# 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 filter ..... 5
2.1.3 drop ..... 7
2.1.4 count-list ..... 8
2.1.5 sorted-wrt ..... 8
2.1.6 insort-wrt and merge-wrt ..... 9
2.2 Recursive Functions ..... 10
2.3 Binary Relations ..... 13

[^0]3 More Properties of Power-Products and Multivariate Poly- nomials ..... 15
3.1 Power-Products ..... 15
3.2 Miscellaneous ..... 16
3.3 ordered-term.lt and ordered-term.higher ..... 18
3.4 gd-term.dgrad-p-set ..... 19
3.5 Regular Sequences ..... 19
4 Signature-Based Algorithms for Computing Gröbner Bases ..... 22
4.1 More Preliminaries ..... 22
4.2 Module Polynomials ..... 23
4.2.1 Signature Reduction ..... 35
4.2.2 Signature Gröbner Bases ..... 57
4.2.3 Rewrite Bases ..... 77
4.2.4 S-Pairs ..... 88
4.2.5 Termination ..... 105
4.2.6 Concrete Rewrite Orders ..... 117
4.2.7 Preparations for Sig-Poly-Pairs ..... 120
4.2.8 Total Reduction ..... 123
4.2.9 Koszul Syzygies ..... 137
4.2.10 Algorithms ..... 144
4.2.11 Minimality of the Computed Basis ..... 199
4.2.12 No Zero-Reductions ..... 206
4.3 Sig-Poly-Pairs ..... 218
5 Sample Computations with Signature-Based Algorithms ..... 235
5.1 Setup ..... 235
5.1.1 Projections of Term Orders to Orders on Power-Products2 ..... 236
5.1.2 Locale Interpretation ..... 238
5.1.3 More Lemmas and Definitions ..... 240
5.2 Computations ..... 243

## 1 Introduction

Signature-based algorithms [3, 1] play are 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. gbsig) 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 \bigwedgei. (\existsx. seq (Suc i)=x # seq i) and i<j
    shows length (seq i) < length (seq j)
proof -
```

```
    from assms(2) obtain \(k\) where \(j=\) Suc \((i+k)\) using less-iff-Suc-add by auto
    show ?thesis unfolding \(\langle j=\operatorname{Suc}(i+k)\) 〉
    proof (induct \(k\) )
        case 0
        from \(\operatorname{assms}(1)\) obtain \(x\) where eq: seq \((S u c i)=x \#\) seq \(i\)..
        show ? case by (simp add: eq)
    next
    case (Suc k)
    from \(\operatorname{assms}(1)\) obtain \(x\) where seq \((S u c(i+S u c k))=x \#\) seq \((i+\) Suc \(k)\)
    hence eq: seq \((S u c(S u c(i+k)))=x \#\) seq \((S u c(i+k))\) by simp
    note Suc
    also have length \((\operatorname{seq}(S u c(i+k)))<\) length \((\operatorname{seq}(S u c(i+S u c k)))\) by \((\operatorname{simp}\)
add: eq)
    finally show ? case .
    qed
qed
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 (cases \(i=j\) )
    case True
    thus ?thesis by simp
next
    case False
    with \(\operatorname{assms}\) (2) have \(i<j\) by simp
    with \(\operatorname{assms}(1)\) have length (seq i) < length (seq j) by (rule list-seq-length-mono)
    thus ?thesis by simp
qed
lemma list-seq-indexE-length:
    fixes seq :: nat \(\Rightarrow\) 'a list
    assumes \(\bigwedge i\). \((\exists x . \operatorname{seq}(\) Suc \(i)=x \#\) seq \(i)\)
    obtains \(j\) where \(i<\) length (seq \(j\) )
proof (induct \(i\) arbitrary: thesis)
    case 0
    have \(0 \leq\) length (seq 0) by \(\operatorname{simp}\)
    also from assms lessI have \(\ldots<\) length (seq (Suc 0)) by (rule list-seq-length-mono)
    finally show? case by (rule 0)
next
    case (Suc \(i\) )
    obtain \(j\) where \(i<\) length (seq \(j\) ) by (rule Suc(1))
    hence Suc \(i \leq\) length (seq \(j\) ) by simp
    also from assms lessI have \(\ldots<\) length (seq (Suc j)) by (rule list-seq-length-mono)
    finally show? case by (rule Suc(2))
qed
```

```
lemma list-seq-nth:
    fixes seq :: nat => 'a list
    assumes }\i.(\existsx.seq (Suc i)=x# seq i) and i<length (seq j) and j\leq
    shows rev (seq k)!i=rev (seq j)!i
proof -
    from assms(3) obtain l where k=j+l using nat-le-iff-add by blast
    show ?thesis unfolding <k=j+l>
    proof (induct l)
        case 0
        show ?case by simp
    next
        case (Suc l)
        note assms(2)
        also from assms(1) le-add1 have length (seq j)\leqlength (seq ( j + l))
            by (rule list-seq-length-mono-weak)
        finally have i:i<length (seq ( j+l)).
        from assms(1) obtain x where seq (Suc (j+l)) = x # seq (j+l)..
        thus ?case by (simp add: nth-append i Suc)
    qed
qed
corollary list-seq-nth':
    fixes seq :: nat => 'a list
    assumes }\i.(\existsx.\operatorname{seq}(Suci)=x# seq i) and i<length (seq j) and i<
length (seq k)
    shows rev (seq k)!i= rev (seq j)!i
proof (rule linorder-cases)
    assume j<k
    hence j\leqk by simp
    with assms(1, 2) show ?thesis by (rule list-seq-nth)
next
    assume k<j
    hence }k\leqj\mathrm{ by simp
    with assms(1, 3) have rev (seq j)!i= rev (seq k)!i by (rule list-seq-nth)
    thus ?thesis by (rule HOL.sym)
next
    assume j=k
    thus ?thesis by simp
qed
```


### 2.1.2 filter

```
lemma filter-merge-wrt-1:
    assumes \y.y f set ys \LongrightarrowPy\Longrightarrow False
    shows filter P (merge-wrt rel xs ys) = filter P xs
    using assms
proof (induct rel xs ys rule: merge-wrt.induct)
    case (1 rel xs)
    show ?case by simp
```

```
next
    case (2 rel y ys)
    hence }Py\Longrightarrow\mathrm{ False and }\bigwedgez.z\in\mathrm{ set ys >Pz> False by auto
    thus ?case by (auto simp: filter-empty-conv)
next
    case (3 rel x xs y ys)
    hence \negP y and x:\z.z\in set ys \LongrightarrowPz\Longrightarrow False by auto
    have a: filter P(merge-wrt rel xs ys) = filter P xs if }x=y\mathrm{ using that }x\mathrm{ by (rule
3(1))
    have b: filter P(merge-wrt rel xs (y#ys)) = filter P xs if }x\not=y\mathrm{ and rel x y
        using that 3(4) by (rule 3(2))
    have c: filter P (merge-wrt rel (x# xs) ys) = filter P (x# xs) if x\not=y and \neg
rel x y
            using that x by (rule 3(3))
    show ?case by (simp add: a b c«\negP y )
qed
lemma filter-merge-wrt-2:
    assumes }\x.x\in\mathrm{ set xs #Px False
    shows filter P (merge-wrt rel xs ys) = filter P ys
    using assms
proof (induct rel xs ys rule: merge-wrt.induct)
    case (1 rel xs)
    thus ?case by (auto simp: filter-empty-conv)
next
    case (2 rel y ys)
    show ?case by simp
next
    case (3 rel x xs y ys)
    hence }\negPx\mathrm{ and }x:\bigwedgez.z\in\mathrm{ set }xs\LongrightarrowPz\Longrightarrow\mathrm{ False by auto
    have a: filter P (merge-wrt rel xs ys) = filter P ys if }x=y\mathrm{ using that }x\mathrm{ by (rule
3(1))
    have b: filter P (merge-wrt rel xs (y#ys))= filter P(y#ys) if }x\not=y\mathrm{ and rel
x y
            using that x by (rule 3(2))
    have c: filter P (merge-wrt rel (x# xs) ys) = filter P ys if }x\not=y\mathrm{ and }\neg\mathrm{ rel x y
        using that 3(4) by (rule 3(3))
    show ?case by (simp add: a b c <\negP P x`)
qed
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 -
    have *: thesis if }a<b\mathrm{ and b< length xs
        and \bigwedgeas bs cs.as @ ((xs!a) # (bs @ ((xs!b) # cs))) = xs \Longrightarrow thesis for a
b thesis
    proof (rule that(3))
```

```
    from that(1, 2) have 1: a< length xs by simp
    with that(1, 2) have 2: b - Suc a < length (drop (Suc a) xs) by simp
    from that(1)<a< length xs` have eq:xs!b=drop (Suc a) xs ! (b - Suc a)
by simp
    show (take a xs) @ ((xs !a) # ((take (b - Suc a) (drop (Suc a) xs)) @ ((xs !
b) #
drop (Suc (b-Suc a)) (drop (Suc a) xs)))) =xs
    by (simp only: eq id-take-nth-drop[OF 1, symmetric] id-take-nth-drop[OF 2,
symmetric])
    qed
    show ?thesis
    proof (rule linorder-cases)
        assume }i<
        then obtain as bs cs where as @ ((xs!i) # (bs @ ((xs!j) # cs)))=xs
            using assms(3) by (rule *)
    hence filter P xs = filter P (as @ ((xs!i) # (bs @ ((xs!j) # cs)))) by simp
    also from assms(4,5) have ... = (filter P as)@ ((xs!i) # ((filter P bs)@
((xs!j) # (filter P cs))))
            by simp
    finally have }\neg\mathrm{ length (filter P xs) }\leq1\mathrm{ by simp
    thus ?thesis using assms(1) ..
    next
    assume j<i
    then obtain as bs cs where as @ ((xs!j) # (bs @ ((xs!i)# cs)))=xs
        using assms(2) by (rule *)
    hence filter P xs = filter P (as @ ((xs!j) # (bs @ ((xs!i) # cs)))) by simp
    also from assms(4,5) have ... = (filter P as) @ ((xs!j) # ((filter P bs)@
((xs!i) # (filter P cs))))
            by simp
    finally have }\neg\mathrm{ length (filter P xs) }\leq1\mathrm{ by simp
    thus ?thesis using assms(1) ..
    qed
qed
lemma length-filter-eq [simp]: length (filter ((=)x) xs) = count-list xs x
    by (induct xs, simp-all)
```


### 2.1.3 drop

```
lemma nth-in-set-dropI:
    assumes j\leqi and i< length xs
    shows xs!i\in set (drop j xs)
    using assms
proof (induct xs arbitrary: ij)
    case Nil
    thus ?case by simp
next
    case (Cons x xs)
    show ?case
```

