Undecidability Results on Orienting Single Rewrite Rules*

René Thiemann, Fabian Mitterwallner, and Aart Middeldorp

University of Innsbruck

May 3, 2024

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 $\ell \to r$, 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

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

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* ⇒ *'a mpoly*) ⇒ *'a* :: *comm-semiring-1 mpoly* ⇒ *'a mpoly* **where**
  substitute σ p = *insertion σ* (*replace-coeff Const p*)

lemma *Const-0*: *Const 0* = *0*
  ⟨*proof*⟩

lemma *Const-1*: *Const 1* = *1*
  ⟨*proof*⟩

lemma *insertion-Var*: *insertion* α (*Var* x) = α x
⟨*proof*⟩

lemma *insertion-Const*: *insertion* α (*Const* a) = a
⟨*proof*⟩

lemma *insertion-power*: *insertion* α (p^n) = (*insertion* α p)^n
⟨*proof*⟩

lemma *insertion-monom-add*: *insertion* α (*monom* (f + g) a) = *insertion* α (*monom* f *1*) ∗ *insertion* α (*monom* g a)
⟨*proof*⟩

lemma *insertion-uminus*: *insertion* α (− p) = − *insertion* α p
⟨*proof*⟩

lemma *insertion-sum-list*: *insertion* α (*sum-list ps*) = *sum-list* (*map* (*insertion* α) *ps*)
⟨*proof*⟩

lemma *coeff-uminus*: *coeff* (− p) m = − *coeff* p m
⟨*proof*⟩

lemma *insertion-substitute*: *insertion* α (*substitute* σ p) = *insertion* (λ x. *insertion* α (σ x)) p
⟨*proof*⟩

lemma *Const-add*: *Const* (x + y) = *Const* x + *Const* y
⟨*proof*⟩

lemma *substitute-add*[*simp*]: *substitute* σ (p + q) = *substitute* σ p + *substitute* σ q
⟨*proof*⟩

lemma *Const-sum*: *Const* (*sum* f A) = *sum* (*Const o* f) A
⟨*proof*⟩

lemma *Const-sum-list*: *Const* (*sum-list* (*map* f *xs*)) = *sum-list* (*map* (*Const o* f) *xs*)
⟨*proof*⟩

lemma *Const-0-eq*[*simp*]: *Const* x = *0* ⟷ x = *0*
⟨*proof*⟩

lemma *Const-sum-any*: *Const* (*Sum-any* f) = *Sum-any* (*Const o* f)
⟨*proof*⟩

lemma *Const-mult*: *Const* (x ∗ y) = *Const* x ∗ *Const* y
⟨*proof*⟩

lemma *Const-power*: *Const* $(x \mathbin{\hat{}} e)$ = *Const* $x \mathbin{\hat{}} e$
⟨*proof*⟩

lemma *lookup-replace-Const*: *lookup* (*mapping-of* (*replace-coeff Const p*)) l = *Const* (*lookup* (*mapping-of p*) l)
⟨*proof*⟩

lemma *replace-coeff-mult*: *replace-coeff Const* $(p * q)$ = *replace-coeff Const* p * *replace-coeff Const* q
⟨*proof*⟩

lemma *substitute-mult*[*simp*]: *substitute* σ $(p * q)$ = *substitute* σ p * *substitute* σ q
⟨*proof*⟩

lemma *replace-coeff-Var*[*simp*]: *replace-coeff Const* (*Var* x) = *Var* x
⟨*proof*⟩

lemma *replace-coeff-Const*[*simp*]: *replace-coeff Const* (*Const* c) = *Const* (*Const* c)
⟨*proof*⟩

lemma *substitute-Var*[*simp*]: *substitute* σ (*Var* x) = σ x
⟨*proof*⟩

lemma *substitute-Const*[*simp*]: *substitute* σ (*Const* c) = *Const* c
⟨*proof*⟩

lemma *substitute-0*[*simp*]: *substitute* σ *0* = *0*
⟨*proof*⟩

lemma *substitute-1*[*simp*]: *substitute* σ *1* = *1*
⟨*proof*⟩

lemma *substitute-power*[*simp*]: *substitute* σ $(p \hat{} e)$ = (*substitute* σ p)$\hat{}e$
⟨*proof*⟩

lemma *substitute-monom*[*simp*]: *substitute* σ (*monom* (*monomial* e x) c) = *Const* c * $(\sigma\ x)\hat{}e$
⟨*proof*⟩

lemma *substitute-sum-list*: *substitute* σ (*sum-list* (*map* f *xs*)) = *sum-list* (*map* (*substitute* σ *o* f) *xs*)
⟨*proof*⟩

lemma *substitute-sum*: *substitute* σ (*sum* f *xs*) = *sum* (*substitute* σ *o* f) *xs*
⟨*proof*⟩

lemma *substitute-prod*: *substitute σ (prod f xs) = prod (substitute σ o f) xs*
⟨*proof*⟩

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

lemma *set-vars-list*[*simp*]: *set (vars-list p) = vars p*
⟨*proof*⟩

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

lemma *mpoly-coeff-filter*: *coeff (mpoly-coeff-filter f p) m = (coeff p m when f (coeff p m))*
⟨*proof*⟩

lemma *total-degree-add*: **assumes** *total-degree p ≤ d total-degree q ≤ d*
shows *total-degree (p + q) ≤ d*
⟨*proof*⟩

lemma *total-degree-Var*[*simp*]: *total-degree (Var x* :: *'a* :: *comm-semiring-1 mpoly) = Suc 0*
⟨*proof*⟩

lemma *total-degree-Const*[*simp*]: *total-degree (Const x) = 0*
⟨*proof*⟩

lemma *total-degree-Const-mult*: **assumes** *total-degree p ≤ d*
shows *total-degree (Const x ∗ p) ≤ d*
⟨*proof*⟩

lemma *vars-0*[*simp*]: *vars 0 = {}*
⟨*proof*⟩

lemma *vars-1*[*simp*]: *vars 1 = {}*
⟨*proof*⟩

lemma *vars-Var*[*simp*]: *vars (Var x* :: *'a* :: *comm-semiring-1 mpoly) = {x}*
⟨*proof*⟩

lemma *vars-Const*[*simp*]: *vars (Const c) = {}*
⟨*proof*⟩

lemma *coeff-sum-list*: *coeff (sum-list ps) m = (∑ p←ps. coeff p m)*
⟨*proof*⟩

lemma *coeff-Const-mult*: *coeff (Const c ∗ p) m = c ∗ coeff p m*
⟨*proof*⟩

lemma *coeff-Const*: *coeff* (*Const c*) *m* = (*if m* = *0 then* (*c* :: *'a* :: *comm-semiring-1*) *else 0*)
⟨*proof*⟩

lemma *coeff-Var*: *coeff* (*Var x*) *m* = (*if m* = *monomial 1 x then 1* :: *'a* :: *comm-semiring-1 else 0*)
⟨*proof*⟩

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

lift-definition *monom-list* :: *'a* :: *comm-semiring-1 mpoly* ⇒ (*monom* × *'a*) *list*
is λ *p*. *map* (λ *m*. (*m*, *lookup p m*)) (*sorted-list-of-set* (*keys p*)) ⟨*proof*⟩

lift-definition *var-list* :: *monom* ⇒ (*var* × *nat*) *list*
is λ *m*. *map* (λ *x*. (*x*, *lookup m x*)) (*sorted-list-of-set* (*keys m*)) ⟨*proof*⟩

lemma *monom-list*: *p* = (∑ (*m*, *c*) ← *monom-list p*. *monom m c*)
⟨*proof*⟩

lemma *monom-list-coeff*: (*m*,*c*) ∈ *set* (*monom-list p*) ⟹ *coeff p m* = *c*
⟨*proof*⟩

lemma *monom-list-keys*: (*m*,*c*) ∈ *set* (*monom-list p*) ⟹ *keys m* ⊆ *vars p*
⟨*proof*⟩

lemma *var-list*: *monom m c* = *Const* (*c* :: *'a* :: *comm-semiring-1*) ∗ (∏ (*x*, *e*) ← *var-list m*. (*Var x*)^*e*)
⟨*proof*⟩

lemma *var-list-keys*: (*x*,*e*) ∈ *set* (*var-list m*) ⟹ *x* ∈ *keys m*
⟨*proof*⟩

lemma *vars-substitute*: **assumes** ⋀ *x*. *vars* (σ *x*) ⊆ *V*
shows *vars* (*substitute* σ *p*) ⊆ *V*
⟨*proof*⟩

lemma *insertion-monom-nonneg*: **assumes** ⋀ *x*. α *x* ≥ *0* **and** *c*: (*c* :: *'a* :: {*linordered-nonzero-semiring*,*ordered-semiring-0*}) ≥ *0*
shows *insertion* α (*monom m c*) ≥ *0*
⟨*proof*⟩

lemma *insertion-nonneg*: **assumes** ⋀ *x*. α *x* ≥ (*0* :: *'a* :: *linordered-idom*)
and ⋀ *m*. *coeff p m* ≥ *0*
shows *insertion* α *p* ≥ *0*
⟨*proof*⟩

lemma *vars-sumlist*: *vars* (*sum-list ps*) ⊆ ⋃ (*vars* ‘ *set ps*)
⟨*proof*⟩

lemma *coefficients-of-linear-poly*: **assumes** *linear*: *total-degree* (*p* :: *'a* :: *comm-semiring-1 mpoly*) ≤ *1*
shows ∃ *c a vs. p* = *Const c* + (∑ *i*←*vs. Const* (*a i*) ∗ *Var i*)
∧ *distinct vs* ∧ *set vs* = *vars p* ∧ *sorted-list-of-set* (*vars p*) = *vs* ∧ (∀ *v* ∈ *set vs. a v* ≠ *0*)
∧ (∀ *i. a i* = *coeff p* (*monomial 1 i*)) ∧ (*c* = *coeff p 0*)
⟨*proof*⟩

