

Signature-Based Gröbner Basis Algorithms

Alexander Maletzky*

September 13, 2023

Abstract

This article formalizes signature-based algorithms for computing Gröbner bases. Such algorithms are, in general, superior to other algorithms in terms of efficiency, and have not been formalized in any proof assistant so far. The present development is both generic, in the sense that most known variants of signature-based algorithms are covered by it, and effectively executable on concrete input thanks to Isabelle’s code generator. Sample computations of benchmark problems show that the verified implementation of signature-based algorithms indeed outperforms the existing implementation of Buchberger’s algorithm in Isabelle/HOL.

Besides total correctness of the algorithms, the article also proves that under certain conditions they a-priori detect and avoid all useless zero-reductions, and always return ‘minimal’ (in some sense) Gröbner bases if an input parameter is chosen in the right way.

The formalization follows the recent survey article by Eder and Faugère.

Contents

1	Introduction	3
2	Preliminaries	3
2.1	Lists	3
2.1.1	Sequences of Lists	3
2.1.2	<i>filter</i>	4
2.1.3	<i>drop</i>	4
2.1.4	<i>count-list</i>	5
2.1.5	<i>sorted-wrt</i>	5
2.1.6	<i>insert-wrt</i> and <i>merge-wrt</i>	5
2.2	Recursive Functions	5
2.3	Binary Relations	6

*Supported by the Austrian Science Fund (FWF): P 29498-N31

3	More Properties of Power-Products and Multivariate Polynomials	8
3.1	Power-Products	8
3.2	Miscellaneous	9
3.3	<i>ordered-term.lt</i> and <i>ordered-term.higher</i>	9
3.4	<i>gd-term.dgrad-p-set</i>	10
3.5	Regular Sequences	10
4	Signature-Based Algorithms for Computing Gröbner Bases	11
4.1	More Preliminaries	11
4.2	Module Polynomials	11
4.2.1	Signature Reduction	17
4.2.2	Signature Gröbner Bases	27
4.2.3	Rewrite Bases	32
4.2.4	S-Pairs	36
4.2.5	Termination	39
4.2.6	Concrete Rewrite Orders	40
4.2.7	Preparations for Sig-Poly-Pairs	42
4.2.8	Total Reduction	43
4.2.9	Koszul Syzygies	46
4.2.10	Algorithms	49
4.2.11	Minimality of the Computed Basis	62
4.2.12	No Zero-Reductions	62
4.3	Sig-Poly-Pairs	64
5	Sample Computations with Signature-Based Algorithms	68
5.1	Setup	69
5.1.1	Projections of Term Orders to Orders on Power-Products	69
5.1.2	Locale Interpretation	69
5.1.3	More Lemmas and Definitions	71
5.2	Computations	72

1 Introduction

Signature-based algorithms [3, 1] play a central role in modern computer algebra systems, as they allow to compute Gröbner bases of ideals of multivariate polynomials much more efficiently than other algorithms. Although they also belong to the class of critical-pair/completion algorithms, as almost all algorithms for computing Gröbner bases, they nevertheless possess some quite unique features that render a formal development in proof assistants challenging. In fact, this is the first formalization of signature-based algorithms in any proof assistant.

The formalization builds upon the existing formalization of Gröbner bases theory [4] and closely follows Sections 4–7 of the excellent survey article [1]. Some proofs were taken from [5, 2].

Summarizing, the main features of the formalization are as follows:

- It is *generic*, in the sense that it considers the computation of so-called *rewrite bases* and neither fixes the term order nor the rewrite-order.
- It is *efficient*, in the sense that all executable algorithms (e.g. *gb-sig*) operate on sig-poly-pairs rather than module elements, and that polynomials are represented efficiently using ordered associative lists.
- It proves that if the input is a regular sequence and the term order is a POT order, there are no useless zero-reductions (Theorem *gb-sig-no-zero-red*).
- It proves that the signature Gröbner bases computed w. r. t. the ‘ratio’ rewrite order are minimal (Theorem *gb-sig-z-is-min-sig-GB*).
- It features sample computations of benchmark problems to illustrate the practical usability of the verified algorithms.

2 Preliminaries

```
theory Prelims
  imports Polynomials.Utils Groebner-Bases.General
begin
```

2.1 Lists

2.1.1 Sequences of Lists

```
lemma list-seq-length-mono:
  fixes seq :: nat ⇒ 'a list
  assumes  $\bigwedge i. (\exists x. seq (Suc i) = x \# seq i)$  and  $i < j$ 
  shows  $length (seq i) < length (seq j)$ 
⟨proof⟩
```

corollary *list-seq-length-mono-weak*:

fixes $seq :: nat \Rightarrow 'a\ list$
assumes $\bigwedge i. (\exists x. seq\ (Suc\ i) = x \# seq\ i)$ **and** $i \leq j$
shows $length\ (seq\ i) \leq length\ (seq\ j)$
<proof>

lemma *list-seq-indexE-length*:

fixes $seq :: nat \Rightarrow 'a\ list$
assumes $\bigwedge i. (\exists x. seq\ (Suc\ i) = x \# seq\ i)$
obtains j **where** $i < length\ (seq\ j)$
<proof>

lemma *list-seq-nth*:

fixes $seq :: nat \Rightarrow 'a\ list$
assumes $\bigwedge i. (\exists x. seq\ (Suc\ i) = x \# seq\ i)$ **and** $i < length\ (seq\ j)$ **and** $j \leq k$
shows $rev\ (seq\ k) ! i = rev\ (seq\ j) ! i$
<proof>

corollary *list-seq-nth'*:

fixes $seq :: nat \Rightarrow 'a\ list$
assumes $\bigwedge i. (\exists x. seq\ (Suc\ i) = x \# seq\ i)$ **and** $i < length\ (seq\ j)$ **and** $i < length\ (seq\ k)$
shows $rev\ (seq\ k) ! i = rev\ (seq\ j) ! i$
<proof>

2.1.2 filter

lemma *filter-merge-wrt-1*:

assumes $\bigwedge y. y \in set\ ys \implies P\ y \implies False$
shows $filter\ P\ (merge-wrt\ rel\ xs\ ys) = filter\ P\ xs$
<proof>

lemma *filter-merge-wrt-2*:

assumes $\bigwedge x. x \in set\ xs \implies P\ x \implies False$
shows $filter\ P\ (merge-wrt\ rel\ xs\ ys) = filter\ P\ ys$
<proof>

lemma *length-filter-le-1*:

assumes $length\ (filter\ P\ xs) \leq 1$ **and** $i < length\ xs$ **and** $j < length\ xs$
and $P\ (xs\ !\ i)$ **and** $P\ (xs\ !\ j)$
shows $i = j$
<proof>

lemma *length-filter-eq [simp]*: $length\ (filter\ ((=)\ x)\ xs) = count-list\ xs\ x$
<proof>

2.1.3 drop

lemma *nth-in-set-dropI*:

assumes $j \leq i$ **and** $i < \text{length } xs$
shows $xs ! i \in \text{set } (\text{drop } j \text{ } xs)$
 $\langle \text{proof} \rangle$

2.1.4 *count-list*

lemma *count-list-upt* [simp]: $\text{count-list } [a..<b] \ x = (\text{if } a \leq x \wedge x < b \text{ then } 1 \text{ else } 0)$
 $\langle \text{proof} \rangle$

2.1.5 *sorted-wrt*

lemma *sorted-wrt-upt-iff*: $\text{sorted-wrt rel } [a..<b] \longleftrightarrow (\forall i \ j. a \leq i \longrightarrow i < j \longrightarrow j < b \longrightarrow \text{rel } i \ j)$
 $\langle \text{proof} \rangle$

2.1.6 *insort-wrt and merge-wrt*

lemma *map-insort-wrt*:

assumes $\bigwedge x. x \in \text{set } xs \implies r2 \ (f \ y) \ (f \ x) \longleftrightarrow r1 \ y \ x$
shows $\text{map } f \ (\text{insort-wrt } r1 \ y \ xs) = \text{insort-wrt } r2 \ (f \ y) \ (\text{map } f \ xs)$
 $\langle \text{proof} \rangle$

lemma *map-merge-wrt*:

assumes $f \ ' \ \text{set } xs \cap f \ ' \ \text{set } ys = \{\}$
and $\bigwedge x \ y. x \in \text{set } xs \implies y \in \text{set } ys \implies r2 \ (f \ x) \ (f \ y) \longleftrightarrow r1 \ x \ y$
shows $\text{map } f \ (\text{merge-wrt } r1 \ xs \ ys) = \text{merge-wrt } r2 \ (\text{map } f \ xs) \ (\text{map } f \ ys)$
 $\langle \text{proof} \rangle$

2.2 Recursive Functions

locale *recursive* =

fixes $h' :: 'b \Rightarrow 'b$
fixes $b :: 'b$
assumes *b-fixpoint*: $h' \ b = b$

begin

context

fixes $Q :: 'a \Rightarrow \text{bool}$
fixes $g :: 'a \Rightarrow 'b$
fixes $h :: 'a \Rightarrow 'a$

begin

function (*domintros*) *recfun-aux* :: $'a \Rightarrow 'b$ **where**
 $\text{recfun-aux } x = (\text{if } Q \ x \ \text{then } g \ x \ \text{else } h' \ (\text{recfun-aux } (h \ x)))$
 $\langle \text{proof} \rangle$

lemmas [*induct del*] = *recfun-aux.pinduct*

definition *dom* :: $'a \Rightarrow \text{bool}$

where $dom\ x \longleftrightarrow (\exists k. Q ((h \sim k) x))$

lemma *domI*:

assumes $\neg Q\ x \implies dom\ (h\ x)$

shows $dom\ x$

<proof>

lemma *domD*:

assumes $dom\ x$ **and** $\neg Q\ x$

shows $dom\ (h\ x)$

<proof>

lemma *recfun-aux-domI*:

assumes $dom\ x$

shows $recfun\ aux\ dom\ x$

<proof>

lemma *recfun-aux-domD*:

assumes $recfun\ aux\ dom\ x$

shows $dom\ x$

<proof>

corollary *recfun-aux-dom-alt*: $recfun\ aux\ dom = dom$

<proof>

definition *fun* :: $'a \Rightarrow 'b$

where $fun\ x = (if\ recfun\ aux\ dom\ x\ then\ recfun\ aux\ x\ else\ b)$

lemma *simps*: $fun\ x = (if\ Q\ x\ then\ g\ x\ else\ h'\ (fun\ (h\ x)))$

<proof>

lemma *eq-fixpointI*: $\neg\ dom\ x \implies fun\ x = b$

<proof>

lemma *pinduct*: $dom\ x \implies (\bigwedge x. dom\ x \implies (\neg Q\ x \implies P\ (h\ x)) \implies P\ x) \implies P\ x$

<proof>

end

end

interpretation *tailrec*: *recursive $\lambda x. x$ undefined*

<proof>

2.3 Binary Relations

lemma *almost-full-on-Int*:

assumes *almost-full-on P1 A1 and almost-full-on P2 A2*

shows *almost-full-on* $(\lambda x y. P1\ x\ y \wedge P2\ x\ y)\ (A1 \cap A2)$ (**is** *almost-full-on* $?P\ ?A$)
<proof>

corollary *almost-full-on-same*:

assumes *almost-full-on* $P1\ A$ **and** *almost-full-on* $P2\ A$
shows *almost-full-on* $(\lambda x y. P1\ x\ y \wedge P2\ x\ y)\ A$
<proof>

context *ord*
begin

definition *is-le-rel* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
where *is-le-rel* $rel = (rel = (=) \vee rel = (\leq) \vee rel = (<))$

lemma *is-le-relI* [*simp*]: *is-le-rel* $(=)$ *is-le-rel* (\leq) *is-le-rel* $(<)$
<proof>

lemma *is-le-relE*:
assumes *is-le-rel* rel
obtains $rel = (=) \mid rel = (\leq) \mid rel = (<)$
<proof>

end

context *preorder*
begin

lemma *is-le-rel-le*:
assumes *is-le-rel* rel
shows $rel\ x\ y \Longrightarrow x \leq y$
<proof>

lemma *is-le-rel-trans*:
assumes *is-le-rel* rel
shows $rel\ x\ y \Longrightarrow rel\ y\ z \Longrightarrow rel\ x\ z$
<proof>

lemma *is-le-rel-trans-le-left*:
assumes *is-le-rel* rel
shows $x \leq y \Longrightarrow rel\ y\ z \Longrightarrow x \leq z$
<proof>

lemma *is-le-rel-trans-le-right*:
assumes *is-le-rel* rel
shows $rel\ x\ y \Longrightarrow y \leq z \Longrightarrow x \leq z$
<proof>

lemma *is-le-rel-trans-less-left*:

assumes *is-le-rel rel*
shows $x < y \implies \text{rel } y \ z \implies x < z$
 ⟨*proof*⟩

lemma *is-le-rel-trans-less-right*:
assumes *is-le-rel rel*
shows $\text{rel } x \ y \implies y < z \implies x < z$
 ⟨*proof*⟩

end

context *order*
begin

lemma *is-le-rel-distinct*:
assumes *is-le-rel rel*
shows $\text{rel } x \ y \implies x \neq y \implies x < y$
 ⟨*proof*⟩

lemma *is-le-rel-antisym*:
assumes *is-le-rel rel*
shows $\text{rel } x \ y \implies \text{rel } y \ x \implies x = y$
 ⟨*proof*⟩

end

end

3 More Properties of Power-Products and Multivariate Polynomials

theory *More-MPoly*
imports *Prelims Polynomials.MPoly-Type-Class-Ordered*
begin

3.1 Power-Products

lemma (**in** *comm-powerprod*) *minus-plus'*: $s \text{ adds } t \implies u + (t - s) = (u + t) - s$
 ⟨*proof*⟩

context *ulcs-powerprod*
begin

lemma *lcs-alt-2*:
assumes $a + x = b + y$
shows $\text{lcs } x \ y = (b + y) - \text{gcs } a \ b$
 ⟨*proof*⟩

corollary *lcs-alt-1*:
assumes $a + x = b + y$
shows $\text{lcs } x \ y = (a + x) - \text{gcs } a \ b$
 $\langle \text{proof} \rangle$

corollary *lcs-minus-1*:
assumes $a + x = b + y$
shows $\text{lcs } x \ y - x = a - \text{gcs } a \ b$
 $\langle \text{proof} \rangle$

corollary *lcs-minus-2*:
assumes $a + x = b + y$
shows $\text{lcs } x \ y - y = b - \text{gcs } a \ b$
 $\langle \text{proof} \rangle$

lemma *gcs-minus*:
assumes $u \text{ adds } s$ **and** $u \text{ adds } t$
shows $\text{gcs } (s - u) \ (t - u) = \text{gcs } s \ t - u$
 $\langle \text{proof} \rangle$

corollary *gcs-minus-gcs*: $\text{gcs } (s - (\text{gcs } s \ t)) \ (t - (\text{gcs } s \ t)) = 0$
 $\langle \text{proof} \rangle$

end

3.2 Miscellaneous

lemma *poly-mapping-rangeE*:
assumes $c \in \text{Poly-Mapping.range } p$
obtains $k \text{ where } k \in \text{keys } p \text{ and } c = \text{lookup } p \ k$
 $\langle \text{proof} \rangle$

lemma *poly-mapping-range-nonzero*: $0 \notin \text{Poly-Mapping.range } p$
 $\langle \text{proof} \rangle$

lemma (in *term-powerprod*) *Keys-range-vectorize-poly*: $\text{Keys } (\text{Poly-Mapping.range } (\text{vectorize-poly } p)) = \text{pp-of-term } \text{'keys } p$
 $\langle \text{proof} \rangle$

3.3 *ordered-term.lt* and *ordered-term.higher*

context *ordered-term*
begin

lemma *lt-lookup-vectorize*: $\text{punit.lt } (\text{lookup } (\text{vectorize-poly } p) \ (\text{component-of-term } (\text{lt } p))) = \text{lp } p$
 $\langle \text{proof} \rangle$

lemma *lower-higher-zeroI*: $u \preceq_t v \implies \text{lower } (\text{higher } p \ v) \ u = 0$

<proof>

lemma *lookup-minus-higher*: $\text{lookup } (p - \text{higher } p \ v) \ u = (\text{lookup } p \ u \ \text{when } u \preceq_t \ v)$

<proof>

lemma *keys-minus-higher*: $\text{keys } (p - \text{higher } p \ v) = \{u \in \text{keys } p. \ u \preceq_t \ v\}$

<proof>

lemma *lt-minus-higher*: $v \in \text{keys } p \implies \text{lt } (p - \text{higher } p \ v) = v$

<proof>

lemma *lc-minus-higher*: $v \in \text{keys } p \implies \text{lc } (p - \text{higher } p \ v) = \text{lookup } p \ v$

<proof>

lemma *tail-minus-higher*: $v \in \text{keys } p \implies \text{tail } (p - \text{higher } p \ v) = \text{lower } p \ v$

<proof>

end

3.4 *gd-term.dgrad-p-set*

lemma (in *gd-term*) *dgrad-p-set-closed-mult-scalar*:

assumes *dickson-grading* d **and** $p \in \text{punit.dgrad-p-set } d \ m$ **and** $r \in \text{dgrad-p-set } d \ m$

shows $p \odot r \in \text{dgrad-p-set } d \ m$

<proof>

3.5 Regular Sequences

definition *is-regular-sequence* :: $(\text{'a}::\text{comm-powerprod} \Rightarrow_0 \text{'b}::\text{comm-ring-1}) \ \text{list} \Rightarrow \text{bool}$

where *is-regular-sequence* $fs \longleftrightarrow (\forall j < \text{length } fs. \forall q. \ q * fs ! j \in \text{ideal } (\text{set } (\text{take } j \ fs))) \longrightarrow$

$$q \in \text{ideal } (\text{set } (\text{take } j \ fs)))$$

lemma *is-regular-sequenceD*:

is-regular-sequence $fs \implies j < \text{length } fs \implies q * fs ! j \in \text{ideal } (\text{set } (\text{take } j \ fs)) \implies$

$$q \in \text{ideal } (\text{set } (\text{take } j \ fs))$$

<proof>

lemma *is-regular-sequence-Nil*: *is-regular-sequence* $[]$

<proof>

lemma *is-regular-sequence-snocI*:

assumes $\bigwedge q. \ q * f \in \text{ideal } (\text{set } fs) \implies q \in \text{ideal } (\text{set } fs)$ **and** *is-regular-sequence* fs

shows *is-regular-sequence* $(fs @ [f])$

<proof>

```

lemma is-regular-sequence-snocD:
  assumes is-regular-sequence (fs @ [f])
  shows  $\bigwedge q. q * f \in \text{ideal } (\text{set } fs) \implies q \in \text{ideal } (\text{set } fs)$ 
  and is-regular-sequence fs
  <proof>

```

```

lemma is-regular-sequence-removeAll-zero:
  assumes is-regular-sequence fs
  shows is-regular-sequence (removeAll 0 fs)
  <proof>

```

```

lemma is-regular-sequence-remdups:
  assumes is-regular-sequence fs
  shows is-regular-sequence (rev (remdups (rev fs)))
  <proof>

```

end

4 Signature-Based Algorithms for Computing Gröbner Bases

```

theory Signature-Groebner
  imports More-MPoly Groebner-Bases.Syzygy Polynomials.Quasi-PM-Power-Products
begin

```

First, we develop the whole theory for elements of the module $K[X]^r$, i. e. objects of type $'t \Rightarrow_0 'b$. Later, we transfer all algorithms defined on such objects to algorithms efficiently operating on sig-poly-pairs, i. e. objects of type $'t \times ('a \Rightarrow_0 'b)$.

4.1 More Preliminaries

```

lemma (in gd-term) lt-spoly-less-lcs:
  assumes  $p \neq 0$  and  $q \neq 0$  and spoly p q  $\neq 0$ 
  shows lt (spoly p q)  $\prec_t$  term-of-pair (lcs (lp p) (lp q), component-of-term (lt p))
  <proof>

```

4.2 Module Polynomials

```

locale qpm-inf-term =
  gd-term pair-of-term term-of-pair ord ord-strict ord-term ord-term-strict
  for pair-of-term:: $'t \Rightarrow ('a::\text{quasi-pm-powerprod} \times \text{nat})$ 
  and term-of-pair:: $('a \times \text{nat}) \Rightarrow 't$ 
  and ord:: $'a \Rightarrow 'a \Rightarrow \text{bool}$  (infixl  $\leq$  50)
  and ord-strict (infixl  $<$  50)
  and ord-term:: $'t \Rightarrow 't \Rightarrow \text{bool}$  (infixl  $\preceq_t$  50)
  and ord-term-strict:: $'t \Rightarrow 't \Rightarrow \text{bool}$  (infixl  $\prec_t$  50)
begin

```

lemma *in-idealE-rep-dgrad-p-set*:

assumes *hom-grading* d **and** $B \subseteq \text{punit.dgrad-p-set } d \text{ } m$ **and** $p \in \text{punit.dgrad-p-set } d \text{ } m$ **and** $p \in \text{ideal } B$

obtains r **where** $\text{keys } r \subseteq B$ **and** $\text{Poly-Mapping.range } r \subseteq \text{punit.dgrad-p-set } d \text{ } m$ **and** $p = \text{ideal.rep } r$
(*proof*)

context **fixes** $fs :: ('a \Rightarrow_0 'b::\text{field}) \text{ list}$
begin

definition *sig-inv-set'* $:: \text{nat} \Rightarrow ('t \Rightarrow_0 'b) \text{ set}$

where $\text{sig-inv-set}' j = \{r. \text{keys } (\text{vectorize-poly } r) \subseteq \{0..<j\}\}$

abbreviation *sig-inv-set* $\equiv \text{sig-inv-set}' (\text{length } fs)$

definition *rep-list* $:: ('t \Rightarrow_0 'b) \Rightarrow ('a \Rightarrow_0 'b)$

where $\text{rep-list } r = \text{ideal.rep } (\text{pm-of-idx-pm } fs \text{ } (\text{vectorize-poly } r))$

lemma *sig-inv-setI*: $\text{keys } (\text{vectorize-poly } r) \subseteq \{0..<j\} \Longrightarrow r \in \text{sig-inv-set}' j$
(*proof*)

lemma *sig-inv-setD*: $r \in \text{sig-inv-set}' j \Longrightarrow \text{keys } (\text{vectorize-poly } r) \subseteq \{0..<j\}$
(*proof*)

lemma *sig-inv-setI'*:

assumes $\bigwedge v. v \in \text{keys } r \Longrightarrow \text{component-of-term } v < j$

shows $r \in \text{sig-inv-set}' j$

(*proof*)

lemma *sig-inv-setD'*:

assumes $r \in \text{sig-inv-set}' j$ **and** $v \in \text{keys } r$

shows $\text{component-of-term } v < j$

(*proof*)

corollary *sig-inv-setD-lt*:

assumes $r \in \text{sig-inv-set}' j$ **and** $r \neq 0$

shows $\text{component-of-term } (\text{lt } r) < j$

(*proof*)

lemma *sig-inv-set-mono*:

assumes $i \leq j$

shows $\text{sig-inv-set}' i \subseteq \text{sig-inv-set}' j$

(*proof*)

lemma *sig-inv-set-zero*: $0 \in \text{sig-inv-set}' j$

(*proof*)

lemma *sig-inv-set-closed-uminus*: $r \in \text{sig-inv-set}' j \Longrightarrow - r \in \text{sig-inv-set}' j$

<proof>

lemma *sig-inv-set-closed-plus*:

assumes $r \in \text{sig-inv-set}' j$ **and** $s \in \text{sig-inv-set}' j$

shows $r + s \in \text{sig-inv-set}' j$

<proof>

lemma *sig-inv-set-closed-minus*:

assumes $r \in \text{sig-inv-set}' j$ **and** $s \in \text{sig-inv-set}' j$

shows $r - s \in \text{sig-inv-set}' j$

<proof>

lemma *sig-inv-set-closed-monom-mult*:

assumes $r \in \text{sig-inv-set}' j$

shows $\text{monom-mult } c \ t \ r \in \text{sig-inv-set}' j$

<proof>

lemma *sig-inv-set-closed-mult-scalar*:

assumes $r \in \text{sig-inv-set}' j$

shows $p \odot r \in \text{sig-inv-set}' j$

<proof>

lemma *rep-list-zero*: $\text{rep-list } 0 = 0$

<proof>

lemma *rep-list-uminus*: $\text{rep-list } (- r) = - \text{rep-list } r$

<proof>

lemma *rep-list-plus*: $\text{rep-list } (r + s) = \text{rep-list } r + \text{rep-list } s$

<proof>

lemma *rep-list-minus*: $\text{rep-list } (r - s) = \text{rep-list } r - \text{rep-list } s$

<proof>

lemma *vectorize-mult-scalar*:

$\text{vectorize-poly } (p \odot q) = \text{MPoly-Type-Class.punit.monom-mult } p \ 0 \ (\text{vectorize-poly } q)$

<proof>

lemma *rep-list-mult-scalar*: $\text{rep-list } (c \odot r) = c * \text{rep-list } r$

<proof>

lemma *rep-list-monom-mult*: $\text{rep-list } (\text{monom-mult } c \ t \ r) = \text{punit.monom-mult } c \ t \ (\text{rep-list } r)$