```
proof (cases j)
    case 0
    with Cons(3) show ?thesis by (metis drop0 nth-mem)
next
    case (Suc j0)
    with Cons(2) Suc-le-D obtain i0 where i:i=Suc i0 by blast
    with Cons(2) have j0\leqi0 by (simp add: <j=Suc j0>)
    moreover from Cons(3) have i0 < length xs by (simp add: i)
    ultimately have xs ! i0 \in set (drop j0 xs) by (rule Cons(1))
    thus ?thesis by (simp add: i<j = Suc j0`)
qed
qed
```


### 2.1.4 count-list

lemma count-list-upt [simp]: count-list $[a . .<b] x=($ if $a \leq x \wedge x<b$ then 1 else 0)
proof (cases $a \leq b$ )
case True
then obtain $k$ where $b=a+k$ using le-Suc-ex by blast
show ?thesis unfolding $\langle b=a+k\rangle$ by (induct $k$, simp-all)
next
case False
thus ?thesis by simp
qed

### 2.1.5 sorted-wrt

lemma sorted-wrt-upt-iff: sorted-wrt rel $[a . .<b] \longleftrightarrow(\forall i j . a \leq i \longrightarrow i<j \longrightarrow j$
$<b \longrightarrow$ rel $i j$ )
proof (cases $a \leq b$ )
case True
then obtain $k$ where $b=a+k$ using le-Suc-ex by blast
show ?thesis unfolding $\langle b=a+k\rangle$
proof (induct $k$ )
case 0
show ?case by simp
next
case (Suc k)
show ?case
proof (simp add: sorted-wrt-append Suc, intro iffI allI ballI impI conjI) fix $i j$ assume $(\forall i \geq a . \forall j>i . j<a+k \longrightarrow$ rel $i j) \wedge(\forall x \in\{a . .<a+k\}$. rel $x(a+$
k))
hence 1: $\bigwedge i^{\prime} j^{\prime} . a \leq i^{\prime} \Longrightarrow i^{\prime}<j^{\prime} \Longrightarrow j^{\prime}<a+k \Longrightarrow$ rel $i^{\prime} j^{\prime}$
and 2: $\wedge x . a \leq x \Longrightarrow x<a+k \Longrightarrow$ rel $x(a+k)$ by simp-all
assume $a \leq i$ and $i<j$
assume $j<\operatorname{Suc}(a+k)$
hence $j<a+k \vee j=a+k$ by auto
thus rel $i j$

```
        proof
            assume j<a+k
            with }\langlea\leqi\rangle\langlei<j\rangle\mathrm{ show ?thesis by (rule 1)
        next
            assume j=a+k
            from <a \leq i\rangle\langlei< j\rangle show ?thesis unfolding <j =a + k\rangle by (rule 2)
        qed
    next
        fix ij
        assume }\foralli\geqa.\forallj>i.j<Suc(a+k)\longrightarrowrel ij\mathrm{ and }a\leqi\mathrm{ and }i<j\mathrm{ and
j<a+k
        thus rel ij by simp
    next
        fix }
        assume }x\in{a..<a+k
        hence }a\leqx\mathrm{ and }x<a+k\mathrm{ by simp-all
        moreover assume }\foralli\geqa.\forallj>i.j<Suc (a+k)\longrightarrow rel i
        ultimately show rel x ( a+k) by simp
        qed
    qed
next
    case False
    thus ?thesis by simp
qed
```


### 2.1.6 insort-wrt and merge-wrt

```
lemma map-insort-wrt:
    assumes \x. x f set xs \Longrightarrowr2 (fy) (fx)\longleftrightarrowr1 y x
    shows map f(insort-wrt r1 y xs) = insort-wrt r2 (fy)(map f xs)
    using assms
proof (induct xs)
    case Nil
    show ?case by simp
next
    case (Cons x xs)
    have }x\in\operatorname{set}(x#xs) by sim
    hence r2 (fy) (fx)=r1 y x by (rule Cons(2))
    moreover have map f(insort-wrt r1 y xs)= insort-wrtr2 (fy)(map fxs)
    proof (rule Cons(1))
        fix }\mp@subsup{x}{}{\prime
        assume \mp@subsup{x}{}{\prime}\in\mathrm{ set xs}
        hence }\mp@subsup{x}{}{\prime}\in\operatorname{set}(x#xs) by sim
        thus r2 (fy) (fx') = r1 y x' by (rule Cons(2))
    qed
    ultimately show ?case by simp
qed
lemma map-merge-wrt:
```

```
    assumes f'set xs \capf'set ys={}
    and \xyy.x\in set xs \Longrightarrowy\in set ys \Longrightarrowr2 (fx) (fy)\longleftrightarrowr1xy
    shows map f(merge-wrt r1 xs ys) = merge-wrt r2 (map fxs) (map fys)
    using assms
proof (induct r1 xs ys rule: merge-wrt.induct)
    case (1 uu xs)
    show ?case by simp
next
    case (2 r1 v va)
    show ?case by simp
next
    case (3 r1 x xs y ys)
    from 3(4) have fx\not=fy and 1:f'set xs\capf'set (y#ys)={}
    and 2: f'set (x# xs)\capf' set ys = {} by auto
    from this(1) have }x\not=y\mathrm{ by auto
    have eq2: map f (merge-wrt r1 xs (y # ys)) = merge-wrt r2 (map fxs) (mapf
(y # ys))
    if r1 x y using <x\not=y> that 1
    proof (rule 3(2))
    fix ab
    assume a f set xs
    hence }a\in\operatorname{set}(x#xs)\mathrm{ by simp
    moreover assume b fet (y#ys)
    ultimately show r2 (fa) (fb)\longleftrightarrowr1 a b by (rule 3(5))
    qed
    have eq3: map f (merge-wrt r1 (x # xs) ys) = merge-wrt r2 (map f (x # xs))
(map fys)
    if \negr1 x y using <x \not= y> that 2
    proof (rule 3(3))
            fix ab
            assume a cet (x#xs)
            assume b f set ys
            hence b\in set (y# ys) by simp
            with}<a\in\operatorname{set}(x#xs)\rangle\mathrm{ show r2 (fa) (fb) «r1 a b by (rule 3(5))
    qed
    have eq4:r2 (f x) (fy)\longleftrightarrowr1 x y by (rule 3(5), simp-all)
    show ?case by (simp add: eq2 eq3 eq4 <f }x\not=fy\rangle\langlex\not=y>
qed
```


### 2.2 Recursive Functions

```
locale recursive \(=\)
fixes \(h^{\prime}::{ }^{\prime} b \Rightarrow{ }^{\prime} b\)
fixes \(b::\) ' \(b\)
assumes \(b\)-fixpoint: \(h^{\prime} b=b\)
begin
context
fixes \(Q::{ }^{\prime} a \Rightarrow\) bool
```

```
    fixes g:: 'a > 'b
    fixes }h::''a=>'
begin
function (domintros) recfun-aux :: ' }a=>\mathrm{ ' 'b where
    recfun-aux x =(if Q x then g x else h'(recfun-aux (hx)))
    by pat-completeness auto
lemmas [induct del] = recfun-aux.pinduct
definition dom :: ' }a=>\mathrm{ bool
    where dom }x\longleftrightarrow(\existsk.Q((h~k)x)
lemma domI:
    assumes \neg Q x dom (h x)
    shows dom x
proof (cases Q x)
    case True
    hence Q ((h~0) x) by simp
    thus ?thesis unfolding dom-def ..
next
    case False
    hence dom (h x) by (rule assms)
    then obtain k where Q (( }\mp@subsup{h}{}{~}k)(hx))\mathrm{ unfolding dom-def ..
    hence Q (( }\mp@subsup{h}{}{~}(Suc k)) x) by (simp add: funpow-swap1
    thus ?thesis unfolding dom-def ..
qed
lemma domD:
    assumes dom x and \negQ x
    shows dom (h x)
proof -
    from assms(1) obtain k where *: Q (( }\mp@subsup{h}{~~}{~}k)x)\mathrm{ unfolding dom-def ..
    with assms(2) have k\not=0 using funpow-0 by fastforce
    then obtain m}\mathrm{ where k=Suc m using nat.exhaust by blast
    with * have Q ((h^^m) (hx)) by (simp add: funpow-swap1)
    thus ?thesis unfolding dom-def ..
qed
lemma recfun-aux-domI:
    assumes dom x
    shows recfun-aux-dom x
proof -
    from assms obtain k where Q ((h~~ k)x) unfolding dom-def ..
    thus ?thesis
    proof (induct k arbitrary: x)
        case 0
        hence Q x by simp
        with recfun-aux.domintros show ?case by blast
```

```
    next
        case (Suc k)
        from Suc(2) have Q ((h^~ k) (hx)) by (simp add: funpow-swap1)
        hence recfun-aux-dom ( }hx\mathrm{ ) by (rule Suc(1))
        with recfun-aux.domintros show ?case by blast
    qed
qed
lemma recfun-aux-domD:
    assumes recfun-aux-dom x
    shows dom x
    using assms
proof (induct x rule: recfun-aux.pinduct)
    case (1 x)
    show ?case
    proof (cases Q x)
        case True
        with domI show ?thesis by blast
    next
        case False
        hence dom (hx) by (rule 1(2))
        thus ?thesis using domI by blast
    qed
qed
corollary recfun-aux-dom-alt: recfun-aux-dom = dom
    by (auto dest: recfun-aux-domI recfun-aux-domD)
definition fun :: ' }a=>\mathrm{ ' '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 (hx)))
proof (cases dom x)
    case True
    hence dom: recfun-aux-dom x by (rule recfun-aux-domI)
    show ?thesis
    proof (cases Q x)
        case True
        with dom show ?thesis by (simp add: fun-def recfun-aux.psimps)
    next
        case False
            have recfun-aux-dom (h x) by (rule recfun-aux-domI, rule domD, fact True,
fact False)
    thus ?thesis by (simp add: fun-def dom False recfun-aux.psimps)
    qed
next
    case False
    moreover have }\negQ
    proof
```