Introduce notion for degree of monom

definition *degree-monom* :: (*var* $\Rightarrow_0$ *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*)))
⟨*proof*⟩

lemma *degree-monon-le-total-degree*: **assumes** *coeff p m* ≠ *0*
shows *degree-monom m* ≤ *total-degree p*
⟨*proof*⟩

lemma *degree-monom-eq-total-degree*: **assumes** *p* ≠ *0*
shows ∃ *m. coeff p m* ≠ *0* ∧ *degree-monom m* = *total-degree p*
⟨*proof*⟩

lemma *degree-add-leI*: *degree p x* ≤ *d* ⟹ *degree q x* ≤ *d* ⟹ *degree* (*p* + *q*) *x* ≤ *d*
⟨*proof*⟩

lemma *degree-sum-leI*: **assumes** ⋀ *i. i* ∈ *A* ⟹ *degree* (*p i*) *x* ≤ *d*
shows *degree* (*sum p A*) *x* ≤ *d*
⟨*proof*⟩

lemma *total-degree-sum-leI*: **assumes** ⋀ *i. i* ∈ *A* ⟹ *total-degree* (*p i*) ≤ *d*
shows *total-degree* (*sum p A*) ≤ *d*
⟨*proof*⟩

lemma *total-degree-monom*: **assumes** *c* ≠ *0*
shows *total-degree* (*monom m c*) = *degree-monom m*
⟨*proof*⟩

lemma *degree-Var*[*simp*]: *degree* (*Var x* :: *'a* :: *comm-semiring-1 mpoly*) *x* = *1*
⟨*proof*⟩

lemma *Var-neq-0*[*simp*]: *Var x* ≠ (*0* :: *'a* :: *comm-semiring-1 mpoly*)
⟨*proof*⟩

lemma *degree-Const*[*simp*]: *degree* (*Const c*) *x* = *0*
⟨*proof*⟩

lemma *vars-add-subI*: $vars\ p \subseteq A \Longrightarrow vars\ q \subseteq A \Longrightarrow vars\ (p + q) \subseteq A$
⟨*proof*⟩

lemma *vars-mult-subI*: $vars\ p \subseteq A \Longrightarrow vars\ q \subseteq A \Longrightarrow vars\ (p * q) \subseteq A$
⟨*proof*⟩

lemma *vars-eqI*: **assumes** $vars\ (p :: 'a :: comm\text{-}ring\text{-}1\ mpoly) \subseteq V$
$\bigwedge v.\ v \in V \Longrightarrow \exists\ a\ b.\ insertion\ a\ p \neq insertion\ (a(v := b))\ p$
shows $vars\ p = V$
⟨*proof*⟩

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* ≡ *MPoly-Type.degree*
abbreviation *mcoeff* **where** *mcoeff* ≡ *MPoly-Type.coeff*
abbreviation *mmonom* **where** *mmonom* ≡ *MPoly-Type.monom*

lemma *range-coeff-poly-to-mpoly*: **assumes** $mcoeff\ (poly\text{-}to\text{-}mpoly\ x\ p)\ m \neq 0$
shows $\exists\ d.\ m = monomial\ d\ x$
⟨*proof*⟩

lemma *degree-poly-to-mpoly*[*simp*]: $mdegree\ (poly\text{-}to\text{-}mpoly\ x\ p)\ x = degree\ p$
⟨*proof*⟩

lemma *degree-mpoly-to-poly*: **assumes** $vars\ p \subseteq \{x\}$
shows $degree\ (mpoly\text{-}to\text{-}poly\ x\ p) = mdegree\ p\ x$
⟨*proof*⟩

lemma *degree-partial-insertion-bound*: $degree\ (partial\text{-}insertion\ a\ x\ p) \leq MPoly\text{-}Type.degree\ p\ x$
⟨*proof*⟩

lemma *insertion-partial-insertion-vars*: **assumes** $\bigwedge y.\ y \neq x \Longrightarrow y \in vars\ p \Longrightarrow \beta\ y = \alpha\ y$

shows *poly* (*partial-insertion* β *x p*) (α *x*) = *insertion* α *p*
⟨*proof*⟩

lemma *degree-mpoly-of-poly*[*simp*]: *mdegree* (*mpoly-of-poly x p*) *x* = *degree p*
⟨*proof*⟩

lemma *mpoly-extI*: **assumes** $\bigwedge$ α. *insertion* α *p* = *insertion* α (*q* :: *'a* :: {*ring-char-0*,*idom*} *mpoly*)
shows *p* = *q*
⟨*proof*⟩

lemma *vars-empty-Const*: **assumes** *vars* (*p* :: *'a* :: {*ring-char-0*,*idom*} *mpoly*) = {}
shows ∃ *c*. *p* = *Const c*
⟨*proof*⟩

context
assumes *ge1*: $\bigwedge$ *c* :: *'a* :: *linordered-idom*. *c* > *0* ⟹ ∃ *x*. *c* ∗ *x* ≥ *1*
begin

lemma *poly-ext-bounded*:
fixes *p q* :: *'a poly*
assumes $\bigwedge$*x*. *x* ≥ *b* ⟹ *poly p x* = *poly q x* **shows** *p* = *q*
⟨*proof*⟩

lemma *mpoly-ext-bounded*:
assumes $\bigwedge$ α. ($\bigwedge$ *x*. α *x* ≥ *b*) ⟹ *insertion* α *p* = *insertion* α (*q* :: *'a* :: *linordered-idom mpoly*)
shows *p* = *q*
⟨*proof*⟩
end

lemma *mpoly-ext-bounded-int*:
assumes $\bigwedge$ α. ($\bigwedge$ *x*. α *x* ≥ *b*) ⟹ *insertion* α *p* = *insertion* α (*q* :: *int mpoly*)
shows *p* = *q*
⟨*proof*⟩

lemma *mpoly-ext-bounded-field*:
assumes $\bigwedge$ α. ($\bigwedge$ *x*. α *x* ≥ *b*) ⟹ *insertion* α *p* = *insertion* α (*q* :: *'a* :: *linordered-field mpoly*)
shows *p* = *q*
⟨*proof*⟩

lemma *mpoly-of-poly-is-poly-to-mpoly*: *mpoly-of-poly* = *poly-to-mpoly*
⟨*proof*⟩

lemma *insertion-poly-to-mpoly* [*simp*]: *insertion f* (*poly-to-mpoly i p*) = *poly p* (*f*

i)
⟨*proof*⟩

lemma *substitute-poly-to-mpoly*:
assumes *x*: α *x* = *poly-to-mpoly y* (*q* :: *'a* :: {*ring-char-0*,*idom*} *poly*)
shows *substitute* α (*poly-to-mpoly x p*) = *poly-to-mpoly y* (*pcompose p q*)
⟨*proof*⟩

lemma *total-degree-add-Const*: *total-degree* (*p* + *Const* (*c* :: *'a* :: *comm-ring-1*)) = *total-degree p*
⟨*proof*⟩

lemma *mpoly-as-sum-any*: (*p* :: *'a* :: *comm-ring-1 mpoly*) = *Sum-any* (λ *m*. *mmonom m* (*mcoeff p m*))
⟨*proof*⟩

lemma *mpoly-as-sum*: (*p* :: *'a* :: *comm-ring-1 mpoly*) = *sum* (λ *m*. *mmonom m* (*mcoeff p m*)) {*m* . *mcoeff p m* $\neq$ *0*}
⟨*proof*⟩

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

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *degree-prod*: **assumes** *prod p A* $\neq$ (*0* :: *'a* :: *idom mpoly*)
shows *mdegree* (*prod p A*) *x* = *sum* (λ *i*. *mdegree* (*p i*) *x*) *A*
⟨*proof*⟩

lemma *degree-prod-le*: **fixes** *p* :: - $\Rightarrow$ *'a* :: *idom mpoly*
shows *mdegree* (*prod p A*) *x* $\leq$ *sum* (λ *i*. *mdegree* (*p i*) *x*) *A*
⟨*proof*⟩

lemma *degree-power*: **assumes** *p* $\neq$ (*0* :: *'a* :: *idom mpoly*)
shows *mdegree* (*p*^*n*) *x* = *n* * *mdegree p x*
⟨*proof*⟩

lemma *mdegree-Const-mult-le*: *mdegree* (*Const* (*c* :: *'a* :: *idom*) * *p*) *x* $\leq$ *mdegree p x*
⟨*proof*⟩

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*⟩

lemma *degree-substitute-same-var*: *mdegree* (*substitute* ($\lambda i.$ *Var x*) (*p* :: *'a* :: *idom mpoly*)) *x* $\leq$ *total-degree p*
⟨*proof*⟩

