insertion ( $\alpha(1 := x)$ ) (eval (l 1)) = h0 + x * h1
    by (simp add: l-def lhs-Q-def Ih h y2-def)
  finally have (g1 + g2 - h1) * x  $\leq$  delt2 unfolding delt2-def
    by (simp add: algebra-simps)
} note ineq2 = this
from bounded-negative-factor[OF this] have g1 + g2  $\leq$  h1 by auto
with g1 g2 have h1:  $h1 \geq 2$  by auto

{
  assume degree q = 1
  from unary-linear[OF - Iq this]
  obtain q0 q1 where Iq': I q-sym = Const q0 + Const q1 * PVar 0
    and q0:  $0 \leq q0$  and q1:  $1 \leq q1$  and q:  $q = [ :q0, q1 : ]$ 
    by auto
  define d1 where d1 = h0 + h0 * h1 + h1 * h1 * q0
  define d2 where d2 = q0 + h0 * q1
  define delt1 where delt1 = d2 + diff 0 - d1
  define fact1 where fact1 = (q1 * h1 * h1 - h1 * q1)
  {
    fix x :: 'a'
    assume  $x \geq 0$ 
    from inequality2[of 0, OF - this]
    have insertion ( $\alpha(0 := x)$ ) (eval (r 0))  $\leq$  insertion ( $\alpha(0 := x)$ ) (eval (l 0))
+ diff 0 by auto
    also have insertion ( $\alpha(0 := x)$ ) (eval (r 0)) = d1 + q1 * h1 * h1 * x
      by (simp add: r-def rhs-Q-def Ih h Iq q y1-def field-simps d1-def)
    also have insertion ( $\alpha(0 := x)$ ) (eval (l 0)) = d2 + h1 * q1 * x
      by (simp add: l-def lhs-Q-def Ih h Iq q y1-def field-simps d2-def)
    finally have fact1 * x  $\leq$  delt1 by (simp add: field-simps delt1-def fact1-def)
  } note ineq1 = this
  from bounded-negative-factor[OF this]
  have fact1  $\leq 0$  .
  from this[unfolded fact1-def] h1 q1 have False by auto
}
with dq have dq: degree q  $\geq 2$  by (cases degree q; cases degree q - 1; auto)