<proof>

lemma *rep-list-monomial*:

assumes *distinct fs*

shows $\text{rep-list } (\text{monomial } c \ u) =$

(*punit.monom-mult* c (*pp-of-term* u) (fs ! (*component-of-term* u))
 when *component-of-term* $u < \text{length } fs$)
 ⟨*proof*⟩

lemma *rep-list-in-ideal-sig-inv-set*:
assumes $r \in \text{sig-inv-set}' j$
shows $\text{rep-list } r \in \text{ideal } (\text{set } (\text{take } j \text{ } fs))$
 ⟨*proof*⟩

corollary *rep-list-subset-ideal-sig-inv-set*:
 $B \subseteq \text{sig-inv-set}' j \implies \text{rep-list } ' B \subseteq \text{ideal } (\text{set } (\text{take } j \text{ } fs))$
 ⟨*proof*⟩

lemma *rep-list-in-ideal*: $\text{rep-list } r \in \text{ideal } (\text{set } fs)$
 ⟨*proof*⟩

corollary *rep-list-subset-ideal*: $\text{rep-list } ' B \subseteq \text{ideal } (\text{set } fs)$
 ⟨*proof*⟩

lemma *in-idealE-rep-list*:
assumes $p \in \text{ideal } (\text{set } fs)$
obtains r **where** $p = \text{rep-list } r$ **and** $r \in \text{sig-inv-set}$
 ⟨*proof*⟩

lemma *keys-rep-list-subset*:
assumes $t \in \text{keys } (\text{rep-list } r)$
obtains $v \ s$ **where** $v \in \text{keys } r$ **and** $s \in \text{Keys } (\text{set } fs)$ **and** $t = \text{pp-of-term } v + s$
 ⟨*proof*⟩

lemma *dgrad-p-set-le-rep-list*:
assumes *dickson-grading* d **and** *dgrad-set-le* d (*pp-of-term* ' *keys* r) (*Keys* (*set* fs))
shows *punit.dgrad-p-set-le* d {*rep-list* r } (*set* fs)
 ⟨*proof*⟩

corollary *dgrad-p-set-le-rep-list-image*:
assumes *dickson-grading* d **and** *dgrad-set-le* d (*pp-of-term* ' *Keys* F) (*Keys* (*set* fs))
shows *punit.dgrad-p-set-le* d (*rep-list* ' F) (*set* fs)
 ⟨*proof*⟩
term *Max*

definition *dgrad-max* :: ($'a \Rightarrow \text{nat}$) $\Rightarrow \text{nat}$
where *dgrad-max* $d = (\text{Max } (d \text{ ' } (\text{insert } 0 \text{ } (\text{Keys } (\text{set } fs))))))$

abbreviation *dgrad-max-set* $d \equiv \text{dgrad-p-set } d$ (*dgrad-max* d)

abbreviation *punit-dgrad-max-set* $d \equiv \text{punit.dgrad-p-set } d$ (*dgrad-max* d)

lemma *dgrad-max-0*: $d \ 0 \leq \text{dgrad-max } d$

<proof>

lemma *dgrad-max-1*: $set\ fs \subseteq punit\ dgrad\ max\ set\ d$
<proof>

lemma *dgrad-max-2*:
 assumes *dickson-grading* d **and** $r \in dgrad\ max\ set\ d$
 shows *rep-list* $r \in punit\ dgrad\ max\ set\ d$
<proof>

corollary *dgrad-max-3*:
 assumes *dickson-grading* d **and** $F \subseteq dgrad\ max\ set\ d$
 shows *rep-list* ' $F \subseteq punit\ dgrad\ max\ set\ d$
<proof>

lemma *punit-dgrad-max-set-subset-dgrad-p-set*:
 assumes *dickson-grading* d **and** $set\ fs \subseteq punit.\ dgrad\ p\ set\ d\ m$ **and** $\neg\ set\ fs \subseteq \{0\}$
 shows $punit.\ dgrad\ max\ set\ d \subseteq punit.\ dgrad\ p\ set\ d\ m$
<proof>

definition *dgrad-sig-set'* :: $nat \Rightarrow ('a \Rightarrow nat) \Rightarrow ('t \Rightarrow_0 'b)\ set$
 where $dgrad\ sig\ set'\ j\ d = dgrad\ max\ set\ d \cap sig\ inv\ set'\ j$

abbreviation *dgrad-sig-set* $\equiv dgrad\ sig\ set'\ (length\ fs)$

lemma *dgrad-sig-set-set-mono*: $i \leq j \implies dgrad\ sig\ set'\ i\ d \subseteq dgrad\ sig\ set'\ j\ d$
<proof>

lemma *dgrad-sig-set-closed-uminus*: $r \in dgrad\ sig\ set'\ j\ d \implies -\ r \in dgrad\ sig\ set'\ j\ d$
<proof>

lemma *dgrad-sig-set-closed-plus*:
 $r \in dgrad\ sig\ set'\ j\ d \implies s \in dgrad\ sig\ set'\ j\ d \implies r + s \in dgrad\ sig\ set'\ j\ d$
<proof>

lemma *dgrad-sig-set-closed-minus*:
 $r \in dgrad\ sig\ set'\ j\ d \implies s \in dgrad\ sig\ set'\ j\ d \implies r - s \in dgrad\ sig\ set'\ j\ d$
<proof>

lemma *dgrad-sig-set-closed-monom-mult*:
 assumes *dickson-grading* d **and** $d\ t \leq dgrad\ max\ d$
 shows $p \in dgrad\ sig\ set'\ j\ d \implies monom\ mult\ c\ t\ p \in dgrad\ sig\ set'\ j\ d$
<proof>

lemma *dgrad-sig-set-closed-monom-mult-zero*:
 $p \in dgrad\ sig\ set'\ j\ d \implies monom\ mult\ c\ 0\ p \in dgrad\ sig\ set'\ j\ d$
<proof>

lemma *dgrad-sig-set-closed-mult-scalar*:

dickson-grading $d \implies p \in \text{punit-dgrad-max-set } d \implies r \in \text{dgrad-sig-set}' j d \implies p \odot r \in \text{dgrad-sig-set}' j d$
<proof>

lemma *dgrad-sig-set-closed-monomial*:

assumes $d (\text{pp-of-term } u) \leq \text{dgrad-max } d$ **and** *component-of-term* $u < j$
shows *monomial* $c u \in \text{dgrad-sig-set}' j d$
<proof>

lemma *rep-list-in-ideal-dgrad-sig-set*:

$r \in \text{dgrad-sig-set}' j d \implies \text{rep-list } r \in \text{ideal } (\text{set } (\text{take } j \text{ fs}))$
<proof>

lemma *in-idealE-rep-list-dgrad-sig-set-take*:

assumes *hom-grading* d **and** $p \in \text{punit-dgrad-max-set } d$ **and** $p \in \text{ideal } (\text{set } (\text{take } j \text{ fs}))$
obtains r **where** $r \in \text{dgrad-sig-set } d$ **and** $r \in \text{dgrad-sig-set}' j d$ **and** $p = \text{rep-list } r$
<proof>

corollary *in-idealE-rep-list-dgrad-sig-set*:

assumes *hom-grading* d **and** $p \in \text{punit-dgrad-max-set } d$ **and** $p \in \text{ideal } (\text{set } \text{fs})$
obtains r **where** $r \in \text{dgrad-sig-set } d$ **and** $p = \text{rep-list } r$
<proof>

lemma *dgrad-sig-setD-lp*:

assumes $p \in \text{dgrad-sig-set}' j d$
shows $d (\text{lp } p) \leq \text{dgrad-max } d$
<proof>

lemma *dgrad-sig-setD-lt*:

assumes $p \in \text{dgrad-sig-set}' j d$ **and** $p \neq 0$
shows *component-of-term* $(\text{lt } p) < j$
<proof>

lemma *dgrad-sig-setD-rep-list-lt*:

assumes *dickson-grading* d **and** $p \in \text{dgrad-sig-set}' j d$
shows $d (\text{punit.lt } (\text{rep-list } p)) \leq \text{dgrad-max } d$
<proof>

definition *spp-of* $:: ('t \Rightarrow_0 'b) \Rightarrow ('t \times ('a \Rightarrow_0 'b))$

where *spp-of* $r = (\text{lt } r, \text{rep-list } r)$

“spp” stands for “sig-poly-pair”.

lemma *fst-spp-of*: $\text{fst } (\text{spp-of } r) = \text{lt } r$

<proof>

lemma *snd-spp-of*: $\text{snd} (\text{spp-of } r) = \text{rep-list } r$
 ⟨*proof*⟩

4.2.1 Signature Reduction

lemma *term-is-le-rel-canc-left*:
assumes *ord-term-lin.is-le-rel rel*
shows $\text{rel } (t \oplus u) (t \oplus v) \longleftrightarrow \text{rel } u v$
 ⟨*proof*⟩

lemma *term-is-le-rel-minus*:
assumes *ord-term-lin.is-le-rel rel* **and** *s adds t*
shows $\text{rel } ((t - s) \oplus u) v \longleftrightarrow \text{rel } (t \oplus u) (s \oplus v)$
 ⟨*proof*⟩

lemma *term-is-le-rel-minus-minus*:
assumes *ord-term-lin.is-le-rel rel* **and** *a adds t* **and** *b adds t*
shows $\text{rel } ((t - a) \oplus u) ((t - b) \oplus v) \longleftrightarrow \text{rel } (b \oplus u) (a \oplus v)$
 ⟨*proof*⟩

lemma *pp-is-le-rel-canc-right*:
assumes *ordered-powerprod-lin.is-le-rel rel*
shows $\text{rel } (s + u) (t + u) \longleftrightarrow \text{rel } s t$
 ⟨*proof*⟩

lemma *pp-is-le-rel-canc-left*: *ordered-powerprod-lin.is-le-rel rel* $\implies \text{rel } (t + u) (t + v) \longleftrightarrow \text{rel } u v$
 ⟨*proof*⟩

definition *sig-red-single* :: $('t \Rightarrow 't \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow 'a \Rightarrow \text{bool}$
where *sig-red-single* *sig-reg* *top-tail* $p \ q \ f \ t \longleftrightarrow$
 $(\text{rep-list } f \neq 0 \wedge \text{lookup } (\text{rep-list } p) (t + \text{punit.lt } (\text{rep-list } f)) \neq 0 \wedge$
 $q = p - \text{monom-mult } ((\text{lookup } (\text{rep-list } p) (t + \text{punit.lt } (\text{rep-list } f))))$
 $/ \text{punit.lc } (\text{rep-list } f)) \ t \ f \wedge$
 $\text{ord-term-lin.is-le-rel } \text{sig-reg} \wedge \text{ordered-powerprod-lin.is-le-rel } \text{top-tail}$
 \wedge
 $\text{sig-reg } (t \oplus \text{lt } f) (\text{lt } p) \wedge \text{top-tail } (t + \text{punit.lt } (\text{rep-list } f)) (\text{punit.lt } (\text{rep-list } p)))$

The first two parameters of *sig-red-single*, *sig-reg* and *top-tail*, specify whether the reduction is a singular/regular/arbitrary top/tail/arbitrary signature-reduction.

- If *sig-reg* is (=), the reduction is singular.
- If *sig-reg* is (\prec_t), the reduction is regular.
- If *sig-reg* is (\preceq_t), the reduction is an arbitrary signature-reduction.

- If *top-tail* is (=), it is a top reduction.
- If *top-tail* is (<), it is a tail reduction.
- If *top-tail* is (\preceq), the reduction is an arbitrary signature-reduction.

definition *sig-red* :: ('t \Rightarrow 't \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('t \Rightarrow_0 'b) set \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow bool

where *sig-red sing-reg top-tail* F p q \longleftrightarrow ($\exists f \in F. \exists t. \text{sig-red-single sing-reg top-tail p q f t}$)

definition *is-sig-red* :: ('t \Rightarrow 't \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('t \Rightarrow_0 'b) set \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow bool

where *is-sig-red sing-reg top-tail* F p \longleftrightarrow ($\exists q. \text{sig-red sing-reg top-tail F p q}$)

lemma *sig-red-singleI*:

assumes *rep-list* f $\neq 0$ **and** t + *punit.lt* (*rep-list* f) \in *keys* (*rep-list* p)

and q = p - *monom-mult* ((*lookup* (*rep-list* p) (t + *punit.lt* (*rep-list* f))) / *punit.lc* (*rep-list* f)) t f

and *ord-term-lin.is-le-rel sing-reg* **and** *ordered-powerprod-lin.is-le-rel top-tail*

and *sig-reg* (t \oplus lt f) (lt p)

and *top-tail* (t + *punit.lt* (*rep-list* f)) (*punit.lt* (*rep-list* p))

shows *sig-red-single sing-reg top-tail* p q f t

<proof>

lemma *sig-red-singleD1*:

assumes *sig-red-single sing-reg top-tail* p q f t

shows *rep-list* f $\neq 0$

<proof>

lemma *sig-red-singleD2*:

assumes *sig-red-single sing-reg top-tail* p q f t

shows t + *punit.lt* (*rep-list* f) \in *keys* (*rep-list* p)

<proof>

lemma *sig-red-singleD3*:

assumes *sig-red-single sing-reg top-tail* p q f t

shows q = p - *monom-mult* ((*lookup* (*rep-list* p) (t + *punit.lt* (*rep-list* f))) / *punit.lc* (*rep-list* f)) t f

<proof>

lemma *sig-red-singleD4*:

assumes *sig-red-single sing-reg top-tail* p q f t

shows *ord-term-lin.is-le-rel sing-reg*

<proof>

lemma *sig-red-singleD5*:

assumes *sig-red-single sing-reg top-tail* p q f t

shows *ordered-powerprod-lin.is-le-rel top-tail*

<proof>

lemma *sig-red-singleD6*:

assumes *sig-red-single sing-reg top-tail p q f t*

shows *sig-reg (t \oplus lt f) (lt p)*

<proof>

lemma *sig-red-singleD7*:

assumes *sig-red-single sing-reg top-tail p q f t*

shows *top-tail (t + punit.lt (rep-list f)) (punit.lt (rep-list p))*

<proof>

lemma *sig-red-singleD8*:

assumes *sig-red-single sing-reg top-tail p q f t*

shows *t \oplus lt f \preceq_t lt p*

<proof>

lemma *sig-red-singleD9*:

assumes *sig-red-single sing-reg top-tail p q f t*

shows *t + punit.lt (rep-list f) \preceq punit.lt (rep-list p)*

<proof>

lemmas *sig-red-singleD = sig-red-singleD1 sig-red-singleD2 sig-red-singleD3 sig-red-singleD4*

sig-red-singleD5 sig-red-singleD6 sig-red-singleD7 sig-red-singleD8

sig-red-singleD9

lemma *sig-red-single-red-single*:

sig-red-single sing-reg top-tail p q f t \implies punit.red-single (rep-list p) (rep-list q)

(rep-list f) t

<proof>

lemma *sig-red-single-regular-lt*:

assumes *sig-red-single (\prec_t) top-tail p q f t*

shows *lt q = lt p*

<proof>

lemma *sig-red-single-regular-lc*:

assumes *sig-red-single (\prec_t) top-tail p q f t*

shows *lc q = lc p*

<proof>

lemma *sig-red-single-lt*:

assumes *sig-red-single sing-reg top-tail p q f t*

shows *lt q \preceq_t lt p*

<proof>

lemma *sig-red-single-lt-rep-list*:

assumes *sig-red-single sing-reg top-tail p q f t*

shows *punit.lt (rep-list q) \preceq punit.lt (rep-list p)*

$\langle \text{proof} \rangle$

lemma *sig-red-single-tail-lt-in-keys-rep-list:*

assumes *sig-red-single sing-reg* $(\prec) p q f t$

shows $\text{punit.lt (rep-list } p) \in \text{keys (rep-list } q)$

$\langle \text{proof} \rangle$

corollary *sig-red-single-tail-lt-rep-list:*

assumes *sig-red-single sing-reg* $(\prec) p q f t$

shows $\text{punit.lt (rep-list } q) = \text{punit.lt (rep-list } p)$

$\langle \text{proof} \rangle$

lemma *sig-red-single-tail-lc-rep-list:*

assumes *sig-red-single sing-reg* $(\prec) p q f t$

shows $\text{punit.lc (rep-list } q) = \text{punit.lc (rep-list } p)$

$\langle \text{proof} \rangle$

lemma *sig-red-single-top-lt-rep-list:*

assumes *sig-red-single sing-reg* $(=) p q f t$ **and** $\text{rep-list } q \neq 0$

shows $\text{punit.lt (rep-list } q) \prec \text{punit.lt (rep-list } p)$

$\langle \text{proof} \rangle$

lemma *sig-red-single-monom-mult:*

assumes *sig-red-single sing-reg top-tail* $p q f t$ **and** $c \neq 0$

shows $\text{sig-red-single sing-reg top-tail (monom-mult } c \text{ } s \text{ } p) (\text{monom-mult } c \text{ } s \text{ } q) f$
 $(s + t)$

$\langle \text{proof} \rangle$

lemma *sig-red-single-sing-reg-cases:*

$\text{sig-red-single } (\preceq_t) \text{ top-tail } p q f t = (\text{sig-red-single } (=) \text{ top-tail } p q f t \vee \text{sig-red-single } (\prec_t) \text{ top-tail } p q f t)$

$\langle \text{proof} \rangle$

corollary *sig-red-single-sing-regI:*

assumes *sig-red-single sing-reg top-tail* $p q f t$

shows $\text{sig-red-single } (\preceq_t) \text{ top-tail } p q f t$

$\langle \text{proof} \rangle$

lemma *sig-red-single-top-tail-cases:*

$\text{sig-red-single sing-reg } (\preceq) p q f t = (\text{sig-red-single sing-reg } (=) p q f t \vee \text{sig-red-single sing-reg } (\prec) p q f t)$

$\langle \text{proof} \rangle$

corollary *sig-red-single-top-tailI:*

assumes *sig-red-single sing-reg top-tail* $p q f t$

shows $\text{sig-red-single sing-reg } (\preceq) p q f t$

$\langle \text{proof} \rangle$

lemma *dgrad-max-set-closed-sig-red-single:*

assumes *dickson-grading* d **and** $p \in \text{dgrad-max-set } d$ **and** $f \in \text{dgrad-max-set } d$
and *sig-red-single sing-red top-tail* $p \ q \ f \ t$
shows $q \in \text{dgrad-max-set } d$
 $\langle \text{proof} \rangle$

lemma *sig-inv-set-closed-sig-red-single*:
assumes $p \in \text{sig-inv-set}$ **and** $f \in \text{sig-inv-set}$ **and** *sig-red-single sing-red top-tail*
 $p \ q \ f \ t$
shows $q \in \text{sig-inv-set}$
 $\langle \text{proof} \rangle$

corollary *dgrad-sig-set-closed-sig-red-single*:
assumes *dickson-grading* d **and** $p \in \text{dgrad-sig-set } d$ **and** $f \in \text{dgrad-sig-set } d$
and *sig-red-single sing-red top-tail* $p \ q \ f \ t$
shows $q \in \text{dgrad-sig-set } d$
 $\langle \text{proof} \rangle$

lemma *sig-red-regular-lt*: *sig-red* (\prec_t) *top-tail* $F \ p \ q \implies \text{lt } q = \text{lt } p$
 $\langle \text{proof} \rangle$

lemma *sig-red-regular-lc*: *sig-red* (\prec_t) *top-tail* $F \ p \ q \implies \text{lc } q = \text{lc } p$
 $\langle \text{proof} \rangle$

lemma *sig-red-lt*: *sig-red sing-reg top-tail* $F \ p \ q \implies \text{lt } q \preceq_t \text{lt } p$
 $\langle \text{proof} \rangle$

lemma *sig-red-tail-lt-rep-list*: *sig-red sing-reg* (\prec) $F \ p \ q \implies \text{punit.lt } (\text{rep-list } q) =$
 $\text{punit.lt } (\text{rep-list } p)$
 $\langle \text{proof} \rangle$

lemma *sig-red-tail-lc-rep-list*: *sig-red sing-reg* (\prec) $F \ p \ q \implies \text{punit.lc } (\text{rep-list } q) =$
 $\text{punit.lc } (\text{rep-list } p)$
 $\langle \text{proof} \rangle$

lemma *sig-red-top-lt-rep-list*:
sig-red sing-reg $(=)$ $F \ p \ q \implies \text{rep-list } q \neq 0 \implies \text{punit.lt } (\text{rep-list } q) \prec \text{punit.lt}$
 $(\text{rep-list } p)$
 $\langle \text{proof} \rangle$

lemma *sig-red-lt-rep-list*: *sig-red sing-reg top-tail* $F \ p \ q \implies \text{punit.lt } (\text{rep-list } q) \preceq$
 $\text{punit.lt } (\text{rep-list } p)$
 $\langle \text{proof} \rangle$

lemma *sig-red-red*: *sig-red sing-reg top-tail* $F \ p \ q \implies \text{punit.red } (\text{rep-list } ' F)$
 $(\text{rep-list } p) (\text{rep-list } q)$
 $\langle \text{proof} \rangle$

lemma *sig-red-monom-mult*:
sig-red sing-reg top-tail $F \ p \ q \implies c \neq 0 \implies \text{sig-red sing-reg top-tail } F (\text{monom-mult}$

$c s p$ (monom-mult $c s q$)
 ⟨proof⟩

lemma *sig-red-sing-reg-cases*:

$\text{sig-red } (\preceq_t) \text{ top-tail } F p q = (\text{sig-red } (=) \text{ top-tail } F p q \vee \text{sig-red } (\prec_t) \text{ top-tail } F p q)$
 ⟨proof⟩

corollary *sig-red-sing-regI*: $\text{sig-red sing-reg top-tail } F p q \implies \text{sig-red } (\preceq_t) \text{ top-tail } F p q$
 ⟨proof⟩

lemma *sig-red-top-tail-cases*:

$\text{sig-red sing-reg } (\preceq) F p q = (\text{sig-red sing-reg } (=) F p q \vee \text{sig-red sing-reg } (\prec) F p q)$
 ⟨proof⟩

corollary *sig-red-top-tailI*: $\text{sig-red sing-reg top-tail } F p q \implies \text{sig-red sing-reg } (\preceq) F p q$
 ⟨proof⟩

lemma *sig-red-wf-dgrad-max-set*:

assumes *dickson-grading* d **and** $F \subseteq \text{dgrad-max-set } d$
shows $\text{wfP } (\text{sig-red sing-reg top-tail } F)^{-1-1}$
 ⟨proof⟩

lemma *dgrad-sig-set-closed-sig-red*:

assumes *dickson-grading* d **and** $F \subseteq \text{dgrad-sig-set } d$ **and** $p \in \text{dgrad-sig-set } d$
and $\text{sig-red sing-red top-tail } F p q$
shows $q \in \text{dgrad-sig-set } d$
 ⟨proof⟩

lemma *sig-red-mono*: $\text{sig-red sing-reg top-tail } F p q \implies F \subseteq F' \implies \text{sig-red sing-reg top-tail } F' p q$
 ⟨proof⟩

lemma *sig-red-Un*:

$\text{sig-red sing-reg top-tail } (A \cup B) p q \iff (\text{sig-red sing-reg top-tail } A p q \vee \text{sig-red sing-reg top-tail } B p q)$
 ⟨proof⟩

lemma *sig-red-subset*:

assumes $\text{sig-red sing-reg top-tail } F p q$ **and** $\text{sing-reg } = (\preceq_t) \vee \text{sing-reg } = (\prec_t)$
shows $\text{sig-red sing-reg top-tail } \{f \in F. \text{sing-reg } (lt f) (lt p)\} p q$
 ⟨proof⟩

lemma *sig-red-regular-rtrancl-lt*:

assumes $(\text{sig-red } (\prec_t) \text{ top-tail } F)^{**} p q$
shows $lt q = lt p$

<proof>

lemma *sig-red-regular-rtrancl-lc:*

assumes $(\text{sig-red } (\prec_t) \text{ top-tail } F)^{**} p q$

shows $lc q = lc p$

<proof>

lemma *sig-red-rtrancl-lt:*

assumes $(\text{sig-red sing-reg top-tail } F)^{**} p q$

shows $lt q \preceq_t lt p$

<proof>

lemma *sig-red-tail-rtrancl-lt-rep-list:*

assumes $(\text{sig-red sing-reg } (\prec) F)^{**} p q$

shows $\text{punit.lt } (\text{rep-list } q) = \text{punit.lt } (\text{rep-list } p)$

<proof>

lemma *sig-red-tail-rtrancl-lc-rep-list:*

assumes $(\text{sig-red sing-reg } (\prec) F)^{**} p q$

shows $\text{punit.lc } (\text{rep-list } q) = \text{punit.lc } (\text{rep-list } p)$

<proof>

lemma *sig-red-rtrancl-lt-rep-list:*

assumes $(\text{sig-red sing-reg top-tail } F)^{**} p q$

shows $\text{punit.lt } (\text{rep-list } q) \preceq \text{punit.lt } (\text{rep-list } p)$

<proof>

lemma *sig-red-red-rtrancl:*

assumes $(\text{sig-red sing-reg top-tail } F)^{**} p q$

shows $(\text{punit.red } (\text{rep-list } ' F))^{**} (\text{rep-list } p) (\text{rep-list } q)$

<proof>

lemma *sig-red-rtrancl-monom-mult:*

assumes $(\text{sig-red sing-reg top-tail } F)^{**} p q$

shows $(\text{sig-red sing-reg top-tail } F)^{**} (\text{monom-mult } c s p) (\text{monom-mult } c s q)$

<proof>

lemma *sig-red-rtrancl-sing-regI:* $(\text{sig-red sing-reg top-tail } F)^{**} p q \implies (\text{sig-red } (\preceq_t) \text{ top-tail } F)^{**} p q$

<proof>

lemma *sig-red-rtrancl-top-tailI:* $(\text{sig-red sing-reg top-tail } F)^{**} p q \implies (\text{sig-red sing-reg } (\preceq) F)^{**} p q$

<proof>

lemma *dgrad-sig-set-closed-sig-red-rtrancl:*

assumes *dickson-grading* d **and** $F \subseteq \text{dgrad-sig-set } d$ **and** $p \in \text{dgrad-sig-set } d$

and $(\text{sig-red sing-red top-tail } F)^{**} p q$

shows $q \in \text{dgrad-sig-set } d$

<proof>

lemma *sig-red-rtrancl-mono*:

assumes $(\text{sig-red sing-reg top-tail } F)^{**} p q$ **and** $F \subseteq F'$

shows $(\text{sig-red sing-reg top-tail } F')^{**} p q$

<proof>

lemma *sig-red-rtrancl-subset*:

assumes $(\text{sig-red sing-reg top-tail } F)^{**} p q$ **and** $\text{sing-reg} = (\preceq_t) \vee \text{sing-reg} = (\prec_t)$

shows $(\text{sig-red sing-reg top-tail } \{f \in F. \text{sing-reg } (lt f) (lt p)\})^{**} p q$