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*⟩

context
assumes *poly-pinfty-ge*: $\bigwedge$ *p b*. *0* < *lead-coeff* (*p* :: *'a* :: *linordered-idom poly*) $\Longrightarrow$ *degree p* $\neq$ *0* $\Longrightarrow$ $\exists$ *n*. $\forall$ *x*$\geq$*n*. *b* $\leq$ *poly p x*
begin
lemma *degree-mono-generic*: **assumes** *pos*: *lead-coeff p* $\geq$ (*0* :: *'a*)
and *le*: $\bigwedge$ *x*. *x* $\geq$ *c* $\Longrightarrow$ *poly p x* $\leq$ *poly q x*
shows *degree p* $\leq$ *degree q*
⟨*proof*⟩

lemma *degree-mono'-generic*: **assumes** *le*: $\bigwedge$ *x*. *x* $\geq$ *c* $\Longrightarrow$ (*bnd* :: *'a*) $\leq$ *poly p x* $\wedge$ *poly p x* $\leq$ *poly q x*
shows *degree p* $\leq$ *degree q*
⟨*proof*⟩

end

definition *nneg-poly* :: *'a* :: {*linordered-semidom*, *semiring-no-zero-divisors*} *poly* $\Rightarrow$ *bool* **where**
nneg-poly p = (($\forall$ *x*. *x* $\geq$ *0* $\longrightarrow$ *poly p x* $\geq$ *0*) $\wedge$ *lead-coeff p* $\geq$ *0*)

lemma *nneg-poly-nneg*: **assumes** *nneg-poly p*
and *x* $\geq$ *0*
shows *poly p x* $\geq$ *0*
⟨*proof*⟩

lemma *nneg-poly-lead-coeff*: **assumes** *nneg-poly p*
shows *p* $\neq$ *0* $\Longrightarrow$ *lead-coeff p* > *0*
⟨*proof*⟩

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*⟩

lemma *nneg-poly-mult*: **assumes** *nneg-poly p nneg-poly q*
shows *nneg-poly* (*p* ∗ *q*)
⟨*proof*⟩

lemma *nneg-poly-const*[*simp*]: *nneg-poly* [:*c*:] = (*c* ≥ *0*)
⟨*proof*⟩

lemma *nneg-poly-pCons*[*simp*]: *a* ≥ *0* ∧ *nneg-poly p* ⟹ *nneg-poly* (*pCons a p*)
⟨*proof*⟩

lemma *nneg-poly-0*[*simp*]: *nneg-poly 0*
⟨*proof*⟩

lemma *nneg-poly-pcompose*: **assumes** *nneg-poly p nneg-poly q*
shows *nneg-poly* (*pcompose p q*)
⟨*proof*⟩

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*⟩

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*⟩

lemma *poly-pinfty-gt-lc*:
fixes *p* :: ′*a* :: *linordered-field poly*
assumes *lead-coeff p* > *0*
shows ∃ *n*. ∀ *x* ≥ *n*. *poly p x* ≥ *lead-coeff p*
⟨*proof*⟩

lemma *poly-pinfty-ge*:
fixes *p* :: ′*a* :: *linordered-field poly*
assumes *lead-coeff p* > *0 degree p* ≠ *0*
shows ∃ *n*. ∀ *x* ≥ *n*. *poly p x* ≥ *b*
⟨*proof*⟩

lemma *nneg-polyI*: **fixes** *p* :: ′*a*::*linordered-field poly*
assumes ⋀ *x*. *0* ≤ *x* ⟹ *0* ≤ *poly p x*
shows *nneg-poly p*
⟨*proof*⟩

lemma *poly-bounded*: **fixes** *x* :: ′*a*:: *linordered-idom*

assumes *abs x* $\leq$ *b*
shows *abs* (*poly p x*) $\leq$ ($\sum$ *i* $\leq$ *degree p. abs* (*coeff p i*) $*$ *b* ^ *i*)
⟨*proof*⟩

lemma *poly-degree-le-large-const*:
assumes *pq*: *degree* (*p* :: *'a* :: *linordered-field poly*) $\geq$ *degree q*
and *p0*: $\bigwedge$ *x*. *x* $\geq$ *0* $\Longrightarrow$ *poly p x* $\geq$ *0*
shows $\exists$ *H*. $\forall$ *h* $\geq$ *H*. $\forall$ *x* $\geq$ *0*. *h* $*$ *poly p x* + *h* $\geq$ *poly q x*
⟨*proof*⟩

lemma *degree-monom-0*[*simp*]: *degree-monom 0* = *0*
⟨*proof*⟩

lemma *degree-monom-monomial*[*simp*]: *degree-monom* (*monomial n x*) = *n*
⟨*proof*⟩

lemma *keys-add*: *keys* (*m* + *n* :: *monom*) = *keys m* $\cup$ *keys n*
⟨*proof*⟩

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

lemma *degree-monom-of-set*: *finite xs* $\Longrightarrow$ *degree-monom* (*monom-of-set xs*) = *card xs*
⟨*proof*⟩

lemma *keys-singletonE*: **assumes** *keys m* = {*x*}
shows $\exists$ *c*. *m* = *monomial c x* $\wedge$ *c* = *degree-monom m* $\wedge$ *c* $\neq$ *0*
⟨*proof*⟩

lemma *binary-degree-2-poly*: **fixes** *p* :: *'a* :: {*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 = *Const a* + *Const b* $*$ *Var x* + *Const c* $*$ *Var y* +
Const d $*$ *Var x* $*$ *Var x* + *Const e* $*$ *Var y* $*$ *Var y* + *Const f* $*$ *Var x* $*$ *Var y*
⟨*proof*⟩

lemma *bounded-negative-factor*: **assumes** $\bigwedge$ *x*. *c* $\leq$ (*x* :: *'a* :: *linordered-field*) $\Longrightarrow$ *a* $*$ *x* $\leq$ *b*
shows *a* $\leq$ *0*
⟨*proof*⟩

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*]: $x < length\ ts \Longrightarrow size\ (ts\ !\ x) < Suc\ (size\text{-}list\ size\ ts)$
$\langle proof \rangle$

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

lemma *assignmentD*: **assumes** $assignment\ \alpha$
shows $\alpha\ x \geq 0$
$\langle proof \rangle$

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

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

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

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

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* ≡ *monotone-fun-wrt gt*

definition *valid-monotone-fun* :: (*'f* × *nat*) ⇒ *bool* **where**
valid-monotone-fun fn = (∀ *f n p*. *fn* = (*f*,*n*) ⟶ *p* = *I f*
⟶ *valid-fun n p* ∧ *monotone-fun n p*)

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

definition *orient-rule* :: (*'f*,*var*)*rule* ⇒ *bool* **where**
orient-rule rule = (*case rule of* (*l*,*r*) ⇒ (∀ α. *assignment* α ⟶ *gt* (*I*⟦*l*⟧α) (*I*⟦*r*⟧α)))
end

locale *omega-term-algebra* = *term-algebra F I* (>) :: *int* ⇒ *int* ⇒ *bool* **for** *F* **and** *I* :: *'f* ⇒ - +
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* ⟨*proof*⟩

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** $\succ_p$ *50*) **where**
(*p* $\succ_p$ *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* {..<*n*} *p* ∧ *vars p* = {..<*n*})

definition *valid-weakly-monotone-poly* :: (*'f* × *nat*) ⇒ *bool* **where**
valid-weakly-monotone-poly fn = (∀ *f n p*. *fn* = (*f*,*n*) ⟶ *p* = *I f*

$\longrightarrow$ *valid-poly p* $\wedge$ *weakly-monotone-poly* $\{..<n\}$ *p* $\wedge$ *vars p* $\subseteq \{..<n\})$

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* $\Rightarrow$ *'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*⟩

lemma *orient-rule*: *orient-rule* (*l*,*r*) = (*eval l* $\succ_p$ *eval r*)
⟨*proof*⟩

lemma *vars-eval*: *vars* (*eval t*) $\subseteq$ *vars-term t*
⟨*proof*⟩

lemma *monotone-imp-weakly-monotone*: **assumes** *valid*: *valid-monotone-poly p*
and *gt*: $\bigwedge$ *x y*. ($x \succ y$) = ($x > y$)
shows *valid-weakly-monotone-poly p*
⟨*proof*⟩

lemma *valid-imp-insertion-eval-pos*: **assumes** *valid*: *valid-monotone-poly-inter*
and *funas-term t* $\subseteq$ *F*
and *assignment* α
shows *insertion* α (*eval t*) $\geq$ *0*
⟨*proof*⟩

end

locale *delta-poly-inter* = *poly-inter F I* (λ *x y*. $x \geq y + \delta$) **for** *F* :: (*'f* $\times$ *nat*) *set* **and** *I* **and**
δ :: *'a* :: {*floor-ceiling*,*linordered-field*} +
assumes *valid*: *valid-monotone-poly-inter*
and *δ0*: $\delta > 0$
begin
definition *termination-by-delta-interpretation* :: (*'f*,*var*) *rule set* $\Rightarrow$ *bool* **where**
termination-by-delta-interpretation R = ($\forall$ (*l*,*r*) $\in$ *R*. *orient-rule* (*l*,*r*) $\wedge$ *funas-term l* $\cup$ *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*⟩

definition *termination-by-poly-interpretation* :: (*'f*,*var*) *rule set* ⇒ *bool* **where**
termination-by-poly-interpretation = *termination-by-interpretation*
end

locale *wm-int-poly-inter* = *poly-inter F I* (>) :: *int* ⇒ *int* ⇒ *bool* **for** *F* :: (*'f* × *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* ∧
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* ⇒ *bool* **where**
hilbert10-problem p = (∃ α. *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 *coeff p m* ≠ (*0*::*'a*) ⟶ (*0*::*'a*) < *coeff p m* for monomials *m*. However, this condition is equivalent to just demand (*0*::*'a*) ≤ *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* ⇒ *bool* **where**
positive-poly p = (∀ *m*. *coeff p m* ≥ *0*)

definition *positive-interpr* :: (*var* ⇒ *'a* :: *linordered-idom*) ⇒ *bool* **where**
positive-interpr α = (∀ *x*. α *x* > *0*)

definition *positive-poly-problem* :: *'a* :: *linordered-idom mpoly* ⇒ *'a mpoly* ⇒ *bool* **where**
positive-poly p ⟹ *positive-poly q* ⟹ *positive-poly-problem p q* =
(∃ α. *positive-interpr* α ∧ *insertion* α *p* ≥ *insertion* α *q*)