have (z-sym, 0)  $\in$  F-Q unfolding F-def F-Q-def by auto
from valid[OF this, unfolded valid-monotone-poly-def, rule-format, OF refl refl]

```

obtain z **where** $Iz: I z\text{-sym} = z$ **and** $\text{vars-}z: \text{vars } z = \{\}$ **and** $\text{valid-}z: \text{valid-poly } z$ **by auto**
from $\text{vars-empty-Const}[OF \text{ vars-}z]$ **obtain** $z0$ **where** $z: z = \text{Const } z0$ **by auto**
from $\text{valid-}z[\text{unfolded valid-poly-def}, \text{rule-format}, OF \alpha, \text{unfolded } z]$ **have** $z0: z0 \geq 0$ **by auto**

{
fix i
assume $i \in V$
hence $v\text{-sym } i \in \{q\text{-sym}, h\text{-sym}\} \cup v\text{-sym } 'V$ **by auto**
note $\text{unary-symbol}[OF \text{ this}]$
}
hence $\forall i. \exists q. i \in V \longrightarrow I (v\text{-sym } i) = \text{poly-to-mpoly } 0 \ q \wedge 0 < \text{degree } q$ **by auto**
from $\text{choice}[OF \text{ this}]$ **obtain** v **where** $Iv: \bigwedge i. i \in V \implies I (v\text{-sym } i) = \text{poly-to-mpoly } 0 \ (v \ i)$
and $dv: \bigwedge i. i \in V \implies 0 < \text{degree } (v \ i)$
by auto

define $\text{const-}t$ **where** $\text{const-}t = \text{insertion } \alpha \ (\text{eval } t\text{-}t)$
have $\text{const-}t: \text{const-}t > 0$
unfolding $\text{const-}t\text{-def}$
by $(\text{rule } \text{eval-}t\text{-}t\text{-gt-}0[OF \text{ Ig}[\text{unfolded } g] \ Iz[\text{unfolded } z]], \text{insert } z0 \ g0 \ g1 \ g2, \text{auto})$

{
define $d1$ **where** $d1 = g0 + g2 * h0 + g2 * h1 * h0 + g2 * h1 * h1 * h0$
define c **where** $c = g0 + g2 * \text{const-}t$
define $\text{delt}4$ **where** $\text{delt}4 = d1 + \text{diff } 3$
have $[\text{simp}]: \text{insertion } a \ (\text{eval } t\text{-}t) = \text{const-}t$ **for a unfolding** $\text{const-}t\text{-def}$
by $(\text{rule } \text{insertion-irrelevant-vars}, \text{insert vars-}t \ \text{vars-eval}, \text{force})$
let $?qq = q \circ_p [:c, g1:] - \text{smult } g1 \ q$
define qq **where** $qq = ?qq$
define hhh **where** $hhh = [:delt4, g2 * h1 * h1 * h1:]$
{
fix $x :: 'a$
assume $x: x \geq 0$
from $\text{inequality2}[of \ 3, OF - \text{this}]$
have $\text{insertion } (\alpha(3 := x)) \ (\text{eval } (r \ 3)) \leq \text{insertion } (\alpha(3 := x)) \ (\text{eval } (l \ 3))$
+ $\text{diff } 3$ **by auto**
also have $\text{insertion } (\alpha(3 := x)) \ (\text{eval } (r \ 3)) = \text{poly } q \ (g0 + g1 * x + g2 * \text{const-}t)$
by $(\text{simp add: } r\text{-def } rhs\text{-}Q\text{-def } y4\text{-def } Iq \ Ig \ g)$
also have $\text{insertion } (\alpha(3 := x)) \ (\text{eval } (l \ 3)) =$
 $g1 * \text{poly } q \ x + g2 * h1 * h1 * h1 * x + d1$
by $(\text{simp add: } l\text{-def } lhs\text{-}Q\text{-def } y4\text{-def } Iq \ Ig \ g \ Ih \ h \ \text{field-simps } d1\text{-def})$
finally have $\text{poly } q \ (g0 + g1 * x + g2 * \text{const-}t) - \text{poly } (\text{smult } g1 \ q) \ x - g2$
}

```

* h1 * h1 * h1 * x ≤ delt4
  by (simp add: delt4-def)
  also have g2 * h1 * h1 * h1 * x = poly [:0, g2 * h1 * h1 * h1:] x by simp
  also have poly q (g0 + g1 * x + g2 * const-t) = poly (pcompose q [:c, g1 :])
x
  by (simp add: poly-pcompose ac-simps c-def)
  finally have poly qq x ≤ poly hhh x
  by (simp add: qq-def hhh-def)
} note ineq3 = this

have lq0: lead-coeff q > 0
proof (rule ccontr)
  assume ¬ ?thesis
  with dq have lq: lead-coeff (- q) > 0 by (cases q = 0, auto)
  from poly-pinfty-ge[OF this, of 1] dq obtain n where  $\bigwedge x. x \geq n \implies \text{poly } q x \leq -1$  by auto
  from this[of max n 0] have 1: poly q (max n 0) ≤ - 1 by auto
  let ?a =  $\lambda x :: \text{var. max } n 0$ 
  have a: assignment ?a unfolding assignment-def by auto
  have (q-sym,1) ∈ F-Q unfolding F-Q-def by auto
  from valid[OF this, unfolded valid-monotone-poly-def, rule-format, OF refl
Iq[symmetric]]
  have valid-poly (poly-to-mpoly 0 q) by auto
  from this[unfolded valid-poly-def, rule-format, OF a]
  have 0 ≤ poly q (max n 0) by auto
  with 1 show False by auto
qed

from const-t g0 g2 have c: c > 0 unfolding c-def
by (metis le-add-same-cancel2 linorder-not-le mult-less-cancel-right2 order-le-less-trans
order-less-le)

have degree hhh ≤ 1 unfolding hhh-def by simp

from criterion-for-degree-2[OF qq-def dq ineq3 this g1 lq0 c]
have degree q = 2 g1 = 1 by auto
}
hence dq: degree q = 2 and g1: g1 = 1 by auto