```
    assume Q x
    hence dom x using domI by blast
    with False show False ..
    qed
    moreover have }\neg\mathrm{ dom ( }hx\mathrm{ )
    proof
    assume dom (h x)
    hence dom x using domI by blast
    with False show False ..
    qed
    ultimately show ?thesis by (simp add: recfun-aux-dom-alt fun-def b-fixpoint
split del: if-split)
qed
lemma eq-fixpointI: ᄀ dom x \Longrightarrow fun }x=
    by (simp add: fun-def recfun-aux-dom-alt)
lemma pinduct: dom x \Longrightarrow(\bigwedgex.dom x \Longrightarrow(\negQ x \LongrightarrowP(hx))\LongrightarrowPx)\Longrightarrow
P x
    unfolding recfun-aux-dom-alt[symmetric] by (fact recfun-aux.pinduct)
end
end
interpretation tailrec: recursive }\lambdax.x\mathrm{ undefined
    by (standard, fact refl)
```


### 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 \((\lambda x y . P 1 x y \wedge P 2 x y)(A 1 \cap A 2)\) (is almost-full-on ?P ?A)
proof (rule almost-full-onI)
fix \(f\) :: nat \(\Rightarrow{ }^{\prime} a\)
assume \(a: \forall i . f i \in ? A\)
define \(g\) where \(g=(\lambda i\). \((f i, f i))\)
from assms have almost-full-on (prod-le P1 P2) ( \(A 1 \times\) A2) by (rule al-most-full-on-Sigma)
moreover from \(a\) have \(\bigwedge i . g i \in A 1 \times A 2\) by (simp add: \(g\)-def)
ultimately obtain \(i j\) where \(i<j\) and prod-le P1 P2 ( \(g i)(g j)\) by (rule almost-full-onD)
from this(2) have ?P \((f i)(f j)\) by (simp add: g-def prod-le-def)
with \(\langle i<j\rangle\) show good? \(f\) by (rule goodI)
qed
corollary almost-full-on-same:
assumes almost-full-on P1 A and almost-full-on P2 A
```

```
    shows almost-full-on ( }\lambdaxy.P1 x y ^P2 x y) A
proof -
    from assms have almost-full-on ( }\lambdaxy.P1 x y ^ P2 x y) (A\capA) by (rule
almost-full-on-Int)
    thus ?thesis by simp
qed
context ord
begin
definition is-le-rel :: (' }a=>\mp@subsup{|}{}{\prime}a=>\mathrm{ bool) ) b 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 (<)
    by (simp-all add: is-le-rel-def)
lemma is-le-relE:
    assumes is-le-rel rel
    obtains rel = (=) | rel=(\leq)| rel=(<)
    using assms unfolding is-le-rel-def by blast
end
context preorder
begin
lemma is-le-rel-le:
    assumes is-le-rel rel
    shows rel x y \Longrightarrowx\leqy
    using assms by (rule is-le-relE, auto dest: less-imp-le)
lemma is-le-rel-trans:
    assumes is-le-rel rel
    shows rel x y rel y z\Longrightarrow rel xz
    using assms by (rule is-le-relE, auto dest:order-trans less-trans)
lemma is-le-rel-trans-le-left:
    assumes is-le-rel rel
    shows }x\leqy\Longrightarrow\mathrm{ rel }yz\Longrightarrowx\leq
    using assms by (rule is-le-relE, auto dest: order-trans le-less-trans less-imp-le)
lemma is-le-rel-trans-le-right:
    assumes is-le-rel rel
    shows rel x y }\Longrightarrowy\leqz\Longrightarrowx\leq
    using assms by (rule is-le-relE, auto dest: order-trans less-le-trans less-imp-le)
lemma is-le-rel-trans-less-left:
    assumes is-le-rel rel
    shows }x<y\Longrightarrow\mathrm{ rel y z 
```

using assms by (rule is-le-relE, auto dest: less-le-trans less-imp-le)

```
lemma is-le-rel-trans-less-right:
    assumes is-le-rel rel
    shows rel x y y<z\Longrightarrowx<z
    using assms by (rule is-le-relE, auto dest: le-less-trans less-imp-le)
end
context order
begin
lemma is-le-rel-distinct:
    assumes is-le-rel rel
    shows rel x y \Longrightarrow x = y \Longrightarrowx<y
    using assms by (rule is-le-relE, auto)
lemma is-le-rel-antisym:
    assumes is-le-rel rel
    shows rel x y rel y x \Longrightarrowx=y
    using assms by (rule is-le-relE, auto)
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 adds $t \Longrightarrow u+(t-s)=(u+t)-$ $s$
using add-commute minus-plus by auto
context ulcs-powerprod
begin
lemma lcs-alt-2:
assumes $a+x=b+y$
shows lcs $x y=(b+y)-g c s a b$
proof -
have $a+($ lcs $x y+$ gcs $a b)=$ lcs $(a+x)(a+y)+g c s a b$ by (simp only: lcs-plus-left ac-simps)

```
    also have ... = lcs (b+y) (a+y) + gcs a b by (simp only: assms)
    also have ... = (lcs a b + y) + gcs a b by (simp only:lcs-plus-right lcs-comm)
    also have ... = (gcs a b lcs a b) + y by (simp only: ac-simps)
    also have ... =a+(b+y) by (simp only: gcs-plus-lcs, simp add: ac-simps)
    finally have (lcs x y + gcs a b) - gcs a b = (b+y) - gcs a b by simp
    thus ?thesis by simp
qed
corollary lcs-alt-1:
    assumes }a+x=b+
    shows lcs x y = (a+x) - gcs a b
proof -
    have lcs x y lcs y x by (simp only:lcs-comm)
    also from assms[symmetric] have ... = (a+x) - gcs b a by (rule lcs-alt-2)
    also have ... = ( a + x) - gcs a b by (simp only: gcs-comm)
    finally show ?thesis.
qed
corollary lcs-minus-1:
    assumes }a+x=b+
    shows lcs x y-x=a-gcs a b
    by (simp add:lcs-alt-1 [OF assms] diff-right-commute)
corollary lcs-minus-2:
    assumes }a+x=b+
    shows lcs x y - y=b - gcs a b
    by (simp add: lcs-alt-2[OF assms] diff-right-commute)
lemma gcs-minus:
    assumes uadds s and u adds t
    shows gcs (s-u) (t-u)=gcs st-u
proof -
    from assms have gcs s t = gcs ((s-u) +u) ((t-u)+u) by (simp add:
minus-plus)
    also have ... =gcs (s-u)(t-u)+u by (simp only:gcs-plus-right)
    finally show?thesis by simp
qed
corollary gcs-minus-gcs: gcs (s-(gcs st)) (t-(gcs s t)) = 0
    by (simp add: gcs-minus gcs-adds gcs-adds-2)
end
```


### 3.2 Miscellaneous

lemma poly-mapping-rangeE:
assumes $c \in$ Poly-Mapping.range $p$
obtains $k$ where $k \in$ keys $p$ and $c=$ lookup $p k$
using assms by (transfer, auto)

```
lemma poly-mapping-range-nonzero: 0 & Poly-Mapping.range p
    by (transfer, auto)
lemma (in term-powerprod) Keys-range-vectorize-poly:Keys (Poly-Mapping.range
(vectorize-poly p)) = pp-of-term'keys p
proof
    show Keys (Poly-Mapping.range (vectorize-poly p))\subseteqpp-of-term'keys p
    proof
        fix }
        assume t Keys (Poly-Mapping.range (vectorize-poly p))
        then obtain q}\mathrm{ where q}\in\mathrm{ Poly-Mapping.range (vectorize-poly p) and t keys
q \text { by (rule in-KeysE)}
            from this(1) obtain k where q: q= lookup (vectorize-poly p) k by (metis
DiffE imageE range.rep-eq)
        with }\langlet\in\mathrm{ keys q> have term-of-pair (t,k) G keys p
            by (metis in-keys-iff lookup-proj-poly lookup-vectorize-poly)
        hence pp-of-term (term-of-pair (t,k)) \in pp-of-term' keys p by (rule imageI)
        thus t\inpp-of-term'keys p by (simp only: pp-of-term-of-pair)
    qed
next
    show pp-of-term'keys p\subseteq Keys (Poly-Mapping.range (vectorize-poly p))
    proof
        fix }
        assume t\inpp-of-term' keys p
        then obtain }x\mathrm{ where }x\in\mathrm{ keys p and t=pp-of-term x ..
        from this(2) have term-of-pair (t, component-of-term x) = x by (simp add:
term-of-pair-pair)
    with }\langlex\inkeys p\rangle have lookup p (term-of-pair (t, component-of-term x)) =0 
        by (simp add: in-keys-iff)
    hence lookup (proj-poly (component-of-term x) p) t\not=0 by (simp add: lookup-proj-poly)
    hence t:t\inkeys (proj-poly (component-of-term x) p)
        by (simp add: in-keys-iff)
    from }\langlex\in\mathrm{ keys p〉 have component-of-term x k keys (vectorize-poly p)
        by (simp add: keys-vectorize-poly)
    from t show t\inKeys (Poly-Mapping.range (vectorize-poly p))
    proof (rule in-KeysI)
    have proj-poly (component-of-term x) p= lookup (vectorize-poly p) (component-of-term
x)
            by (simp only: lookup-vectorize-poly)
            also from <component-of-term x keys (vectorize-poly p)>
        have ...\in Poly-Mapping.range (vectorize-poly p) by (rule in-keys-lookup-in-range)
            finally show proj-poly (component-of-term x) p\in Poly-Mapping.range
(vectorize-poly p).
    qed
    qed
qed
```


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

```
context ordered-term
begin
```

lemma lt-lookup-vectorize: punit.lt (lookup (vectorize-poly p) (component-of-term
$(l t p)))=l p p$
proof (cases $p=0$ )
case True
thus ?thesis by (simp add: vectorize-zero min-term-def pp-of-term-of-pair)
next
case False
show ?thesis
proof (rule punit.lt-eqI-keys)
from False have $l t p \in$ keys $p$ by (rule lt-in-keys)
thus $l p p \in$ keys (lookup (vectorize-poly $p$ ) (component-of-term (lt p)))
by (simp add: lookup-vectorize-poly keys-proj-poly)
next
fix $t$
assume $t \in$ keys (lookup (vectorize-poly p) (component-of-term (lt p)))
also have $\ldots=p p$-of-term ' $\{x \in k e y s p$. component-of-term $x=$ component-of-term
(lt p) \}
by (simp only: lookup-vectorize-poly keys-proj-poly)
finally obtain $v$ where $v \in$ keys $p$ and 1: component-of-term $v=$ compo-
nent-of-term (lt p)
and $t: t=p p$-of-term $v$ by auto
from this(1) have $v \preceq_{t} l t p$ by (rule lt-max-keys)
show $t \preceq l p p$
proof (rule ccontr)
assume $\neg t \preceq l p p$
hence $l p p \prec p p$-of-term $v$ by (simp add: $t$ )
hence $l p p \neq p p$-of-term $v$ and $l p p \preceq p p$-of-term $v$ by simp-all
note this(2)
moreover from 1 have component-of-term (lt p) $\leq$ component-of-term $v$ by
simp
ultimately have $l t p \preceq_{t} v$ by (rule ord-termI)
with $\left\langle v \preceq_{t} l t p\right\rangle$ have $v=l t p$
by $\operatorname{simp}$
with $\langle l p p \neq p p$-of-term $v\rangle$ show False by simp
qed
qed
qed
lemma lower-higher-zeroI: $u \preceq_{t} v \Longrightarrow$ lower (higher $p$ v) $u=0$
by (simp add: lower-eq-zero-iff lookup-higher-when)
lemma lookup-minus-higher: lookup $(p-$ higher $p v) u=\left(\right.$ lookup $p u$ when $u \preceq_{t}$
v)
by (auto simp: lookup-minus lookup-higher-when when-def)
lemma keys-minus-higher: keys $(p-$ higher $p v)=\left\{u \in\right.$ keys $\left.p . u \preceq_{t} v\right\}$
by (rule set-eqI, simp add: lookup-minus-higher conj-commute flip: lookup-not-eq-zero-eq-in-keys)

## lemma lt-minus-higher: $v \in$ keys $p \Longrightarrow l t(p-\operatorname{higher} p v)=v$

by (rule lt-eqI-keys, simp-all add: keys-minus-higher)
lemma lc-minus-higher: $v \in$ keys $p \Longrightarrow l c(p-\operatorname{higher} p v)=$ lookup $p v$
by (simp add: lc-def lt-minus-higher lookup-minus-higher)
lemma tail-minus-higher: $v \in$ keys $p \Longrightarrow$ tail $(p-$ higher $p v)=$ lower $p v$
by (rule poly-mapping-eqI, simp add: lookup-tail-when lt-minus-higher lookup-lower-when lookup-minus-higher cong: when-cong)
end

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

lemma (in gd-term) dgrad-p-set-closed-mult-scalar:
assumes dickson-grading $d$ and $p \in$ punit.dgrad-p-set $d m$ and $r \in d g r a d-p$-set
d $m$
shows $p \odot r \in d g r a d-p$-set $d m$
proof (rule dgrad-p-setI)
fix $v$
assume $v \in$ keys $(p \odot r)$
then obtain $t u$ where $t \in$ keys $p$ and $u \in$ keys $r$ and $v: v=t \oplus u$
by (rule in-keys-mult-scalarE)
from assms(2) $\langle t \in$ keys $p\rangle$ have $d t \leq m$ by (rule punit.dgrad- $p$-setD[simplified])
moreover from assms(3) $\langle u \in$ keys $r\rangle$ have $d$ (pp-of-term $u) \leq m$ by (rule dgrad-p-setD)
ultimately have $d(t+p p$-of-term $u) \leq m$ using assms $(1)$ by (simp add: dickson-gradingD1)
thus $d$ ( $p$ p-of-term $v$ ) $\leq m$ by (simp only: v pp-of-term-splus)
qed

### 3.5 Regular Sequences

definition is-regular-sequence :: ('a::comm-powerprod $\Rightarrow_{0}{ }^{\prime} b::$ comm-ring-1) list $\Rightarrow$ bool
where is-regular-sequence $f s \longleftrightarrow(\forall j<$ length $f s . \forall q . q * f s!j \in$ ideal (set (take $j f s)) \longrightarrow$

$$
\left.q \in \operatorname{ideal}\left(\operatorname{set}\left(\text { take } j f_{s}\right)\right)\right)
$$

lemma is-regular-sequenceD:
is-regular-sequence $f s \Longrightarrow j<$ length $f s \Longrightarrow q * f s!j \in$ ideal $($ set $($ take $j f s)) \Longrightarrow$ $q \in$ ideal (set (take j fs))
by (simp add: is-regular-sequence-def)
lemma is-regular-sequence-Nil: is-regular-sequence []
by (simp add: is-regular-sequence-def)