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

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

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

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

definition *correct-flags* :: *var set* ⇒ (*var* ⇒ *flag*) ⇒ (*var* ⇒ *'a* :: *ordered-comm-ring*) ⇒ *bool* **where**
correct-flags V flag α = (∀ *x* ∈ *V*. *flag x* = *flag-of* (α *x*))

lemma *correct-flag-substitutions*: **fixes** *p* :: *'a* :: *linordered-idom mpoly*
assumes *vars p* ⊆ *V*
and *beta*: β = *assignment-flag V flag* α
and *sigma*: σ = *subst-flag V flag*
and *q*: *q* = *substitute* σ *p*
and *corr*: *correct-flags V flag* α
shows *insertion* β *q* = *insertion* α *p positive-interpr* β
⟨*proof*⟩

definition *hilbert-encode1* :: *int mpoly* ⇒ *int mpoly list* **where**
hilbert-encode1 r = (*let r2* = *r*^*2*;
V = *vars-list r2*;
flag-lists = *product-lists* (*map* (λ *x*. *map* (λ *f*. (*x*,*f*)) [*Positive*,*Negative*,*Zero*]) *V*);
subst = (λ *fl*. *subst-flag* (*set V*) (λ *x*. *case map-of fl x of Some f* ⇒ *f* | *None* ⇒ *Zero*))
in map (λ *fl*. *substitute* (*subst fl*) *r2*) *flag-lists*)

lemma *hilbert-encode1*:
hilbert10-problem r ⟷ (∃ *p* ∈ *set* (*hilbert-encode1 r*). ∃ α. *positive-interpr* α ∧ *insertion* α *p* ≤ *0*)
⟨*proof*⟩

lemma *pos-neg-split*: *mpoly-coeff-filter* (λ *x*. (*x* :: *'a* :: *linordered-idom*) > *0*) *p* + *mpoly-coeff-filter* (λ *x*. *x* < *0*) *p* = *p* (**is** *?l* + *?r* = *p*)
⟨*proof*⟩

definition *hilbert-encode2* :: *int mpoly* ⇒ *int mpoly* × *int mpoly* **where**
hilbert-encode2 p =
(− *mpoly-coeff-filter* (λ *x*. *x* < *0*) *p*, *mpoly-coeff-filter* (λ *x*. *x* > *0*) *p*)

lemma *hilbert-encode2*: **assumes** *hilbert-encode2 p* = (*r*,*s*)
shows *positive-poly r positive-poly s insertion* α *p* ≤ *0* ⟷ *insertion* α *r* ≥ *insertion* α *s*
⟨*proof*⟩

definition *hilbert-encode* :: *int mpoly* ⇒ (*int mpoly* × *int mpoly*)*list* **where**

hilbert-encode = *map hilbert-encode2 o hilbert-encode1*

Lemma 2.2 in paper

lemma *hilbert-encode-positive*: *hilbert10-problem p*
⟷ (∃ (*r*,*s*) ∈ *set* (*hilbert-encode p*). *positive-poly-problem r s*)
⟨*proof*⟩

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

definition *encode-num* :: *var* ⇒ *int* ⇒ (*symbol*,*var*)*term* **where**
encode-num x n = ((λ *t*. *a-t* (*Var x*) *t*)^^(*nat n*)) *z-t*

definition *encode-monom* :: *var* ⇒ *monom* ⇒ *int* ⇒ (*symbol*,*var*)*term* **where**
encode-monom x m c = *rec-list* (*encode-num x c*) (λ (*i*,*e*) -. (λ *t*. *v-t i t*)^^*e*) (*var-list m*)

definition *encode-poly* :: *var* ⇒ *int mpoly* ⇒ (*symbol*,*var*)*term* **where**
encode-poly x r = *rec-list z-t* (λ (*m*,*c*) - *t*. *a-t* (*encode-monom x m c*) *t*) (*monom-list r*)

lemma *vars-encode-num*: *vars-term* (*encode-num x n*) ⊆ {*x*}
⟨*proof*⟩

lemma *vars-encode-monom*: *vars-term* (*encode-monom x m c*) ⊆ {*x*}

⟨*proof*⟩

lemma *vars-encode-poly*: *vars-term* (*encode-poly x r*) ⊆ {*x*}
⟨*proof*⟩

definition *V* **where** *V* = *vars p* ∪ *vars q*

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

lemma *y-vars*: *y1* ≠ *y2 y2* ≠ *y3 y1* ≠ *y3*
⟨*proof*⟩

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*)} ∪ (λ *i*. (*v-sym i*, *1* :: *nat*)) ‘ *V*
definition *F-R* **where** *F-R* = {(*f-sym*,*4*), (*o-sym*, *0*)} ∪ *F*

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

definition *V-list* **where** *V-list* = *sorted-list-of-set V*

definition *contexts* :: (*symbol* × *nat* × *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 (λ *i*. (*v-sym i*, *1*,*0*)) *V-list*

replace t by f(z,...z,t,z,...,z)

definition *z-context* :: *symbol* × *nat* × *nat* ⇒ (*symbol*, *var*)*term* ⇒ (*symbol*, *var*) *term* **where**
z-context c t = (*case c of* (*f*,*n*,*i*) ⇒ *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* ⇒ (*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*) ⊆ *F*
⟨*proof*⟩

lemma *funas-encode-monom*: **assumes** *keys m* ⊆ *V*
shows *funas-term* (*encode-monom x m c*) ⊆ *F*
⟨*proof*⟩

lemma *funas-encode-poly*: **assumes** *vars r* ⊆ *V* **shows** *funas-term* (*encode-poly x r*) ⊆ *F*
⟨*proof*⟩

lemma *funas-encode-poly-p*: *funas-term* (*encode-poly x p*) ⊆ *F*
⟨*proof*⟩

lemma *funas-encode-poly-q*: *funas-term* (*encode-poly x q*) ⊆ *F*
⟨*proof*⟩

lemma *lhs-R-F*: *funas-term lhs-R* ⊆ *F-R*
⟨*proof*⟩

lemma *rhs-R-F*: *funas-term rhs-R* ⊆ *F-R*
⟨*proof*⟩

lemma *finite-V*: *finite V* ⟨*proof*⟩

lemma *V-list*: *set V-list* = *V* ⟨*proof*⟩

lemma *contexts*: **assumes** (*f*,*n*,*i*) ∈ *set contexts*
shows (*f*,*n*) ∈ *F-R* *i* < *n*
⟨*proof*⟩

lemma *z-contexts-append*: *z-contexts* (*cs* @ *ds*) *t* = *z-contexts cs* (*z-contexts ds t*)
⟨*proof*⟩

lemma *z-context*: **assumes** (*f*,*n*) ∈ *F-R* *i* < *n* **and** *funas-term t* ⊆ *F-R*
shows *funas-term* (*z-context* (*f*,*n*,*i*) *t*) ⊆ *F-R*
⟨*proof*⟩

lemma *funas-all-symbol-pos-ctxt*: **assumes** *funas-term t* ⊆ *F-R*
shows *funas-term* (*all-symbol-pos-ctxt t*) ⊆ *F-R*
⟨*proof*⟩

lemma *lhs-R'-F*: *funas-term lhs-R'* ⊆ *F-R*
⟨*proof*⟩

lemma *rhs-R′-F*: *funas-term rhs-R′* ⊆ *F-R*
⟨*proof*⟩
end

lemma *insertion-positive-poly*: **assumes** ⋀ *x*. α *x* ≥ (*0* :: ′*a* :: *linordered-idom*)
and *positive-poly p*
shows *insertion* α *p* ≥ *0*
⟨*proof*⟩

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

definition α **where** α = (*SOME* α. *positive-interpr* α ∧ *insertion* α *q* ≤ *insertion* α *p*)

lemma α: *positive-interpr* α *insertion* α *q* ≤ *insertion* α *p*
⟨*proof*⟩

lemma α*1*: α *x* > *0* ⟨*proof*⟩

context
fixes *I* :: *symbol* ⇒ *int mpoly*
assumes *inter*: *I a-sym* = *PVar 0* + *PVar 1*
I z-sym = *0*
I o-sym = *1*
I (*v-sym i*) = *Const* (α *i*) ∗ *PVar 0*
begin

lemma *inter-encode-num*: **assumes** *c* ≥ *0*
shows *poly-inter.eval I* (*encode-num x c*) = *Const c* ∗ *PVar x*
⟨*proof*⟩

lemma *inter-v-pow-e*: *poly-inter.eval I* ((*v-t x* ^^ *e*) *t*) = *Const* ((α *x*)^*e*) ∗ *poly-inter.eval I t*
⟨*proof*⟩

lemma *inter-encode-monom*: **assumes** *c*: *c* ≥ *0*
shows *poly-inter.eval I* (*encode-monom y m c*) = *Const* (*insertion* α (*monom m c*)) ∗ *PVar y*
⟨*proof*⟩

lemma *inter-foldr-v-t*:
poly-inter.eval I (*foldr v-t xs t*) = *Const* (*prod-list* (*map* α *xs*)) ∗ *poly-inter.eval I t*
⟨*proof*⟩

lemma *inter-encode-poly-generic*: **assumes** *positive-poly r*
  shows *poly-inter.eval I (encode-poly x r) = Const (insertion α r) ∗ PVar x*
⟨*proof*⟩

lemma *valid-monotone-inter-F*: **assumes** *positive-interpr α*
  and *inF*: *fn ∈ F*
shows *poly-inter.valid-monotone-poly I (>) fn*
⟨*proof*⟩

end

fun *I-R* :: *symbol ⇒ int mpoly* **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-R (v-sym i) = Const (α i) ∗ PVar 0*

interpretation *inter-R*: *poly-inter F-R I-R (>)* ⟨*proof*⟩

lemma *inter-R-encode-poly*: **assumes** *positive-poly r*
  shows *inter-R.eval (encode-poly x r) = Const (insertion α r) ∗ PVar x*
  ⟨*proof*⟩

lemma *valid-monotone-inter-R*: *inter-R.valid-monotone-poly-inter* ⟨*proof*⟩

sublocale *inter-R*: *linear-int-poly-inter F-R I-R*
⟨*proof*⟩

lemma *orient-R-main*: **assumes** *assignment β*
  shows *insertion β (inter-R.eval lhs-R) > insertion β (inter-R.eval rhs-R)*
⟨*proof*⟩