<proof>

lemma *is-sig-red-is-red*: $\text{is-sig-red sing-reg top-tail } F p \implies \text{punit.is-red } (\text{rep-list } F) (\text{rep-list } p)$

<proof>

lemma *is-sig-red-monom-mult*:

assumes $\text{is-sig-red sing-reg top-tail } F p$ **and** $c \neq 0$

shows $\text{is-sig-red sing-reg top-tail } F (\text{monom-mult } c s p)$

<proof>

lemma *is-sig-red-sing-reg-cases*:

$\text{is-sig-red } (\preceq_t) \text{ top-tail } F p = (\text{is-sig-red } (=) \text{ top-tail } F p \vee \text{is-sig-red } (\prec_t) \text{ top-tail } F p)$

<proof>

corollary *is-sig-red-sing-regI*: $\text{is-sig-red sing-reg top-tail } F p \implies \text{is-sig-red } (\preceq_t) \text{ top-tail } F p$

<proof>

lemma *is-sig-red-top-tail-cases*:

$\text{is-sig-red sing-reg } (\preceq) F p = (\text{is-sig-red sing-reg } (=) F p \vee \text{is-sig-red sing-reg } (\prec) F p)$

<proof>

corollary *is-sig-red-top-tailII*: $\text{is-sig-red sing-reg top-tail } F p \implies \text{is-sig-red sing-reg } (\preceq) F p$

<proof>

lemma *is-sig-red-singletonI*:

assumes $\text{is-sig-red sing-reg top-tail } F r$

obtains f **where** $f \in F$ **and** $\text{is-sig-red sing-reg top-tail } \{f\} r$

<proof>

lemma *is-sig-red-singletonD*:

assumes $\text{is-sig-red sing-reg top-tail } \{f\} r$ **and** $f \in F$

shows $\text{is-sig-red sing-reg top-tail } F r$

<proof>

lemma *is-sig-redD1*:

assumes *is-sig-red sing-reg top-tail F p*
shows *ord-term-lin.is-le-rel sing-reg*
(*proof*)

lemma *is-sig-redD2*:

assumes *is-sig-red sing-reg top-tail F p*
shows *ordered-powerprod-lin.is-le-rel top-tail*
(*proof*)

lemma *is-sig-red-addsI*:

assumes $f \in F$ **and** $t \in \text{keys } (\text{rep-list } p)$ **and** $\text{rep-list } f \neq 0$ **and** $\text{punit.lt } (\text{rep-list } f)$ *adds t*
and *ord-term-lin.is-le-rel sing-reg* **and** *ordered-powerprod-lin.is-le-rel top-tail*
and *sing-reg* $(t \oplus \text{lt } f)$ $(\text{punit.lt } (\text{rep-list } f) \oplus \text{lt } p)$ **and** *top-tail t* $(\text{punit.lt } (\text{rep-list } p))$
shows *is-sig-red sing-reg top-tail F p*
(*proof*)

lemma *is-sig-red-addsE*:

assumes *is-sig-red sing-reg top-tail F p*
obtains $f t$ **where** $f \in F$ **and** $t \in \text{keys } (\text{rep-list } p)$ **and** $\text{rep-list } f \neq 0$
and $\text{punit.lt } (\text{rep-list } f)$ *adds t*
and *sing-reg* $(t \oplus \text{lt } f)$ $(\text{punit.lt } (\text{rep-list } f) \oplus \text{lt } p)$
and *top-tail t* $(\text{punit.lt } (\text{rep-list } p))$
(*proof*)

lemma *is-sig-red-top-addsI*:

assumes $f \in F$ **and** $\text{rep-list } f \neq 0$ **and** $\text{rep-list } p \neq 0$
and $\text{punit.lt } (\text{rep-list } f)$ *adds* $\text{punit.lt } (\text{rep-list } p)$ **and** *ord-term-lin.is-le-rel*
sing-reg
and *sing-reg* $(\text{punit.lt } (\text{rep-list } p) \oplus \text{lt } f)$ $(\text{punit.lt } (\text{rep-list } f) \oplus \text{lt } p)$
shows *is-sig-red sing-reg (=) F p*
(*proof*)

lemma *is-sig-red-top-addsE*:

assumes *is-sig-red sing-reg (=) F p*
obtains f **where** $f \in F$ **and** $\text{rep-list } f \neq 0$ **and** $\text{rep-list } p \neq 0$
and $\text{punit.lt } (\text{rep-list } f)$ *adds* $\text{punit.lt } (\text{rep-list } p)$
and *sing-reg* $(\text{punit.lt } (\text{rep-list } p) \oplus \text{lt } f)$ $(\text{punit.lt } (\text{rep-list } f) \oplus \text{lt } p)$
(*proof*)

lemma *is-sig-red-top-plusE*:

assumes *is-sig-red sing-reg (=) F p* **and** *is-sig-red sing-reg (=) F q*
and $\text{lt } p \preceq_t \text{lt } (p + q)$ **and** $\text{lt } q \preceq_t \text{lt } (p + q)$ **and** *sing-reg =* $(\preceq_t) \vee \text{sing-reg}$
 $= (\prec_t)$
assumes 1: *is-sig-red sing-reg (=) F (p + q) \implies thesis*
assumes 2: $\text{punit.lt } (\text{rep-list } p) = \text{punit.lt } (\text{rep-list } q) \implies \text{punit.lt } (\text{rep-list } p) +$

$punit.lt (rep-list\ q) = 0 \implies thesis$

shows $thesis$

$\langle proof \rangle$

lemma $is-sig-red-singleton-monom-multD$:

assumes $is-sig-red\ sing-reg\ top-tail\ \{monom-mult\ c\ t\ f\}\ p$

shows $is-sig-red\ sing-reg\ top-tail\ \{f\}\ p$

$\langle proof \rangle$

lemma $is-sig-red-top-singleton-monom-multI$:

assumes $is-sig-red\ sing-reg\ (=)\ \{f\}\ p$ **and** $c \neq 0$

and t **adds** $punit.lt (rep-list\ p) - punit.lt (rep-list\ f)$

shows $is-sig-red\ sing-reg\ (=)\ \{monom-mult\ c\ t\ f\}\ p$

$\langle proof \rangle$

lemma $is-sig-red-cong'$:

assumes $is-sig-red\ sing-reg\ top-tail\ F\ p$ **and** $lt\ p = lt\ q$ **and** $rep-list\ p = rep-list$

q

shows $is-sig-red\ sing-reg\ top-tail\ F\ q$

$\langle proof \rangle$

lemma $is-sig-red-cong$:

$lt\ p = lt\ q \implies rep-list\ p = rep-list\ q \implies$

$is-sig-red\ sing-reg\ top-tail\ F\ p \longleftrightarrow is-sig-red\ sing-reg\ top-tail\ F\ q$

$\langle proof \rangle$

lemma $is-sig-red-top-cong$:

assumes $is-sig-red\ sing-reg\ (=)\ F\ p$ **and** $rep-list\ q \neq 0$ **and** $lt\ p = lt\ q$

and $punit.lt (rep-list\ p) = punit.lt (rep-list\ q)$

shows $is-sig-red\ sing-reg\ (=)\ F\ q$

$\langle proof \rangle$

lemma $sig-irredE-dgrad-max-set$:

assumes $dickson-grading\ d$ **and** $F \subseteq dgrad-max-set\ d$

obtains q **where** $(sig-red\ sing-reg\ top-tail\ F)^{**}\ p\ q$ **and** $\neg is-sig-red\ sing-reg\ top-tail\ F\ q$

$\langle proof \rangle$

lemma $is-sig-red-mono$:

$is-sig-red\ sing-reg\ top-tail\ F\ p \implies F \subseteq F' \implies is-sig-red\ sing-reg\ top-tail\ F'\ p$

$\langle proof \rangle$

lemma $is-sig-red-Un$:

$is-sig-red\ sing-reg\ top-tail\ (A \cup B)\ p \longleftrightarrow (is-sig-red\ sing-reg\ top-tail\ A\ p \vee is-sig-red\ sing-reg\ top-tail\ B\ p)$

$\langle proof \rangle$

lemma $is-sig-redD-lt$:

assumes $is-sig-red\ (\preceq_t)\ top-tail\ \{f\}\ p$

shows $lt\ f \preceq_t\ lt\ p$
 $\langle proof \rangle$

lemma *is-sig-red-regularD-lt*:
assumes *is-sig-red* (\prec_t) *top-tail* $\{f\}$ p
shows $lt\ f \prec_t\ lt\ p$
 $\langle proof \rangle$

lemma *sig-irred-regular-self*: \neg *is-sig-red* (\prec_t) *top-tail* $\{p\}$ p
 $\langle proof \rangle$

4.2.2 Signature Gröbner Bases

definition *sig-red-zero* :: $('t \Rightarrow 't \Rightarrow bool) \Rightarrow ('t \Rightarrow_0 'b)\ set \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow bool$
where *sig-red-zero* *sig-reg* $F\ r \longleftrightarrow (\exists s. (sig-red\ sing-reg\ (\preceq)\ F)^{**}\ r\ s \wedge rep-list\ s = 0)$

definition *is-sig-GB-in* :: $('a \Rightarrow nat) \Rightarrow ('t \Rightarrow_0 'b)\ set \Rightarrow 't \Rightarrow bool$
where *is-sig-GB-in* $d\ G\ u \longleftrightarrow (\forall r. lt\ r = u \longrightarrow r \in dgrad-sig-set\ d \longrightarrow sig-red-zero\ (\preceq_t)\ G\ r)$

definition *is-sig-GB-upt* :: $('a \Rightarrow nat) \Rightarrow ('t \Rightarrow_0 'b)\ set \Rightarrow 't \Rightarrow bool$
where *is-sig-GB-upt* $d\ G\ u \longleftrightarrow$
 $(G \subseteq dgrad-sig-set\ d \wedge (\forall v. v \prec_t\ u \longrightarrow d\ (pp-of-term\ v) \leq dgrad-max\ d \longrightarrow$
 $component-of-term\ v < length\ fs \longrightarrow is-sig-GB-in\ d\ G\ v))$

definition *is-min-sig-GB* :: $('a \Rightarrow nat) \Rightarrow ('t \Rightarrow_0 'b)\ set \Rightarrow bool$
where *is-min-sig-GB* $d\ G \longleftrightarrow G \subseteq dgrad-sig-set\ d \wedge$
 $(\forall u. d\ (pp-of-term\ u) \leq dgrad-max\ d \longrightarrow component-of-term\ u < length\ fs \longrightarrow$
 $is-sig-GB-in\ d\ G\ u) \wedge$
 $(\forall g \in G. \neg is-sig-red\ (\preceq_t)\ (=)\ (G - \{g\})\ g)$

definition *is-syz-sig* :: $('a \Rightarrow nat) \Rightarrow 't \Rightarrow bool$
where *is-syz-sig* $d\ u \longleftrightarrow (\exists s \in dgrad-sig-set\ d. s \neq 0 \wedge lt\ s = u \wedge rep-list\ s = 0)$

lemma *sig-red-zeroI*:
assumes $(sig-red\ sing-reg\ (\preceq)\ F)^{**}\ r\ s$ **and** $rep-list\ s = 0$
shows *sig-red-zero* *sig-reg* $F\ r$
 $\langle proof \rangle$

lemma *sig-red-zeroE*:
assumes *sig-red-zero* *sig-reg* $F\ r$
obtains s **where** $(sig-red\ sing-reg\ (\preceq)\ F)^{**}\ r\ s$ **and** $rep-list\ s = 0$
 $\langle proof \rangle$

lemma *sig-red-zero-monom-mult*:

assumes *sig-red-zero sing-reg* F r

shows *sig-red-zero sing-reg* F (*monom-mult* c t r)

<proof>

lemma *sig-red-zero-sing-regI*:

assumes *sig-red-zero sing-reg* G p

shows *sig-red-zero* (\preceq_t) G p

<proof>

lemma *sig-red-zero-nonzero*:

assumes *sig-red-zero sing-reg* F r **and** *rep-list* $r \neq 0$ **and** *sig-reg* $= (\preceq_t) \vee$
sig-reg $= (\prec_t)$

shows *is-sig-red sing-reg* $(=)$ F r

<proof>

lemma *sig-red-zero-mono*: *sig-red-zero sing-reg* F $p \implies F \subseteq F' \implies$ *sig-red-zero*
sig-reg F' p

<proof>

lemma *sig-red-zero-subset*:

assumes *sig-red-zero sing-reg* F p **and** *sig-reg* $= (\preceq_t) \vee$ *sig-reg* $= (\prec_t)$

shows *sig-red-zero sing-reg* $\{f \in F. \text{sig-reg } (lt\ f) (lt\ p)\}$ p

<proof>

lemma *sig-red-zero-idealI*:

assumes *sig-red-zero sing-reg* F p

shows *rep-list* $p \in \text{ideal } (\text{rep-list } 'F)$

<proof>

lemma *is-sig-GB-inI*:

assumes $\bigwedge r. lt\ r = u \implies r \in \text{dgrad-sig-set } d \implies$ *sig-red-zero* (\preceq_t) G r

shows *is-sig-GB-in* d G u

<proof>

lemma *is-sig-GB-inD*:

assumes *is-sig-GB-in* d G u **and** $r \in \text{dgrad-sig-set } d$ **and** $lt\ r = u$

shows *sig-red-zero* (\preceq_t) G r

<proof>

lemma *is-sig-GB-inI-triv*:

assumes $\neg d$ (*pp-of-term* u) $\leq \text{dgrad-max } d \vee \neg$ *component-of-term* $u < \text{length}$
 fs

shows *is-sig-GB-in* d G u

<proof>

lemma *is-sig-GB-in-mono*: *is-sig-GB-in* d G $u \implies G \subseteq G' \implies$ *is-sig-GB-in* d G'
 u

<proof>

lemma *is-sig-GB-uptI*:
assumes $G \subseteq \text{dgrad-sig-set } d$
and $\bigwedge v. v \prec_t u \implies d (\text{pp-of-term } v) \leq \text{dgrad-max } d \implies \text{component-of-term } v < \text{length } fs \implies$
 $\text{is-sig-GB-in } d \ G \ v$
shows $\text{is-sig-GB-upt } d \ G \ u$
 $\langle \text{proof} \rangle$

lemma *is-sig-GB-uptD1*:
assumes $\text{is-sig-GB-upt } d \ G \ u$
shows $G \subseteq \text{dgrad-sig-set } d$
 $\langle \text{proof} \rangle$

lemma *is-sig-GB-uptD2*:
assumes $\text{is-sig-GB-upt } d \ G \ u$ **and** $v \prec_t u$
shows $\text{is-sig-GB-in } d \ G \ v$
 $\langle \text{proof} \rangle$

lemma *is-sig-GB-uptD3*:
assumes $\text{is-sig-GB-upt } d \ G \ u$ **and** $r \in \text{dgrad-sig-set } d$ **and** $lt \ r \prec_t \ u$
shows $\text{sig-red-zero } (\preceq_t) \ G \ r$
 $\langle \text{proof} \rangle$

lemma *is-sig-GB-upt-le*:
assumes $\text{is-sig-GB-upt } d \ G \ u$ **and** $v \preceq_t \ u$
shows $\text{is-sig-GB-upt } d \ G \ v$
 $\langle \text{proof} \rangle$

lemma *is-sig-GB-upt-mono*:
 $\text{is-sig-GB-upt } d \ G \ u \implies G \subseteq G' \implies G' \subseteq \text{dgrad-sig-set } d \implies \text{is-sig-GB-upt } d \ G' \ u$
 $\langle \text{proof} \rangle$

lemma *is-sig-GB-upt-is-Groebner-basis*:
assumes $\text{dickson-grading } d$ **and** $\text{hom-grading } d$ **and** $G \subseteq \text{dgrad-sig-set}' \ j \ d$
and $\bigwedge u. \text{component-of-term } u < j \implies \text{is-sig-GB-in } d \ G \ u$
shows $\text{punit.is-Groebner-basis } (\text{rep-list } ' G)$
 $\langle \text{proof} \rangle$

lemma *is-sig-GB-is-Groebner-basis*:
assumes $\text{dickson-grading } d$ **and** $\text{hom-grading } d$ **and** $G \subseteq \text{dgrad-max-set } d$ **and**
 $\bigwedge u. \text{is-sig-GB-in } d \ G \ u$
shows $\text{punit.is-Groebner-basis } (\text{rep-list } ' G)$
 $\langle \text{proof} \rangle$

lemma *sig-red-zero-is-red*:
assumes $\text{sig-red-zero sing-reg } F \ r$ **and** $\text{rep-list } r \neq 0$
shows $\text{is-sig-red sing-reg } (\preceq) \ F \ r$

<proof>

lemma *is-sig-red-sing-top-is-red-zero*:

assumes *dickson-grading* d **and** *is-sig-GB-upt* d G u **and** $a \in$ *dgrad-sig-set* d
and $lt\ a = u$

and *is-sig-red* $(=)$ $(=)$ G a **and** \neg *is-sig-red* (\prec_t) $(=)$ G a

shows *sig-red-zero* (\preceq_t) G a

<proof>

lemma *sig-regular-reduced-unique*:

assumes *is-sig-GB-upt* d G $(lt\ q)$ **and** $p \in$ *dgrad-sig-set* d **and** $q \in$ *dgrad-sig-set* d

and $lt\ p = lt\ q$ **and** $lc\ p = lc\ q$ **and** \neg *is-sig-red* (\prec_t) (\preceq) G p **and** \neg *is-sig-red* (\prec_t) (\preceq) G q

shows *rep-list* $p = rep-list\ q$

<proof>

corollary *sig-regular-reduced-unique'*:

assumes *is-sig-GB-upt* d G $(lt\ q)$ **and** $p \in$ *dgrad-sig-set* d **and** $q \in$ *dgrad-sig-set* d

and $lt\ p = lt\ q$ **and** \neg *is-sig-red* (\prec_t) (\preceq) G p **and** \neg *is-sig-red* (\prec_t) (\preceq) G q

shows *punit.monom-mult* $(lc\ q)$ 0 $(rep-list\ p) = punit.monom-mult$ $(lc\ p)$ 0
 $(rep-list\ q)$

<proof>

lemma *sig-regular-top-reduced-lt-lc-unique*:

assumes *dickson-grading* d **and** *is-sig-GB-upt* d G $(lt\ q)$ **and** $p \in$ *dgrad-sig-set* d **and** $q \in$ *dgrad-sig-set* d

and $lt\ p = lt\ q$ **and** $(p = 0) \longleftrightarrow (q = 0)$ **and** \neg *is-sig-red* (\prec_t) $(=)$ G p **and**
 \neg *is-sig-red* (\prec_t) $(=)$ G q

shows *punit.lt* $(rep-list\ p) = punit.lt$ $(rep-list\ q) \wedge lc\ q * punit.lc$ $(rep-list\ p) =$
 $lc\ p * punit.lc$ $(rep-list\ q)$

<proof>

corollary *sig-regular-top-reduced-lt-unique*:

assumes *dickson-grading* d **and** *is-sig-GB-upt* d G $(lt\ q)$ **and** $p \in$ *dgrad-sig-set* d

and $q \in$ *dgrad-sig-set* d **and** $lt\ p = lt\ q$ **and** $p \neq 0$ **and** $q \neq 0$

and \neg *is-sig-red* (\prec_t) $(=)$ G p **and** \neg *is-sig-red* (\prec_t) $(=)$ G q

shows *punit.lt* $(rep-list\ p) = punit.lt$ $(rep-list\ q)$

<proof>

corollary *sig-regular-top-reduced-lc-unique*:

assumes *dickson-grading* d **and** *is-sig-GB-upt* d G $(lt\ q)$ **and** $p \in$ *dgrad-sig-set* d **and** $q \in$ *dgrad-sig-set* d

and $lt\ p = lt\ q$ **and** $lc\ p = lc\ q$ **and** \neg *is-sig-red* (\prec_t) $(=)$ G p **and** \neg *is-sig-red* (\prec_t) $(=)$ G q

shows *punit.lc* $(rep-list\ p) = punit.lc$ $(rep-list\ q)$

<proof>

Minimal signature Gröbner bases are indeed minimal, at least up to sig-lead-pairs:

lemma *is-min-sig-GB-minimal*:

assumes *is-min-sig-GB* d G **and** $G' \subseteq \text{dgrad-sig-set } d$
and $\bigwedge u. d(\text{pp-of-term } u) \leq \text{dgrad-max } d \implies \text{component-of-term } u < \text{length } fs \implies \text{is-sig-GB-in } d$ $G' u$
and $g \in G$ **and** $\text{rep-list } g \neq 0$
obtains g' **where** $g' \in G'$ **and** $\text{rep-list } g' \neq 0$ **and** $\text{lt } g' = \text{lt } g$
and $\text{punit.lt } (\text{rep-list } g') = \text{punit.lt } (\text{rep-list } g)$
 $\langle \text{proof} \rangle$

lemma *sig-red-zero-regularI-adds*:

assumes *dickson-grading* d **and** *is-sig-GB-upt* d G $(\text{lt } q)$
and $p \in \text{dgrad-sig-set } d$ **and** $q \in \text{dgrad-sig-set } d$ **and** $p \neq 0$ **and** *sig-red-zero* (\prec_t) G p
and $\text{lt } p \text{ adds}_t \text{lt } q$
shows *sig-red-zero* (\prec_t) G q
 $\langle \text{proof} \rangle$

lemma *is-syz-sigI*:

assumes $s \neq 0$ **and** $\text{lt } s = u$ **and** $s \in \text{dgrad-sig-set } d$ **and** $\text{rep-list } s = 0$
shows *is-syz-sig* d u
 $\langle \text{proof} \rangle$

lemma *is-syz-sigE*:

assumes *is-syz-sig* d u
obtains r **where** $r \neq 0$ **and** $\text{lt } r = u$ **and** $r \in \text{dgrad-sig-set } d$ **and** $\text{rep-list } r = 0$
 $\langle \text{proof} \rangle$

lemma *is-syz-sig-adds*:

assumes *dickson-grading* d **and** *is-syz-sig* d u **and** $u \text{ adds}_t v$
and $d(\text{pp-of-term } v) \leq \text{dgrad-max } d$
shows *is-syz-sig* d v
 $\langle \text{proof} \rangle$

lemma *szygy-crit*:

assumes *dickson-grading* d **and** *is-sig-GB-upt* d G u **and** *is-syz-sig* d u
and $p \in \text{dgrad-sig-set } d$ **and** $\text{lt } p = u$
shows *sig-red-zero* (\prec_t) G p
 $\langle \text{proof} \rangle$

lemma *lemma-21*:

assumes *dickson-grading* d **and** *is-sig-GB-upt* d G $(\text{lt } p)$ **and** $p \in \text{dgrad-sig-set } d$ **and** $g \in G$
and $\text{rep-list } p \neq 0$ **and** $\text{rep-list } g \neq 0$ **and** $\text{lt } g \text{ adds}_t \text{lt } p$
and $\text{punit.lt } (\text{rep-list } g) \text{ adds } \text{punit.lt } (\text{rep-list } p)$
shows *is-sig-red* (\succeq_t) $(=)$ G p
 $\langle \text{proof} \rangle$

4.2.3 Rewrite Bases

definition *is-rewrite-ord* :: $((t \times (a \Rightarrow_0 b)) \Rightarrow (t \times (a \Rightarrow_0 b)) \Rightarrow \text{bool}) \Rightarrow \text{bool}$
where *is-rewrite-ord* *rword* \longleftrightarrow $(\text{reflp } rword \wedge \text{transp } rword \wedge (\forall a b. rword a b \vee rword b a) \wedge$

$(\forall a b. rword a b \longrightarrow rword b a \longrightarrow \text{fst } a = \text{fst } b) \wedge$
 $(\forall d G a b. \text{dickson-grading } d \longrightarrow \text{is-sig-GB-upt } d G (lt b) \longrightarrow$
 $a \in G \longrightarrow b \in G \longrightarrow a \neq 0 \longrightarrow b \neq 0 \longrightarrow lt a$
 $\text{adds}_t lt b \longrightarrow$
 $\neg \text{is-sig-red } (\prec_t) (=) G b \longrightarrow rword (spp\text{-of } a)$
 $(spp\text{-of } b)))$

definition *is-canon-rewriter* :: $((t \times (a \Rightarrow_0 b)) \Rightarrow (t \times (a \Rightarrow_0 b)) \Rightarrow \text{bool}) \Rightarrow (t \Rightarrow_0 b) \text{ set} \Rightarrow t \Rightarrow (t \Rightarrow_0 b) \Rightarrow \text{bool}$

where *is-canon-rewriter* *rword* *A* *u* *p* \longleftrightarrow
 $(p \in A \wedge p \neq 0 \wedge lt p \text{ adds}_t u \wedge (\forall a \in A. a \neq 0 \longrightarrow lt a \text{ adds}_t u$
 $\longrightarrow rword (spp\text{-of } a) (spp\text{-of } p)))$

definition *is-RB-in* :: $(a \Rightarrow \text{nat}) \Rightarrow ((t \times (a \Rightarrow_0 b)) \Rightarrow (t \times (a \Rightarrow_0 b)) \Rightarrow \text{bool}) \Rightarrow (t \Rightarrow_0 b) \text{ set} \Rightarrow t \Rightarrow \text{bool}$

where *is-RB-in* *d* *rword* *G* *u* \longleftrightarrow
 $((\exists g. \text{is-canon-rewriter } rword G u g \wedge \neg \text{is-sig-red } (\prec_t) (=) G (\text{monom-mult } 1 (pp\text{-of-term } u - lp g) g)) \vee$
 $\text{is-syz-sig } d u)$

definition *is-RB-upt* :: $(a \Rightarrow \text{nat}) \Rightarrow ((t \times (a \Rightarrow_0 b)) \Rightarrow (t \times (a \Rightarrow_0 b)) \Rightarrow \text{bool}) \Rightarrow (t \Rightarrow_0 b) \text{ set} \Rightarrow t \Rightarrow \text{bool}$

where *is-RB-upt* *d* *rword* *G* *u* \longleftrightarrow
 $(G \subseteq \text{dgrad-sig-set } d \wedge (\forall v. v \prec_t u \longrightarrow d (pp\text{-of-term } v) \leq \text{dgrad-max } d$
 \longrightarrow
 $\text{component-of-term } v < \text{length } fs \longrightarrow \text{is-RB-in } d$
 $rword G v))$

lemma *is-rewrite-ordI*:

assumes *reflp* *rword* **and** *transp* *rword* **and** $\bigwedge a b. rword a b \vee rword b a$
and $\bigwedge a b. rword a b \Longrightarrow rword b a \Longrightarrow \text{fst } a = \text{fst } b$
and $\bigwedge d G a b. \text{dickson-grading } d \Longrightarrow \text{is-sig-GB-upt } d G (lt b) \Longrightarrow a \in G \Longrightarrow$
 $b \in G \Longrightarrow$
 $a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow lt a \text{ adds}_t lt b \Longrightarrow \neg \text{is-sig-red } (\prec_t) (=) G b$
 $\Longrightarrow rword (spp\text{-of } a) (spp\text{-of } b)$
shows *is-rewrite-ord* *rword*
 $\langle \text{proof} \rangle$

lemma *is-rewrite-ordD1*: *is-rewrite-ord* *rword* $\Longrightarrow rword a a$

$\langle \text{proof} \rangle$

lemma *is-rewrite-ordD2*: *is-rewrite-ord* *rword* $\Longrightarrow rword a b \Longrightarrow rword b c \Longrightarrow rword a c$