```
lemma is-regular-sequence-snocI:
    assumes \q. q*f\inideal (set fs)\Longrightarrowq\inideal (set fs) and is-regular-sequence
fs
    shows is-regular-sequence (fs @ [f])
proof (simp add: is-regular-sequence-def, intro impI allI)
    fix jq
    assume 1: j< Suc (length fs) and 2: q*(fs @ [f])!j\in ideal (set (take jfs))
    show q\inideal (set (take jfs))
    proof (cases j = length fs)
        case True
        from 2 have q*f\inideal (set fs) by (simp add: True)
        hence q\in ideal (set fs) by (rule assms(1))
        thus ?thesis by (simp add: True)
    next
        case False
        with 1 have j< length fs by simp
        with 2 have q*fs ! j\in ideal (set (take j fs)) by (simp add: nth-append)
            with assms(2)}\langlej<length fs> show q \in ideal (set (take j fs)) by (rule
is-regular-sequenceD)
    qed
qed
lemma is-regular-sequence-snocD:
    assumes is-regular-sequence (fs @ [f])
    shows }\q.q*f\in\mathrm{ ideal (set fs) }\Longrightarrowq\inideal (set fs
        and is-regular-sequence fs
proof -
    fix q
    assume 1:q*f\in ideal (set fs)
    note assms
    moreover have length fs < length (fs @ [f]) by simp
    moreover from 1 have q*(fs @ [f])! (length fs) \in ideal (set (take (length fs)
(fs @ [f])))
            by simp
    ultimately have q\inideal (set (take (length fs) (fs @ [f]))) by (rule is-regular-sequenceD)
    thus q\inideal (set fs) by simp
next
    show is-regular-sequence fs unfolding is-regular-sequence-def
    proof (intro impI allI)
        fix j q
        assume 1:j< length fs and 2: q* fs !j i ideal (set (take j fs))
        note assms
        moreover from 1 have j<length (fs @ [f]) by simp
        moreover from 1 2 have q*(fs @ [f])!j i\in ideal (set (take j (fs @ [f])))
            by (simp add: nth-append)
        ultimately have q\inideal (set (take j(fs @ [f]))) by (rule is-regular-sequenceD)
        with 1 show q}\in\mathrm{ ideal (set (take jfs)) by simp
    qed
qed
```

```
lemma is-regular-sequence-removeAll-zero:
    assumes is-regular-sequence fs
    shows is-regular-sequence (removeAll 0 fs)
    using assms
proof (induct fs rule: rev-induct)
    case Nil
    show ?case by (simp add: is-regular-sequence-Nil)
next
    case (snoc f fs)
    have set (removeAll 0 fs) = set fs - {0} by simp
    also have ideal ... = ideal (set fs) by (fact ideal.span-Diff-zero)
    finally have eq: ideal (set (removeAll 0 fs)) =ideal (set fs).
    from snoc(2) have *: is-regular-sequence fs by (rule is-regular-sequence-snocD)
    show ?case
    proof (simp, intro conjI impI)
        show is-regular-sequence (removeAll 0 fs @ [f])
        proof (rule is-regular-sequence-snocI)
            fix q
            assume q*f\inideal (set (removeAll 0 fs))
            hence q*f\inideal (set fs) by (simp only: eq)
            with snoc(2) have q\inideal (set fs) by (rule is-regular-sequence-snocD)
            thus q}\in\mathrm{ ideal (set (removeAll 0 fs)) by (simp only: eq)
        next
            from * show is-regular-sequence (removeAll 0 fs) by (rule snoc.hyps)
        qed
    next
        from * show is-regular-sequence (removeAll 0 fs) by (rule snoc.hyps)
    qed
qed
lemma is-regular-sequence-remdups:
    assumes is-regular-sequence fs
    shows is-regular-sequence (rev (remdups (rev fs)))
    using assms
proof (induct fs rule: rev-induct)
    case Nil
    show ?case by (simp add: is-regular-sequence-Nil)
next
    case (snoc ffs)
    from snoc(2) have *: is-regular-sequence fs by (rule is-regular-sequence-snocD)
    show ?case
    proof (simp, intro conjI impI)
        show is-regular-sequence (rev (remdups (rev fs)) @ [f])
        proof (rule is-regular-sequence-snocI)
            fix q
            assume q*f\inideal (set (rev (remdups (rev fs))))
            hence q*f\inideal (set fs) by simp
            with snoc(2) have q\inideal (set fs) by (rule is-regular-sequence-snocD)
```

```
            thus q\inideal (set (rev (remdups (rev fs)))) by simp
    next
            from * show is-regular-sequence (rev (remdups (rev fs))) by (rule snoc.hyps)
        qed
    next
        from * show is-regular-sequence (rev (remdups (rev fs))) by (rule snoc.hyps)
    qed
qed
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 ' $t \Rightarrow_{0}{ }^{\prime} b$. Later, we transfer all algorithms defined on such objects to algorithms efficiently operating on sig-poly-pairs, i. e. objects of type ${ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)$.

### 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 -
let $? l=l c s(l p p)(l p q)$
let $? p=$ monom-mult $(1 / l c p)(? l-l p p) p$
let $? q=$ monom-mult $(1 / l c q)(? l-l p q) q$
from $\operatorname{assms}(3)$ have eq1: component-of-term (lt p) $=$ component-of-term (lt q)
and eq2: spoly $p q=? p-? q$
by (simp-all add: spoly-def Let-def lc-def split: if-split-asm)
from $\langle p \neq 0\rangle$ have $l c p \neq 0$ by (rule lc-not-0)
with $\operatorname{assms}(1)$ have $l t ? p=(? l-l p p) \oplus l t p$ and $l c ? p=1$ by (simp-all add:
lt-monom-mult)
from this(1) have lt-p:lt ?p = term-of-pair (?l, component-of-term (lt p))
by (simp add: splus-def adds-minus adds-lcs)
from $\langle q \neq 0\rangle$ have $l c q \neq 0$ by (rule lc-not-0)
with $\operatorname{assms}(2)$ have $l t ? q=(? l-l p q) \oplus l t q$ and $l c ? q=1$ by (simp-all add:
lt-monom-mult)
from this(1) have lt-q: lt ?q = term-of-pair (?l, component-of-term (lt p))
by (simp add: eq1 splus-def adds-minus adds-lcs-2)
from $\operatorname{assms}(3)$ have $? p-? q \neq 0$ by (simp add: eq2)
moreover have $l t ? q=l t ? p$ by (simp only: lt-p lt-q)
moreover have $l c ? q=l c ? p$ by (simp only: $\langle l c ? p=1\rangle\langle l c ? q=1\rangle)$
ultimately have lt (?p - ?q) $\prec_{t} l t$ ?p by (rule lt-minus-lessI) thus ?thesis by (simp only: eq2 lt-p)
qed