{
  have l: eval (l 4) = poly-to-mpoly 4 q unfolding
  l-def lhs-Q-def using qy by (simp add: y5-def)
  have eval (r 4) = eval (Fun a-sym (replicate 2 (TVar y5))) unfolding r-def
rhs-Q-def
  apply (simp)
  apply (intro arg-cong[of - -  $\lambda x. \text{substitute } x$  -] ext)
  subgoal for i by (cases i; cases i - 1; auto)
  done
}

```

```

also have ... = poly-to-mpoly y5 au by fact
finally have r: eval (r 4) = poly-to-mpoly 4 au by (auto simp: y5-def)
from deg-inequality[of 4, unfolded upoly-def l r poly-to-mpoly-inverse]
have degree au ≤ degree q by auto
with subst-same-var-monotone-imp-same-degree[OF mono-a poly-a]
have total-degree a ≤ degree q by auto
}
hence d-aq: total-degree a ≤ degree q by auto

{
  have r: eval (r 5) = poly-to-mpoly 5 q unfolding
    r-def rhs-Q-def using qy by (simp add: y6-def)
  have eval (l 5) = eval (Fun a-sym (replicate 2 (TVar y6))) unfolding l-def
lhs-Q-def
  apply (simp)
  apply (intro arg-cong[of - - λ x. substitute x -] ext)
  subgoal for i by (cases i; cases i - 1; auto)
  done
  also have ... = poly-to-mpoly y6 au' by fact
  finally have l: eval (l 5) = poly-to-mpoly 5 au' by (auto simp: y6-def)
  from deg-inequality[of 5, unfolded upoly-def l r poly-to-mpoly-inverse]
  have degree q ≤ degree au' by auto
  with subst-same-var-monotone-imp-same-degree[OF mono-a poly-a'] da'
  have degree q ≤ total-degree a by auto
}

with d-aq
have d-aq: total-degree a = degree q by auto

with dq have da: total-degree a = 2 by simp
have vars a = {0,1} unfolding vars-a by code-simp

from binary-degree-2-poly[OF - this] da
obtain a0 a1 a2 a3 a4 a5 where a: a = Const a0 + Const a1 * PVar 0 +
Const a2 * PVar 1 +
  Const a3 * PVar 0 * PVar 0 + Const a4 * PVar 1 * PVar 1 +
  Const a5 * PVar 0 * PVar 1 by auto

define d1 where d1 = a0 + a1 * z0 + a3 * z0 * z0
define d2 where d2 = (a2 + a5 * z0)
define delt7 where delt7 = diff 6 - d1
{
  fix x
  assume x ≥ (0 :: 'a)
  from inequality2[of 6, OF - this]
  have insertion (α(6 := x)) (eval (r 6)) ≤ insertion (α(6 := x)) (eval (l 6)) +
diff 6 by auto
}

```

```

also have insertion ( $\alpha(6 := x)$ ) (eval (r 6)) =  $a_4 * x * x + d_2 * x + d_1$ 
  by (simp add: r-def rhs-Q-def Ig g y7-def Ia a Iz z algebra-simps d1-def d2-def)
also have insertion ( $\alpha(6 := x)$ ) (eval (l 6)) =  $x$ 
  by (simp add: l-def lhs-Q-def Ih h y7-def)
finally have  $0 \geq \text{poly } [-\text{delt}7, d_2 - 1, a_4:] x$  unfolding delt7-def
  by (simp add: algebra-simps)
} note ineq7 = this
{
  define p where p =  $[-\text{delt}7, d_2 - 1, a_4:]$ 
  assume  $a_4 > 0$ 
  hence lead-coeff p > 0 degree p > 0 by (auto simp: p-def)
  with poly-pinfty-ge[OF this(1), of 1] obtain n where  $\bigwedge x. x \geq n \implies 1 \leq \text{poly}$ 
  p x by blast
  from this[of max n 0] ineq7[of max n 0] have False unfolding p-def by auto
}
hence  $a_4: a_4 \leq 0$  by force