$\langle \text{proof} \rangle$

lemma *is-rewrite-ordD3*:

assumes *is-rewrite-ord* *rword*

and $rword\ a\ b \implies thesis$

and $\neg rword\ a\ b \implies rword\ b\ a \implies thesis$

shows *thesis*

<proof>

lemma *is-rewrite-ordD4*:

assumes *is-rewrite-ord* *rword* **and** *rword* *a* *b* **and** *rword* *b* *a*

shows $fst\ a = fst\ b$

<proof>

lemma *is-rewrite-ordD4'*:

assumes *is-rewrite-ord* *rword* **and** *rword* (*spp-of* *a*) (*spp-of* *b*) **and** *rword* (*spp-of* *b*) (*spp-of* *a*)

shows $lt\ a = lt\ b$

<proof>

lemma *is-rewrite-ordD5*:

assumes *is-rewrite-ord* *rword* **and** *dickson-grading* *d* **and** *is-sig-GB-upt* *d* *G* (*lt* *b*)

and $a \in G$ **and** $b \in G$ **and** $a \neq 0$ **and** $b \neq 0$ **and** $lt\ a\ adds_t\ lt\ b$

and $\neg is-sig-red\ (\prec_t)\ (=)\ G\ b$

shows *rword* (*spp-of* *a*) (*spp-of* *b*)

<proof>

lemma *is-canon-rewriterI*:

assumes $p \in A$ **and** $p \neq 0$ **and** $lt\ p\ adds_t\ u$

and $\bigwedge a. a \in A \implies a \neq 0 \implies lt\ a\ adds_t\ u \implies rword\ (spp-of\ a)\ (spp-of\ p)$

shows *is-canon-rewriter* *rword* *A* *u* *p*

<proof>

lemma *is-canon-rewriterD1*: *is-canon-rewriter* *rword* *A* *u* *p* $\implies p \in A$

<proof>

lemma *is-canon-rewriterD2*: *is-canon-rewriter* *rword* *A* *u* *p* $\implies p \neq 0$

<proof>

lemma *is-canon-rewriterD3*: *is-canon-rewriter* *rword* *A* *u* *p* $\implies lt\ p\ adds_t\ u$

<proof>

lemma *is-canon-rewriterD4*:

is-canon-rewriter *rword* *A* *u* *p* $\implies a \in A \implies a \neq 0 \implies lt\ a\ adds_t\ u \implies rword\ (spp-of\ a)\ (spp-of\ p)$

<proof>

lemmas *is-canon-rewriterD* = *is-canon-rewriterD1* *is-canon-rewriterD2* *is-canon-rewriterD3* *is-canon-rewriterD4*

lemma *is-rewrite-ord-finite-canon-rewriterE*:

assumes *is-rewrite-ord rword* **and** *finite A* **and** $a \in A$ **and** $a \neq 0$ **and** $lt\ a\ adds_t\ u$
obtains p **where** *is-canon-rewriter rword A u p*
<proof>

lemma *is-rewrite-ord-canon-rewriterD1*:

assumes *is-rewrite-ord rword* **and** *is-canon-rewriter rword A u p* **and** *is-canon-rewriter rword A v q*
and $lt\ p\ adds_t\ v$ **and** $lt\ q\ adds_t\ u$
shows $lt\ p = lt\ q$
<proof>

corollary *is-rewrite-ord-canon-rewriterD2*:

assumes *is-rewrite-ord rword* **and** *is-canon-rewriter rword A u p* **and** *is-canon-rewriter rword A u q*
shows $lt\ p = lt\ q$
<proof>

lemma *is-rewrite-ord-canon-rewriterD3*:

assumes *is-rewrite-ord rword* **and** *dickson-grading d* **and** *is-canon-rewriter rword A u p*
and $a \in A$ **and** $a \neq 0$ **and** $lt\ a\ adds_t\ u$ **and** *is-sig-GB-upt d A (lt a)*
and $lt\ p\ adds_t\ lt\ a$ **and** $\neg is-sig-red\ (\prec_t)\ (=)\ A\ a$
shows $lt\ p = lt\ a$
<proof>

lemma *is-RB-inI1*:

assumes *is-canon-rewriter rword G u g* **and** $\neg is-sig-red\ (\prec_t)\ (=)\ G\ (monom-mult\ 1\ (pp-of-term\ u - lp\ g)\ g)$
shows *is-RB-in d rword G u*
<proof>

lemma *is-RB-inI2*:

assumes *is-syz-sig d u*
shows *is-RB-in d rword G u*
<proof>

lemma *is-RB-inE*:

assumes *is-RB-in d rword G u*
and *is-syz-sig d u* $\implies thesis$
and $\bigwedge g. \neg is-syz-sig\ d\ u \implies is-canon-rewriter\ rword\ G\ u\ g \implies$
 $\neg is-sig-red\ (\prec_t)\ (=)\ G\ (monom-mult\ 1\ (pp-of-term\ u - lp\ g)\ g) \implies$
thesis
shows *thesis*
<proof>

lemma *is-RB-inD*:

assumes *dickson-grading* d **and** $G \subseteq \text{dgrad-sig-set } d$ **and** *is-RB-in* d *rword* G u
and $\neg \text{is-syz-sig } d$ u **and** $d(\text{pp-of-term } u) \leq \text{dgrad-max } d$
and *is-canon-rewriter* *rword* G u g
shows *rep-list* $g \neq 0$
 $\langle \text{proof} \rangle$

lemma *is-RB-uptI*:
assumes $G \subseteq \text{dgrad-sig-set } d$
and $\bigwedge v. v \prec_t u \implies d(\text{pp-of-term } v) \leq \text{dgrad-max } d \implies \text{component-of-term } v < \text{length } fs \implies$
is-RB-in d *canon* G v
shows *is-RB-upt* d *canon* G u
 $\langle \text{proof} \rangle$

lemma *is-RB-uptD1*:
assumes *is-RB-upt* d *canon* G u
shows $G \subseteq \text{dgrad-sig-set } d$
 $\langle \text{proof} \rangle$

lemma *is-RB-uptD2*:
assumes *is-RB-upt* d *canon* G u **and** $v \prec_t u$ **and** $d(\text{pp-of-term } v) \leq \text{dgrad-max } d$
and *component-of-term* $v < \text{length } fs$
shows *is-RB-in* d *canon* G v
 $\langle \text{proof} \rangle$

lemma *is-RB-in-UnI*:
assumes *is-RB-in* d *rword* G u **and** $\bigwedge h. h \in H \implies u \prec_t \text{lt } h$
shows *is-RB-in* d *rword* $(H \cup G)$ u
 $\langle \text{proof} \rangle$

corollary *is-RB-in-insertI*:
assumes *is-RB-in* d *rword* G u **and** $u \prec_t \text{lt } g$
shows *is-RB-in* d *rword* $(\text{insert } g$ $G)$ u
 $\langle \text{proof} \rangle$

corollary *is-RB-upt-UnI*:
assumes *is-RB-upt* d *rword* G u **and** $H \subseteq \text{dgrad-sig-set } d$ **and** $\bigwedge h. h \in H \implies$
 $u \preceq_t \text{lt } h$
shows *is-RB-upt* d *rword* $(H \cup G)$ u
 $\langle \text{proof} \rangle$

corollary *is-RB-upt-insertI*:
assumes *is-RB-upt* d *rword* G u **and** $g \in \text{dgrad-sig-set } d$ **and** $u \preceq_t \text{lt } g$
shows *is-RB-upt* d *rword* $(\text{insert } g$ $G)$ u
 $\langle \text{proof} \rangle$

lemma *is-RB-upt-is-sig-GB-upt*:
assumes *dickson-grading* d **and** *is-RB-upt* d *rword* G u

shows *is-sig-GB-upt* $d G u$
 ⟨*proof*⟩

corollary *is-RB-upt-is-syz-sigD*:

assumes *dickson-grading* d **and** *is-RB-upt* d *rword* $G u$
and *is-syz-sig* $d u$ **and** $p \in \text{dgrad-sig-set } d$ **and** $\text{lt } p = u$
shows *sig-red-zero* $(\prec_t) G p$
 ⟨*proof*⟩

4.2.4 S-Pairs

definition *spair* :: $(t \Rightarrow_0 'b) \Rightarrow (t \Rightarrow_0 'b) \Rightarrow (t \Rightarrow_0 'b)$

where *spair* $p q = (\text{let } t1 = \text{punit.lt } (\text{rep-list } p); t2 = \text{punit.lt } (\text{rep-list } q); l = \text{lcs } t1 t2$ in

$$(\text{monom-mult } (1 / \text{punit.lc } (\text{rep-list } p)) (l - t1) p) - \\ (\text{monom-mult } (1 / \text{punit.lc } (\text{rep-list } q)) (l - t2) q)$$

definition *is-regular-spair* :: $(t \Rightarrow_0 'b) \Rightarrow (t \Rightarrow_0 'b) \Rightarrow \text{bool}$

where *is-regular-spair* $p q \longleftrightarrow$
 $(\text{rep-list } p \neq 0 \wedge \text{rep-list } q \neq 0 \wedge$
 $(\text{let } t1 = \text{punit.lt } (\text{rep-list } p); t2 = \text{punit.lt } (\text{rep-list } q); l = \text{lcs } t1$
 $t2$ in

$$(l - t1) \oplus \text{lt } p \neq (l - t2) \oplus \text{lt } q)$$

lemma *rep-list-spair*: $\text{rep-list } (\text{spair } p q) = \text{punit.spoly } (\text{rep-list } p) (\text{rep-list } q)$
 ⟨*proof*⟩

lemma *spair-comm*: $\text{spair } p q = - \text{spair } q p$
 ⟨*proof*⟩

lemma *dgrad-sig-set-closed-spair*:

assumes *dickson-grading* d **and** $p \in \text{dgrad-sig-set } d$ **and** $q \in \text{dgrad-sig-set } d$
shows $\text{spair } p q \in \text{dgrad-sig-set } d$
 ⟨*proof*⟩

lemma *lt-spair*:

assumes $\text{rep-list } p \neq 0$ **and** $\text{punit.lt } (\text{rep-list } p) \oplus \text{lt } q \prec_t \text{punit.lt } (\text{rep-list } q) \oplus \text{lt } p$

shows $\text{lt } (\text{spair } p q) = (\text{lcs } (\text{punit.lt } (\text{rep-list } p)) (\text{punit.lt } (\text{rep-list } q)) - \text{punit.lt } (\text{rep-list } p)) \oplus \text{lt } p$
 ⟨*proof*⟩

lemma *lt-spair'*:

assumes $\text{rep-list } p \neq 0$ **and** $a + \text{punit.lt } (\text{rep-list } p) = b + \text{punit.lt } (\text{rep-list } q)$
and $b \oplus \text{lt } q \prec_t a \oplus \text{lt } p$

shows $\text{lt } (\text{spair } p q) = (a - \text{gcs } a b) \oplus \text{lt } p$
 ⟨*proof*⟩

lemma *lt-rep-list-spair*:

assumes $\text{rep-list } p \neq 0$ **and** $\text{rep-list } q \neq 0$ **and** $\text{rep-list } (\text{spair } p \ q) \neq 0$
and $a + \text{punit.lt } (\text{rep-list } p) = b + \text{punit.lt } (\text{rep-list } q)$
shows $\text{punit.lt } (\text{rep-list } (\text{spair } p \ q)) \prec (a - \text{gcs } a \ b) + \text{punit.lt } (\text{rep-list } p)$
 $\langle \text{proof} \rangle$

lemma *is-regular-spair-sym*: $\text{is-regular-spair } p \ q \implies \text{is-regular-spair } q \ p$
 $\langle \text{proof} \rangle$

lemma *is-regular-spairI*:
assumes $\text{rep-list } p \neq 0$ **and** $\text{rep-list } q \neq 0$
and $\text{punit.lt } (\text{rep-list } q) \oplus \text{lt } p \neq \text{punit.lt } (\text{rep-list } p) \oplus \text{lt } q$
shows $\text{is-regular-spair } p \ q$
 $\langle \text{proof} \rangle$

lemma *is-regular-spairI'*:
assumes $\text{rep-list } p \neq 0$ **and** $\text{rep-list } q \neq 0$
and $a + \text{punit.lt } (\text{rep-list } p) = b + \text{punit.lt } (\text{rep-list } q)$ **and** $a \oplus \text{lt } p \neq b \oplus \text{lt } q$
shows $\text{is-regular-spair } p \ q$
 $\langle \text{proof} \rangle$

lemma *is-regular-spairD1*: $\text{is-regular-spair } p \ q \implies \text{rep-list } p \neq 0$
 $\langle \text{proof} \rangle$

lemma *is-regular-spairD2*: $\text{is-regular-spair } p \ q \implies \text{rep-list } q \neq 0$
 $\langle \text{proof} \rangle$

lemma *is-regular-spairD3*:
fixes $p \ q$
defines $t1 \equiv \text{punit.lt } (\text{rep-list } p)$
defines $t2 \equiv \text{punit.lt } (\text{rep-list } q)$
assumes $\text{is-regular-spair } p \ q$
shows $t2 \oplus \text{lt } p \neq t1 \oplus \text{lt } q$ (**is** *?thesis1*)
and $\text{lt } (\text{monom-mult } (1 / \text{punit.lc } (\text{rep-list } p)) (\text{lcs } t1 \ t2 - t1) \ p) \neq$
 $\text{lt } (\text{monom-mult } (1 / \text{punit.lc } (\text{rep-list } q)) (\text{lcs } t1 \ t2 - t2) \ q)$ (**is** *?l* \neq *?r*)
 $\langle \text{proof} \rangle$

lemma *is-regular-spair-nonzero*: $\text{is-regular-spair } p \ q \implies \text{spair } p \ q \neq 0$
 $\langle \text{proof} \rangle$

lemma *is-regular-spair-lt*:
assumes $\text{is-regular-spair } p \ q$
shows $\text{lt } (\text{spair } p \ q) = \text{ord-term-lin.max}$
 $((\text{lcs } (\text{punit.lt } (\text{rep-list } p)) (\text{punit.lt } (\text{rep-list } q)) - \text{punit.lt } (\text{rep-list } p))$
 $\oplus \text{lt } p)$
 $((\text{lcs } (\text{punit.lt } (\text{rep-list } p)) (\text{punit.lt } (\text{rep-list } q)) - \text{punit.lt } (\text{rep-list } q))$
 $\oplus \text{lt } q)$
 $\langle \text{proof} \rangle$

lemma *is-regular-spair-lt-ge-1*:

assumes *is-regular-spair* p q
shows $lt\ p \preceq_t\ lt\ (spair\ p\ q)$
 $\langle proof \rangle$

corollary *is-regular-spair-lt-ge-2*:

assumes *is-regular-spair* p q
shows $lt\ q \preceq_t\ lt\ (spair\ p\ q)$
 $\langle proof \rangle$

lemma *is-regular-spair-component-lt-cases*:

assumes *is-regular-spair* p q
shows $component-of-term\ (lt\ (spair\ p\ q)) = component-of-term\ (lt\ p) \vee$
 $component-of-term\ (lt\ (spair\ p\ q)) = component-of-term\ (lt\ q)$
 $\langle proof \rangle$

lemma *lemma-9*:

assumes *dickson-grading* d **and** *is-rewrite-ord* $rword$ **and** *is-RB-upt* d $rword$ G
 u
and *inj-on* $lt\ G$ **and** \neg *is-syz-sig* d u **and** *is-canon-rewriter* $rword\ G\ u\ g1$ **and**
 $h \in G$
and *is-sig-red* $(\prec_t) (=) \{h\}$ (*monom-mult* 1 (*pp-of-term* $u - lp\ g1$) $g1$)
and d (*pp-of-term* u) \leq *dgrad-max* d
shows $lcs\ (punit.lt\ (rep-list\ g1))\ (punit.lt\ (rep-list\ h)) - punit.lt\ (rep-list\ g1) =$
 $pp-of-term\ u - lp\ g1$ (**is** *?thesis1*)
and $lcs\ (punit.lt\ (rep-list\ g1))\ (punit.lt\ (rep-list\ h)) - punit.lt\ (rep-list\ h) =$
 $((pp-of-term\ u - lp\ g1) + punit.lt\ (rep-list\ g1)) - punit.lt\ (rep-list\ h)$
(**is** *?thesis2*)
and *is-regular-spair* $g1\ h$ (**is** *?thesis3*)
and $lt\ (spair\ g1\ h) = u$ (**is** *?thesis4*)
 $\langle proof \rangle$

lemma *is-RB-upt-finite*:

assumes *dickson-grading* d **and** *is-rewrite-ord* $rword$ **and** $G \subseteq$ *dgrad-sig-set* d
and *inj-on* $lt\ G$
and *finite* G
and $\bigwedge g1\ g2. g1 \in G \implies g2 \in G \implies is-regular-spair\ g1\ g2 \implies lt\ (spair\ g1\ g2) \prec_t\ u \implies$
 $is-RB-in\ d\ rword\ G\ (lt\ (spair\ g1\ g2))$
and $\bigwedge i. i < length\ fs \implies term-of-pair\ (0, i) \prec_t\ u \implies is-RB-in\ d\ rword\ G$
(*term-of-pair* $(0, i)$)
shows *is-RB-upt* $d\ rword\ G\ u$
 $\langle proof \rangle$

Note that the following lemma actually holds for *all* regularly reducible power-products in *rep-list* p , not just for the leading power-product.

lemma *lemma-11*:

assumes *dickson-grading* d **and** *is-rewrite-ord* $rword$ **and** *is-RB-upt* $d\ rword\ G$
 $(lt\ p)$
and $p \in$ *dgrad-sig-set* d **and** *is-sig-red* $(\prec_t) (=) G\ p$

obtains $u\ g$ **where** $u \prec_t lt\ p$ **and** $d\ (pp\text{-of-term}\ u) \leq dgrad\text{-max}\ d$ **and** *component-of-term* $u < length\ fs$
and $\neg\ is\text{-syz-sig}\ d\ u$ **and** *is-canon-rewriter* $rword\ G\ u\ g$
and $u = (punit.lt\ (rep\text{-list}\ p) - punit.lt\ (rep\text{-list}\ g)) \oplus lt\ g$ **and** *is-sig-red* (\prec_t)
 $(=)\ \{g\}\ p$
 $\langle proof \rangle$

4.2.5 Termination

definition *term-pp-rel* $:: ('t \Rightarrow 't \Rightarrow bool) \Rightarrow ('t \times 'a) \Rightarrow ('t \times 'a) \Rightarrow bool$
where *term-pp-rel* $r\ a\ b \longleftrightarrow r\ (snd\ b \oplus fst\ a)\ (snd\ a \oplus fst\ b)$

definition *canon-term-pp-pair* $:: ('t \times 'a) \Rightarrow bool$
where *canon-term-pp-pair* $a \longleftrightarrow (gcs\ (pp\text{-of-term}\ (fst\ a))\ (snd\ a) = 0)$

definition *cancel-term-pp-pair* $:: ('t \times 'a) \Rightarrow ('t \times 'a)$
where *cancel-term-pp-pair* $a = (fst\ a \ominus (gcs\ (pp\text{-of-term}\ (fst\ a))\ (snd\ a)),\ snd\ a - (gcs\ (pp\text{-of-term}\ (fst\ a))\ (snd\ a)))$

lemma *term-pp-rel-refl*: $reflp\ r \Longrightarrow term\text{-pp-rel}\ r\ a\ a$
 $\langle proof \rangle$

lemma *term-pp-rel-irrefl*: $irreflp\ r \Longrightarrow \neg\ term\text{-pp-rel}\ r\ a\ a$
 $\langle proof \rangle$

lemma *term-pp-rel-sym*: $symp\ r \Longrightarrow term\text{-pp-rel}\ r\ a\ b \Longrightarrow term\text{-pp-rel}\ r\ b\ a$
 $\langle proof \rangle$

lemma *term-pp-rel-trans*:
assumes *ord-term-lin.is-le-rel* r **and** *term-pp-rel* $r\ a\ b$ **and** *term-pp-rel* $r\ b\ c$
shows *term-pp-rel* $r\ a\ c$
 $\langle proof \rangle$

lemma *term-pp-rel-trans-eq-left*:
assumes *ord-term-lin.is-le-rel* r **and** *term-pp-rel* $(=)\ a\ b$ **and** *term-pp-rel* $r\ b\ c$
shows *term-pp-rel* $r\ a\ c$
 $\langle proof \rangle$

lemma *term-pp-rel-trans-eq-right*:
assumes *ord-term-lin.is-le-rel* r **and** *term-pp-rel* $r\ a\ b$ **and** *term-pp-rel* $(=)\ b\ c$
shows *term-pp-rel* $r\ a\ c$
 $\langle proof \rangle$

lemma *canon-term-pp-cancel*: *canon-term-pp-pair* $(cancel\text{-term-pp-pair}\ a)$
 $\langle proof \rangle$

lemma *term-pp-rel-cancel*:
assumes *reflp* r
shows *term-pp-rel* $r\ a\ (cancel\text{-term-pp-pair}\ a)$

<proof>

lemma *canon-term-pp-rel-id:*

assumes *term-pp-rel* (=) *a b* **and** *canon-term-pp-pair a* **and** *canon-term-pp-pair b*

shows $a = b$

<proof>

lemma *min-set-finite:*

fixes $seq :: nat \Rightarrow ('t \Rightarrow_0 'b::field)$

assumes *dickson-grading d* **and** $range\ seq \subseteq dgrad\text{-}sig\text{-}set\ d$ **and** $0 \notin rep\text{-}list\ 'a\ range\ seq$

and $\bigwedge i\ j. i < j \implies lt\ (seq\ i) \prec_t\ lt\ (seq\ j)$

shows $finite\ \{i. \neg (\exists j < i. lt\ (seq\ j)\ adds_t\ lt\ (seq\ i) \wedge$

$punit.lt\ (rep\text{-}list\ (seq\ j))\ adds\ punit.lt\ (rep\text{-}list\ (seq\ i))\}\}$

<proof>

lemma *rb-termination:*

fixes $seq :: nat \Rightarrow ('t \Rightarrow_0 'b::field)$

assumes *dickson-grading d* **and** $range\ seq \subseteq dgrad\text{-}sig\text{-}set\ d$ **and** $0 \notin rep\text{-}list\ 'a\ range\ seq$

and $\bigwedge i\ j. i < j \implies lt\ (seq\ i) \prec_t\ lt\ (seq\ j)$

and $\bigwedge i. \neg is\text{-}sig\text{-}red\ (\prec_t)\ (\preceq)\ (seq\ ' \{0..<i\})\ (seq\ i)$

and $\bigwedge i. (\exists j < length\ fs. lt\ (seq\ i) = lt\ (monomial\ (1::'b)\ (term\text{-}of\text{-}pair\ (0, j))))$

\wedge

$punit.lt\ (rep\text{-}list\ (seq\ i)) \preceq\ punit.lt\ (rep\text{-}list\ (monomial\ 1\ (term\text{-}of\text{-}pair\ (0, j)))) \vee$

$(\exists j\ k. is\text{-}regular\text{-}spair\ (seq\ j)\ (seq\ k) \wedge rep\text{-}list\ (spair\ (seq\ j)\ (seq\ k)) \neq 0 \wedge$

$lt\ (seq\ i) = lt\ (spair\ (seq\ j)\ (seq\ k)) \wedge$

$punit.lt\ (rep\text{-}list\ (seq\ i)) \preceq\ punit.lt\ (rep\text{-}list\ (spair\ (seq\ j)\ (seq\ k))))$

and $\bigwedge i. is\text{-}sig\text{-}GB\text{-}upt\ d\ (seq\ ' \{0..<i\})\ (lt\ (seq\ i))$

shows *thesis*

<proof>

4.2.6 Concrete Rewrite Orders

definition *is-strict-rewrite-ord* :: $((t \times ('a \Rightarrow_0 'b)) \Rightarrow (t \times ('a \Rightarrow_0 'b)) \Rightarrow bool) \Rightarrow bool$

where *is-strict-rewrite-ord rel* $\longleftrightarrow is\text{-}rewrite\text{-}ord\ (\lambda x\ y. \neg rel\ y\ x)$

lemma *is-strict-rewrite-ordI:* $is\text{-}rewrite\text{-}ord\ (\lambda x\ y. \neg rel\ y\ x) \implies is\text{-}strict\text{-}rewrite\text{-}ord\ rel$

<proof>

lemma *is-strict-rewrite-ordD:* $is\text{-}strict\text{-}rewrite\text{-}ord\ rel \implies is\text{-}rewrite\text{-}ord\ (\lambda x\ y. \neg rel\ y\ x)$