### 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::quasi-pm-powerprod $\times$ nat $)$
and term-of-pair::(' $a \times n a t) \Rightarrow{ }^{\prime} t$
and ord::' $a \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infixl $\left.\preceq 50\right)$
and ord-strict (infixl $\prec 50$ )
and ord-term: :' $t \Rightarrow$ ' $t \Rightarrow$ bool (infixl $\preceq_{t} 50$ )
and ord-term-strict::' $t \Rightarrow{ }^{\prime} t \Rightarrow$ bool (infixl $\left.\prec_{t} 50\right)$
begin
lemma in-idealE-rep-dgrad-p-set:
assumes hom-grading $d$ and $B \subseteq$ punit.dgrad-p-set $d m$ and $p \in$ punit.dgrad-p-set
$d m$ and $p \in$ ideal $B$
obtains $r$ where keys $r \subseteq B$ and Poly-Mapping.range $r \subseteq$ punit.dgrad-p-set d
$m$ and $p=$ ideal.rep $r$
proof -
from assms obtain $A q$ where finite $A$ and $A \subseteq B$ and $0: \Lambda b . q b \in$ punit.dgrad-p-set $d m$
and $p: p=\left(\sum a \in A . q a * a\right)$ by (rule punit.in-pmdlE-dgrad-p-set[simplified], blast)
define $r$ where $r=$ Abs-poly-mapping ( $\lambda k . q k$ when $k \in A$ )
have 1 : lookup $r=(\lambda k$. $q k$ when $k \in A)$ unfolding $r$-def
by (rule Abs-poly-mapping-inverse, simp add: <finite $A\rangle$ )
have 2: keys $r \subseteq A$ by (auto simp: in-keys-iff 1)
show ?thesis
proof
show Poly-Mapping.range $r \subseteq$ punit.dgrad- $p$-set $d m$
proof
fix $f$
assume $f \in$ Poly-Mapping.range $r$
then obtain $b$ where $b \in$ keys $r$ and $f: f=$ lookup $r b$ by (rule poly-mapping-range $E$ )
from this(1) 2 have $b \in A$..
hence $f=q b$ by (simp add: $f 1$ )
show $f \in$ punit.dgrad- $p$-set $d m$ unfolding $\langle f=q b\rangle$ by (rule 0 )
qed
next
have $p=\left(\sum a \in A\right.$. lookup $\left.r a * a\right)$ unfolding $p$ by (rule sum.cong, simp-all add: 1)
also from 〈finite $A$ 〉 2 have $\ldots=\left(\sum a \in k e y s\right.$ r. lookup $\left.r a * a\right)$
proof (rule sum.mono-neutral-right)
show $\forall a \in A-k e y s$ r. lookup $r a * a=0$
by (simp add: in-keys-iff)
qed

```
    finally show p =ideal.rep r by (simp only: ideal.rep-def)
    next
    from 2 <A\subseteqB` show keys r\subseteqB by (rule subset-trans)
    qed
qed
context fixes fs :: (' }a\not=0\mp@subsup{}{0}{\prime}b::field) lis
begin
definition sig-inv-set' :: nat => ('t 䅐 'b) set
    where sig-inv-set' j = {r. keys (vectorize-poly r)\subseteq{0..<j}}
abbreviation sig-inv-set \equiv sig-inv-set' (length fs)
definition rep-list :: ('t =\mp@subsup{0}{0}{\prime}}\mp@subsup{}{}{\prime})=>(\mp@subsup{}{}{\prime}a=\mp@subsup{=}{0}{\prime}'b
    where rep-list r = ideal.rep (pm-of-idx-pm fs (vectorize-poly r))
lemma sig-inv-setI: keys (vectorize-poly r)\subseteq{0..<j}\Longrightarrowr\insig-inv-set'j
    by (simp add: sig-inv-set'-def)
lemma sig-inv-setD: r sig-inv-set' j\Longrightarrow keys (vectorize-poly r)\subseteq{0..<j}
    by (simp add: sig-inv-set'-def)
lemma sig-inv-setI':
    assumes \v.v\in keys r\Longrightarrow component-of-term v<j
    shows r f sig-inv-set' }
proof (rule sig-inv-setI, rule)
    fix }
    assume k}\in\mathrm{ keys (vectorize-poly r)
    then obtain v}\mathrm{ where vekeys r and k:k=component-of-term v unfolding
keys-vectorize-poly ..
    from this(1) have k<j unfolding k by (rule assms)
    thus }k\in{0..<j} by sim
qed
lemma sig-inv-setD':
    assumes r sig-inv-set' }j\mathrm{ and }v\inkeys 
    shows component-of-term v<j
proof -
    from assms(2) have component-of-term v\incomponent-of-term'keys r by (rule
imageI)
    also have ... = keys (vectorize-poly r) by (simp only: keys-vectorize-poly)
    also from assms(1) have ...\subseteq{0..<j} by (rule sig-inv-setD)
    finally show ?thesis by simp
qed
corollary sig-inv-setD-lt:
    assumes r\in sig-inv-set' j and r\not=0
    shows component-of-term (lt r)<j
```

```
    by (rule sig-inv-setD', fact, rule lt-in-keys, fact)
lemma sig-inv-set-mono:
    assumes i\leqj
    shows sig-inv-set' i}\subseteq\mathrm{ sig-inv-set' }
proof
    fix r
    assume r\in sig-inv-set' }
    hence keys (vectorize-poly r)\subseteq{0..<i} by (rule sig-inv-setD)
    also from assms have ...\subseteq{0..<j} by fastforce
    finally show }r\in\mathrm{ sig-inv-set' j by (rule sig-inv-setI)
qed
lemma sig-inv-set-zero: 0 \in sig-inv-set' j
    by (rule sig-inv-setI', simp)
lemma sig-inv-set-closed-uminus: r sig-inv-set' j\Longrightarrow-r\insig-inv-set' j
    by (auto dest!: sig-inv-setD' intro!: sig-inv-setI' simp: keys-uminus)
lemma sig-inv-set-closed-plus:
    assumes r}\in\mathrm{ sig-inv-set' j and se sig-inv-set' j
    shows }r+s\in\mathrm{ sig-inv-set' }
proof (rule sig-inv-setI')
    fix v
    assume v\in keys (r+s)
    hence v}\in\mathrm{ keys }r\cup\mathrm{ keys s using Poly-Mapping.keys-add ..
    thus component-of-term v<j
    proof
        assume v\in keys r
        with assms(1) show ?thesis by (rule sig-inv-setD')
    next
        assume v\in keys s
        with assms(2) show ?thesis by (rule sig-inv-setD')
    qed
qed
lemma sig-inv-set-closed-minus:
    assumes r\in sig-inv-set' }j\mathrm{ and }s\in\mathrm{ sig-inv-set' }
    shows }r-s\in\mathrm{ sig-inv-set' }
proof (rule sig-inv-setI')
    fix }
    assume v\in keys (r-s)
    hence v\in keys r \cup keys s using keys-minus ..
    thus component-of-term v<j
    proof
        assume v\in keys r
        with assms(1) show ?thesis by (rule sig-inv-setD')
    next
    assume v\in keys s
```