note valid-a = valid-a[unfolded a valid-poly-def, rule-format]
{
  define p where p =  $[-a_0, -a_2, -a_4:]$ 
  assume  $a_4 < 0$ 
  hence p: lead-coeff p > 0 degree p  $\neq 0$  unfolding p-def by auto
  {
    fix x :: 'a
    assume  $x \geq 0$ 
    hence assignment ( $\lambda v. \text{if } v = 1 \text{ then } x \text{ else } 0$ ) unfolding assignment-def by
    auto
    from valid-a[OF this]
    have  $0 \geq \text{poly } p x$  by (auto simp: algebra-simps p-def)
  }
  with poly-pinfty-ge[OF p] have False
  by (metis (no-types, opaque-lifting) dual-order.trans nle-le not-one-le-zero)
}
with  $a_4$  have  $a_4: a_4 = 0$  by force

define d1 where  $d_1 = a_0 + a_2 * z_0$ 
define d2 where  $d_2 = (a_5 * z_0 + a_1)$ 
define delt8 where  $\text{delt}8 = \text{diff } 7 - d_1$ 
{
  fix x
  assume  $x \geq (0 :: 'a)$ 
  from inequality2[of 7, OF - this]
  have insertion ( $\alpha(7 := x)$ ) (eval (r 7))  $\leq$  insertion ( $\alpha(7 := x)$ ) (eval (l 7)) +
  diff 7 by auto
  also have insertion ( $\alpha(7 := x)$ ) (eval (r 7)) =  $d_1 + a_3 * (x * x) + d_2 * x$ 
  by (simp add: r-def rhs-Q-def Ig g y8-def Ia a  $a_4$  Iz z algebra-simps d1-def
  d2-def)
  also have insertion ( $\alpha(7 := x)$ ) (eval (l 7)) =  $x$ 

```

```

    by (simp add: l-def lhs-Q-def Ih h y8-def)
  finally have 0 ≥ poly [:-delt8,d2 - 1,a3:] x unfolding delt8-def
    by (simp add: algebra-simps)
} note ineq8 = this
{
  define p where p = [:-delt8,d2 - 1,a3:]
  assume a3 > 0
  hence lead-coeff p > 0 degree p > 0 by (auto simp: p-def)
  with poly-pinfty-ge[OF this(1), of 1] obtain n where  $\bigwedge x. x \geq n \implies 1 \leq \text{poly}$ 
p x by blast
  from this[of max n 0] ineq8[of max n 0] have False unfolding p-def by auto
}
hence a3: a3 ≤ 0 by force
{
  define p where p = [:-a0,-a1,-a3:]
  assume a3 < 0
  hence p: lead-coeff p > 0 degree p ≠ 0 unfolding p-def by auto
  {
    fix x :: 'a
    assume x ≥ 0
    hence assignment (λ v. if v = 0 then x else 0) unfolding assignment-def by
auto
    from valid-a[OF this, simplified]
    have 0 ≥ poly p x by (auto simp: algebra-simps p-def)
  }
  with poly-pinfty-ge[OF p] have False
    by (metis (no-types, opaque-lifting) dual-order.trans nle-le not-one-le-zero)
}
with a3 have a3: a3 = 0 by force

from a a3 a4 have a: a = Const a5 * PVar 0 * PVar 1 + Const a1 * PVar 0
+ Const a2 * PVar 1 + Const a0 by simp
note valid-a = valid-a[unfolded a3 a4]
from valid-a[OF α, simplified, unfolded α-def]
have a0: a0 ≥ 0 by auto

note mono-a' = mono-a[unfolded monotone-poly-wrt-def, rule-format, unfolded
vars-a, OF α, unfolded a, simplified,
  unfolded α-def, simplified]
from mono-a'[of 0] have a1: δ ≤ x  $\implies$  δ ≤ a1 * x for x by auto
from mono-a'[of 1] have a2: δ ≤ x  $\implies$  δ ≤ a2 * x for x by auto
{
  fix a
  assume a ∈ {a1,a2}
  with a1 a2 have δ ≤ x  $\implies$  δ ≤ a * x for x by auto
  with δ0 have a ≥ 1
  using mult-le-cancel-right1 by auto
  hence a > 0 by simp
}

```