<proof>

lemma *is-strict-rewrite-ord-antisym*:

assumes *is-strict-rewrite-ord rel* **and** $\neg \text{rel } x \ y$ **and** $\neg \text{rel } y \ x$

shows $\text{fst } x = \text{fst } y$

<proof>

lemma *is-strict-rewrite-ord-asy*:

assumes *is-strict-rewrite-ord rel* **and** $\text{rel } x \ y$

shows $\neg \text{rel } y \ x$

<proof>

lemma *is-strict-rewrite-ord-irrefl*: *is-strict-rewrite-ord rel* $\implies \neg \text{rel } x \ x$

<proof>

definition *rw-rat* :: $('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b)) \Rightarrow \text{bool}$

where *rw-rat* $p \ q \longleftrightarrow (\text{let } u = \text{punit.lt } (\text{snd } q) \oplus \text{fst } p; v = \text{punit.lt } (\text{snd } p) \oplus \text{fst } q \text{ in}$

$$u \prec_t v \vee (u = v \wedge \text{fst } p \preceq_t \text{fst } q))$$

definition *rw-rat-strict* :: $('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b)) \Rightarrow \text{bool}$

where *rw-rat-strict* $p \ q \longleftrightarrow (\text{let } u = \text{punit.lt } (\text{snd } q) \oplus \text{fst } p; v = \text{punit.lt } (\text{snd } p) \oplus \text{fst } q \text{ in}$

$$u \prec_t v \vee (u = v \wedge \text{fst } p \prec_t \text{fst } q))$$

definition *rw-add* :: $('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b)) \Rightarrow \text{bool}$

where *rw-add* $p \ q \longleftrightarrow (\text{fst } p \preceq_t \text{fst } q)$

definition *rw-add-strict* :: $('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b)) \Rightarrow \text{bool}$

where *rw-add-strict* $p \ q \longleftrightarrow (\text{fst } p \prec_t \text{fst } q)$

lemma *rw-rat-alt*: $\text{rw-rat} = (\lambda p \ q. \neg \text{rw-rat-strict } q \ p)$

<proof>

lemma *rw-rat-is-rewrite-ord*: *is-rewrite-ord* *rw-rat*

<proof>

lemma *rw-rat-strict-is-strict-rewrite-ord*: *is-strict-rewrite-ord* *rw-rat-strict*

<proof>

lemma *rw-add-alt*: $\text{rw-add} = (\lambda p \ q. \neg \text{rw-add-strict } q \ p)$

<proof>

lemma *rw-add-is-rewrite-ord*: *is-rewrite-ord* *rw-add*

<proof>

lemma *rw-add-strict-is-strict-rewrite-ord*: *is-strict-rewrite-ord* *rw-add-strict*

<proof>

4.2.7 Preparations for Sig-Poly-Pairs

context

fixes $dgrad :: 'a \Rightarrow nat$

begin

definition $spp-rel :: ('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow bool$

where $spp-rel\ sp\ r \longleftrightarrow (r \neq 0 \wedge r \in dgrad\text{-sig-set}\ dgrad \wedge lt\ r = fst\ sp \wedge rep\text{-list}\ r = snd\ sp)$

definition $spp-inv :: ('t \times ('a \Rightarrow_0 'b)) \Rightarrow bool$

where $spp-inv\ sp \longleftrightarrow \exists x\ (spp-rel\ sp)$

definition $vec-of :: ('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \Rightarrow_0 'b)$

where $vec-of\ sp = (if\ spp-inv\ sp\ then\ Eps\ (spp-rel\ sp)\ else\ 0)$

lemma $spp-inv\text{-}spp\text{-}of$:

assumes $r \neq 0$ **and** $r \in dgrad\text{-sig-set}\ dgrad$

shows $spp-inv\ (spp-of\ r)$

$\langle proof \rangle$

context

fixes $sp :: 't \times ('a \Rightarrow_0 'b)$

assumes $spl: spp-inv\ sp$

begin

lemma $sig\text{-}poly\text{-}rel\text{-}vec\text{-}of$: $spp-rel\ sp\ (vec-of\ sp)$

$\langle proof \rangle$

lemma $vec-of\text{-}nonzero$: $vec-of\ sp \neq 0$

$\langle proof \rangle$

lemma $lt\text{-}vec\text{-}of$: $lt\ (vec-of\ sp) = fst\ sp$

$\langle proof \rangle$

lemma $rep\text{-}list\text{-}vec\text{-}of$: $rep\text{-}list\ (vec-of\ sp) = snd\ sp$

$\langle proof \rangle$

lemma $spp\text{-}of\text{-}vec\text{-}of$: $spp-of\ (vec-of\ sp) = sp$

$\langle proof \rangle$

end

lemma $map\text{-}spp\text{-}of\text{-}vec\text{-}of$:

assumes $list\text{-}all\ spp\text{-}inv\ sps$

shows $map\ (spp-of \circ vec-of)\ sps = sps$

$\langle proof \rangle$

lemma $vec-of\text{-}dgrad\text{-}sig\text{-}set$: $vec-of\ sp \in dgrad\text{-sig-set}\ dgrad$

$\langle proof \rangle$

lemma *spp-invD-fst*:

assumes *spp-inv sp*

shows $dgrad (pp\text{-of-term } (fst\ sp)) \leq dgrad\text{-max } dgrad$ **and** *component-of-term*
 $(fst\ sp) < length\ fs$

<proof>

lemma *spp-invD-snd*:

assumes *dickson-grading dgrad and spp-inv sp*

shows $snd\ sp \in punit\text{-dgrad-max-set } dgrad$

<proof>

lemma *vec-of-inj*:

assumes *spp-inv sp and vec-of sp = vec-of sp'*

shows $sp = sp'$

<proof>

lemma *spp-inv-alt*: $spp\text{-inv } sp \longleftrightarrow (vec\text{-of } sp \neq 0)$

<proof>

lemma *spp-of-vec-of-spp-of*:

assumes $p \in dgrad\text{-sig-set } dgrad$

shows $spp\text{-of } (vec\text{-of } (spp\text{-of } p)) = spp\text{-of } p$

<proof>

4.2.8 Total Reduction

primrec *find-sig-reducer* :: $('t \times ('a \Rightarrow_0 'b))\ list \Rightarrow 't \Rightarrow 'a \Rightarrow nat \Rightarrow nat\ option$

where

$find\text{-sig-reducer } []\ _ \ _ = None$

$find\text{-sig-reducer } (b \# bs)\ u\ t\ i =$

$(if\ snd\ b \neq 0 \wedge punit.lt\ (snd\ b)\ adds\ t \wedge (t - punit.lt\ (snd\ b)) \oplus fst\ b \prec_t$
 $u\ then\ Some\ i$

$else\ find\text{-sig-reducer } bs\ u\ t\ (Suc\ i))$

lemma *find-sig-reducer-SomeD-aux*:

assumes $find\text{-sig-reducer } bs\ u\ t\ i = Some\ j$

shows $i \leq j$ **and** $j - i < length\ bs$

<proof>

lemma *find-sig-reducer-SomeD'*:

assumes $find\text{-sig-reducer } bs\ u\ t\ i = Some\ j$ **and** $b = bs ! (j - i)$

shows $b \in set\ bs$ **and** $snd\ b \neq 0$ **and** $punit.lt\ (snd\ b)\ adds\ t$ **and** $(t - punit.lt$
 $(snd\ b)) \oplus fst\ b \prec_t\ u$

<proof>

corollary *find-sig-reducer-SomeD*:

assumes $find\text{-sig-reducer } (map\ spp\text{-of } bs)\ u\ t\ 0 = Some\ i$

shows $i < length\ bs$ **and** $rep\text{-list } (bs ! i) \neq 0$ **and** $punit.lt\ (rep\text{-list } (bs ! i))\ adds$

t
and $(t - \text{punit.lt } (\text{rep-list } (bs ! i))) \oplus \text{lt } (bs ! i) \prec_t u$
 $\langle \text{proof} \rangle$

lemma *find-sig-reducer-NoneE*:

assumes *find-sig-reducer* $bs\ u\ t\ i = \text{None}$ **and** $b \in \text{set } bs$
assumes $\text{snd } b = 0 \implies \text{thesis}$ **and** $\text{snd } b \neq 0 \implies \neg \text{punit.lt } (\text{snd } b) \text{ adds } t \implies$
thesis
and $\text{snd } b \neq 0 \implies \text{punit.lt } (\text{snd } b) \text{ adds } t \implies \neg (t - \text{punit.lt } (\text{snd } b)) \oplus \text{fst } b$
 $\prec_t u \implies \text{thesis}$
shows *thesis*
 $\langle \text{proof} \rangle$

lemma *find-sig-reducer-SomeD-red-single*:

assumes $t \in \text{keys } (\text{rep-list } p)$ **and** *find-sig-reducer* $(\text{map } \text{spp-of } bs) (\text{lt } p) t\ 0 =$
Some i
shows *sig-red-single* $(\prec_t) (\preceq) p (p - \text{monom-mult } (\text{lookup } (\text{rep-list } p) t / \text{punit.lc}$
 $(\text{rep-list } (bs ! i)))$
 $(t - \text{punit.lt } (\text{rep-list } (bs ! i))) (bs ! i) (bs ! i) (t - \text{punit.lt } (\text{rep-list } (bs$
 $! i)))$
 $\langle \text{proof} \rangle$

corollary *find-sig-reducer-SomeD-red*:

assumes $t \in \text{keys } (\text{rep-list } p)$ **and** *find-sig-reducer* $(\text{map } \text{spp-of } bs) (\text{lt } p) t\ 0 =$
Some i
shows *sig-red* $(\prec_t) (\preceq) (\text{set } bs) p (p - \text{monom-mult } (\text{lookup } (\text{rep-list } p) t /$
 $\text{punit.lc } (\text{rep-list } (bs ! i)))$
 $(t - \text{punit.lt } (\text{rep-list } (bs ! i))) (bs ! i)$
 $\langle \text{proof} \rangle$

context

fixes $bs :: ('t \Rightarrow_0 'b) \text{ list}$
begin

definition *sig-trd-term* $:: ('a \Rightarrow \text{nat}) \Rightarrow (('a \times ('t \Rightarrow_0 'b)) \times ('a \times ('t \Rightarrow_0 'b)))$
set

where *sig-trd-term* $d = \{(x, y). \text{punit.dgrad-p-set-le } d \{\text{rep-list } (\text{snd } x)\}$
 $(\text{insert } (\text{rep-list } (\text{snd } y)) (\text{rep-list } ' \text{set } bs)) \wedge$
 $\text{fst } x \in \text{keys } (\text{rep-list } (\text{snd } x)) \wedge \text{fst } y \in \text{keys } (\text{rep-list}$
 $(\text{snd } y)) \wedge$
 $\text{fst } x \prec \text{fst } y\}$

lemma *sig-trd-term-wf*:

assumes *dickson-grading* d
shows *wf* $(\text{sig-trd-term } d)$
 $\langle \text{proof} \rangle$

function $(\text{domintros}) \text{sig-trd-aux} :: ('a \times ('t \Rightarrow_0 'b)) \Rightarrow ('t \Rightarrow_0 'b)$ **where**
 $\text{sig-trd-aux } (t, p) =$

```

    (let p' =
      (case find-sig-reducer (map spp-of bs) (lt p) t 0 of
        None => p
        | Some i => p - monom-mult (lookup (rep-list p) t / punit.lc (rep-list (bs !
i)))
          (t - punit.lt (rep-list (bs ! i))) (bs ! i));
      p'' = punit.lower (rep-list p') t in
      if p'' = 0 then p' else sig-trd-aux (punit.lt p'', p')
    )
  <proof>

```

lemma *sig-trd-aux-domI*:
assumes $\text{fst } \text{args0} \in \text{keys } (\text{rep-list } (\text{snd } \text{args0}))$
shows *sig-trd-aux-dom* args0
 <proof>

definition *sig-trd* :: $('t \Rightarrow_0 'b) \Rightarrow ('t \Rightarrow_0 'b)$
where *sig-trd* $p = (\text{if } \text{rep-list } p = 0 \text{ then } p \text{ else } \text{sig-trd-aux } (\text{punit.lt } (\text{rep-list } p), p))$

lemma *sig-trd-aux-red-rtrancl*:
assumes $\text{fst } \text{args0} \in \text{keys } (\text{rep-list } (\text{snd } \text{args0}))$
shows $(\text{sig-red } (\prec_t) (\preceq) (\text{set } \text{bs}))^{**} (\text{snd } \text{args0}) (\text{sig-trd-aux } \text{args0})$
 <proof>

corollary *sig-trd-red-rtrancl*: $(\text{sig-red } (\prec_t) (\preceq) (\text{set } \text{bs}))^{**} p (\text{sig-trd } p)$
 <proof>

lemma *sig-trd-aux-irred*:
assumes $\text{fst } \text{args0} \in \text{keys } (\text{rep-list } (\text{snd } \text{args0}))$
and $\bigwedge b s. b \in \text{set } \text{bs} \implies \text{rep-list } b \neq 0 \implies \text{fst } \text{args0} \prec s + \text{punit.lt } (\text{rep-list } b) \implies$
 $s \oplus \text{lt } b \prec_t \text{lt } (\text{snd } (\text{args0})) \implies \text{lookup } (\text{rep-list } (\text{snd } \text{args0})) (s + \text{punit.lt } (\text{rep-list } b)) = 0$
shows $\neg \text{is-sig-red } (\prec_t) (\preceq) (\text{set } \text{bs}) (\text{sig-trd-aux } \text{args0})$
 <proof>

corollary *sig-trd-irred*: $\neg \text{is-sig-red } (\prec_t) (\preceq) (\text{set } \text{bs}) (\text{sig-trd } p)$
 <proof>

end

context
fixes $\text{bs} :: ('t \times ('a \Rightarrow_0 'b)) \text{ list}$
begin

context
fixes $v :: 't$
begin

fun *sig-trd-spp-body* :: (('a \Rightarrow_0 'b) \times ('a \Rightarrow_0 'b)) \Rightarrow (('a \Rightarrow_0 'b) \times ('a \Rightarrow_0 'b))
where
sig-trd-spp-body (p, r) =
 (case *find-sig-reducer* bs v (punit.lt p) 0 of
 None \Rightarrow (punit.tail p, r + monomial (punit.lc p) (punit.lt p))
 | Some i \Rightarrow let b = snd (bs ! i) in
 (punit.tail p - punit.monom-mult (punit.lc p / punit.lc b) (punit.lt p -
 punit.lt b) (punit.tail b), r))

definition *sig-trd-spp-aux* :: (('a \Rightarrow_0 'b) \times ('a \Rightarrow_0 'b)) \Rightarrow ('a \Rightarrow_0 'b)
where *sig-trd-spp-aux-def* [code del]: *sig-trd-spp-aux* = *tailrec.fun* (λx . *fst* x = 0) *snd sig-trd-spp-body*

lemma *sig-trd-spp-aux-simps* [code]:
sig-trd-spp-aux (p, r) = (if p = 0 then r else *sig-trd-spp-aux* (*sig-trd-spp-body* (p, r)))
 ⟨*proof*⟩

end

fun *sig-trd-spp* :: ('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b)) **where**
sig-trd-spp (v, p) = (v, *sig-trd-spp-aux* v (p, 0))

We define function *sig-trd-spp*, operating on sig-poly-pairs, already here, to have its definition in the right context. Lemmas are proved about it below in Section *Sig-Poly-Pairs*.

end

4.2.9 Koszul Syzygies

A *Koszul syzygy* of the list *fs* of scalar polynomials is a syzygy of the form *fs* ! *i* \odot *monomial* (1::*b*) (*term-of-pair* (0::*a*, *j*)) - *fs* ! *j* \odot *monomial* (1::*b*) (*term-of-pair* (0::*a*, *i*)), for *i* < *j* and *j* < *length fs*.

primrec *Koszul-syz-sigs-aux* :: ('a \Rightarrow_0 'b) list \Rightarrow nat \Rightarrow 't list **where**
Koszul-syz-sigs-aux [] i = [] |
Koszul-syz-sigs-aux (b # bs) i =
 map-idx ($\lambda b'$ j. ord-term-lin.max (*term-of-pair* (punit.lt b, j)) (*term-of-pair* (punit.lt b', i))) bs (Suc i) @
Koszul-syz-sigs-aux bs (Suc i)

definition *Koszul-syz-sigs* :: ('a \Rightarrow_0 'b) list \Rightarrow 't list
where *Koszul-syz-sigs* bs = *filter-min* (*adds_t*) (*Koszul-syz-sigs-aux* bs 0)

fun *new-syz-sigs* :: 't list \Rightarrow ('t \Rightarrow_0 'b) list \Rightarrow (('t \Rightarrow_0 'b) \times ('t \Rightarrow_0 'b)) + nat \Rightarrow 't list
where
new-syz-sigs ss bs (*Inl* (a, b)) = ss |
new-syz-sigs ss bs (*Inr* j) =

(if is-pot-ord then
 filter-min-append (adds_t) ss (filter-min (adds_t) (map (λb. term-of-pair
 (punit.lt (rep-list b), j)) bs))
 else ss)

fun new-syz-sigs-spp :: 't list ⇒ ('t × ('a ⇒₀ 'b)) list ⇒ (('t × ('a ⇒₀ 'b)) × ('t
 × ('a ⇒₀ 'b))) + nat ⇒ 't list

where

new-syz-sigs-spp ss bs (Inl (a, b)) = ss |
 new-syz-sigs-spp ss bs (Inr j) =
 (if is-pot-ord then
 filter-min-append (adds_t) ss (filter-min (adds_t) (map (λb. term-of-pair
 (punit.lt (snd b), j)) bs))
 else ss)

lemma Koszul-syz-sigs-auxI:

assumes $i < j$ **and** $j < \text{length } bs$
shows $\text{ord-term-lin.max } (\text{term-of-pair } (\text{punit.lt } (bs ! i), k + j)) (\text{term-of-pair } (\text{punit.lt } (bs ! j), k + i)) \in$
 $\text{set } (\text{Koszul-syz-sigs-aux } bs k)$
 ⟨proof⟩

lemma Koszul-syz-sigs-auxE:

assumes $v \in \text{set } (\text{Koszul-syz-sigs-aux } bs k)$
obtains $i j$ **where** $i < j$ **and** $j < \text{length } bs$
and $v = \text{ord-term-lin.max } (\text{term-of-pair } (\text{punit.lt } (bs ! i), k + j)) (\text{term-of-pair } (\text{punit.lt } (bs ! j), k + i))$
 ⟨proof⟩

lemma lt-Koszul-syz-comp:

assumes $0 \notin \text{set } fs$ **and** $i < \text{length } fs$
shows $lt ((fs ! i) \odot \text{monomial } 1 (\text{term-of-pair } (0, j))) = \text{term-of-pair } (\text{punit.lt } (fs ! i), j)$
 ⟨proof⟩

lemma Koszul-syz-nonzero-lt:

assumes $\text{rep-list } a \neq 0$ **and** $\text{rep-list } b \neq 0$ **and** $\text{component-of-term } (lt a) < \text{component-of-term } (lt b)$
shows $\text{rep-list } a \odot b - \text{rep-list } b \odot a \neq 0$ (**is** ?p - ?q ≠ 0)
and $lt (\text{rep-list } a \odot b - \text{rep-list } b \odot a) =$
 $\text{ord-term-lin.max } (\text{punit.lt } (\text{rep-list } a) \oplus lt b) (\text{punit.lt } (\text{rep-list } b) \oplus lt a)$
 (**is** - = ?r)
 ⟨proof⟩

lemma Koszul-syz-is-syz: $\text{rep-list } (\text{rep-list } a \odot b - \text{rep-list } b \odot a) = 0$

⟨proof⟩

lemma dgrad-sig-set-closed-Koszul-syz:

assumes $\text{dickson-grading } dgrad$ **and** $a \in dgrad\text{-sig-set } dgrad$ **and** $b \in dgrad\text{-sig-set}$

dgrad
shows $\text{rep-list } a \odot b - \text{rep-list } b \odot a \in \text{dgrad-sig-set } dgrad$
 ⟨proof⟩

corollary *Koszul-syz-is-syz-sig*:

assumes *dickson-grading* *dgrad* **and** $a \in \text{dgrad-sig-set } dgrad$ **and** $b \in \text{dgrad-sig-set } dgrad$

and $\text{rep-list } a \neq 0$ **and** $\text{rep-list } b \neq 0$ **and** $\text{component-of-term } (lt \ a) < \text{component-of-term } (lt \ b)$

shows *is-syz-sig* *dgrad* ($\text{ord-term-lin.max } (\text{punit.lt } (\text{rep-list } a) \oplus lt \ b) (\text{punit.lt } (\text{rep-list } b) \oplus lt \ a)$)

⟨proof⟩

corollary *lt-Koszul-syz-in-Koszul-syz-sigs-aux*:

assumes *distinct* *fs* **and** $0 \notin \text{set } fs$ **and** $i < j$ **and** $j < \text{length } fs$

shows $lt \ ((fs \ ! \ i) \odot \text{monomial } 1 \ (\text{term-of-pair } (0, j)) - (fs \ ! \ j) \odot \text{monomial } 1 \ (\text{term-of-pair } (0, i))) \in$

$\text{set } (\text{Koszul-syz-sigs-aux } fs \ 0)$ **(is** $?l \in ?K$ **)**

⟨proof⟩

corollary *lt-Koszul-syz-in-Koszul-syz-sigs*:

assumes $\neg \text{is-pot-ord}$ **and** *distinct* *fs* **and** $0 \notin \text{set } fs$ **and** $i < j$ **and** $j < \text{length } fs$

obtains *v* **where** $v \in \text{set } (\text{Koszul-syz-sigs } fs)$

and $v \text{ adds}_t \ lt \ ((fs \ ! \ i) \odot \text{monomial } 1 \ (\text{term-of-pair } (0, j)) - (fs \ ! \ j) \odot \text{monomial } 1 \ (\text{term-of-pair } (0, i)))$

⟨proof⟩

lemma *lt-Koszul-syz-init*:

assumes $0 \notin \text{set } fs$ **and** $i < j$ **and** $j < \text{length } fs$

shows $lt \ ((fs \ ! \ i) \odot \text{monomial } 1 \ (\text{term-of-pair } (0, j)) - (fs \ ! \ j) \odot \text{monomial } 1 \ (\text{term-of-pair } (0, i))) =$

$\text{ord-term-lin.max } (\text{term-of-pair } (\text{punit.lt } (fs \ ! \ i), j)) \ (\text{term-of-pair } (\text{punit.lt } (fs \ ! \ j), i))$

(is $lt \ (?p - ?q) = ?r$ **)**

⟨proof⟩

corollary *Koszul-syz-sigs-auxE-lt-Koszul-syz*:

assumes $0 \notin \text{set } fs$ **and** $v \in \text{set } (\text{Koszul-syz-sigs-aux } fs \ 0)$

obtains $i \ j$ **where** $i < j$ **and** $j < \text{length } fs$

and $v = lt \ ((fs \ ! \ i) \odot \text{monomial } 1 \ (\text{term-of-pair } (0, j)) - (fs \ ! \ j) \odot \text{monomial } 1 \ (\text{term-of-pair } (0, i)))$