with assms(2) show ?thesis by (rule sig-inv-setD')
qed
qed
lemma sig-inv-set-closed-monom-mult:
assumes $r \in$ sig-inv-set' $j$
shows monom-mult c $t r \in$ sig-inv-set' $j$
proof (rule sig-inv-setI')
fix $v$
assume $v \in$ keys (monom-mult ctr)
hence $v \in(\oplus) t$ ' keys $r$ using keys-monom-mult-subset ..
then obtain $u$ where $u \in$ keys $r$ and $v: v=t \oplus u$..
from assms this(1) have component-of-term $u<j$ by (rule sig-inv-setD')
thus component-of-term $v<j$ by (simp add: v term-simps)
qed
lemma sig-inv-set-closed-mult-scalar:
assumes $r \in$ sig-inv-set' $j$
shows $p \odot r \in$ sig-inv-set' $j$
proof (rule sig-inv-setI')
fix $v$
assume $v \in$ keys $(p \odot r)$
then obtain $t u$ where $u \in$ keys $r$ and $v: v=t \oplus u$ by (rule in-keys-mult-scalarE)
from assms this(1) have component-of-term $u<j$ by (rule sig-inv-setD')
thus component-of-term $v<j$ by (simp add: v term-simps)
qed
lemma rep-list-zero: rep-list $0=0$
by (simp add: rep-list-def vectorize-zero)
lemma rep-list-uminus: rep-list $(-r)=-$ rep-list $r$ by (simp add: rep-list-def vectorize-uminus pm-of-idx-pm-uminus)
lemma rep-list-plus: rep-list $(r+s)=$ rep-list $r+$ rep-list $s$
by (simp add: rep-list-def vectorize-plus pm-of-idx-pm-plus ideal.rep-plus)
lemma rep-list-minus: rep-list $(r-s)=$ rep-list $r-r e p-l i s t ~ s$
by (simp add: rep-list-def vectorize-minus pm-of-idx-pm-minus ideal.rep-minus)
lemma vectorize-mult-scalar:
vectorize-poly $(p \odot q)=$ MPoly-Type-Class.punit.monom-mult p 0 (vectorize-poly
q)
by (rule poly-mapping-eqI, simp add: lookup-vectorize-poly MPoly-Type-Class.punit.lookup-monom-mult-zero proj-mult-scalar)
lemma rep-list-mult-scalar: rep-list $(c \odot r)=c *$ rep-list $r$ by (simp add: rep-list-def vectorize-mult-scalar pm-of-idx-pm-monom-mult punit.rep-mult-scalar[simplified])
lemma rep-list-monom-mult: rep-list (monom-mult ctr)=punit.monom-mult $c$

```
t (rep-list r)
    unfolding mult-scalar-monomial[symmetric] times-monomial-left[symmetric] by
(rule rep-list-mult-scalar)
lemma rep-list-monomial:
    assumes distinct fs
    shows rep-list (monomial c u)=
                            (punit.monom-mult c (pp-of-term u) (fs !(component-of-term u))
                        when component-of-term u< length fs)
    by (simp add: rep-list-def vectorize-monomial pm-of-idx-pm-monomial[OF assms]
when-def times-monomial-left)
lemma rep-list-in-ideal-sig-inv-set:
    assumes r\in sig-inv-set' j
    shows rep-list r ideal (set (take j fs))
proof -
    let ?fs = take j fs
    from assms have keys (vectorize-poly r)\subseteq{0..<j} by (rule sig-inv-setD)
    hence eq:pm-of-idx-pm fs (vectorize-poly r) = pm-of-idx-pm?fs (vectorize-poly
r)
    by (simp only: pm-of-idx-pm-take)
    have rep-list r i ideal (keys (pm-of-idx-pm fs (vectorize-poly r)))
        unfolding rep-list-def by (rule ideal.rep-in-span)
    also have ... = ideal (keys (pm-of-idx-pm ?fs (vectorize-poly r)) ) by (simp only:
eq)
    also from keys-pm-of-idx-pm-subset have ... \subseteqideal (set ?fs) by (rule ideal.span-mono)
    finally show ?thesis.
qed
corollary rep-list-subset-ideal-sig-inv-set:
    B\subseteq sig-inv-set' }j\Longrightarrow\mathrm{ rep-list ' }B\subseteq\mathrm{ ideal (set (take j fs))
    by (auto dest: rep-list-in-ideal-sig-inv-set)
lemma rep-list-in-ideal: rep-list r i ideal (set fs)
proof -
    have rep-list r ideal (keys (pm-of-idx-pm fs (vectorize-poly r)))
        unfolding rep-list-def by (rule ideal.rep-in-span)
    also from keys-pm-of-idx-pm-subset have ... \subseteqideal (set fs) by (rule ideal.span-mono)
    finally show ?thesis .
qed
corollary rep-list-subset-ideal: rep-list' }B\subseteq\mathrm{ ideal (set fs)
    by (auto intro: rep-list-in-ideal)
lemma in-idealE-rep-list:
    assumes p}\in\mathrm{ ideal (set fs)
    obtains r where p=rep-list r and r\in sig-inv-set
proof -
    from assms obtain r0 where r0: keys r0\subseteq set fs and p:p= ideal.rep r0
```

```
    by (rule ideal.spanE-rep)
    show ?thesis
    proof
    show p = rep-list (atomize-poly (idx-pm-of-pm fs r0))
        by (simp add: rep-list-def vectorize-atomize-poly pm-of-idx-pm-of-pm[OF r0]
p)
    next
        show atomize-poly (idx-pm-of-pm fs r0) \in sig-inv-set
        by (rule sig-inv-setI, simp add: vectorize-atomize-poly keys-idx-pm-of-pm-subset)
    qed
qed
lemma keys-rep-list-subset:
    assumes t\in keys (rep-list r)
    obtains vs where v\in keys r and s\inKeys (set fs) and t=pp-of-term v +s
proof -
    from assms obtain v0 s where v0:v0\in Keys (Poly-Mapping.range (pm-of-idx-pm
fs (vectorize-poly r)))
    and s:s\inKeys (keys (pm-of-idx-pm fs (vectorize-poly r))) and t:t=v0 +s
    unfolding rep-list-def by (rule punit.keys-rep-subset[simplified])
    note s
    also from keys-pm-of-idx-pm-subset have Keys (keys (pm-of-idx-pm fs (vectorize-poly
r)))}\subseteqK\mp@code{Keys (set fs)
    by (rule Keys-mono)
    finally have s Keys (set fs).
    note v0
    also from range-pm-of-idx-pm-subset'
    have Keys (Poly-Mapping.range (pm-of-idx-pm fs (vectorize-poly r))) \subseteq Keys
(Poly-Mapping.range (vectorize-poly r))
    by (rule Keys-mono)
    also have ... = pp-of-term'keys r by (fact Keys-range-vectorize-poly)
    finally obtain v where v\inkeys r and v0 = pp-of-term v ..
    from this(2) have t=pp-of-term v + s by (simp only: t)
    with }\langlev\in\mathrm{ keys }r\rangle\langles\in\mathrm{ Keys (set fs)> show ?thesis ..
qed
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 (simp add: punit.dgrad-p-set-le-def Keys-insert, rule dgrad-set-leI)
    fix }
    assume t\in keys (rep-list r)
    then obtain vs1 where v\in keys r and s1\inKeys (set fs) and t:t=pp-of-term
v+s1
    by (rule keys-rep-list-subset)
    from this(1) have pp-of-term v\inpp-of-term' keys r by fastforce
    with assms(2) obtain s2 where s2 \in Keys (set fs) and d (pp-of-term v) \leqd
s2
```

```
    by (rule dgrad-set-leE)
    from assms(1) have dt=ord-class.max (d (pp-of-term v)) (d s1) unfolding t
    by (rule dickson-gradingD1)
    hence dt=d(pp-of-term v) \veedt=d s1 by (simp add:ord-class.max-def)
    thus }\existss\in\mathrm{ Keys (set fs).dt 
    proof
    assume dt=d(pp-of-term v)
    with <d (pp-of-term v) \leqd s2\rangle have dt\leqd s2 by simp
    with <s2 \in Keys (set fs)\rangle show ?thesis ..
    next
    assume d t=d s1
    hence dt\leqd s1 by simp
    with <s1 \in Keys (set fs)\rangle show ?thesis ..
    qed
qed
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 (rule punit.dgrad-p-set-leI, elim imageE, simp)
    fix f
    assume f}\in
    have pp-of-term'keys f\subseteqpp-of-term'Keys F by (rule image-mono, rule
keys-subset-Keys, fact)
    hence dgrad-set-le d (pp-of-term 'keys f) (pp-of-term' Keys F) by (rule dgrad-set-le-subset)
    hence dgrad-set-le d (pp-of-term' keys f) (Keys (set fs)) using assms(2) by
(rule dgrad-set-le-trans)
    with assms(1) show punit.dgrad-p-set-le d {rep-list f} (set fs) by (rule dgrad-p-set-le-rep-list)
qed
term Max
definition dgrad-max :: ('a m nat) => nat
    where dgrad-max d = (Max (d`'(insert 0 (Keys (set fs)))))
abbreviation dgrad-max-set d \equivdgrad-p-set d (dgrad-max d)
abbreviation punit-dgrad-max-set d \equiv punit.dgrad-p-set d (dgrad-max d)
lemma dgrad-max-0:d 0 \leqdgrad-max d
proof -
    from finite-Keys have finite (d' 'insert 0 (Keys (set fs))) by auto
    moreover have d 0 & d' insert 0 (Keys (set fs)) by blast
    ultimately show ?thesis unfolding dgrad-max-def by (rule Max-ge)
qed
lemma dgrad-max-1: set fs \subseteq punit-dgrad-max-set d
proof (cases Keys (set fs)={})
    case True
    show ?thesis
```

```
    proof (rule, rule punit.dgrad-p-setI[simplified])
    fix fv
    assume f}\in\mathrm{ set fs and v}\mathrm{ vekeys}
    with True show d v}\leqdgrad-max d by (auto simp: Keys-def
    qed
next
    case False
    show ?thesis
    proof (rule subset-trans)
        from finite-set show set fs \subseteq punit.dgrad-p-set d (Max (d'(Keys (set fs))))
            by (rule punit.dgrad-p-set-exhaust-expl[simplified])
    next
        from finite-set have finite (Keys (set fs)) by (rule finite-Keys)
        hence finite (d'Keys (set fs)) by (rule finite-imageI)
        moreover from False have 2: d' Keys (set fs) }={{}\mathrm{ by simp
    ultimately have dgrad-max d = ord-class.max (d 0) (Max (d'Keys (set fs)))
        by (simp add: dgrad-max-def)
    hence Max (d'(Keys (set fs))) \leqdgrad-max d by simp
    thus punit.dgrad-p-set d (Max (d'(Keys (set fs)))) \subseteq punit-dgrad-max-set d
        by (rule punit.dgrad-p-set-subset)
    qed
qed
lemma dgrad-max-2:
    assumes dickson-grading d and r\indgrad-max-set d
    shows rep-list r f punit-dgrad-max-set d
proof (rule punit.dgrad-p-setI[simplified])
    fix }
    assume t\in keys (rep-list r)
    then obtain vs where v\in keys r and s\inKeys (set fs) and t:t=pp-of-term
v+s
    by (rule keys-rep-list-subset)
    from assms(2) <v \in keys r> have d (pp-of-term v) \leq dgrad-max d by (rule
dgrad-p-setD)
    moreover have ds\leqdgrad-max d by (simp add: <s\inKeys (set fs)>dgrad-max-def
finite-Keys)
    ultimately show d t \leq dgrad-max d by (simp add: t dickson-gradingD1[OF
assms(1)])
qed
corollary dgrad-max-3:
    assumes dickson-grading d and F\subseteqdgrad-max-set d
    shows rep-list ' F\subseteq punit-dgrad-max-set d
proof (rule, elim imageE, simp)
    fix f
    assume f}\in
    hence f}\in\mathrm{ dgrad-p-set d (dgrad-max d) using assms(2) ..
    with assms(1) show rep-list f\in punit.dgrad-p-set d (dgrad-max d) by (rule
dgrad-max-2)
```


## qed

lemma punit-dgrad-max-set-subset-dgrad-p-set:
assumes dickson-grading $d$ and set $f s \subseteq$ punit.dgrad-p-set $d m$ and $\neg$ set $f s \subseteq$ $\{0\}$
shows punit-dgrad-max-set $d \subseteq$ punit.dgrad-p-set $d m$
proof (rule punit.dgrad-p-set-subset)
show dgrad-max $d \leq m$ unfolding dgrad-max-def
proof (rule Max.boundedI)
show finite ( $d$ ' insert 0 (Keys ( set fs))) by (simp add: finite-Keys)
next
show $d$ ' insert 0 (Keys $($ set $f s)) \neq\{ \}$ by simp
next
fix $a$
assume $a \in d^{\prime}$ insert 0 (Keys (set fs))
then obtain $t$ where $t \in \operatorname{insert} 0($ Keys $($ set fs)) and $a=d t$..
from this(1) show $a \leq m$ unfolding $\langle a=d t\rangle$
proof
assume $t=0$
from $\operatorname{assms}(3)$ obtain $f$ where $f \in$ set $f s$ and $f \neq 0$ by auto
from this(1) assms(2) have $f \in$ punit.dgrad-p-set $d m$..
from $\langle f \neq 0\rangle$ have keys $f \neq\{ \}$ by simp
then obtain $s$ where $s \in$ keys $f$ by blast
have $d s=d(t+s)$ by ( $\operatorname{simp}$ add: $\langle t=0\rangle$ )
also from $\operatorname{assms}(1)$ have $\ldots=\operatorname{ord}$-class.max $(d t)(d s)$ by (rule dick-son-gradingD1)
finally have $d t \leq d s$ by (simp add: max-def)
also from $\langle f \in$ punit.dgrad-p-set $d m\rangle\langle s \in$ keys $f\rangle$ have $\ldots \leq m$
by (rule punit.dgrad-p-setD[simplified])
finally show $d t \leq m$.
next
assume $t \in$ Keys (set fs)
then obtain $f$ where $f \in$ set $f s$ and $t \in$ keys $f$ by (rule in-KeysE)
from this(1) assms(2) have $f \in$ punit.dgrad-p-set $d m$..
thus $d t \leq m$ using $\langle t \in$ keys $f\rangle$ by (rule punit.dgrad-p-set $D[$ simplified])
qed
qed
qed
definition dgrad-sig-set' $::$ nat $\Rightarrow\left({ }^{\prime} a \Rightarrow\right.$ nat $) \Rightarrow\left({ }^{\prime} t \Rightarrow{ }_{0}{ }^{\prime} b\right)$ set where dgrad-sig-set' $j d=d g r a d-m a x-s e t ~ d \cap$ sig-inv-set ${ }^{\prime} j$
abbreviation dgrad-sig-set $\equiv$ dgrad-sig-set' $($ length $f s)$
lemma dgrad-sig-set-set-mono: $i \leq j \Longrightarrow d g r a d$-sig-set' $i d \subseteq d g r a d$-sig-set' $j d$ by (auto simp: dgrad-sig-set'-def dest: sig-inv-set-mono)
lemma dgrad-sig-set-closed-uminus: $r \in d g r a d-s i g-s e t ' ~ j d \Longrightarrow-r \in d g r a d-s i g-s e t ' ~$ jd
unfolding dgrad-sig-set'-def by (auto intro: dgrad-p-set-closed-uminus sig-inv-set-closed-uminus)
lemma dgrad-sig-set-closed-plus:
$r \in d g r a d-s i g-s e t^{\prime} j d \Longrightarrow s \in d g r a d-s i g-s e t^{\prime} j d \Longrightarrow r+s \in d g r a d-s i g-s e t^{\prime} j d$
unfolding dgrad-sig-set'-def by (auto intro: dgrad-p-set-closed-plus sig-inv-set-closed-plus)
lemma dgrad-sig-set-closed-minus: $r \in d g r a d-s i g-s e t^{\prime} j d \Longrightarrow s \in d g r a d-s i g-s e t^{\prime} j d \Longrightarrow r-s \in d g r a d-s i g-s e t^{\prime} j d$
unfolding dgrad-sig-set'-def by (auto intro: dgrad-p-set-closed-minus sig-inv-set-closed-minus)
lemma dgrad-sig-set-closed-monom-mult:
assumes dickson-grading $d$ and $d t \leq d g r a d-m a x d$
shows $p \in d g r a d-s i g-s e t^{\prime} j d \Longrightarrow$ monom-mult $c$ t $p \in d g r a d$-sig-set ${ }^{\prime} j d$
unfolding dgrad-sig-set'-def by (auto intro: assms dgrad-p-set-closed-monom-mult
sig-inv-set-closed-monom-mult)
lemma dgrad-sig-set-closed-monom-mult-zero: $p \in$ dgrad-sig-set' $j d \Longrightarrow$ monom-mult $c 0 p \in$ dgrad-sig-set $^{\prime} j d$ unfolding dgrad-sig-set'-def by (auto intro: dgrad-p-set-closed-monom-mult-zero sig-inv-set-closed-monom-mult)
lemma dgrad-sig-set-closed-mult-scalar:
dickson-grading $d \Longrightarrow p \in$ punit-dgrad-max-set $d \Longrightarrow r \in$ dgrad-sig-set' $j d \Longrightarrow$
$p \odot r \in$ dgrad-sig-set' $j d$
unfolding dgrad-sig-set'-def by (auto intro: dgrad-p-set-closed-mult-scalar sig-inv-set-closed-mult-scalar)
lemma dgrad-sig-set-closed-monomial:
assumes $d(p p$-of-term $u) \leq d g r a d-m a x d$ and component-of-term $u<j$
shows monomial c $u \in d g r a d-s i g-s e t ' j d$
proof (simp add: dgrad-sig-set'-def, rule)
show monomial c $u \in$ dgrad-max-set $d$
proof (rule dgrad-p-setI)
fix $v$
assume $v \in$ keys (monomial c $u$ )
also have $\ldots \subseteq\{u\}$ by $\operatorname{simp}$
finally show $d$ (pp-of-term $v) \leq d g r a d-m a x ~ d$ using assms(1) by simp
qed
next
show monomial c $u \in \operatorname{sig}$-inv-set' $j$
proof (rule sig-inv-setI')
fix $v$
assume $v \in$ keys (monomial cu)
also have $\ldots \subseteq\{u\}$ by $\operatorname{simp}$
finally show component-of-term $v<j$ using assms(2) by simp
qed
qed
lemma rep-list-in-ideal-dgrad-sig-set:
$r \in$ dgrad-sig-set' $j d \Longrightarrow$ rep-list $r \in$ ideal $($ set (take $j f s))$
by (auto simp: dgrad-sig-set'-def dest: rep-list-in-ideal-sig-inv-set)
lemma in-idealE-rep-list-dgrad-sig-set-take:
assumes hom-grading $d$ and $p \in$ punit-dgrad-max-set $d$ and $p \in$ ideal (set (take $j f s)$ )
obtains $r$ where $r \in d g r a d$-sig-set $d$ and $r \in d g r a d-s i g-s e t ' ~ j d$ and $p=r e p-l i s t$ $r$
proof -
let $? f s=$ take $j f s$
from set-take-subset dgrad-max-1 have set ?fs $\subseteq$ punit-dgrad-max-set d
by (rule subset-trans)
with $\operatorname{assms}(1)$ obtain $r 0$ where $r 0$ : keys r0 $\subseteq$ set ?fs
and 1: Poly-Mapping.range r0 $\subseteq$ punit-dgrad-max-set $d$ and $p: p=$ ideal.rep
r0
using $\operatorname{assms}(2,3)$ by (rule in-idealE-rep-dgrad-p-set)
define $q$ where $q=i d x$-pm-of-pm? ${ }^{\text {fs } r 0}$
have keys $q \subseteq\{0 . .<$ length ?fs $\}$ unfolding $q$-def by (rule keys-idx-pm-of-pm-subset)
also have $\ldots \subseteq\{0 . .<j\}$ by fastforce
finally have keys-q: keys $q \subseteq\{0 . .<j\}$.
have $*$ : atomize-poly $q \in$ dgrad-max-set $d$
proof
fix $v$
assume $v \in$ keys (atomize-poly $q$ )
then obtain $i$ where $i: i \in$ keys $q$
and $v$-in: $v \in(\lambda t$. term-of-pair $(t, i))$ 'keys (lookup q i)
unfolding keys-atomize-poly ..
from $i$ keys-idx-pm-of-pm-subset[of ?fs r0] have $i<l e n g t h ? f s$ by (auto simp:
$q-d e f)$
from $v$-in obtain $t$ where $t \in$ keys (lookup $q i$ ) and $v: v=$ term-of-pair $(t$,
i) ..
from this(1)〈i<length ?fs〉 have $t: t \in$ keys (lookup r0 (?fs!i))
by (simp add: lookup-idx-pm-of-pm q-def)
hence lookup r0 $(? f s!i) \neq 0$ by fastforce
hence lookup r0 $(? f s!i) \in$ Poly-Mapping.range r0 by (simp add: in-keys-iff)
hence lookup r0 $(? f s!i) \in$ punit-dgrad-max-set $d$ using 1 ..
hence $d t \leq d g r a d-m a x d$ using $t$ by (rule punit.dgrad- $p$-set $D[$ simplified $]$ )