```

}
hence a1: a1 > 0 and a2: a2 > 0 by auto

{
  assume a5: a5 = 0
  from da[unfolded a a5]
  have 2 = total-degree (Const a1 * PVar 0 + Const a2 * PVar (Suc 0) +
Const a0) by simp
  also have ... ≤ 1
    by (intro total-degree-add total-degree-Const-mult, auto)
  finally have False by simp
}
hence a5: a5 ≠ 0 by force
{
  define p where p = [-a0, -a1 -a2, - a5:]
  assume a5: a5 < 0
  hence p: lead-coeff p > 0 degree p ≠ 0 by (auto simp: p-def)
  {
    fix x :: 'a
    assume x ≥ 0
    hence assignment (λ -. x) by (auto simp: assignment-def)
    from valid-a[OF this]
    have 0 ≥ poly p x by (simp add: p-def algebra-simps)
  }
  with poly-pinfy-ge[OF p] have False
    by (metis (no-types, opaque-lifting) dual-order.trans nle-le not-one-le-zero)
}
with a5 have a5: a5 > 0 by force

define I' where I' = (λ f. if f ∈ v-sym ' (UNIV - V) then PVar 0 else I f)
define v' where v' = (λ i. if i ∈ V then v i else [:0,1:])
have Iv': I' (v-sym i) = poly-to-mpoly 0 (v' i) for i
  unfolding I'-def v'-def using Iv by (auto simp: mpoly-of-poly-is-poly-to-mpoly[symmetric])
have dv': 0 < degree (v' i) for i using dv[of i] by (auto simp: v'-def)
have Ia': I' a-sym = a unfolding I'-def using Ia by auto
have Iz': I' z-sym = z unfolding I'-def using Iz by auto
{
  fix i
  have nneg-poly (v' i)
  proof (cases i ∈ V)
    case False
      thus ?thesis by (auto simp: v'-def)
  next
    case i: True
      hence id: v' i = v i by (auto simp: v'-def)
      from i have (v-sym i, 1) ∈ F-Q unfolding F-Q-def F-def by auto
      from valid[OF this, unfolded valid-monotone-poly-def] Iv[OF i]
      have valid: valid-poly (poly-to-mpoly 0 (v i) ) by auto
      define p where p = v i

```



```

have valid:  $0 \leq x \implies 0 \leq \text{poly } p \ x$  for  $x$  unfolding p-def
  using valid[unfolded valid-poly-def, rule-format, of  $\lambda \ . \ x$ ]
  by (auto simp: assignment-def)
hence nneg-poly p by (intro nneg-polyI, auto)
thus ?thesis unfolding id p-def .
qed
} note nneg-v = this

{
  fix  $r \ x$ 
  assume  $r \in \{p, ?q\}$ 
  with pq funas-encode-poly-p[of  $x$ ] funas-encode-poly-q[of  $x$ ]
  have pos: positive-poly r and inF: funas-term (encode-poly x r)  $\subseteq F$  by auto
  from degree-eval-encode-poly-generic[of  $I'$ , unfolded mpoly-of-poly-is-poly-to-mpoly,

    OF Ia'[unfolded a] Iz'[unfolded z] - a5 a1 a2 a0 z0, of v', OF Iv' nneg-v dv'
    pos refl, of x]
  obtain rr where id: poly-to-mpoly x rr = poly-inter.eval I' (encode-poly x r)
    and deg: int (degree rr) = insertion ( $\lambda i. \text{int (degree (v' i))}$ ) r
    and nneg: nneg-poly rr
    by auto
  have poly-to-mpoly x rr = poly-inter.eval I (encode-poly x r) unfolding id
  proof (rule poly-inter-eval-cong)
    fix  $f \ a$ 
    assume  $(f, a) \in \text{funas-term (encode-poly x r)}$ 
    hence  $(f, a) \in F$  using inF by auto
    thus  $I' f = I f$  unfolding F-def I'-def by auto
  qed
  with deg nneg have  $\exists p. \text{mpoly-of-poly } x \ p = \text{eval (encode-poly x r)} \wedge$ 
     $\text{int (degree } p) = \text{insertion } (\lambda i. \text{int (degree (v' i))) } r \wedge \text{nneg-poly } p$ 
    by (auto simp: mpoly-of-poly-is-poly-to-mpoly)
} note encode = this
from encode[of  $p \ y9$ ]
obtain pp where pp: mpoly-of-poly y9 pp = eval (encode-poly y9 p)
   $\text{int (degree } pp) = \text{insertion } (\lambda i. \text{int (degree (v' i))) } p$ 
  nneg-poly pp by auto