The easy direction of Theorem 3.4

lemma *orient-R*: *inter-R.termination-by-poly-interpretation R*
  ⟨*proof*⟩

lemma *solution-imp-linear-termination-R*: *termination-by-linear-int-poly-interpretation F-R R*
  ⟨*proof*⟩
end

context *poly-input*
begin

lemma *inter-z-context*:
  assumes *i*: *i < n* **and** *I*: *I f = Const c0 + (sum-list (map (λ j. Const (c j) ∗ PVar j) [0..<n]))*

and *Ize*: *I z-sym* = *Const d0*
shows ∃ *d*. ∀ *t*. *poly-inter.eval I* (*z-context* (*f*,*n*,*i*) *t*) = *Const d* + *Const* (*c i*) ∗ *poly-inter.eval I t*
⟨*proof*⟩

lemma *inter-z-contexts*:
assumes *cs*: ⋀ *f n i*. (*f*,*n*,*i*) ∈ *set cs* ⟹ *i* < *n* ∧ *I f* = *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*⟩

lemma *inter-all-symbol-pos-ctxt-generic*:
assumes *f*: *I f-sym* = *Const fc* + *Const f0* ∗ *PVar 0* + *Const f1* ∗ *PVar 1* + *Const f2* ∗ *PVar 2* + *Const f3* ∗ *PVar 3*
and *a*: *I a-sym* = *Const ac* + *Const a0* ∗ *PVar 0* + *Const a1* ∗ *PVar 1*
and *v*: ⋀ *i*. *i* ∈ *V* ⟹ *I* (*v-sym i*) = *Const* (*vc i*) + *Const* (*v0 i*) ∗ *PVar 0*
and *I z-sym* = *Const zc*
shows ∃ *d*. ∀ *t*. *poly-inter.eval I* (*all-symbol-pos-ctxt t*) = *Const d* + *Const* (*prod-list* ([*a0*, *a1*, *f0*, *f1*, *f2*, *f3*] @ *map v0 V-list*))
∗ *poly-inter.eval I t*
⟨*proof*⟩
end

context *solvable-poly-problem*
begin

lemma *inter-all-symbol-pos-ctxt*:
∃ *d e*. *e* ≥ *1* ∧ (∀ *t*. *inter-R.eval* (*all-symbol-pos-ctxt t*) = *Const d* + *Const e* ∗ *inter-R.eval t*)
⟨*proof*⟩

The easy direction of Theorem 3.4 for R'

lemma *orient-R′*: *inter-R.termination-by-poly-interpretation R′*
⟨*proof*⟩

lemma *solution-imp-linear-termination-R′*: *termination-by-linear-int-poly-interpretation F-R R′*
⟨*proof*⟩
end

Now for the other direction of Theorem 3.4

lemma *monotone-linear-poly-to-coeffs*: **fixes** *p* :: *int mpoly*
assumes *linear*: *total-degree p* ≤ *1*
and *poly*: *valid-poly p*
and *mono*: *poly-inter.monotone-poly* (>) {..<*n*} *p*
and *vars*: *vars p* = {..<*n*}
shows ∃ *c a*. *p* = *Const c* + (∑ *i*←[*0*..<*n*]. *Const* (*a i*) ∗ *PVar i*)

$\wedge$ *c* $\geq$ *0* $\wedge$ ($\forall$ *i* < *n*. *a i* > *0*)
⟨*proof*⟩

locale *poly-input-to-solution-common* = *poly-input p q* +
poly-inter F′ I (>) :: *int* ⇒ *int* ⇒ *bool* **for** *p q I* **and** *F′* :: (*poly-input.symbol* × *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* ⟹

∃ *c a*. *I g* = *Const c* + (∑ *i*←[*0*..<*n*]. *Const* (*a i*) ∗ *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*⟩

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*⟩

lemma *extract-z-poly*: ∃ *ze0*. *I z-sym* = *Const ze0* $\wedge$ *ze0* $\geq$ *0*
⟨*proof*⟩

lemma *solution*: *positive-poly-problem p q*
⟨*proof*⟩
end

locale *solution-poly-input-R* = *poly-input p q* + *poly-inter F-R I* (>) :: *int* ⇒ - **for** *p q I* +
assumes *orient*: *orient-rule* (*lhs-R*,*rhs-R*)
and *linear-mono-interpretation*: (*g*,*n*) ∈ *F-R* ⟹
∃ *c a*. *I g* = *Const c* + (∑ *i*←[*0*..<*n*]. *Const* (*a i*) ∗ *PVar i*)
$\wedge$ *c* $\geq$ *0* $\wedge$ ($\forall$ *i* < *n*. *a i* > *0*)
begin

lemma *solution*: *positive-poly-problem p q*
⟨*proof*⟩
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* = (*SOME I*. *linear-int-poly-inter F-R I* ∧ *int-poly-inter*.*termination-by-poly-interpretati*
F-R I R)

lemma *I*: *linear-int-poly-inter F-R I int-poly-inter*.*termination-by-poly-interpretation*
F-R I R
⟨*proof*⟩

sublocale *linear-int-poly-inter F-R I* ⟨*proof*⟩

lemma *orient*: *orient-rule* (*lhs-R*,*rhs-R*)
⟨*proof*⟩

lemma *extract-linear-poly*: **assumes** *g*: (*g*,*n*) ∈ *F-R*
shows ∃ *c a*. *I g* = *Const c* + (∑ *i*←[*0*..<*n*]. *Const* (*a i*) ∗ *PVar i*)
∧ *c* ≥ *0* ∧ (∀ *i* < *n*. *a i* > *0*)
⟨*proof*⟩

lemma *solution*: *positive-poly-problem p q*
⟨*proof*⟩
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* ∧ *wm-int-poly-inter*.*oriented-by-interpretati*
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′
⟨*proof*⟩

sublocale *linear-wm-int-poly-inter F-R I* ⟨*proof*⟩

lemma *orient-R′*: *orient-rule* (*lhs-R′*,*rhs-R′*)
⟨*proof*⟩

lemma *extract-linear-poly*: **assumes** *g*: (*g*,*n*) ∈ *F-R*
shows ∃ *c a*. *I g* = *Const c* + (∑ *i*←[*0*..<*n*]. *Const* (*a i*) ∗ *PVar i*)
∧ *c* ≥ *0* ∧ (∀ *i* < *n*. *a i* ≥ *0*)
⟨*proof*⟩

lemma *extract-a-poly*: ∃ *a0 a1 a2*. *I a-sym* = *Const a0* + *Const a1* ∗ *PVar 0* + *Const a2* ∗ *PVar 1*
∧ *a0* ≥ *0* ∧ *a1* ≥ *0* ∧ *a2* ≥ *0*
⟨*proof*⟩

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* + *Const f4* ∗ *PVar 3*
∧ *f0* ≥ *0* ∧ *f1* ≥ *0* ∧ *f2* ≥ *0* ∧ *f3* ≥ *0* ∧ *f4* ≥ *0*
⟨*proof*⟩

lemma *solution*: *positive-poly-problem p q*
⟨*proof*⟩
end

context *poly-input*
begin

Theorem 3.4 in paper

theorem *linear-polynomial-termination-with-natural-numbers-undecidable*:
positive-poly-problem p q ⟷ *termination-by-linear-int-poly-interpretation F-R R*
⟨*proof*⟩

Theorem 3.9

theorem *orientation-by-linear-wm-int-poly-interpretation-undecidable*:
positive-poly-problem p q ⟷ *orientation-by-linear-wm-int-poly-interpretation F-R R′*
⟨*proof*⟩

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* (>) ⟨*proof*⟩

lemma *inter-encode-num*: **assumes** $c \geq 0$
shows *inter-R.eval* (*encode-num x c*) = *Const c* ∗ *PVar x*
⟨*proof*⟩

lemma *inter-v-pow-e*: *inter-R.eval* ((*v-t x* ^^ *e*) *t*) = *inter-R.eval t*
⟨*proof*⟩

lemma *inter-encode-monom*: **assumes** *c*: $c \geq 0$
shows *inter-R.eval* (*encode-monom y m c*) = *Const* (*insertion* (λ -.*1*) (*monom m c*)) ∗ *PVar y*
⟨*proof*⟩

lemma *inter-encode-poly*: **assumes** *positive-poly r*
shows *inter-R.eval* (*encode-poly x r*) = *Const* (*insertion* (λ -.*1*) *r*) ∗ *PVar x*
⟨*proof*⟩

lemma *valid-monotone-inter*: *inter-R.valid-monotone-poly-inter*
⟨*proof*⟩

Lemma 3.6 in the paper

lemma *orient-R-main*: **assumes** *assignment* β
shows *insertion* β (*inter-R.eval lhs-R*) > *insertion* β (*inter-R.eval rhs-R*)
⟨*proof*⟩

lemma *polynomial-termination-R*: *termination-by-int-poly-interpretation F-R R*
⟨*proof*⟩

lemma *polynomial-termination-R′*: *termination-by-int-poly-interpretation F-R R′*
⟨*proof*⟩