⟨proof⟩

corollary *Koszul-syz-sigs-is-syz-sig*:

assumes *dickson-grading* *dgrad* **and** *distinct* *fs* **and** $0 \notin \text{set } fs$ **and** $v \in \text{set } (\text{Koszul-syz-sigs } fs)$

shows *is-syz-sig* *dgrad* *v*

⟨proof⟩

lemma *Koszul-syz-sigs-minimal*:

assumes $u \in \text{set } (\text{Koszul-syz-sigs } fs)$ **and** $v \in \text{set } (\text{Koszul-syz-sigs } fs)$ **and** u
adds _{t} v
shows $u = v$
<proof>

lemma *Koszul-syz-sigs-distinct*: *distinct* (*Koszul-syz-sigs* fs)
<proof>

4.2.10 Algorithms

definition *spair-spp* :: $('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b))$
where *spair-spp* $p\ q = (\text{let } t1 = \text{punit.lt } (\text{snd } p); t2 = \text{punit.lt } (\text{snd } q); l = \text{lcs } t1\ t2$ *in*

$$(\text{ord-term-lin.max } ((l - t1) \oplus \text{fst } p) ((l - t2) \oplus \text{fst } q), \\ \text{punit.monom-mult } (1 / \text{punit.lc } (\text{snd } p)) (l - t1) (\text{snd } p) - \\ \text{punit.monom-mult } (1 / \text{punit.lc } (\text{snd } q)) (l - t2) (\text{snd } q)))$$

definition *is-regular-spair-spp* :: $('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b)) \Rightarrow \text{bool}$

where *is-regular-spair-spp* $p\ q \longleftrightarrow$
 $(\text{snd } p \neq 0 \wedge \text{snd } q \neq 0 \wedge \text{punit.lt } (\text{snd } q) \oplus \text{fst } p \neq \text{punit.lt } (\text{snd } p) \oplus \text{fst } q)$

definition *spair-sigs* :: $('t \Rightarrow_0 'b) \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow ('t \times 't)$

where *spair-sigs* $p\ q =$
 $(\text{let } t1 = \text{punit.lt } (\text{rep-list } p); t2 = \text{punit.lt } (\text{rep-list } q); l = \text{lcs } t1\ t2$ *in*
 $((l - t1) \oplus \text{lt } p, (l - t2) \oplus \text{lt } q))$

definition *spair-sigs-spp* :: $('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times 't)$

where *spair-sigs-spp* $p\ q =$
 $(\text{let } t1 = \text{punit.lt } (\text{snd } p); t2 = \text{punit.lt } (\text{snd } q); l = \text{lcs } t1\ t2$ *in*
 $((l - t1) \oplus \text{fst } p, (l - t2) \oplus \text{fst } q))$

fun *poly-of-pair* :: $((('t \Rightarrow_0 'b) \times ('t \Rightarrow_0 'b)) + \text{nat}) \Rightarrow ('t \Rightarrow_0 'b)$

where
poly-of-pair (*Inl* (p, q)) = *spair* $p\ q$ |
poly-of-pair (*Inr* j) = *monomial* 1 (*term-of-pair* $(0, j)$)

fun *spp-of-pair* :: $((('t \times ('a \Rightarrow_0 'b)) \times ('t \times ('a \Rightarrow_0 'b))) + \text{nat}) \Rightarrow ('t \times ('a \Rightarrow_0 'b))$

where
spp-of-pair (*Inl* (p, q)) = *spair-spp* $p\ q$ |
spp-of-pair (*Inr* j) = (*term-of-pair* $(0, j)$, *fs* ! j)

fun *sig-of-pair* :: $((('t \Rightarrow_0 'b) \times ('t \Rightarrow_0 'b)) + \text{nat}) \Rightarrow 't$

where
sig-of-pair (*Inl* (p, q)) = (*let* $(u, v) = \text{spair-sigs } p\ q$ *in* *ord-term-lin.max* $u\ v$) |

sig-of-pair (*Inr j*) = *term-of-pair* (0, *j*)

fun *sig-of-pair-spp* :: (((*t* × (*a* ⇒₀ *b*)) × (*t* × (*a* ⇒₀ *b*))) + *nat*) ⇒ *t*
where
sig-of-pair-spp (*Inl* (*p*, *q*)) = (*let* (*u*, *v*) = *spair-sigs-spp* *p q* *in ord-term-lin.max* *u v*) |
sig-of-pair-spp (*Inr j*) = *term-of-pair* (0, *j*)

definition *pair-ord* :: (((*t* ⇒₀ *b*) × (*t* ⇒₀ *b*)) + *nat*) ⇒ (((*t* ⇒₀ *b*) × (*t* ⇒₀ *b*)) + *nat*) ⇒ *bool*
where *pair-ord* *x y* ⇔ (*sig-of-pair* *x* ≤_{*t*} *sig-of-pair* *y*)

definition *pair-ord-spp* :: (((*t* × (*a* ⇒₀ *b*)) × (*t* × (*a* ⇒₀ *b*))) + *nat*) ⇒ (((*t* × (*a* ⇒₀ *b*)) × (*t* × (*a* ⇒₀ *b*))) + *nat*) ⇒ *bool*
where *pair-ord-spp* *x y* ⇔ (*sig-of-pair-spp* *x* ≤_{*t*} *sig-of-pair-spp* *y*)

primrec *new-spairs* :: (*t* ⇒₀ *b*) *list* ⇒ (*t* ⇒₀ *b*) ⇒ (((*t* ⇒₀ *b*) × (*t* ⇒₀ *b*)) + *nat*) *list* **where**
new-spairs [] *p* = [] |
new-spairs (*b* # *bs*) *p* =
(*if is-regular-spair* *p b* *then insert-wrt pair-ord* (*Inl* (*p*, *b*)) (*new-spairs* *bs p*) *else new-spairs* *bs p*)

primrec *new-spairs-spp* :: (*t* × (*a* ⇒₀ *b*)) *list* ⇒ (*t* × (*a* ⇒₀ *b*)) ⇒ (((*t* × (*a* ⇒₀ *b*)) × (*t* × (*a* ⇒₀ *b*))) + *nat*) *list* **where**
new-spairs-spp [] *p* = [] |
new-spairs-spp (*b* # *bs*) *p* =
(*if is-regular-spair-spp* *p b* *then insert-wrt pair-ord-spp* (*Inl* (*p*, *b*)) (*new-spairs-spp* *bs p*) *else new-spairs-spp* *bs p*)

definition *add-spairs* :: (((*t* ⇒₀ *b*) × (*t* ⇒₀ *b*)) + *nat*) *list* ⇒ (*t* ⇒₀ *b*) *list* ⇒ (*t* ⇒₀ *b*) ⇒
(((*t* ⇒₀ *b*) × (*t* ⇒₀ *b*)) + *nat*) *list*
where *add-spairs* *ps bs p* = *merge-wrt pair-ord* (*new-spairs* *bs p*) *ps*

definition *add-spairs-spp* :: (((*t* × (*a* ⇒₀ *b*)) × (*t* × (*a* ⇒₀ *b*))) + *nat*) *list* ⇒ (*t* × (*a* ⇒₀ *b*)) *list* ⇒ (*t* × (*a* ⇒₀ *b*)) ⇒
(((*t* × (*a* ⇒₀ *b*)) × (*t* × (*a* ⇒₀ *b*))) + *nat*) *list*
where *add-spairs-spp* *ps bs p* = *merge-wrt pair-ord-spp* (*new-spairs-spp* *bs p*) *ps*

lemma *spair-alt-spair-sigs*:
spair *p q* = *monom-mult* (1 / *punit.lc* (*rep-list* *p*)) (*pp-of-term* (*fst* (*spair-sigs* *p q*)) - *lp* *p*) *p* -
monom-mult (1 / *punit.lc* (*rep-list* *q*)) (*pp-of-term* (*snd* (*spair-sigs* *p q*)) - *lp* *q*) *q*
⟨*proof*⟩

lemma *sig-of-spair*:

assumes *is-regular-spair* p q
shows *sig-of-pair* (Inl (p , q)) = lt (*spair* p q)
 \langle *proof* \rangle

lemma *sig-of-spair-commute*: *sig-of-pair* (Inl (p , q)) = *sig-of-pair* (Inl (q , p))
 \langle *proof* \rangle

lemma *in-new-spairsI*:
assumes $b \in set$ bs **and** *is-regular-spair* p b
shows Inl (p , b) $\in set$ (*new-spairs* bs p)
 \langle *proof* \rangle

lemma *in-new-spairsD*:
assumes Inl (a , b) $\in set$ (*new-spairs* bs p)
shows $a = p$ **and** $b \in set$ bs **and** *is-regular-spair* p b
 \langle *proof* \rangle

corollary *in-new-spairs-iff*:
 Inl (p , b) $\in set$ (*new-spairs* bs p) \longleftrightarrow ($b \in set$ $bs \wedge is-regular-spair$ p b)
 \langle *proof* \rangle

lemma *Inr-not-in-new-spairs*: Inr $j \notin set$ (*new-spairs* bs p)
 \langle *proof* \rangle

lemma *sum-prodE*:
assumes $\bigwedge a$ b . $p = Inl$ (a , b) $\implies thesis$ **and** $\bigwedge j$. $p = Inr$ $j \implies thesis$
shows *thesis*
 \langle *proof* \rangle

corollary *in-new-spairsE*:
assumes $q \in set$ (*new-spairs* bs p)
obtains b **where** $b \in set$ bs **and** *is-regular-spair* p b **and** $q = Inl$ (p , b)
 \langle *proof* \rangle

lemma *new-spairs-sorted*: *sorted-wrt pair-ord* (*new-spairs* bs p)
 \langle *proof* \rangle

lemma *sorted-add-spairs*:
assumes *sorted-wrt pair-ord* ps
shows *sorted-wrt pair-ord* (*add-spairs* ps bs p)
 \langle *proof* \rangle

context

fixes *rword-strict* :: ($'t \times ('a \Rightarrow_0 'b)$) \Rightarrow ($'t \times ('a \Rightarrow_0 'b)$) $\Rightarrow bool$ — Must be a *strict* rewrite order.

begin

qualified definition *rword* :: ($'t \times ('a \Rightarrow_0 'b)$) \Rightarrow ($'t \times ('a \Rightarrow_0 'b)$) $\Rightarrow bool$
where *rword* x $y \longleftrightarrow \neg rword-strict$ y x

definition *is-pred-syz* :: 't list \Rightarrow 't \Rightarrow bool
where *is-pred-syz* ss u = ($\exists v \in \text{set ss. } v \text{ adds}_t u$)

definition *is-rewritable* :: ('t \Rightarrow_0 'b) list \Rightarrow ('t \Rightarrow_0 'b) \Rightarrow 't \Rightarrow bool
where *is-rewritable* bs p u = ($\exists b \in \text{set bs. } b \neq 0 \wedge \text{lt } b \text{ adds}_t u \wedge \text{rword-strict (spp-of } p) \text{ (spp-of } b)$)

definition *is-rewritable-spp* :: ('t \times ('a \Rightarrow_0 'b)) list \Rightarrow ('t \times ('a \Rightarrow_0 'b)) \Rightarrow 't \Rightarrow bool
where *is-rewritable-spp* bs p u = ($\exists b \in \text{set bs. } \text{fst } b \text{ adds}_t u \wedge \text{rword-strict } p \text{ } b$)

fun *sig-crit* :: ('t \Rightarrow_0 'b) list \Rightarrow 't list \Rightarrow (((('t \Rightarrow_0 'b) \times ('t \Rightarrow_0 'b)) + nat) \Rightarrow bool
where
sig-crit bs ss (Inl (p, q)) =
 (let (u, v) = *spair-sigs* p q in
is-pred-syz ss u \vee *is-pred-syz* ss v \vee *is-rewritable* bs p u \vee *is-rewritable* bs q v) |
sig-crit bs ss (Inr j) = *is-pred-syz* ss (term-of-pair (0, j))

fun *sig-crit'* :: ('t \Rightarrow_0 'b) list \Rightarrow (((('t \Rightarrow_0 'b) \times ('t \Rightarrow_0 'b)) + nat) \Rightarrow bool
where
sig-crit' bs (Inl (p, q)) =
 (let (u, v) = *spair-sigs* p q in
is-syz-sig dgrad u \vee *is-syz-sig* dgrad v \vee *is-rewritable* bs p u \vee *is-rewritable* bs q v) |
sig-crit' bs (Inr j) = *is-syz-sig* dgrad (term-of-pair (0, j))

fun *sig-crit-spp* :: ('t \times ('a \Rightarrow_0 'b)) list \Rightarrow 't list \Rightarrow (((('t \times ('a \Rightarrow_0 'b)) \times ('t \times ('a \Rightarrow_0 'b))) + nat) \Rightarrow bool
where
sig-crit-spp bs ss (Inl (p, q)) =
 (let (u, v) = *spair-sigs-spp* p q in
is-pred-syz ss u \vee *is-pred-syz* ss v \vee *is-rewritable-spp* bs p u \vee *is-rewritable-spp* bs q v) |
sig-crit-spp bs ss (Inr j) = *is-pred-syz* ss (term-of-pair (0, j))

sig-crit is used in algorithms, *sig-crit'* is only needed for proving.

fun *rb-spp-body* ::
 (((('t \times ('a \Rightarrow_0 'b)) list \times 't list \times (((('t \times ('a \Rightarrow_0 'b)) \times ('t \times ('a \Rightarrow_0 'b))) + nat) list) \times nat) \Rightarrow
 (((('t \times ('a \Rightarrow_0 'b)) list \times 't list \times (((('t \times ('a \Rightarrow_0 'b)) \times ('t \times ('a \Rightarrow_0 'b))) + nat) list) \times nat)
where
rb-spp-body ((bs, ss, []), z) = ((bs, ss, []), z) |
rb-spp-body ((bs, ss, p # ps), z) =
 (let ss' = *new-syz-sigs-spp* ss bs p in
 if *sig-crit-spp* bs ss' p then
 ((bs, ss', ps), z)

else
let $p' = \text{sig-trd-spp } bs \text{ (spp-of-pair } p) \text{ in}$
if $\text{snd } p' = 0$ *then*
 $((bs, \text{fst } p' \# ss', ps), \text{Suc } z)$
else
 $((p' \# bs, ss', \text{add-spairs-spp } ps \text{ } bs \text{ } p'), z)$

definition $rb\text{-spp-aux} ::$

$((('t \times ('a \Rightarrow_0 'b)) \text{ list} \times 't \text{ list} \times (((('t \times ('a \Rightarrow_0 'b)) \times ('t \times ('a \Rightarrow_0 'b))) + \text{nat}) \text{ list}) \times \text{nat}) \Rightarrow$
 $((('t \times ('a \Rightarrow_0 'b)) \text{ list} \times 't \text{ list} \times (((('t \times ('a \Rightarrow_0 'b)) \times ('t \times ('a \Rightarrow_0 'b))) + \text{nat}) \text{ list}) \times \text{nat})$
where $rb\text{-spp-aux-def}$ [code del]: $rb\text{-spp-aux} = \text{tailrec.fun } (\lambda x. \text{snd } (\text{snd } (\text{fst } x)))$
 $= []$ $(\lambda x. x) \text{ } rb\text{-spp-body}$

lemma $rb\text{-spp-aux-Nil}$ [code]: $rb\text{-spp-aux } ((bs, ss, []), z) = ((bs, ss, []), z)$
 $\langle \text{proof} \rangle$

lemma $rb\text{-spp-aux-Cons}$ [code]:

$rb\text{-spp-aux } ((bs, ss, p \# ps), z) = rb\text{-spp-aux } (rb\text{-spp-body } ((bs, ss, p \# ps), z))$
 $\langle \text{proof} \rangle$

The last parameter / return value of $rb\text{-spp-aux}$, z , counts the number of zero-reductions. Below we will prove that this number remains 0 under certain conditions.

context

assumes $rword\text{-is-strict-rewrite-ord}$: $is\text{-strict-rewrite-ord } rword\text{-strict}$

assumes $dgrad$: $dickson\text{-grading } dgrad$

begin

lemma $rword$: $is\text{-rewrite-ord } rword$

$\langle \text{proof} \rangle$

lemma $sig\text{-crit}'\text{-sym}$: $sig\text{-crit}' \text{ } bs \text{ (Inl } (p, q)) \Longrightarrow sig\text{-crit}' \text{ } bs \text{ (Inl } (q, p))$

$\langle \text{proof} \rangle$

lemma $is\text{-rewritable-ConsD}$:

assumes $is\text{-rewritable } (b \# bs) \text{ } p \text{ } u$ **and** $u \prec_t \text{ } lt \text{ } b$

shows $is\text{-rewritable } bs \text{ } p \text{ } u$

$\langle \text{proof} \rangle$

lemma $sig\text{-crit}'\text{-ConsD}$:

assumes $sig\text{-crit}' \text{ } (b \# bs) \text{ } p$ **and** $sig\text{-of-pair } p \prec_t \text{ } lt \text{ } b$

shows $sig\text{-crit}' \text{ } bs \text{ } p$

$\langle \text{proof} \rangle$

definition $rb\text{-aux-inv1} :: ('t \Rightarrow_0 'b) \text{ list} \Rightarrow \text{bool}$

where $rb\text{-aux-inv1 } bs =$

$(\text{set } bs \subseteq dgrad\text{-sig-set } dgrad \wedge 0 \notin \text{rep-list ' } \text{set } bs \wedge$

$$\begin{aligned}
& \text{sorted-wrt } (\lambda x y. \text{lt } y \prec_t \text{lt } x) \text{ } bs \wedge \\
& (\forall i < \text{length } bs. \neg \text{is-sig-red } (\prec_t) (\preceq) (\text{set } (\text{drop } (\text{Suc } i) \text{ } bs)) (bs ! i)) \wedge \\
& (\forall i < \text{length } bs. \\
& ((\exists j < \text{length } fs. \text{lt } (bs ! i) = \text{lt } (\text{monomial } (1::'b) (\text{term-of-pair } (0, j))) \wedge \\
& \quad \text{punit.lt } (\text{rep-list } (bs ! i)) \preceq \text{punit.lt } (\text{rep-list } (\text{monomial } 1 (\text{term-of-pair} \\
& (0, j)))))) \vee \\
& (\exists p \in \text{set } bs. \exists q \in \text{set } bs. \text{is-regular-spair } p \ q \wedge \text{rep-list } (\text{spair } p \ q) \neq 0 \wedge \\
& \quad \text{lt } (bs ! i) = \text{lt } (\text{spair } p \ q) \wedge \text{punit.lt } (\text{rep-list } (bs ! i)) \preceq \text{punit.lt } (\text{rep-list} \\
& (\text{spair } p \ q)))) \wedge \\
& (\forall i < \text{length } bs. \text{is-RB-upt } \text{dgrad } \text{rword } (\text{set } (\text{drop } (\text{Suc } i) \text{ } bs)) (\text{lt } (bs ! \\
& i))))
\end{aligned}$$

fun *rb-aux-inv* :: (('t \Rightarrow_0 'b) list \times 't list \times (('t \Rightarrow_0 'b) \times ('t \Rightarrow_0 'b)) + nat) list \Rightarrow bool

where *rb-aux-inv* (bs, ss, ps) =
(*rb-aux-inv1* bs \wedge
 $(\forall u \in \text{set } ss. \text{is-syz-sig } \text{dgrad } u) \wedge$
 $(\forall p \ q. \text{Inl } (p, q) \in \text{set } ps \longrightarrow (\text{is-regular-spair } p \ q \wedge p \in \text{set } bs \wedge q \in \text{set}$
bs)) \wedge
 $(\forall j. \text{Inr } j \in \text{set } ps \longrightarrow (j < \text{length } fs \wedge (\forall b \in \text{set } bs. \text{lt } b \prec_t \text{term-of-pair}$
(0, j))) \wedge
 $\text{length } (\text{filter } (\lambda q. \text{sig-of-pair } q = \text{term-of-pair } (0, j)) \text{ } ps)$
 $\leq 1)$ \wedge
 $(\text{sorted-wrt } \text{pair-ord } ps) \wedge$
 $(\forall p \in \text{set } ps. (\forall b1 \in \text{set } bs. \forall b2 \in \text{set } bs. \text{is-regular-spair } b1 \ b2 \longrightarrow$
 $\text{sig-of-pair } p \prec_t \text{lt } (\text{spair } b1 \ b2) \longrightarrow (\text{Inl } (b1, b2) \in \text{set } ps \vee$
 $\text{Inl } (b2, b1) \in \text{set } ps)) \wedge$
 $(\forall j < \text{length } fs. \text{sig-of-pair } p \prec_t \text{term-of-pair } (0, j) \longrightarrow \text{Inr } j \in$
set ps)) \wedge
 $(\forall b \in \text{set } bs. \forall p \in \text{set } ps. \text{lt } b \preceq_t \text{sig-of-pair } p) \wedge$
 $(\forall a \in \text{set } bs. \forall b \in \text{set } bs. \text{is-regular-spair } a \ b \longrightarrow \text{Inl } (a, b) \notin \text{set } ps \longrightarrow$
 $\text{Inl } (b, a) \notin \text{set } ps \longrightarrow$
 $\neg \text{is-RB-in } \text{dgrad } \text{rword } (\text{set } bs) (\text{lt } (\text{spair } a \ b)) \longrightarrow$
 $(\exists p \in \text{set } ps. \text{sig-of-pair } p = \text{lt } (\text{spair } a \ b) \wedge \neg \text{sig-crit}' \text{ } bs \ p)) \wedge$
 $(\forall j < \text{length } fs. \text{Inr } j \notin \text{set } ps \longrightarrow (\text{is-RB-in } \text{dgrad } \text{rword } (\text{set } bs)$
 $(\text{term-of-pair } (0, j)) \wedge$
 $\text{rep-list } (\text{monomial } (1::'b) (\text{term-of-pair } (0, j))) \in \text{ideal } (\text{rep-list ' } \text{set}$
bs))))

lemmas [*simp del*] = *rb-aux-inv.simps*

lemma *rb-aux-inv1-D1*: *rb-aux-inv1* bs \Longrightarrow set bs \subseteq *dgrad-sig-set* *dgrad*
⟨*proof*⟩

lemma *rb-aux-inv1-D2*: *rb-aux-inv1* bs \Longrightarrow 0 \notin *rep-list ' set* bs
⟨*proof*⟩

lemma *rb-aux-inv1-D3*: *rb-aux-inv1* bs \Longrightarrow *sorted-wrt* ($\lambda x y. \text{lt } y \prec_t \text{lt } x$) bs
⟨*proof*⟩

lemma *rb-aux-inv1-D4*:

rb-aux-inv1 bs $\implies i < \text{length } bs \implies \neg \text{is-sig-red } (\prec_t) (\preceq) (\text{set } (\text{drop } (\text{Suc } i) bs))$
 $(bs ! i)$
<proof>

lemma *rb-aux-inv1-D5*:

rb-aux-inv1 bs $\implies i < \text{length } bs \implies \text{is-RB-upt dgrad rword } (\text{set } (\text{drop } (\text{Suc } i) bs))$
 $(lt (bs ! i))$
<proof>

lemma *rb-aux-inv1-E*:

assumes *rb-aux-inv1 bs* **and** $i < \text{length } bs$
and $\bigwedge j. j < \text{length } fs \implies lt (bs ! i) = lt (\text{monomial } (1::'b) (\text{term-of-pair } (0, j)))$ \implies
 $\text{punit.lt } (\text{rep-list } (bs ! i)) \preceq \text{punit.lt } (\text{rep-list } (\text{monomial } 1 (\text{term-of-pair } (0, j))))$ $\implies \text{thesis}$
and $\bigwedge p q. p \in \text{set } bs \implies q \in \text{set } bs \implies \text{is-regular-spair } p q \implies \text{rep-list } (\text{spair } p q) \neq 0 \implies$
 $lt (bs ! i) = lt (\text{spair } p q) \implies \text{punit.lt } (\text{rep-list } (bs ! i)) \preceq \text{punit.lt } (\text{rep-list } (\text{spair } p q))$ $\implies \text{thesis}$
shows *thesis*
<proof>

lemmas *rb-aux-inv1-D = rb-aux-inv1-D1 rb-aux-inv1-D2 rb-aux-inv1-D3 rb-aux-inv1-D4 rb-aux-inv1-D5*

lemma *rb-aux-inv1-distinct-lt*:

assumes *rb-aux-inv1 bs*
shows *distinct (map lt bs)*
<proof>

corollary *rb-aux-inv1-lt-inj-on*:

assumes *rb-aux-inv1 bs*
shows *inj-on lt (set bs)*
<proof>

lemma *canon-rewriter-unique*:

assumes *rb-aux-inv1 bs* **and** *is-canon-rewriter rword (set bs) u a*
and *is-canon-rewriter rword (set bs) u b*
shows $a = b$
<proof>

lemma *rb-aux-inv-D1*: *rb-aux-inv (bs, ss, ps) \implies rb-aux-inv1 bs*
<proof>

lemma *rb-aux-inv-D2*: *rb-aux-inv (bs, ss, ps) $\implies u \in \text{set } ss \implies \text{is-syz-sig dgrad } u$*
<proof>

lemma *rb-aux-inv-D3*:

assumes *rb-aux-inv* (*bs*, *ss*, *ps*) **and** $\text{Inl } (p, q) \in \text{set } ps$
shows $p \in \text{set } bs$ **and** $q \in \text{set } bs$ **and** *is-regular-spair* p q
<proof>

lemma *rb-aux-inv-D4*:

assumes *rb-aux-inv* (*bs*, *ss*, *ps*) **and** $\text{Inr } j \in \text{set } ps$
shows $j < \text{length } fs$ **and** $\bigwedge b. b \in \text{set } bs \implies \text{lt } b \prec_t \text{term-of-pair } (0, j)$
and $\text{length } (\text{filter } (\lambda q. \text{sig-of-pair } q = \text{term-of-pair } (0, j)) \text{ } ps) \leq 1$
<proof>

lemma *rb-aux-inv-D5*: *rb-aux-inv* (*bs*, *ss*, *ps*) \implies *sorted-wrt pair-ord* *ps*
<proof>

lemma *rb-aux-inv-D6-1*:

assumes *rb-aux-inv* (*bs*, *ss*, *ps*) **and** $p \in \text{set } ps$ **and** $b1 \in \text{set } bs$ **and** $b2 \in \text{set } bs$
and *is-regular-spair* $b1$ $b2$ **and** *sig-of-pair* $p \prec_t \text{lt } (\text{spair } b1 \text{ } b2)$
obtains $\text{Inl } (b1, b2) \in \text{set } ps \mid \text{Inl } (b2, b1) \in \text{set } ps$
<proof>

lemma *rb-aux-inv-D6-2*:

rb-aux-inv (*bs*, *ss*, *ps*) $\implies p \in \text{set } ps \implies j < \text{length } fs \implies \text{sig-of-pair } p \prec_t$
term-of-pair $(0, j) \implies$
 $\text{Inr } j \in \text{set } ps$
<proof>

lemma *rb-aux-inv-D7*: *rb-aux-inv* (*bs*, *ss*, *ps*) $\implies b \in \text{set } bs \implies p \in \text{set } ps \implies$
 $\text{lt } b \preceq_t \text{sig-of-pair } p$
<proof>

lemma *rb-aux-inv-D8*:

assumes *rb-aux-inv* (*bs*, *ss*, *ps*) **and** $a \in \text{set } bs$ **and** $b \in \text{set } bs$ **and** *is-regular-spair*
 a b
and $\text{Inl } (a, b) \notin \text{set } ps$ **and** $\text{Inl } (b, a) \notin \text{set } ps$ **and** $\neg \text{is-RB-in dgrad rword}$
 $(\text{set } bs) (\text{lt } (\text{spair } a \text{ } b))$
obtains p **where** $p \in \text{set } ps$ **and** *sig-of-pair* $p = \text{lt } (\text{spair } a \text{ } b)$ **and** $\neg \text{sig-crit'}$
 bs p
<proof>

lemma *rb-aux-inv-D9*:

assumes *rb-aux-inv* (*bs*, *ss*, *ps*) **and** $j < \text{length } fs$ **and** $\text{Inr } j \notin \text{set } ps$
shows *is-RB-in dgrad rword* (*set* *bs*) (*term-of-pair* $(0, j)$)
and $\text{rep-list } (\text{monomial } (1::'b) (\text{term-of-pair } (0, j))) \in \text{ideal } (\text{rep-list ' set } bs)$
<proof>

lemma *rb-aux-inv-is-RB-upt*:

assumes *rb-aux-inv* (*bs*, *ss*, *ps*) **and** $\bigwedge p. p \in \text{set } ps \implies u \preceq_t \text{sig-of-pair } p$
shows *is-RB-upt dgrad rword* (*set* *bs*) u

<proof>

lemma *rb-aux-inv-is-RB-upt-Cons*:

assumes *rb-aux-inv* (*bs*, *ss*, *p* # *ps*)

shows *is-RB-upt dgrad rword* (*set bs*) (*sig-of-pair p*)

<proof>

lemma *Inr-in-tailD*:

assumes *rb-aux-inv* (*bs*, *ss*, *p* # *ps*) **and** *Inr j* ∈ *set ps*

shows *sig-of-pair p* ≠ *term-of-pair* (*0*, *j*)