qed
show ?thesis
proof
have atomize-poly $q \in$ sig-inv-set ${ }^{\prime} j$
by (rule sig-inv-setI, simp add: vectorize-atomize-poly keys-q)
with $*$ show atomize-poly $q \in d g r a d-s i g-s e t ' j d$ unfolding dgrad-sig-set'-def
..
next
from «keys $q \subseteq\{0 . .<$ length ?fs $\}\rangle$ have keys- $q^{\prime}:$ keys $q \subseteq\{0 . .<$ length $f s\}$ by
auto
have atomize-poly $q \in$ sig-inv-set
by (rule sig-inv-setI, simp add: vectorize-atomize-poly keys-q')
with $*$ show atomize-poly $q \in$ dgrad-sig-set d unfolding dgrad-sig-set'-def .. next
from keys- $q$ have $p m$-of-idx-pm fs $q=p m$-of-idx-pm? fs $q$ by (simp only: pm-of-idx-pm-take)
thus $p=$ rep-list (atomize-poly $q$ )
by (simp add: rep-list-def vectorize-atomize-poly pm-of-idx-pm-of-pm[OF r0] $p q-d e f)$
qed
qed
corollary in-idealE-rep-list-dgrad-sig-set:
assumes hom-grading $d$ and $p \in$ punit-dgrad-max-set $d$ and $p \in$ ideal (set $f_{s}$ )
obtains $r$ where $r \in d$ grad-sig-set $d$ and $p=$ rep-list $r$
proof -
from $\operatorname{assms}(3)$ have $p \in$ ideal (set (take (length fs) fs)) by simp
with $\operatorname{assms}(1,2)$ obtain $r$ where $r \in d g r a d$-sig-set $d$ and $p=$ rep-list $r$
by (rule in-idealE-rep-list-dgrad-sig-set-take)
thus ?thesis..
qed
lemma dgrad-sig-setD-lp:
assumes $p \in d g r a d$-sig-set' $j d$
shows $d(l p p) \leq d g r a d-m a x d$
proof (cases $p=0$ )
case True
show ?thesis by (simp add: True min-term-def pp-of-term-of-pair dgrad-max-0)
next
case False

thus ?thesis using False by (rule dgrad-p-setD-lp)
qed
lemma dgrad-sig-setD-lt:
assumes $p \in d g r a d-s i g-s e t^{\prime} j d$ and $p \neq 0$
shows component-of-term (lt $p$ ) $<j$
proof -
from assms have $p \in$ sig-inv-set' $j$ by (simp add: dgrad-sig-set'-def)
thus ?thesis using assms(2) by (rule sig-inv-setD-lt)
qed
lemma dgrad-sig-setD-rep-list-lt:

shows $d$ (punit.lt $($ rep-list p) $) \leq$ dgrad-max $d$
proof (cases rep-list $p=0$ )
case True
show ?thesis by (simp add: True dgrad-max-0)
next
case False

with assms(1) have rep-list $p \in$ punit-dgrad-max-set $d$ by (rule dgrad-max-2) thus ?thesis using False by (rule punit.dgrad-p-setD-lp[simplified])

## qed

definition spp-of :: ('t $\left.\Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)$
where spp-of $r=(l t r$, rep-list $r)$
"spp" stands for "sig-poly-pair".
lemma fst-spp-of: fst (spp-of r) $=l t r$
by (simp add: spp-of-def)
lemma snd-spp-of: snd (spp-of $r$ ) $=$ rep-list $r$
by (simp add: spp-of-def)

### 4.2.1 Signature Reduction

lemma term-is-le-rel-canc-left:
assumes ord-term-lin.is-le-rel rel
shows rel $(t \oplus u)(t \oplus v) \longleftrightarrow$ rel $u v$
using assms
by (rule ord-term-lin.is-le-relE,
auto simp: splus-left-canc dest: ord-term-canc ord-term-strict-canc splus-mono splus-mono-strict)
lemma term-is-le-rel-minus:
assumes ord-term-lin.is-le-rel rel and $s$ adds $t$
shows $r e l((t-s) \oplus u) v \longleftrightarrow r e l(t \oplus u)(s \oplus v)$
proof -
from assms(2) have eq: $s+(t-s)=t$ unfolding add.commute[of $s]$ by (rule adds-minus)
from $\operatorname{assms}(1)$ have $r e l((t-s) \oplus u) v=r e l(s \oplus((t-s) \oplus u))(s \oplus v)$ by (simp only: term-is-le-rel-canc-left)
also have $\ldots=\operatorname{rel}(t \oplus u)(s \oplus v)$ by (simp only: splus-assoc[symmetric] eq)
finally show ?thesis.
qed
lemma term-is-le-rel-minus-minus:
assumes ord-term-lin.is-le-rel rel and $a$ adds $t$ and $b$ adds $t$
shows rel $((t-a) \oplus u)((t-b) \oplus v) \longleftrightarrow r e l(b \oplus u)(a \oplus v)$
proof -
from $\operatorname{assms}(2)$ have eq1: $a+(t-a)=t$ unfolding add.commute[of a] by (rule adds-minus)
from $\operatorname{assms}(3)$ have eq2: $b+(t-b)=t$ unfolding add.commute[of $b]$ by (rule adds-minus)
from $\operatorname{assms}(1)$ have $r e l((t-a) \oplus u)((t-b) \oplus v)=r e l((a+b) \oplus((t-a)$ $\oplus u))((a+b) \oplus((t-b) \oplus v))$
by (simp only: term-is-le-rel-canc-left)
also have $\ldots=\operatorname{rel}((t+b) \oplus u)((t+a) \oplus v)$ unfolding splus-assoc[symmetric] by (metis (no-types, lifting) add.assoc add.commute eq1 eq2)

```
    also from assms(1) have ... = rel (b\oplusu) (a\oplusv) by (simp only: splus-assoc
term-is-le-rel-canc-left)
    finally show ?thesis.
qed
lemma pp-is-le-rel-canc-right:
    assumes ordered-powerprod-lin.is-le-rel rel
    shows rel (s+u)(t+u)\longleftrightarrow relst
    using assms
    by (rule ordered-powerprod-lin.is-le-relE, auto dest: ord-canc ord-strict-canc plus-monotone
plus-monotone-strict)
lemma pp-is-le-rel-canc-left:ordered-powerprod-lin.is-le-rel rel \Longrightarrowrel (t+u) (t
+v)\longleftrightarrow rel uv
    by (simp add: add.commute[of t] pp-is-le-rel-canc-right)
```



```
('t #00'b) => ('t =00'b) =>''a=> bool
    where sig-red-single sing-reg top-tail p qft\longleftrightarrow
    (rep-list f}\not=0\wedge lookup (rep-list p) (t + punit.lt (rep-list f)) # = ^ ^
    q=p - monom-mult ((lookup (rep-list p) (t + punit.lt (rep-list f)))
/ punit.lc (rep-list f)) tf ^
    ord-term-lin.is-le-rel sing-reg ^ ordered-powerprod-lin.is-le-rel top-tail
^
    sing-reg (t \opluslt f) (lt p)^top-tail (t + punit.lt (rep-list f)) (punit.lt
(rep-list p)))
```