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* (λ *m*. *count m x*) *ms*)
⟨*proof*⟩

lemma *sum-list-scf-list-prod*: *sum-list* (*map f* (*scf-list scf as*)) = *sum-list* (*map* (λ *i*. *scf i* ∗ *f* (*as* ! *i*)) [*0*..<*length as*])
⟨*proof*⟩

lemma *count-vars-term-different-var*: **assumes** *x*: $x \notin$ *vars-term t*
shows *count* (*vars-term-ms* (*scf-term scf t*)) *x* = *0*
⟨*proof*⟩

context *kbo*
begin
definition *kbo-orientation* :: ('*f*,'*v*)*rule set* ⇒ *bool* **where**
kbo-orientation R = (∀ (*l*,*r*) ∈ *R*. *fst* (*kbo l r*))
end

definition *kbo-with-sc-termination* :: ('*f*,'*v*)*rule set* ⇒ *bool* **where**
kbo-with-sc-termination R = (∃ *w w0 sc least pr-strict pr-weak*. *admissible-kbo w w0 pr-strict pr-weak least sc*
∧ *kbo.kbo-orientation w w0 sc least pr-strict pr-weak R*)

context *poly-input*
begin

context
fixes *sc*
assumes *sc*: *sc* (*a-sym*, *Suc* (*Suc 0*)) *0* = (*1* :: *nat*)
sc (*a-sym*, *Suc* (*Suc 0*)) (*Suc 0*) = *1*
begin
lemma *count-vars-term-encode-num-nat*:
count (*vars-term-ms* (*scf-term sc* (*encode-num x* (*int n*)))) *x* = *n*
⟨*proof*⟩

lemma *count-vars-term-encode-num*:
$c \geq 0 \Longrightarrow$ *int* (*count* (*vars-term-ms* (*scf-term sc* (*encode-num x c*))) *x*) = *c*
⟨*proof*⟩

lemma *count-vars-term-v-pow-e*:
count (*vars-term-ms* (*scf-term sc* ((*v-t x* ^^ *e*) *t*))) *y*
= (*sc* (*v-sym x*,*1*) *0*)^*e* ∗ *count* (*vars-term-ms* (*scf-term sc t*)) *y*
⟨*proof*⟩

lemma *count-vars-term-encode-monom*: **assumes** *c*: $c \geq 0$
shows *int* (*count* (*vars-term-ms* (*scf-term sc* (*encode-monom x m c*))) *x*)
= *insertion* (λ *v*. *int* (*sc* (*v-sym v*,*1*) *0*)) (*monom m c*)
⟨*proof*⟩

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*⟩
end

Theorem 4.6

theorem *kbo-sc-termination-R-imp-solution*:
assumes *kbo-with-sc-termination R*
shows *positive-poly-problem p q*
⟨*proof*⟩
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* ⟨*proof*⟩

sublocale *admissible-kbo w w0* (λ - -. *False*) (=) *least sc*
⟨*proof*⟩

lemma *insertion-pos*: *positive-poly r* ⟹ *insertion* α *r* ≥ *0*
⟨*proof*⟩

lemma *count-vars-term-encode-poly*: **assumes** *positive-poly r*
shows *count* (*vars-term-ms* (*SCF* (*encode-poly x r*))) *y* = (*nat* (*insertion* α *r*) *when x* = *y*)
⟨*proof*⟩

Theorem 4.7 in context

theorem *kbo-with-sc-termination*: *kbo-with-sc-termination R*
⟨*proof*⟩

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*
  ⟨*proof*⟩

Combining 4.6 and 4.7

corollary *solvable-iff-kbo-with-sc-termination*:
  positive-poly-problem p q ⟷ *kbo-with-sc-termination R*
  ⟨*proof*⟩
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 *y4* :: *var* **where** *y4* = *3*
definition *y5* :: *var* **where** *y5* = *4*
definition *y6* :: *var* **where** *y6* = *5*
definition *y7* :: *var* **where** *y7* = *6*

abbreviation *q-t* **where** *q-t t* ≡ *Fun q-sym* [*t*]
abbreviation *h-t* **where** *h-t t* ≡ *Fun h-sym* [*t*]
abbreviation *g-t* **where** *g-t t1 t2* ≡ *Fun g-sym* [*t1*, *t2*]

Definition 5.1

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

definition *rhs-S* = *Fun f-sym* [
  a-t (*Var y1*) *z-t*,
  a-t z-t (*Var y2*),
  a-t (*encode-poly y3 q*) (*Var y3*),
  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*]

definition *S* **where** *S* = {(*lhs-S*, *rhs-S*)}

definition *F-S* **where** *F-S* = {(*f-sym*,*7*), (*h-sym*,*1*), (*g-sym*,*2*), (*o-sym*,*0*), (*q-sym*,*1*)} ∪ *F*

lemma *lhs-S-F*: *funas-term lhs-S* ⊆ *F-S*
⟨*proof*⟩

lemma *funas-fold-vs*[*simp*]: *funas-term* (*foldr v-t V-list t*) = (λ *i*. (*v-sym i*,*1*)) ' *V* ∪ *funas-term t*
⟨*proof*⟩

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

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

  ⟨*proof*⟩

lemma *rhs-S-F*: *funas-term rhs-S* ⊆ *F-S*
⟨*proof*⟩
end

lemma *poly-inter-eval-cong*: **assumes** ⋀ *f a*. (*f*,*a*) ∈ *funas-term t* ⟹ *I f* = *I′ f*
  shows *poly-inter.eval I t* = *poly-inter.eval I′ t*
  ⟨*proof*⟩

The easy direction of Theorem 5.4

context *solvable-poly-problem*
begin

definition *c-S* **where** *c-S* = *max 7* (*2* ∗ *prod-list* (*map* α *V-list*))

lemma *c-S*: *c-S* > *0* ⟨*proof*⟩

fun *I-S* :: *symbol* ⇒ *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* (*α i*) ∗ *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* (>) ⟨*proof*⟩

lemma *inter-S-encode-poly*: **assumes** *positive-poly r*
shows *inter-S.eval* (*encode-poly x r*) = *Const* (*insertion α r*) ∗ *PVar x*
⟨*proof*⟩

lemma *valid-monotone-inter-S*: *inter-S.valid-monotone-poly-inter*
⟨*proof*⟩

interpretation *inter-S*: *int-poly-inter F-S I-S*
⟨*proof*⟩

lemma *orient-trs*: *inter-S.termination-by-poly-interpretation S*
⟨*proof*⟩

lemma *solution-imp-poly-termination*: *termination-by-int-poly-interpretation F-S S*
⟨*proof*⟩

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*
⟨*proof*⟩

context
fixes *gt* :: *'a* :: *linordered-idom* ⇒ *'a* ⇒ *bool*
assumes *trans-gt*: *transp gt*
and *gt-imp-ge*: $\bigwedge$ *x y*. *gt x y* ⟹ *x* ≥ *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* ∈ *xs* ⟹ *gt*$^{==}$ (*b x*) (*a x*)
$\bigwedge$ *x*. *x* ∉ *xs* ⟹ *a x* = *b x*
shows *gt*$^{==}$ (*insertion b p*) (*insertion a p*)
⟨*proof*⟩

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* ∈ *vars p* ⟹ *gt*$^{==}$ (*b x*) (*a x*)
shows *gt*$^{==}$ (*insertion b p*) (*insertion a p*)
⟨*proof*⟩

lemma *partial-insertion-mono-wrt*: **assumes** *mono*: *monotone-poly-wrt gt* (*vars p*) *p*
and *a*: *assignment a*
and *b*: $\bigwedge$ *y*. *y* ≠ *x* ⟹ *gt*$^{==}$ (*b y*) (*a y*)
and *d*: $\bigwedge$ *y*. *y* ≥ *d* ⟹ *gt*$^{==}$ *y 0*
shows ∃ *c*. ∀ *y*. *y* ≥ *d* ⟶ *c* ≤ *poly* (*partial-insertion a x p*) *y*
∧ *poly* (*partial-insertion a x p*) *y* ≤ *poly* (*partial-insertion b x p*) *y*
⟨*proof*⟩

context
assumes *poly-pinfty-ge*: $\bigwedge$ *p b*. *0* < *lead-coeff* (*p* :: *'a poly*) ⟹ *degree p* ≠ *0* ⟹ ∃ *n*. ∀ *x*≥*n*. *b* ≤ *poly p x*
begin

context
fixes *p d*
assumes *mono*: *monotone-poly-wrt gt* (*vars p*) *p*
and *d*: $\bigwedge$ *y*. *y* ≥ *d* ⟹ *gt*$^{==}$ *y 0*
begin

lemma *degree-partial-insertion-mono-generic*: **assumes**
a: *assignment a*
and *b*: $\bigwedge$ *y*. *y* ≠ *x* ⟹ *gt*$^{==}$ (*b y*) (*a y*)
shows *degree* (*partial-insertion a x p*) ≤ *degree* (*partial-insertion b x p*)
⟨*proof*⟩

lemma *degree-partial-insertion-stays-constant-generic*:
∃ *a*. *assignment a* ∧
(∀ *b*. (∀ *y*. *gt*$^{==}$ (*b y*) (*a y*)) ⟶ *degree* (*partial-insertion a x p*) = *degree* (*partial-insertion b x p*))
⟨*proof*⟩
end

lemma *monotone-poly-partial-insertion-generic*:
assumes *delta-order*: $\bigwedge$ *x y*. *gt y x* ⟷ *y* ≥ *x* + δ
and *delta*: δ > *0*
and *eps-delta*: ε ∗ δ ≥ *1*
and *ceil-nat*: $\bigwedge$ *x* :: *'a*. *of-nat* (*ceil-nat x*) ≥ *x*
assumes *x*: *x* ∈ *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*