<proof>

lemma *pair-list-aux*:

assumes *rb-aux-inv* (*bs*, *ss*, *ps*) **and** *p* ∈ *set ps*

shows *sig-of-pair p* = *lt* (*poly-of-pair p*) ∧ *poly-of-pair p* ≠ *0* ∧ *poly-of-pair p* ∈ *dgrad-sig-set dgrad*

<proof>

corollary *pair-list-sig-of-pair*:

rb-aux-inv (*bs*, *ss*, *ps*) ⇒ *p* ∈ *set ps* ⇒ *sig-of-pair p* = *lt* (*poly-of-pair p*)

<proof>

corollary *pair-list-nonzero*: *rb-aux-inv* (*bs*, *ss*, *ps*) ⇒ *p* ∈ *set ps* ⇒ *poly-of-pair p* ≠ *0*

<proof>

corollary *pair-list-dgrad-sig-set*:

rb-aux-inv (*bs*, *ss*, *ps*) ⇒ *p* ∈ *set ps* ⇒ *poly-of-pair p* ∈ *dgrad-sig-set dgrad*

<proof>

lemma *is-rewritableI-is-canon-rewriter*:

assumes *rb-aux-inv1 bs* **and** *b* ∈ *set bs* **and** *b* ≠ *0* **and** *lt b adds_t u*

and ¬ *is-canon-rewriter rword* (*set bs*) *u b*

shows *is-rewritable bs b u*

<proof>

lemma *is-rewritableD-is-canon-rewriter*:

assumes *rb-aux-inv1 bs* **and** *is-rewritable bs b u*

shows ¬ *is-canon-rewriter rword* (*set bs*) *u b*

<proof>

lemma *lemma-12*:

assumes *rb-aux-inv* (*bs*, *ss*, *ps*) **and** *is-RB-upt dgrad rword* (*set bs*) *u*

and *dgrad* (*pp-of-term u*) ≤ *dgrad-max dgrad* **and** *is-canon-rewriter rword* (*set bs*) *u a*

and ¬ *is-syz-sig dgrad u* **and** *is-sig-red* (<_t) (=) (*set bs*) (*monom-mult 1* (*pp-of-term u* - *lp a*) *a*)

obtains *p q* **where** *p* ∈ *set bs* **and** *q* ∈ *set bs* **and** *is-regular-spair p q* **and** *lt* (*spair p q*) = *u*

and $\neg \text{sig-crit}' bs (\text{Inl } (p, q))$
 $\langle \text{proof} \rangle$

lemma *is-canon-rewriterI-eq-sig*:
assumes *rb-aux-inv1* *bs* **and** $b \in \text{set } bs$
shows *is-canon-rewriter* *rword* (*set* *bs*) (*lt* *b*) *b*
 $\langle \text{proof} \rangle$

lemma *not-sig-crit*:
assumes *rb-aux-inv* (*bs*, *ss*, $p \# ps$) **and** $\neg \text{sig-crit } bs (\text{new-syz-sigs } ss \text{ } bs \text{ } p)$
and $b \in \text{set } bs$
shows $lt \text{ } b \prec_t \text{ sig-of-pair } p$
 $\langle \text{proof} \rangle$

context
assumes *fs-distinct*: *distinct* *fs*
assumes *fs-nonzero*: $0 \notin \text{set } fs$
begin

lemma *rep-list-monomial'*: *rep-list* (*monomial* 1 (*term-of-pair* (*0*, *j*))) = (*fs* ! *j*)
when $j < \text{length } fs$
 $\langle \text{proof} \rangle$

lemma *new-syz-sigs-is-syz-sig*:
assumes *rb-aux-inv* (*bs*, *ss*, $p \# ps$) **and** $v \in \text{set } (\text{new-syz-sigs } ss \text{ } bs \text{ } p)$
shows *is-syz-sig* *dgrad* *v*
 $\langle \text{proof} \rangle$

lemma *new-syz-sigs-minimal*:
assumes $\bigwedge u' v'. u' \in \text{set } ss \implies v' \in \text{set } ss \implies u' \text{ adds}_t v' \implies u' = v'$
assumes $u \in \text{set } (\text{new-syz-sigs } ss \text{ } bs \text{ } p)$ **and** $v \in \text{set } (\text{new-syz-sigs } ss \text{ } bs \text{ } p)$ **and**
 $u \text{ adds}_t v$
shows $u = v$
 $\langle \text{proof} \rangle$

lemma *new-syz-sigs-distinct*:
assumes *distinct* *ss*
shows *distinct* (*new-syz-sigs* *ss* *bs* *p*)
 $\langle \text{proof} \rangle$

lemma *sig-crit'I-sig-crit*:
assumes *rb-aux-inv* (*bs*, *ss*, $p \# ps$) **and** *sig-crit* *bs* (*new-syz-sigs* *ss* *bs* *p*) *p*
shows *sig-crit'* *bs* *p*
 $\langle \text{proof} \rangle$

lemma *rb-aux-inv-preserved-0*:
assumes *rb-aux-inv* (*bs*, *ss*, $p \# ps$)
and $\bigwedge s. s \in \text{set } ss' \implies \text{is-syz-sig } dgrad \text{ } s$
and $\bigwedge a \ b. a \in \text{set } bs \implies b \in \text{set } bs \implies \text{is-regular-spair } a \ b \implies \text{Inl } (a, b) \notin$

$set\ ps \implies$
 $Inl\ (b, a) \notin set\ ps \implies \neg is-RB-in\ dgrad\ rword\ (set\ bs)\ (lt\ (spair\ a\ b)) \implies$
 $\exists q \in set\ ps.\ sig-of-pair\ q = lt\ (spair\ a\ b) \wedge \neg sig-crit'\ bs\ q$
and $\bigwedge j.\ j < length\ fs \implies p = Inr\ j \implies Inr\ j \notin set\ ps \implies is-RB-in\ dgrad$
 $rword\ (set\ bs)\ (term-of-pair\ (0, j)) \wedge$
 $rep-list\ (monomial\ 1\ (term-of-pair\ (0, j))) \in ideal\ (rep-list\ 'set\ bs)$
shows $rb-aux-inv\ (bs, ss', ps)$
 $\langle proof \rangle$

lemma $rb-aux-inv-preserved-1$:
assumes $rb-aux-inv\ (bs, ss, p \# ps)$ **and** $sig-crit\ bs\ (new-syz-sigs\ ss\ bs\ p)$ p
shows $rb-aux-inv\ (bs, new-syz-sigs\ ss\ bs\ p, ps)$
 $\langle proof \rangle$

lemma $rb-aux-inv-preserved-2$:
assumes $rb-aux-inv\ (bs, ss, p \# ps)$ **and** $rep-list\ (sig-trd\ bs\ (poly-of-pair\ p)) = 0$
shows $rb-aux-inv\ (bs, lt\ (sig-trd\ bs\ (poly-of-pair\ p)) \# new-syz-sigs\ ss\ bs\ p, ps)$
 $\langle proof \rangle$

lemma $rb-aux-inv-preserved-3$:
assumes $rb-aux-inv\ (bs, ss, p \# ps)$ **and** $\neg sig-crit\ bs\ (new-syz-sigs\ ss\ bs\ p)$ p
and $rep-list\ (sig-trd\ bs\ (poly-of-pair\ p)) \neq 0$
shows $rb-aux-inv\ ((sig-trd\ bs\ (poly-of-pair\ p)) \# bs, new-syz-sigs\ ss\ bs\ p,$
 $add-spairs\ ps\ bs\ (sig-trd\ bs\ (poly-of-pair\ p)))$
and $lt\ (sig-trd\ bs\ (poly-of-pair\ p)) \notin lt\ 'set\ bs$
 $\langle proof \rangle$

lemma $rb-aux-inv-init$: $rb-aux-inv\ ([], Koszul-syz-sigs\ fs, map\ Inr\ [0..<length\ fs])$
 $\langle proof \rangle$

corollary $rb-aux-inv-init-fst$:
 $rb-aux-inv\ (fst\ ([], Koszul-syz-sigs\ fs, map\ Inr\ [0..<length\ fs]), z)$
 $\langle proof \rangle$

function (*domintros*) $rb-aux :: (((t \Rightarrow_0 'b)\ list \times 't\ list \times (((t \Rightarrow_0 'b) \times (t \Rightarrow_0 'b)) + nat)\ list) \times nat) \Rightarrow$
 $((t \Rightarrow_0 'b)\ list \times 't\ list \times (((t \Rightarrow_0 'b) \times (t \Rightarrow_0 'b)) + nat)\ list) \times nat)$

where
 $rb-aux\ ((bs, ss, []), z) = ((bs, ss, []), z) \mid$
 $rb-aux\ ((bs, ss, p \# ps), z) =$
 $(let\ ss' = new-syz-sigs\ ss\ bs\ p\ in$
 $if\ sig-crit\ bs\ ss'\ p\ then$
 $rb-aux\ ((bs, ss', ps), z)$
 $else$
 $let\ p' = sig-trd\ bs\ (poly-of-pair\ p)\ in$
 $if\ rep-list\ p' = 0\ then$
 $rb-aux\ ((bs, lt\ p' \# ss', ps), Suc\ z)$
 $else$

$rb\text{-aux} ((p' \# bs, ss', \text{add-pairs } ps \text{ } bs \text{ } p'), z)$
 $\langle \text{proof} \rangle$

definition $rb :: ('t \Rightarrow_0 'b) \text{list} \times \text{nat}$

where $rb = (\text{let } ((bs, -, -), z) = rb\text{-aux} ([], \text{Koszul-syz-sigs } fs, \text{map Inr } [0..<\text{length } fs]), 0) \text{ in } (bs, z)$

rb is only an auxiliary function used for stating some theorems about rewrite bases and their computation in a readable way. Actual computations (of Gröbner bases) are performed by function sig-gb , defined below. The second return value of rb is the number of zero-reductions. It is only needed for proving that under certain assumptions, there are no such zero-reductions.

Termination

qualified definition $rb\text{-aux-term1} \equiv \{(x, y). \exists z. x = z \# y\}$

qualified definition $rb\text{-aux-term2} \equiv \{(x, y). (\text{fst } x, \text{fst } y) \in rb\text{-aux-term1} \vee (\text{fst } x = \text{fst } y \wedge \text{length } (\text{snd } (\text{snd } x)) < \text{length } (\text{snd } (\text{snd } y)))\}$

qualified definition $rb\text{-aux-term} \equiv rb\text{-aux-term2} \cap \{(x, y). rb\text{-aux-inv } x \wedge rb\text{-aux-inv } y\}$

lemma $wfp\text{-on-}rb\text{-aux-term1}$: $wfp\text{-on } (\lambda x y. (x, y) \in rb\text{-aux-term1})$ ($\text{Collect } rb\text{-aux-inv1}$)
 $\langle \text{proof} \rangle$

lemma $wfp\text{-on-}rb\text{-aux-term2}$: $wfp\text{-on } (\lambda x y. (x, y) \in rb\text{-aux-term2})$ ($\text{Collect } rb\text{-aux-inv}$)
 $\langle \text{proof} \rangle$

corollary $wf\text{-}rb\text{-aux-term}$: $wf \text{ } rb\text{-aux-term}$
 $\langle \text{proof} \rangle$

lemma $rb\text{-aux-domI}$:

assumes $rb\text{-aux-inv } (\text{fst } args)$

shows $rb\text{-aux-dom } args$

$\langle \text{proof} \rangle$

Invariant

lemma $rb\text{-aux-inv-invariant}$:

assumes $rb\text{-aux-inv } (\text{fst } args)$

shows $rb\text{-aux-inv } (\text{fst } (rb\text{-aux } args))$

$\langle \text{proof} \rangle$

lemma $rb\text{-aux-inv-last-Nil}$:

assumes $rb\text{-aux-dom } args$

shows $\text{snd } (\text{snd } (\text{fst } (rb\text{-aux } args))) = []$

$\langle \text{proof} \rangle$

corollary $rb\text{-aux-shape}$:

assumes $rb\text{-aux-dom } args$

obtains $bs\ ss\ z$ **where** $rb\text{-aux}\ args = ((bs, ss, []), z)$
 ⟨proof⟩

lemma $rb\text{-aux-is-RB-upt}$:

$is\text{-RB-upt}\ dgrad\ rword\ (set\ (fst\ (fst\ (rb\text{-aux}\ ([],\ Koszul\text{-syz}\text{-sigs}\ fs,\ map\ Inr\ [0..\<length\ fs]),\ z))))\ u$
 ⟨proof⟩

corollary $rb\text{-is-RB-upt}$: $is\text{-RB-upt}\ dgrad\ rword\ (set\ (fst\ rb))\ u$
 ⟨proof⟩

corollary $rb\text{-aux-is-sig-GB-upt}$:

$is\text{-sig-GB-upt}\ dgrad\ (set\ (fst\ (fst\ (rb\text{-aux}\ ([],\ Koszul\text{-syz}\text{-sigs}\ fs,\ map\ Inr\ [0..\<length\ fs]),\ z))))\ u$
 ⟨proof⟩

corollary $rb\text{-aux-is-sig-GB-in}$:

$is\text{-sig-GB-in}\ dgrad\ (set\ (fst\ (fst\ (rb\text{-aux}\ ([],\ Koszul\text{-syz}\text{-sigs}\ fs,\ map\ Inr\ [0..\<length\ fs]),\ z))))\ u$
 ⟨proof⟩

corollary $rb\text{-aux-is-Groebner-basis}$:

assumes $hom\text{-grading}\ dgrad$
shows $punit.is\text{-Groebner-basis}\ (set\ (map\ rep\text{-list}\ (fst\ (fst\ (rb\text{-aux}\ ([],\ Koszul\text{-syz}\text{-sigs}\ fs,\ map\ Inr\ [0..\<length\ fs]),\ z))))\ u$
 ⟨proof⟩

lemma $ideal\text{-rb-aux}$:

$ideal\ (set\ (map\ rep\text{-list}\ (fst\ (fst\ (rb\text{-aux}\ ([],\ Koszul\text{-syz}\text{-sigs}\ fs,\ map\ Inr\ [0..\<length\ fs]),\ z))))\ u) =$
 $ideal\ (set\ fs)\ (is\ ideal\ ?l = ideal\ ?r)$
 ⟨proof⟩

corollary $ideal\text{-rb}$: $ideal\ (rep\text{-list}\ 'set\ (fst\ rb)) = ideal\ (set\ fs)$
 ⟨proof⟩

lemma

shows $dgrad\text{-max-set-closed-rb-aux}$:
 $set\ (map\ rep\text{-list}\ (fst\ (fst\ (rb\text{-aux}\ ([],\ Koszul\text{-syz}\text{-sigs}\ fs,\ map\ Inr\ [0..\<length\ fs]),\ z))))\ u \subseteq$
 $punit\text{-dgrad-max-set}\ dgrad\ (is\ ?thesis1)$
and $rb\text{-aux-nonzero}$:
 $0 \notin set\ (map\ rep\text{-list}\ (fst\ (fst\ (rb\text{-aux}\ ([],\ Koszul\text{-syz}\text{-sigs}\ fs,\ map\ Inr\ [0..\<length\ fs]),\ z))))\ u$
 (is ?thesis2)
 ⟨proof⟩

4.2.11 Minimality of the Computed Basis

lemma *rb-aux-top-irred'*:

assumes *rword-strict* = *rw-rat-strict* **and** *rb-aux-inv* (*bs*, *ss*, *p* # *ps*)
and \neg *sig-crit* *bs* (*new-syz-sigs* *ss* *bs* *p*) *p*
shows \neg *is-sig-red* (\preceq_t) (=) (*set* *bs*) (*sig-trd* *bs* (*poly-of-pair* *p*))
 \langle *proof* \rangle

lemma *rb-aux-top-irred*:

assumes *rword-strict* = *rw-rat-strict* **and** *rb-aux-inv* (*fst* *args*) **and** $b \in \text{set } (\text{fst } (\text{fst } (\text{rb-aux } \text{args})))$
and $\bigwedge b0. b0 \in \text{set } (\text{fst } (\text{fst } \text{args})) \implies \neg \text{is-sig-red } (\preceq_t) (=) (\text{set } (\text{fst } (\text{fst } \text{args})) - \{b0\}) b0$
shows $\neg \text{is-sig-red } (\preceq_t) (=) (\text{set } (\text{fst } (\text{fst } (\text{rb-aux } \text{args}))) - \{b\}) b$
 \langle *proof* \rangle

corollary *rb-aux-is-min-sig-GB*:

assumes *rword-strict* = *rw-rat-strict*
shows *is-min-sig-GB* *dgrad* (*set* (*fst* (*fst* (*rb-aux* ($[\]$, *Koszul-syz-sigs* *fs*, *map* *Inr* $[0..<\text{length } fs]$), *z*))))))
(*is* *is-min-sig-GB* - (*set* (*fst* (*fst* (*rb-aux* ?*args*))))))
 \langle *proof* \rangle

corollary *rb-is-min-sig-GB*:

assumes *rword-strict* = *rw-rat-strict*
shows *is-min-sig-GB* *dgrad* (*set* (*fst* *rb*))
 \langle *proof* \rangle

4.2.12 No Zero-Reductions

fun *rb-aux-inv2* :: ($'t \Rightarrow_0 'b$) *list* \times $'t$ *list* \times ($(('t \Rightarrow_0 'b) \times ('t \Rightarrow_0 'b)) + \text{nat})$ *list*) \Rightarrow *bool*

where *rb-aux-inv2* (*bs*, *ss*, *ps*) =
(*rb-aux-inv* (*bs*, *ss*, *ps*) \wedge
 $(\forall j < \text{length } fs. \text{Inr } j \notin \text{set } ps \longrightarrow$
 $(fs ! j \in \text{ideal } (\text{rep-list } ' \text{set } (\text{filter } (\lambda b. \text{component-of-term } (lt \ b) < \text{Suc } j) \ bs)) \wedge$
 $(\forall b \in \text{set } bs. \text{component-of-term } (lt \ b) < j \longrightarrow$
 $(\exists s \in \text{set } ss. s \text{ adds}_t \text{ term-of-pair } (\text{punit.lt } (\text{rep-list } b), j))))))$

lemma *rb-aux-inv2-D1*: *rb-aux-inv2* *args* \implies *rb-aux-inv* *args*

\langle *proof* \rangle

lemma *rb-aux-inv2-D2*:

rb-aux-inv2 (*bs*, *ss*, *ps*) $\implies j < \text{length } fs \implies \text{Inr } j \notin \text{set } ps \implies$
 $fs ! j \in \text{ideal } (\text{rep-list } ' \text{set } (\text{filter } (\lambda b. \text{component-of-term } (lt \ b) < \text{Suc } j) \ bs))$
 \langle *proof* \rangle

lemma *rb-aux-inv2-E*:

assumes *rb-aux-inv2* (*bs*, *ss*, *ps*) **and** $j < \text{length } fs$ **and** $\text{Inr } j \notin \text{set } ps$ **and** $b \in$

set bs
and *component-of-term (lt b) < j*
obtains *s where s ∈ set ss and s adds_t term-of-pair (punit.lt (rep-list b), j)*
<proof>

context
assumes *pot: is-pot-ord*
begin

lemma *sig-red-zero-filter:*
assumes *sig-red-zero (≼_t) (set bs) r and component-of-term (lt r) < j*
shows *sig-red-zero (≼_t) (set (filter (λb. component-of-term (lt b) < j) bs)) r*
<proof>

lemma *rb-aux-inv2-preserved-0:*
assumes *rb-aux-inv2 (bs, ss, p # ps) and j < length fs and Inr j ∉ set ps*
and *b ∈ set bs and component-of-term (lt b) < j*
shows $\exists s \in \text{set } (new\text{-syz}\text{-sigs } ss \text{ } bs \text{ } p). s \text{ adds}_t \text{ term-of-pair } (punit.lt (rep\text{-list } b), j)$
<proof>

lemma *rb-aux-inv2-preserved-1:*
assumes *rb-aux-inv2 (bs, ss, p # ps) and sig-crit bs (new-syz-sigs ss bs p) p*
shows *rb-aux-inv2 (bs, new-syz-sigs ss bs p, ps)*
<proof>

lemma *rb-aux-inv2-preserved-3:*
assumes *rb-aux-inv2 (bs, ss, p # ps) and ¬ sig-crit bs (new-syz-sigs ss bs p) p*
and *rep-list (sig-trd bs (poly-of-pair p)) ≠ 0*
shows *rb-aux-inv2 (sig-trd bs (poly-of-pair p) # bs, new-syz-sigs ss bs p,*
add-spairs ps bs (sig-trd bs (poly-of-pair p)))
<proof>

lemma *rb-aux-inv2-ideal-subset:*
assumes *rb-aux-inv2 (bs, ss, ps) and $\bigwedge p0. p0 \in \text{set } ps \implies j \leq \text{component-of-term}$*
(sig-of-pair p0)
shows *ideal (set (take j fs)) ⊆ ideal (rep-list ‘ set (filter (λb. component-of-term*
(lt b) < j) bs))
(is ideal ?B ⊆ ideal ?A)
<proof>

lemma *rb-aux-inv-is-Groebner-basis:*
assumes *hom-grading dgrad and rb-aux-inv (bs, ss, ps)*
and $\bigwedge p0. p0 \in \text{set } ps \implies j \leq \text{component-of-term } (sig\text{-of-pair } p0)$
shows *punit.is-Groebner-basis (rep-list ‘ set (filter (λb. component-of-term (lt b)*
< j) bs))
(is punit.is-Groebner-basis (rep-list ‘ set ?bs))
<proof>

lemma *rb-aux-inv2-no-zero-red*:

assumes *hom-grading dgrad* **and** *is-regular-sequence fs* **and** *rb-aux-inv2 (bs, ss, p # ps)*
and \neg *sig-crit bs (new-syz-sigs ss bs p) p*
shows *rep-list (sig-trd bs (poly-of-pair p)) \neq 0*
 \langle *proof* \rangle

corollary *rb-aux-no-zero-red'*:

assumes *hom-grading dgrad* **and** *is-regular-sequence fs* **and** *rb-aux-inv2 (fst args)*
shows *snd (rb-aux args) = snd args*
 \langle *proof* \rangle

corollary *rb-aux-no-zero-red*:

assumes *hom-grading dgrad* **and** *is-regular-sequence fs*
shows *snd (rb-aux ([], Koszul-syz-sigs fs, map Inr [0..*length fs*]), z) = z*
 \langle *proof* \rangle

corollary *rb-no-zero-red*:

assumes *hom-grading dgrad* **and** *is-regular-sequence fs*
shows *snd rb = 0*
 \langle *proof* \rangle

end

4.3 Sig-Poly-Pairs

We now prove that the algorithms defined for sig-poly-pairs (i. e. those whose names end with *-spp*) behave exactly as those defined for module elements. More precisely, if *A* is some algorithm defined for module elements, we prove something like *spp-of (A x) = A-spp (spp-of x)*.

fun *spp-inv-pair* :: $((('t \times ('a \Rightarrow_0 'b)) \times ('t \times ('a \Rightarrow_0 'b))) + nat) \Rightarrow bool$ **where**
spp-inv-pair (Inl (p, q)) = (spp-inv p \wedge spp-inv q) |
spp-inv-pair (Inr j) = True

fun *app-pair* :: $('x \Rightarrow 'y) \Rightarrow (('x \times 'x) + nat) \Rightarrow (('y \times 'y) + nat)$ **where**
app-pair f (Inl (p, q)) = Inl (f p, f q) |
app-pair f (Inr j) = Inr j

fun *app-args* :: $('x \Rightarrow 'y) \Rightarrow (('x \text{ list} \times 'z \times (((('x \times 'x) + nat) \text{ list})) \times nat) \Rightarrow$
 $((('y \text{ list} \times 'z \times (((('y \times 'y) + nat) \text{ list})) \times nat)) \times nat)$ **where**
app-args f ((as, bs, cs), n) = ((map f as, bs, map (app-pair f) cs), n)

lemma *app-pair-spp-of-vec-of*:

assumes *spp-inv-pair p*
shows *app-pair spp-of (app-pair vec-of p) = p*
 \langle *proof* \rangle

lemma *map-app-pair-spp-of-vec-of*:

assumes *list-all spp-inv-pair ps*
shows $\text{map } (\text{app-pair spp-of} \circ \text{app-pair vec-of}) \text{ ps} = \text{ps}$
 ⟨*proof*⟩

lemma *snd-app-args*: $\text{snd } (\text{app-args } f \text{ args}) = \text{snd args}$
 ⟨*proof*⟩

lemma *new-syz-sigs-alt-spp*:
 $\text{new-syz-sigs } ss \text{ bs } p = \text{new-syz-sigs-spp } ss \text{ (map spp-of bs) (app-pair spp-of p)}$
 ⟨*proof*⟩

lemma *is-rewritable-alt-spp*:
assumes $0 \notin \text{set bs}$
shows $\text{is-rewritable } bs \text{ p } u = \text{is-rewritable-spp } (\text{map spp-of bs}) \text{ (spp-of p) } u$
 ⟨*proof*⟩

lemma *spair-sigs-alt-spp*: $\text{spair-sigs } p \text{ q} = \text{spair-sigs-spp } (\text{spp-of p}) \text{ (spp-of q)}$
 ⟨*proof*⟩

lemma *sig-crit-alt-spp*:
assumes $0 \notin \text{set bs}$
shows $\text{sig-crit } bs \text{ ss } p = \text{sig-crit-spp } (\text{map spp-of bs}) \text{ ss } (\text{app-pair spp-of p})$
 ⟨*proof*⟩

lemma *spair-alt-spp*:
assumes *is-regular-spair p q*
shows $\text{spp-of } (\text{spair } p \text{ q}) = \text{spair-spp } (\text{spp-of p}) \text{ (spp-of q)}$
 ⟨*proof*⟩

lemma *sig-trd-spp-body-alt-Some*:
assumes $\text{find-sig-reducer } (\text{map spp-of bs}) \text{ v } (\text{punit.lt } p) \text{ 0} = \text{Some } i$
shows $\text{sig-trd-spp-body } (\text{map spp-of bs}) \text{ v } (p, r) =$
 $(\text{punit.lower } (p - \text{local.punit.monom-mult } (\text{punit.lc } p / \text{punit.lc } (\text{rep-list } (bs ! i))))$
 $(\text{punit.lt } p - \text{punit.lt } (\text{rep-list } (bs ! i))) \text{ (rep-list } (bs ! i))) (\text{punit.lt } p), r)$
 (is *?thesis1*)
and $\text{sig-trd-spp-body } (\text{map spp-of bs}) \text{ v } (p, r) =$
 $(p - \text{local.punit.monom-mult } (\text{punit.lc } p / \text{punit.lc } (\text{rep-list } (bs ! i))))$
 $(\text{punit.lt } p - \text{punit.lt } (\text{rep-list } (bs ! i))) \text{ (rep-list } (bs ! i)), r)$
 (is *?thesis2*)
 ⟨*proof*⟩

lemma *sig-trd-aux-alt-spp*:
assumes $\text{fst args} \in \text{keys } (\text{rep-list } (\text{snd args}))$
shows $\text{rep-list } (\text{sig-trd-aux } bs \text{ args}) =$
 $\text{sig-trd-spp-aux } (\text{map spp-of bs}) \text{ (lt } (\text{snd args}))$
 $(\text{rep-list } (\text{snd args}) - \text{punit.higher } (\text{rep-list } (\text{snd args})) \text{ (fst args)},$
 $\text{punit.higher } (\text{rep-list } (\text{snd args})) \text{ (fst args)})$