from encode[of  $?q \ y9$ ]
obtain qq where qq: mpoly-of-poly y9 qq = eval (encode-poly y9 ?q)
   $\text{int (degree } qq) = \text{insertion } (\lambda i. \text{int (degree (v' i))) } ?q$ 
  nneg-poly qq by auto

define ppp where ppp =  $(pp * [:a1, a5:] + [:a0, a2:])$ 
from deg-inequality[of  $8$ ]
have degree (upoly r 8)  $\leq$  degree (upoly l 8) by simp
also have upoly r 8 = mpoly-to-poly 8
   $(\text{mpoly-of-poly } y9 \ [ : a1, a5 : ] * \text{mpoly-of-poly } y9 \ qq + \text{mpoly-of-poly } y9 \ [ : a0,$ 
   $a2 : ])$ 

```

```

    unfolding r-def rhs-Q-def by (simp add: upoly-def Ia a qq algebra-simps)
  also have ... = qq * [:a1, a5:] + [:a0, a2:] unfolding mpoly-of-poly-add[symmetric]
    mpoly-of-poly-mult[symmetric]
    unfolding mpoly-of-poly-is-poly-to-mpoly y9-def poly-to-mpoly-inverse by simp

  also have degree ... = 1 + degree qq
    by (rule nneg-poly-degree-add-1[OF qq(3)], insert a5 a2, auto)
  also have upoly l 8 = mpoly-to-poly 8
    (mpoly-of-poly y9 [: h0 :] + mpoly-of-poly y9 [: h1:] * (mpoly-of-poly y9 [: a1,
    a5 :] * mpoly-of-poly y9 pp + mpoly-of-poly y9 [: a0, a2:]))
    unfolding l-def lhs-Q-def by (simp add: upoly-def Ih h mpoly-of-poly-is-poly-to-mpoly[symmetric]
    Ia a pp algebra-simps)
  also have ... = [:h0:] + [: h1 :] * ppp unfolding mpoly-of-poly-add[symmetric]
    mpoly-of-poly-mult[symmetric] ppp-def
    unfolding mpoly-of-poly-is-poly-to-mpoly y9-def poly-to-mpoly-inverse by simp

  also have degree ... = degree ([:h1:] * ppp)
    by (metis degree-add-eq-right degree-add-le degree-pCons-0 le-zero-eq zero-less-iff-neq-zero)
  also have ... = degree ppp using h1 by simp
  also have ... = 1 + degree pp unfolding ppp-def
    by (rule nneg-poly-degree-add-1[OF pp(3)], insert a5 a2, auto)
  finally have deg-qq-pp: int (degree qq) ≤ int (degree pp) by simp

  show ?thesis unfolding positive-poly-problem-def[OF pq]
  proof (intro exI[of - (λi. int (Polynomial.degree (v' i)))] conjI deg-qq-pp[unfolded
  pp(2) qq(2)])
    show positive-interpr (λi. int (Polynomial.degree (v' i)))
      unfolding positive-interpr-def using dv' by auto
    qed
  qed
end

context poly-input
begin

corollary polynomial-termination-with-delta-orders-undecidable:
  positive-poly-problem p q ↔
  termination-by-delta-poly-interpretation (TYPE('a :: floor-ceiling)) F-Q Q
proof
  show positive-poly-problem p q ⇒ termination-by-delta-poly-interpretation TYPE('a)
  F-Q Q
    using solution-impl-delta-termination-of-Q by blast
  assume termination-by-delta-poly-interpretation TYPE('a) F-Q Q
  interpret term-delta-poly-input p q TYPE('a)
    by (unfold-locales, fact)
  from solution show positive-poly-problem p q by auto
qed

end

```

end

References

- [1] D. Lankford. On proving term rewrite systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
- [2] Y. Y. Matijasevic. Enumerable sets are diophantine (translated from Russian). In *Soviet Mathematics Doklady*, volume 11, pages 354–358, 1970.
- [3] F. Mitterwallner, A. Middeldorp, and R. Thiemann. Linear termination is undecidable. In *Proceedings of the 39th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, 2024. To appear.