$valid\text{-}poly\ p \Longrightarrow poly\ (partial\text{-}insertion\ a\ x\ p)\ (\delta * of\text{-}nat\ y) \geq \delta * of\text{-}nat\ y$
⟨*proof*⟩
end
end

context *poly-inter*
begin

lemma *monotone-poly-eval-generic*:
assumes *valid*: *valid-monotone-poly-inter*
and *trans-gt*: *transp* (≻)
and *gt-imp-ge*: $\bigwedge x\ y.\ x \succ y \Longrightarrow y \leq x$
and *gt-exists*: $\bigwedge x.\ x \geq 0 \Longrightarrow \exists\ y.\ y \succ x$
and *gt-irrefl*: $\bigwedge x.\ \neg\ (x \succ x)$
and *tF*: $funas\text{-}term\ t \subseteq F$
shows $monotone\text{-}poly\ (vars\text{-}term\ t)\ (eval\ t)\ vars\ (eval\ t) = vars\text{-}term\ t$
⟨*proof*⟩
end

context *int-poly-inter*
begin

lemma *degree-mono*: **assumes** *pos*: $lead\text{-}coeff\ p \geq (0 :: int)$
and *le*: $\bigwedge x.\ x \geq c \Longrightarrow poly\ p\ x \leq poly\ q\ x$
shows $degree\ p \leq degree\ q$
⟨*proof*⟩

lemma *degree-mono'*: **assumes** $\bigwedge x.\ x \geq c \Longrightarrow (bnd :: int) \leq poly\ p\ x \wedge poly\ p\ x \leq poly\ q\ x$
shows $degree\ p \leq degree\ q$
⟨*proof*⟩

lemma *weakly-monotone-insertion*: **assumes** $weakly\text{-}monotone\text{-}poly\ (vars\ p)\ p$
and $assignment\ (a :: \text{-} \Rightarrow int)$
and $\bigwedge x.\ x \in vars\ p \Longrightarrow a\ x \leq b\ x$
shows $insertion\ a\ p \leq insertion\ b\ p$
⟨*proof*⟩

Lemma 5.2

lemma *degree-partial-insertion-stays-constant*: **assumes** *mono*: $monotone\text{-}poly\ (vars\ p)\ p$
shows $\exists\ a.\ assignment\ (a :: \text{-} \Rightarrow int) \wedge (\forall\ b.\ (\forall\ y.\ a\ y \leq b\ y) \longrightarrow degree\ (partial\text{-}insertion\ a\ x\ p) = degree\ (partial\text{-}insertion\ b\ x\ p))$
⟨*proof*⟩

lemma *degree-partial-insertion-stays-constant-wm*: **assumes** *wm*: $weakly\text{-}monotone\text{-}poly\ (vars\ p)\ p$

shows ∃ *a*. *assignment* (*a* :: - ⇒ *int*) ∧
(∀ *b*. (∀ *y*. *a y* ≤ *b y*) ⟶ *degree* (*partial-insertion a x p*) = *degree* (*partial-insertion b x p*))
⟨*proof*⟩

Lemma 5.3

lemma *subst-same-var-weakly-monotone-imp-same-degree*:
assumes *wm*: *weakly-monotone-poly* (*vars p*) (*p* :: *int mpoly*)
and *dq*: *degree q* = *d*
and *d0*: *d* ≠ *0*
and *qp*: *poly-to-mpoly x q* = *substitute* (λ*i*. *PVar x*) *p*
shows *total-degree p* = *d*
⟨*proof*⟩

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*
valid-poly p ⟹ *insertion a p* ≥ *a x*
⟨*proof*⟩

end

context *int-poly-inter*
begin

lemma *insertion-eval-pos*: **assumes** *funas-term t* ⊆ *F*
and *assignment* α
shows *insertion* α (*eval t*) ≥ *0*
⟨*proof*⟩

lemma *monotone-poly-eval*: **assumes** *funas-term t* ⊆ *F*
shows *monotone-poly* (*vars-term t*) (*eval t*) *vars* (*eval t*) = *vars-term t*
⟨*proof*⟩
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* ∧ *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*

⟨proof⟩

sublocale *int-poly-inter F-S I* ⟨proof⟩

lemma *orient*: *orient-rule* (*lhs-S*,*rhs-S*)
⟨proof⟩

lemma *solution*: *positive-poly-problem p q*
⟨proof⟩
end

context *poly-input*
begin

Theorem 5.4 in paper

theorem *polynomial-termination-with-natural-numbers-undecidable*:
positive-poly-problem p q ⟷ *termination-by-int-poly-interpretation F-S S*
⟨proof⟩

end

Now head for Lemma 5.6

locale *poly-input-omega-solution* = *poly-input*
begin

fun *I* :: *symbol* ⇒ *int list* ⇒ *int* **where**
I o-sym xs = *insertion* (λ -. *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* ≥ *0*
shows *I*⟦*encode-num x c*⟧α = *c* ∗ α *x*
⟨proof⟩

lemma *I-v-pow-e*: *I* ⟦(*v-t x* ^^ *e*) *t*⟧α = *I* ⟦*t*⟧α
⟨proof⟩

lemma *I-encode-monom*: **assumes** *c*: *c* ≥ *0*
shows *I*⟦*encode-monom x m c*⟧α = *c* ∗ α *x*
⟨proof⟩

lemma *I-encode-poly*: **assumes** *positive-poly r*
shows *I* ⟦*encode-poly x r*⟧α = *insertion* (λ -. *1*) *r* ∗ α *x*
⟨proof⟩

end

lemma *length2-cases*: *length xs* = *2* ⟹ ∃ *x y*. *xs* = [*x*,*y*]
⟨*proof*⟩

lemma *length7-cases*: *length xs* = *7* ⟹ ∃ *x1 x2 x3 x4 x5 x6 x7*. *xs* = [*x1*,*x2*,*x3*,*x4*,*x5*,*x6*,*x7*]
⟨*proof*⟩

lemma *length1-cases*: *length xs* = *Suc 0* ⟹ ∃ *x*. *xs* = [*x*]
⟨*proof*⟩

lemma *less2-cases*: *i* < *2* ⟹ *i* = *0* ∨ (*i* :: *nat*) = *1*
⟨*proof*⟩

lemma *less7-cases*: *i* < *7* ⟹ *i* = *0* ∨ (*i* :: *nat*) = *1* ∨ *i* = *2* ∨ *i* = *3* ∨ *i* = *4* ∨ *i* = *5* ∨ *i* = *6*
⟨*proof*⟩

context *poly-input-omega-solution*
begin

sublocale *inter-S*: *term-algebra F-S I* (>) ⟨*proof*⟩
sublocale *inter-S*: *omega-term-algebra F-S I*
⟨*proof*⟩

Lemma 5.6

lemma *S-is-omega-terminating*: *omega-termination F-S S*
⟨*proof*⟩
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* :: - ⇒ (*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* :: (*symbol*,*var*)*term* **where** *t-t* = (*g-list* ((*z-sym*,*0*) # *symbol-list*))

definition *rhs-Q* = *Fun f-sym* [
h-t (*h-t* (*q-t* (*Var y1*))),
g-t (*Var y2*) (*Var y2*),
Fun f-sym (*replicate 9* (*Var y3*)),
q-t (*g-t* (*Var y4*) *t-t*),
a-t (*Var y5*) (*Var y5*),
q-t (*Var y6*),
a-t z-t (*Var y7*),
a-t (*Var y8*) *z-t*,
a-t (*encode-poly y9 q*) (*Var y9*)]

definition *Q* **where** *Q* = {(*lhs-Q*, *rhs-Q*)}

definition *F-Q* **where** *F-Q* = {(*f-sym*,*9*), (*h-sym*,*1*), (*g-sym*,*2*), (*q-sym*,*1*)} ∪ *F*

lemma *lhs-Q-F*: *funas-term lhs-Q* ⊆ *F-Q*
⟨*proof*⟩

lemma *g-list-F*: *set zs* ⊆ *F-Q* ⟹ *funas-term* (*g-list zs*) ⊆ *F-Q*
⟨*proof*⟩

lemma *symbol-list*: *set symbol-list* ⊆ *F-Q* ⟨*proof*⟩

lemma *t-F*: *funas-term t-t* ⊆ *F-Q*
⟨*proof*⟩

lemma *vars-g-list*[*simp*]: *vars-term* (*g-list zs*) = {}
⟨*proof*⟩

lemma *vars-t*: *vars-term t-t* = {}

⟨*proof*⟩

lemma *rhs-Q-F*: *funas-term rhs-Q* ⊆ *F-Q*
⟨*proof*⟩

context
fixes *I* :: *symbol* ⇒ *'a* :: *linordered-field mpoly* **and** *δ* :: *'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* ≥ *0*
and *z*: *z0* ≥ *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*:])
⟨*proof*⟩

lemma *degree-combination*: **assumes** *nneg-poly r*
shows *degree* ([:*a1*, *a3*:] ∗ *r* + [:*a0*, *a2*:]) = *Suc* (*degree r*)
⟨*proof*⟩

lemma *degree-eval-encode-num*: **assumes** *c*: *c* ≥ *0*
shows ∃ *p*. *mpoly-of-poly x p* = *poly-inter.eval I* (*encode-num x c*) ∧ *nneg-poly p* ∧ *int* (*degree p*) = *c*
⟨*proof*⟩

lemma *degree-eval-encode-monom*: **assumes** *c*: *c* > *0*
and *α*: *α* = (*λ i*. *int* (*degree* (*v i*)))
shows ∃ *p*. *mpoly-of-poly y p* = *poly-inter.eval I* (*encode-monom y m c*) ∧ *nneg-poly p* ∧
int (*degree p*) = *insertion α* (*mmonom m c*) ∧ *degree p* > *0*
⟨*proof*⟩