$\langle proof \rangle$

lemma *sig-trd-alt-spp*: $spp\text{-of } (sig\text{-trd } bs \ p) = sig\text{-trd-spp } (map \ spp\text{-of } bs) (spp\text{-of } p)$
 $\langle proof \rangle$

lemma *is-regular-spair-alt-spp*: $is\text{-regular-spair } p \ q \longleftrightarrow is\text{-regular-spair-spp } (spp\text{-of } p) (spp\text{-of } q)$
 $\langle proof \rangle$

lemma *sig-of-spair-alt-spp*: $sig\text{-of-pair } p = sig\text{-of-pair-spp } (app\text{-pair } spp\text{-of } p)$
 $\langle proof \rangle$

lemma *pair-ord-alt-spp*: $pair\text{-ord } x \ y \longleftrightarrow pair\text{-ord-spp } (app\text{-pair } spp\text{-of } x) (app\text{-pair } spp\text{-of } y)$
 $\langle proof \rangle$

lemma *new-spairs-alt-spp*:
 $map (app\text{-pair } spp\text{-of}) (new\text{-spairs } bs \ p) = new\text{-spairs-spp } (map \ spp\text{-of } bs) (spp\text{-of } p)$
 $\langle proof \rangle$

lemma *add-spairs-alt-spp*:
assumes $\bigwedge x. x \in set \ bs \implies Inl (spp\text{-of } p, spp\text{-of } x) \notin app\text{-pair } spp\text{-of } \text{' set } ps$
shows $map (app\text{-pair } spp\text{-of}) (add\text{-spairs } ps \ bs \ p) =$
 $add\text{-spairs-spp } (map (app\text{-pair } spp\text{-of}) ps) (map \ spp\text{-of } bs) (spp\text{-of } p)$
 $\langle proof \rangle$

lemma *rb-aux-invD-app-args*:
assumes $rb\text{-aux-inv } (fst (app\text{-args } vec\text{-of } ((bs, ss, ps), z)))$
shows $list\text{-all } spp\text{-inv } bs \ \mathbf{and} \ list\text{-all } spp\text{-inv-pair } ps$
 $\langle proof \rangle$

lemma *app-args-spp-of-vec-of*:
assumes $rb\text{-aux-inv } (fst (app\text{-args } vec\text{-of } args))$
shows $app\text{-args } spp\text{-of } (app\text{-args } vec\text{-of } args) = args$
 $\langle proof \rangle$

lemma *poly-of-pair-alt-spp*:
assumes $distinct \ fs \ \mathbf{and} \ rb\text{-aux-inv } (bs, ss, p \ \# \ ps)$
shows $spp\text{-of } (poly\text{-of-pair } p) = spp\text{-of-pair } (app\text{-pair } spp\text{-of } p)$
 $\langle proof \rangle$

lemma *rb-aux-alt-spp*:
assumes $rb\text{-aux-inv } (fst \ args)$
shows $app\text{-args } spp\text{-of } (rb\text{-aux } args) = rb\text{-spp-aux } (app\text{-args } spp\text{-of } args)$
 $\langle proof \rangle$

corollary *rb-spp-aux-alt*:

$rb\text{-aux}\text{-inv} (fst (app\text{-args} \text{vec-of} \text{args})) \implies$
 $rb\text{-spp}\text{-aux} \text{args} = app\text{-args} \text{spp-of} (rb\text{-aux} (app\text{-args} \text{vec-of} \text{args}))$
 <proof>

corollary $rb\text{-spp}\text{-aux}$:

$hom\text{-grading} \text{dgrad} \implies$
 $punit.is\text{-Groebner}\text{-basis} (set (map \text{snd} (fst (fst (rb\text{-spp}\text{-aux} ([], \text{Koszul}\text{-syz}\text{-sigs} \text{fs}, map \text{Inr} [0..\text{length} \text{fs}], z))))))$
 (is $\implies ?thesis1$)
 $ideal (set (map \text{snd} (fst (fst (rb\text{-spp}\text{-aux} ([], \text{Koszul}\text{-syz}\text{-sigs} \text{fs}, map \text{Inr} [0..\text{length} \text{fs}], z)))))) = ideal (set \text{fs})$
 (is $?thesis2$)
 $set (map \text{snd} (fst (fst (rb\text{-spp}\text{-aux} ([], \text{Koszul}\text{-syz}\text{-sigs} \text{fs}, map \text{Inr} [0..\text{length} \text{fs}], z)))))) \subseteq punit\text{-dgrad}\text{-max}\text{-set} \text{dgrad}$
 (is $?thesis3$)
 $0 \notin set (map \text{snd} (fst (fst (rb\text{-spp}\text{-aux} ([], \text{Koszul}\text{-syz}\text{-sigs} \text{fs}, map \text{Inr} [0..\text{length} \text{fs}], z))))))$
 (is $?thesis4$)
 $hom\text{-grading} \text{dgrad} \implies is\text{-pot}\text{-ord} \implies is\text{-regular}\text{-sequence} \text{fs} \implies$
 $\text{snd} (rb\text{-spp}\text{-aux} ([], \text{Koszul}\text{-syz}\text{-sigs} \text{fs}, map \text{Inr} [0..\text{length} \text{fs}], z)) = z$
 (is $\implies - \implies - \implies ?thesis5$)
 $rword\text{-strict} = rw\text{-rat}\text{-strict} \implies p \in set (fst (fst (rb\text{-spp}\text{-aux} ([], \text{Koszul}\text{-syz}\text{-sigs} \text{fs}, map \text{Inr} [0..\text{length} \text{fs}], z)))) \implies$
 $q \in set (fst (fst (rb\text{-spp}\text{-aux} ([], \text{Koszul}\text{-syz}\text{-sigs} \text{fs}, map \text{Inr} [0..\text{length} \text{fs}], z)))) \implies p \neq q \implies$
 $punit.lt (\text{snd} p) \text{adds} punit.lt (\text{snd} q) \implies punit.lt (\text{snd} p) \oplus fst q \prec_t punit.lt$
 $(\text{snd} q) \oplus fst p$
 <proof>

end

end

end

end

end

definition $gb\text{-sig}\text{-z} ::$

$((t \times ('a \Rightarrow_0 'b)) \Rightarrow (t \times ('a \Rightarrow_0 'b)) \Rightarrow bool) \Rightarrow ('a \Rightarrow_0 'b) \text{list} \Rightarrow ((t \times ('a \Rightarrow_0 'b :: \text{field})) \text{list} \times \text{nat})$
where $gb\text{-sig}\text{-z} \text{rword}\text{-strict} \text{fs0} =$
 $(let \text{fs} = rev (\text{remdups} (rev (\text{removeAll} 0 \text{fs0})));$
 $\text{res} = rb\text{-spp}\text{-aux} \text{fs} \text{rword}\text{-strict} ([], \text{Koszul}\text{-syz}\text{-sigs} \text{fs}, map \text{Inr} [0..\text{length} \text{fs}], 0) \text{in}$
 $(fst (fst \text{res}), \text{snd} \text{res}))$

The second return value of $gb\text{-sig}\text{-z}$ is the total number of zero-reductions.

definition $gb\text{-}sig :: ('t \times ('a \Rightarrow_0 'b)) \Rightarrow ('t \times ('a \Rightarrow_0 'b)) \Rightarrow bool \Rightarrow ('a \Rightarrow_0 'b)$
 $list \Rightarrow ('a \Rightarrow_0 'b)::field) list$

where $gb\text{-}sig\ rword\text{-}strict\ fs0 = map\ snd\ (fst\ (gb\text{-}sig\text{-}z\ rword\text{-}strict\ fs0))$

theorem

assumes $\bigwedge fs. is\text{-}strict\text{-}rewrite\text{-}ord\ fs\ rword\text{-}strict$

shows $gb\text{-}sig\text{-}isGB: punit.is\text{-}Groebner\text{-}basis\ (set\ (gb\text{-}sig\ rword\text{-}strict\ fs))\ (is\ ?thesis1)$

and $gb\text{-}sig\text{-}ideal: ideal\ (set\ (gb\text{-}sig\ rword\text{-}strict\ fs)) = ideal\ (set\ fs)\ (is\ ?thesis2)$

and $dgrad\text{-}p\text{-}set\text{-}closed\text{-}gb\text{-}sig:$

$dickson\text{-}grading\ d \Longrightarrow set\ fs \subseteq punit.dgrad\text{-}p\text{-}set\ d\ m \Longrightarrow set\ (gb\text{-}sig\ rword\text{-}strict\ fs) \subseteq punit.dgrad\text{-}p\text{-}set\ d\ m$

$(is\ - \Longrightarrow - \Longrightarrow ?thesis3)$

and $gb\text{-}sig\text{-}nonzero: 0 \notin set\ (gb\text{-}sig\ rword\text{-}strict\ fs)\ (is\ ?thesis4)$

and $gb\text{-}sig\text{-}no\text{-}zero\text{-}red: is\text{-}pot\text{-}ord \Longrightarrow is\text{-}regular\text{-}sequence\ fs \Longrightarrow snd\ (gb\text{-}sig\text{-}z\ rword\text{-}strict\ fs) = 0$

$\langle proof \rangle$

theorem $gb\text{-}sig\text{-}z\text{-}is\text{-}min\text{-}sig\text{-}GB:$

assumes $p \in set\ (fst\ (gb\text{-}sig\text{-}z\ rw\text{-}rat\text{-}strict\ fs))$ **and** $q \in set\ (fst\ (gb\text{-}sig\text{-}z\ rw\text{-}rat\text{-}strict\ fs))$

and $p \neq q$ **and** $punit.lt\ (snd\ p)\ adds\ punit.lt\ (snd\ q)$

shows $punit.lt\ (snd\ p) \oplus fst\ q \prec_t punit.lt\ (snd\ q) \oplus fst\ p$

$\langle proof \rangle$

Summarizing, these are the four main results proved in this theory:

- $(\bigwedge fs. is\text{-}strict\text{-}rewrite\text{-}ord\ fs\ ?rword\text{-}strict) \Longrightarrow punit.is\text{-}Groebner\text{-}basis\ (set\ (gb\text{-}sig\ ?rword\text{-}strict\ ?fs)),$
- $(\bigwedge fs. is\text{-}strict\text{-}rewrite\text{-}ord\ fs\ ?rword\text{-}strict) \Longrightarrow ideal\ (set\ (gb\text{-}sig\ ?rword\text{-}strict\ ?fs)) = ideal\ (set\ ?fs),$
- $\llbracket \bigwedge fs. is\text{-}strict\text{-}rewrite\text{-}ord\ fs\ ?rword\text{-}strict; is\text{-}pot\text{-}ord; is\text{-}regular\text{-}sequence\ ?fs \rrbracket \Longrightarrow snd\ (gb\text{-}sig\text{-}z\ ?rword\text{-}strict\ ?fs) = 0,$ and
- $\llbracket ?p \in set\ (fst\ (gb\text{-}sig\text{-}z\ rw\text{-}rat\text{-}strict\ ?fs)); ?q \in set\ (fst\ (gb\text{-}sig\text{-}z\ rw\text{-}rat\text{-}strict\ ?fs)); ?p \neq ?q; punit.lt\ (snd\ ?p)\ adds\ punit.lt\ (snd\ ?q) \rrbracket \Longrightarrow punit.lt\ (snd\ ?p) \oplus fst\ ?q \prec_t punit.lt\ (snd\ ?q) \oplus fst\ ?p.$

end

end

5 Sample Computations with Signature-Based Algorithms

theory *Signature-Examples*

imports *Signature-Groebner Groebner-Bases.Benchmarks Groebner-Bases.Code-Target-Rat*
begin

5.1 Setup

lift-definition *except-pp* :: ('a, 'b) pp ⇒ 'a set ⇒ ('a, 'b::zero) pp is except ⟨proof⟩

lemma *hom-grading-varnum-pp*: hom-grading (varnum-pp::('a::countable, 'b::add-wellorder) pp ⇒ nat)
⟨proof⟩

instance pp :: (countable, add-wellorder) quasi-pm-powerprod
⟨proof⟩

5.1.1 Projections of Term Orders to Orders on Power-Products

definition *proj-comp* :: (('a::nat, 'b::nat) pp × nat) nat-term-order ⇒ ('a, 'b) pp
⇒ ('a, 'b) pp ⇒ order
where *proj-comp cmp* = (λx y. nat-term-compare cmp (x, 0) (y, 0))

definition *proj-ord* :: (('a::nat, 'b::nat) pp × nat) nat-term-order ⇒ ('a, 'b) pp
nat-term-order
where *proj-ord cmp* = Abs-nat-term-order (proj-comp cmp)

In principle, *proj-comp* and *proj-ord* could be defined more generally on type $'a \times \text{nat}$, but then $'a$ would have to belong to some new type-class which is the intersection of *nat-pp-term* and *nat-pp-compare* and additionally requires *rep-nat-term* $x = (\text{rep-nat-pp } x, 0)$.

lemma *comparator-proj-comp*: comparator (proj-comp cmp)
⟨proof⟩

lemma *nat-term-comp-proj-comp*: nat-term-comp (proj-comp cmp)
⟨proof⟩

corollary *nat-term-compare-proj-ord*: nat-term-compare (proj-ord cmp) = proj-comp
cmp
⟨proof⟩

lemma *proj-ord-LEX* [code]: proj-ord LEX = LEX
⟨proof⟩

lemma *proj-ord-DRLEX* [code]: proj-ord DRLEX = DRLEX
⟨proof⟩

lemma *proj-ord-DEG* [code]: proj-ord (DEG to) = DEG (proj-ord to)
⟨proof⟩

lemma *proj-ord-POT* [code]: proj-ord (POT to) = proj-ord to
⟨proof⟩

5.1.2 Locale Interpretation

locale *qpm-nat-inf-term* = gd-nat-term λx. x λx. x to

for $to :: ('a :: nat, 'b :: nat) pp \times nat$ *nat-term-order*
begin

sublocale *aux*: *qpm-inf-term* $\lambda x. x \lambda x. x$
le-of-nat-term-order (*proj-ord to*)
lt-of-nat-term-order (*proj-ord to*)
le-of-nat-term-order to
lt-of-nat-term-order to
 $\langle proof \rangle$

end

We must define the following two constants outside the global interpretation, since otherwise their types are too general.

definition *splus-pprod* :: $('a :: nat, 'b :: nat) pp \Rightarrow -$
where *splus-pprod* = *pprod.splus*

definition *adds-term-pprod* :: $(('a :: nat, 'b :: nat) pp \times -) \Rightarrow -$
where *adds-term-pprod* = *pprod.adds-term*

global-interpretation *pprod'*: *qpm-nat-inf-term to*
rewrites *pprod.pp-of-term* = *fst*
and *pprod.component-of-term* = *snd*
and *pprod.splus* = *splus-pprod*
and *pprod.adds-term* = *adds-term-pprod*
and *punit.monom-mult* = *monom-mult-punit*
and *pprod'.aux.punit.lt* = *lt-punit (proj-ord to)*
and *pprod'.aux.punit.lc* = *lc-punit (proj-ord to)*
and *pprod'.aux.punit.tail* = *tail-punit (proj-ord to)*
for $to :: (('a :: nat, 'b :: nat) pp \times nat)$ *nat-term-order*
defines *max-pprod* = *pprod'.ord-term-lin.max*
and *Koszul-syz-sigs-aux-pprod* = *pprod'.aux.Koszul-syz-sigs-aux*
and *Koszul-syz-sigs-pprod* = *pprod'.aux.Koszul-syz-sigs*
and *find-sig-reducer-pprod* = *pprod'.aux.find-sig-reducer*
and *sig-trd-spp-body-pprod* = *pprod'.aux.sig-trd-spp-body*
and *sig-trd-spp-aux-pprod* = *pprod'.aux.sig-trd-spp-aux*
and *sig-trd-spp-pprod* = *pprod'.aux.sig-trd-spp*
and *spair-sigs-spp-pprod* = *pprod'.aux.spair-sigs-spp*
and *is-pred-syz-pprod* = *pprod'.aux.is-pred-syz*
and *is-rewritable-spp-pprod* = *pprod'.aux.is-rewritable-spp*
and *sig-crit-spp-pprod* = *pprod'.aux.sig-crit-spp*
and *spair-spp-pprod* = *pprod'.aux.spair-spp*
and *spp-of-pair-pprod* = *pprod'.aux.spp-of-pair*
and *pair-ord-spp-pprod* = *pprod'.aux.pair-ord-spp*
and *sig-of-pair-spp-pprod* = *pprod'.aux.sig-of-pair-spp*
and *new-spairs-spp-pprod* = *pprod'.aux.new-spairs-spp*
and *is-regular-spair-spp-pprod* = *pprod'.aux.is-regular-spair-spp*
and *add-spairs-spp-pprod* = *pprod'.aux.add-spairs-spp*
and *is-pot-ord-pprod* = *pprod'.is-pot-ord*

and *new-syz-sigs-spp-pprod* = *pprod'.aux.new-syz-sigs-spp*
and *rb-spp-body-pprod* = *pprod'.aux.rb-spp-body*
and *rb-spp-aux-pprod* = *pprod'.aux.rb-spp-aux*
and *gb-sig-z-pprod'* = *pprod'.aux.gb-sig-z*
and *gb-sig-pprod'* = *pprod'.aux.gb-sig*
and *rw-rat-strict-pprod* = *pprod'.aux.rw-rat-strict*
and *rw-add-strict-pprod* = *pprod'.aux.rw-add-strict*
 ⟨*proof*⟩

5.1.3 More Lemmas and Definitions

lemma *compute-adds-term-pprod* [*code*]:
adds-term-pprod *u v* = (*snd u* = *snd v* ∧ *adds-pp-add-linorder* (*fst u*) (*fst v*))
 ⟨*proof*⟩

lemma *compute-splus-pprod* [*code*]: *splus-pprod* *t (s, i)* = (*t + s, i*)
 ⟨*proof*⟩

lemma *compute-sig-trd-spp-body-pprod* [*code*]:
sig-trd-spp-body-pprod *to bs v (p, r)* =
 (case *find-sig-reducer-pprod* *to bs v (lt-punit (proj-ord to) p) 0* of
 None ⇒ (*tail-punit (proj-ord to) p*, *plus-monomial-less r (lc-punit (proj-ord to) p) (lt-punit (proj-ord to) p)*)
 | *Some i* ⇒ let *b = snd (bs ! i)* in
 (*tail-punit (proj-ord to) p - monom-mult-punit (lc-punit (proj-ord to) p / lc-punit (proj-ord to) b)*
 (*lt-punit (proj-ord to) p - lt-punit (proj-ord to) b) (tail-punit (proj-ord to) b), r*))
 ⟨*proof*⟩

lemma *compute-sig-trd-spp-pprod* [*code*]:
sig-trd-spp-pprod *to bs (v, p)* ≡ (*v*, *sig-trd-spp-aux-pprod* *to bs v (p, change-ord (proj-ord to) 0)*)
 ⟨*proof*⟩

lemmas [*code*] = *conversep-iff*

lemma *compute-is-pot-ord* [*code*]:
is-pot-ord-pprod (*LEX::('a::nat, 'b::nat) pp × nat*) *nat-term-order*) = *False*
 (is *is-pot-ord-pprod* ?*lex* = -)
is-pot-ord-pprod (*DRLEX::('a::nat, 'b::nat) pp × nat*) *nat-term-order*) = *False*
 (is *is-pot-ord-pprod* ?*drlex* = -)
is-pot-ord-pprod (*DEG (to::('a::nat, 'b::nat) pp × nat) nat-term-order*) = *False*
is-pot-ord-pprod (*POT (to::('a::nat, 'b::nat) pp × nat) nat-term-order*) = *True*
 ⟨*proof*⟩

corollary *is-pot-ord-POT*: *is-pot-ord-pprod* (*POT to*)
 ⟨*proof*⟩

definition $gb\text{-}sig\text{-}z\text{-}pprod\ to\ rword\text{-}strict\ fs \equiv$
 $(let\ res = gb\text{-}sig\text{-}z\text{-}pprod'\ to\ (rword\text{-}strict\ to)\ (map\ (change\text{-}ord$
 $(proj\text{-}ord\ to))\ fs)\ in$
 $(length\ (fst\ res),\ snd\ res))$

definition $gb\text{-}sig\text{-}pprod\ to\ rword\text{-}strict\ fs \equiv gb\text{-}sig\text{-}pprod'\ to\ (rword\text{-}strict\ to)\ (map$
 $(change\text{-}ord\ (proj\text{-}ord\ to))\ fs)$

lemma $snd\text{-}gb\text{-}sig\text{-}z\text{-}pprod'\text{-}eq\text{-}gb\text{-}sig\text{-}z\text{-}pprod:$
 $snd\ (gb\text{-}sig\text{-}z\text{-}pprod'\ to\ (rword\text{-}strict\ to)\ fs) = snd\ (gb\text{-}sig\text{-}z\text{-}pprod\ to\ rword\text{-}strict$
 $fs)$
 $\langle proof \rangle$

lemma $gb\text{-}sig\text{-}pprod'\text{-}eq\text{-}gb\text{-}sig\text{-}pprod:$
 $gb\text{-}sig\text{-}pprod'\ to\ (rword\text{-}strict\ to)\ fs = gb\text{-}sig\text{-}pprod\ to\ rword\text{-}strict\ fs$
 $\langle proof \rangle$

thm $pprod'.aux.gb\text{-}sig\text{-}isGB[OF\ pprod'.aux.rw\text{-}rat\text{-}strict\text{-}is\text{-}strict\text{-}rewrite\text{-}ord,\ sim$
 $plified\ gb\text{-}sig\text{-}pprod'\text{-}eq\text{-}gb\text{-}sig\text{-}pprod]$

thm $pprod'.aux.gb\text{-}sig\text{-}no\text{-}zero\text{-}red[OF\ pprod'.aux.rw\text{-}rat\text{-}strict\text{-}is\text{-}strict\text{-}rewrite\text{-}ord$
 $is\text{-}pot\text{-}ord\text{-}POT,\ simplified\ snd\text{-}gb\text{-}sig\text{-}z\text{-}pprod'\text{-}eq\text{-}gb\text{-}sig\text{-}z\text{-}pprod]$

5.2 Computations

experiment begin interpretation $trivariate_0\text{-}rat\ \langle proof \rangle$

lemma
 $gb\text{-}sig\text{-}pprod\ DRLEX\ rw\text{-}rat\text{-}strict\text{-}pprod\ [X^2 * Z^3 + 3 * X^2 * Y,\ X * Y * Z$
 $+ 2 * Y^2] =$
 $[C_0\ (3 / 4) * X^3 * Y^2 - 2 * Y^4,\ -4 * Y^3 * Z - 3 * X^2 * Y^2,\ X$
 $* Y * Z + 2 * Y^2,\ X^2 * Z^3 + 3 * X^2 * Y]$
 $\langle proof \rangle$

end

Recall that the first return value of $gb\text{-}sig\text{-}z\text{-}pprod$ is the size of the computed Gröbner basis, and the second return value is the total number of useless zero-reductions:

lemma
 $gb\text{-}sig\text{-}z\text{-}pprod\ (POT\ DRLEX)\ rw\text{-}rat\text{-}strict\text{-}pprod\ ((cyclic\ DRLEX\ 6)::(- \Rightarrow_0\ rat)$
 $list) = (155,\ 8)$
 $\langle proof \rangle$

lemma
 $gb\text{-}sig\text{-}z\text{-}pprod\ (POT\ DRLEX)\ rw\text{-}rat\text{-}strict\text{-}pprod\ ((katsura\ DRLEX\ 5)::(- \Rightarrow_0$
 $rat)\ list) = (29,\ 0)$
 $\langle proof \rangle$

lemma


```
gb-sig-z-pprod (POT DRLEX) rw-rat-strict-pprod ((eco DRLEX 8)::(- =>0 rat)
list) = (76, 0)
⟨proof⟩
```

lemma

```
gb-sig-z-pprod (POT DRLEX) rw-rat-strict-pprod ((noon DRLEX 5)::(- =>0 rat)
list) = (83, 0)
⟨proof⟩
```

end

References

- [1] C. Eder and J.-C. Faugère. A Survey on Signature-Based Algorithms for Computing Gröbner Bases. *J. Symb. Comput.*, 80(3):719–784, 2017.
- [2] C. Eder and B. H. Roune. Signature Rewriting in Gröbner Basis Computation. In *Proceedings of ISSAC'13*, pages 331–338. ACM, 2013.
- [3] J.-C. Faugère. A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F_5). In T. Mora, editor, *Proceedings of ISSAC'02*, pages 61–88. ACM, 2002.
- [4] F. Immler and A. Maletzky. Gröbner Bases Theory. *Archive of Formal Proofs*, 2016. http://isa-afp.org/entries/Groebner_Bases.html, Formal proof development.
- [5] B. H. Roune and M. Stillman. Practical Gröbner Basis Computation. In *Proceedings of ISSAC'12*, pages 203–210. ACM, 2012.