Lemma 6.2

lemma *degree-eval-encode-poly-generic*: **assumes** *positive-poly r*
and *α*: *α* = (*λ i*. *int* (*degree* (*v i*)))
shows ∃ *p*. *poly-to-mpoly x p* = *poly-inter.eval I* (*encode-poly x r*) ∧ *nneg-poly p* ∧
int (*degree p*) = *insertion α r*
⟨*proof*⟩
end
end

context *delta-poly-inter*
begin

lemma *transp-gt-delta*: *transp* $(\lambda\ x\ y.\ x \geq y + \delta)$ ⟨*proof*⟩

lemma *gt-delta-imp-ge*: $y + \delta \leq x \Longrightarrow y \leq x$ ⟨*proof*⟩

lemma *weakly-monotone-insertion*: **assumes** *mono*: *monotone-poly* (*vars p*) *p*
  and *a*: *assignment* $(a :: - \Rightarrow 'a)$
  and *gt*: $\bigwedge x.\ x \in vars\ p \Longrightarrow a\ x + \delta \leq b\ x$
shows *insertion a p* $\leq$ *insertion b p*
  ⟨*proof*⟩

Lemma 6.5

lemma *degree-partial-insertion-stays-constant*: **assumes** *mono*: *monotone-poly* (*vars p*) *p*
  shows $\exists\ a.$ *assignment a* $\wedge$
   $(\forall\ b.\ (\forall\ y.\ a\ y + \delta \leq b\ y) \longrightarrow$ *degree* (*partial-insertion a x p*) = *degree* (*partial-insertion b x p*))
  ⟨*proof*⟩

lemma *degree-mono*: **assumes** *pos*: *lead-coeff p* $\geq (0 :: 'a)$
  and *le*: $\bigwedge x.\ x \geq c \Longrightarrow$ *poly p x* $\leq$ *poly q x*
shows *degree p* $\leq$ *degree q*
  ⟨*proof*⟩

lemma *degree-mono'*: **assumes** $\bigwedge x.\ x \geq c \Longrightarrow (bnd :: 'a) \leq$ *poly p x* $\wedge$ *poly p x* $\leq$ *poly q x*
  shows *degree p* $\leq$ *degree q*
  ⟨*proof*⟩

Lemma 6.6

lemma *subst-same-var-monotone-imp-same-degree*:
  assumes *mono*: *monotone-poly* (*vars p*) $(p :: 'a\ mpoly)$
    and *dq*: *degree q* $= d$
    and *d0*: $d \neq 0$
    and *qp*: *poly-to-mpoly x q* = *substitute* $(\lambda i.\ PVar\ x)$ *p*
  shows *total-degree p* $= d$
⟨*proof*⟩

lemma *monotone-poly-partial-insertion*:
  assumes *x*: $x \in xs$
  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 $\Longrightarrow y \geq 0 \Longrightarrow$ *poly* (*partial-insertion a x p*) $y \geq y - \delta$
   valid-poly p $\Longrightarrow$ *insertion a p* $\geq a\ x - \delta$
⟨*proof*⟩
end

context *solvable-poly-problem*
begin

context
  assumes *SORT-CONSTRAINT*(*'a* :: *floor-ceiling*)
begin

context
  fixes *h* :: *'a*
begin

fun *IQ* :: *symbol* ⇒ *'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* (*λx y. x* ≥ *y* + (*1* :: *'a*)) ⟨*proof*⟩

Lemma 6.2 specialized for this interpretation

lemma *degree-eval-encode-poly*: **assumes** *positive-poly r*
  shows ∃ *p. poly-to-mpoly y9 p* = *interQ.eval* (*encode-poly y9 r*) ∧ *nneg-poly p* ∧ *int* (*degree p*) = *insertion α r*
⟨*proof*⟩

definition *pp* **where** *pp* = (*SOME pp. poly-to-mpoly y9 pp* = *interQ.eval* (*encode-poly y9 p*) ∧ *nneg-poly pp* ∧ *int* (*degree pp*) = *insertion α p*)

lemma *pp*: *interQ.eval* (*encode-poly y9 p*) = *poly-to-mpoly y9 pp*
  nneg-poly pp int (*degree pp*) = *insertion α p*
  ⟨*proof*⟩

definition *qq* **where** *qq* = (*SOME qq. poly-to-mpoly y9 qq* = *interQ.eval* (*encode-poly y9 q*) ∧ *nneg-poly qq* ∧ *int* (*degree qq*) = *insertion α q*)

lemma *qq*: *interQ.eval* (*encode-poly y9 q*) = *poly-to-mpoly y9 qq*
  nneg-poly qq int (*degree qq*) = *insertion α q*
  ⟨*proof*⟩

definition *ppp* = *pp* ∗ [:*1*,*1*:] + [:*0*,*1*:]
definition *qqq* = *qq* ∗ [:*1*,*1*:] + [:*0*,*1*:]

lemma *degree-ppp*: *int* (*degree ppp*) = *1* + *insertion α p*
  ⟨*proof*⟩

lemma *degree-qqq*: *int* (*degree qqq*) = *1* + *insertion* α *q*
⟨*proof*⟩

lemma *ppp-qqq*: *degree ppp* $\geq$ *degree qqq*
⟨*proof*⟩

lemma *nneg-ppp*: *nneg-poly ppp*
⟨*proof*⟩

definition *H* **where** *H* = (*SOME H*. $\forall$ *h* $\geq$ *H*. $\forall$ *x*$\geq$*0*. *poly qqq x* $\leq$ *h* $*$ *poly ppp x* + *h*)

lemma *H*: *h* $\geq$ *H* $\Longrightarrow$ *x* $\geq$ *0* $\Longrightarrow$ *poly qqq x* $\leq$ *h* $*$ *poly ppp x* + *h*
⟨*proof*⟩
end

definition *h* **where** *h* = *max 9* (*H 1*)

lemma *h*: *h* $\geq$ *1* ⟨*proof*⟩

abbreviation *I-Q* **where** *I-Q* $\equiv$ *IQ h*

interpretation *inter-Q*: *poly-inter F-Q I-Q* (λ*x y*. *x* $\geq$ *y* + (*1* :: *'a*)) ⟨*proof*⟩

Well-definedness of Interpretation in Theorem 6.4

lemma *valid-monotone-inter-Q*:
inter-Q.valid-monotone-poly-inter
⟨*proof*⟩

lemma *I-Q-delta-poly-inter*: *delta-poly-inter F-Q I-Q* (*1* :: *'a*)
⟨*proof*⟩

interpretation *inter-Q*: *delta-poly-inter F-Q I-Q 1* :: *'a* ⟨*proof*⟩

Orientation part of Theorem 6.4

lemma *orient-Q*: *inter-Q.orient-rule* (*lhs-Q*, *rhs-Q*)
⟨*proof*⟩
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*⟩

end

context *delta-poly-inter*
begin

lemma *insertion-eval-pos*: **assumes** *funas-term t* $\subseteq$ *F*
  and *assignment* α
shows *insertion* α (*eval t*) $\geq$ *0*
  ⟨*proof*⟩

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*⟩

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* = *Const c* + ($\sum$ *i*←[*0*..<*n*]. *Const* (*a i*) $*$ *PVar i*)
    $\wedge$ *c* $\geq$ *0* $\wedge$ ($\forall$ *i* < *n*. *a i* $\geq$ *1*)
⟨*proof*⟩

end

Lemma 6.7

lemma *criterion-for-degree-2*: **assumes** *qq-def*: *qq* = *q* $\circ_p$ [:*c*, *a*:] − *smult a q*
  and *dq*: *degree q* $\geq$ *2*
  and *ineq*: $\bigwedge$ *x* :: *'a* :: *linordered-field*. *x* $\geq$ *0* $\Longrightarrow$ *poly qq x* $\leq$ *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*⟩

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* = (*SOME I*. $\exists$ δ. *delta-poly-inter F-Q I* (δ :: *'a*) $\wedge$
    delta-poly-inter.termination-by-delta-interpretation F-Q I δ *Q*)

definition δ **where** $\delta = (SOME\ \delta.\ delta\text{-}poly\text{-}inter\ F\text{-}Q\ I\ (\delta :: 'a) \wedge$
$delta\text{-}poly\text{-}inter.termination\text{-}by\text{-}delta\text{-}interpretation\ F\text{-}Q\ I\ \delta\ Q)$

lemma *I*: *delta-poly-inter F-Q I δ delta-poly-inter.termination-by-delta-interpretation F-Q I δ Q*
⟨*proof*⟩

sublocale *delta-poly-inter F-Q I δ* ⟨*proof*⟩

lemma *orient*: *orient-rule* (*lhs-Q*,*rhs-Q*)
⟨*proof*⟩

lemma *eval-t-t-gt-0*: **assumes** *Ig*: $I\ g\text{-}sym = Const\ g0 + Const\ g1 * PVar\ 0 + Const\ g2 * PVar\ 1$
and *Iz*: $I\ z\text{-}sym = Const\ z0$
and *z0*: $z0 \geq 0$
and *g0*: $g0 \geq 0$
and *g12*: $g1 > 0\ g2 > 0$
shows $insertion\ \beta\ (eval\ t\text{-}t) > 0$
⟨*proof*⟩

Theorem 6.8

theorem *solution*: *positive-poly-problem p q*
⟨*proof*⟩
end

context *poly-input*
begin

corollary *polynomial-termination-with-delta-orders-undecidable*:
$positive\text{-}poly\text{-}problem\ p\ q \longleftrightarrow$
$termination\text{-}by\text{-}delta\text{-}poly\text{-}interpretation\ (TYPE('a :: floor\text{-}ceiling))\ F\text{-}Q\ Q$
⟨*proof*⟩

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.