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

- If sing-reg is (=), the reduction is singular.
- If sing-reg is $\left(\prec_{t}\right)$, the reduction is regular.
- If sing-reg is $\left(\preceq_{t}\right)$, the reduction is an arbitrary signature-reduction.
- If top-tail is $(=)$, it is a top reduction.
- If top-tail is $(\prec)$, it is a tail reduction.
- If top-tail is $(\preceq)$, the reduction is an arbitrary signature-reduction.
definition sig-red :: $\left({ }^{\prime} t \Rightarrow{ }^{\prime} t \Rightarrow\right.$ bool $) \Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow\left({ }^{\prime} t \Rightarrow{ }^{\prime} b\right)$ set $\Rightarrow\left({ }^{\prime} t\right.$ $\left.\Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow$ bool
where sig-red sing-reg top-tail $F p q \longleftrightarrow(\exists f \in F . \exists t$. sig-red-single sing-reg top-tail p q ft)
definition is-sig-red :: $\left({ }^{\prime} t \Rightarrow{ }^{\prime} t \Rightarrow\right.$ bool $) \Rightarrow\left({ }^{\prime} a \Rightarrow '^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ set $\Rightarrow$ $\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow$ bool
where is-sig-red sing-reg top-tail $F p \longleftrightarrow(\exists q$. 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)) tf
and ord-term-lin.is-le-rel sing-reg and ordered-powerprod-lin.is-le-rel top-tail and sing-reg $(t \oplus l t f$ ) (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 qft
unfolding sig-red-single-def using assms by blast
lemma sig-red-singleD1:
assumes sig-red-single sing-reg top-tail p qft
shows rep-list $f \neq 0$
using assms unfolding sig-red-single-def by blast
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 $p$ )
using assms unfolding sig-red-single-def by (simp add: in-keys-iff)
lemma sig-red-singleD3:
assumes sig-red-single sing-reg top-tail $p q f t$
shows $q=p$ - monom-mult $((l o o k u p(r e p-l i s t ~ p)(t+$ punit.lt (rep-list $f))) /$
punit.lc (rep-list f)) tf
using assms unfolding sig-red-single-def by blast
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
using assms unfolding sig-red-single-def by blast
lemma sig-red-singleD5:
assumes sig-red-single sing-reg top-tail p qft
shows ordered-powerprod-lin.is-le-rel top-tail
using assms unfolding sig-red-single-def by blast
lemma sig-red-singleD6:
assumes sig-red-single sing-reg top-tail p qft
shows sing-reg $(t \oplus l t f)$ (lt p)
using assms unfolding sig-red-single-def by blast
lemma sig-red-singleD7:
assumes sig-red-single sing-reg top-tail p qft
shows top-tail $(t+$ punit.lt (rep-list f)) (punit.lt (rep-list p))
using assms unfolding sig-red-single-def by blast
lemma sig-red-singleD8:

```
    assumes sig-red-single sing-reg top-tail p qft
    shows t}\oplusltf\mp@subsup{\preceq}{t}{}lt
proof -
    from assms have ord-term-lin.is-le-rel sing-reg and sing-reg (t\opluslt f) (lt p)
    by (rule sig-red-singleD4, rule sig-red-singleD6)
    thus ?thesis by (rule ord-term-lin.is-le-rel-le)
qed
lemma sig-red-singleD9:
    assumes sig-red-single sing-reg top-tail p qft
    shows t + punit.lt (rep-list f)\preceq punit.lt (rep-list p)
proof -
    from assms have ordered-powerprod-lin.is-le-rel top-tail
        and top-tail (t + punit.lt (rep-list f)) (punit.lt (rep-list p))
        by (rule sig-red-singleD5, rule sig-red-singleD7)
    thus ?thesis by (rule ordered-powerprod-lin.is-le-rel-le)
qed
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 qft\Longrightarrow punit.red-single (rep-list p)(rep-list q)
(rep-list f) t
    by (simp add: sig-red-single-def punit.red-single-def rep-list-minus rep-list-monom-mult)
lemma sig-red-single-regular-lt:
    assumes sig-red-single ( }\mp@subsup{\prec}{t}{})\mathrm{ top-tail p qft
    shows lt q=lt p
proof -
    let ?f = monom-mult ((lookup (rep-list p) (t + punit.lt (rep-list f))) / punit.lc
(rep-list f)) tf
    from assms have lt: t\oplusltf\mp@subsup{\prec}{t}{}lt p and q: q=p - ?f
        by (rule sig-red-singleD6, rule sig-red-singleD3)
    from lt-monom-mult-le lt have lt ?f }\mp@subsup{\prec}{t}{}lt p by (rule ord-term-lin.order.strict-trans1
    thus ?thesis unfolding q}\mathrm{ by (rule lt-minus-eqI-2)
qed
lemma sig-red-single-regular-lc:
    assumes sig-red-single ( }\mp@subsup{\prec}{t}{})\mathrm{ top-tail p qft
    shows lc q=lc p
proof -
    from assms have lt q=lt p by (rule sig-red-single-regular-lt)
    from assms have lt:t\opluslt f < lt p
            and q: q = p-monom-mult ((lookup (rep-list p) (t+ punit.lt (rep-list f))) /
punit.lc (rep-list f)) tf
            (is - = - - ?f) by (rule sig-red-singleD6, rule sig-red-singleD3)
    from lt-monom-mult-le lt have lt ?f }\mp@subsup{\prec}{t}{}lt p by (rule ord-term-lin.order.strict-trans1)
```

```
    hence lookup ?f (lt p) = 0 using lt-max ord-term-lin.leD by blast
    thus ?thesis unfolding lc-def <lt q=lt p\rangle by (simp add: q lookup-minus)
qed
lemma sig-red-single-lt:
    assumes sig-red-single sing-reg top-tail p qft
    shows lt q}\mp@subsup{\preceq}{t}{}lt 
proof -
    from assms have lt: t\opluslt f\preceq\preceqt lt p
        and q=p - monom-mult ((lookup (rep-list p) (t + punit.lt (rep-list f))) /
punit.lc (rep-list f)) tf
    by (rule sig-red-singleD8, rule sig-red-singleD3)
    from this(2) have q:q=p+ monom-mult (- (lookup (rep-list p) (t+ punit.lt
(rep-list f))) / punit.lc (rep-list f)) tf
    (is - = - + ?f) by (simp add: monom-mult-uminus-left)
    from lt-monom-mult-le lt have 1: lt ?f }\mp@subsup{\preceq}{t}{}\mathrm{ lt p by (rule ord-term-lin.order.trans)
    have lt q}\mp@subsup{\preceq}{t}{}\mathrm{ ord-term-lin.max (lt p) (lt ?f) unfolding q by (fact lt-plus-le-max)
    also from 1 have ord-term-lin.max (lt p) (lt ?f) = lt p by (rule ord-term-lin.max.absorb1)
    finally show ?thesis.
qed
lemma sig-red-single-lt-rep-list:
    assumes sig-red-single sing-reg top-tail p qft
    shows punit.lt (rep-list q)\preceq punit.lt (rep-list p)
proof -
    from assms have punit.red-single (rep-list p) (rep-list q) (rep-list f)t
        by (rule sig-red-single-red-single)
    hence punit.ord-strict-p (rep-list q) (rep-list p) by (rule punit.red-single-ord)
    hence punit.ord-p (rep-list q) (rep-list p) by simp
    thus ?thesis by (rule punit.ord-p-lt)
qed
lemma sig-red-single-tail-lt-in-keys-rep-list:
    assumes sig-red-single sing-reg (\prec) pqft
    shows punit.lt (rep-list p) \in keys (rep-list q)
proof -
    from assms have q=p - monom-mult ((lookup (rep-list p) (t+punit.lt (rep-list
f))) / punit.lc (rep-list f))tf
    by (rule sig-red-singleD3)
    hence q: q=p+monom-mult (- (lookup (rep-list p) (t+ punit.lt (rep-list f)))
/ punit.lc (rep-list f)) tf
    by (simp add: monom-mult-uminus-left)
    show ?thesis unfolding q rep-list-plus rep-list-monom-mult
    proof (rule in-keys-plusI1)
    from assms have t+ punit.lt (rep-list f) \in keys (rep-list p) by (rule sig-red-singleD2)
        hence rep-list p\not=0 by auto
        thus punit.lt (rep-list p)\in keys (rep-list p) by (rule punit.lt-in-keys)
    next
    show punit.lt (rep-list p) }
```

```
            keys (punit.monom-mult (- lookup (rep-list p) (t + punit.lt (rep-list f)) /
punit.lc (rep-list f))t(rep-list f))
            (is - & keys?f)
    proof
        assume punit.lt (rep-list p)\in keys ?f
        hence punit.lt (rep-list p)\preceq punit.lt ?f by (rule punit.lt-max-keys)
    also have ... \preceqt+ punit.lt (rep-list f) by (fact punit.lt-monom-mult-le[simplified])
        also from assms have ... \prec punit.lt (rep-list p) by (rule sig-red-singleD7)
        finally show False by simp
    qed
    qed
qed
corollary sig-red-single-tail-lt-rep-list:
    assumes sig-red-single sing-reg (\prec) pqft
    shows punit.lt (rep-list q) = punit.lt (rep-list p)
proof (rule ordered-powerprod-lin.order-antisym)
    from assms show punit.lt (rep-list q)\preceq punit.lt (rep-list p) by (rule sig-red-single-lt-rep-list)
next
    from assms have punit.lt (rep-list p) \in keys (rep-list q) by (rule sig-red-single-tail-lt-in-keys-rep-list)
    thus punit.lt (rep-list p)\preceq punit.lt (rep-list q) by (rule punit.lt-max-keys)
qed
lemma sig-red-single-tail-lc-rep-list:
    assumes sig-red-single sing-reg (\prec) pqft
    shows punit.lc (rep-list q) = punit.lc (rep-list p)
proof -
    from assms have *: punit.lt (rep-list q) = punit.lt (rep-list p)
        by (rule sig-red-single-tail-lt-rep-list)
    from assms have lt: t + punit.lt (rep-list f) \prec punit.lt (rep-list p)
        and q: q = p-monom-mult ((lookup (rep-list p) (t+ punit.lt (rep-list f))) /
punit.lc (rep-list f)) tf
    (is - = - - ?f) by (rule sig-red-singleD7, rule sig-red-singleD3)
    from punit.lt-monom-mult-le[simplified] lt have punit.lt (rep-list ?f) \prec punit.lt
(rep-list p)
    unfolding rep-list-monom-mult by (rule ordered-powerprod-lin.order.strict-trans1)
    hence lookup (rep-list ?f) (punit.lt (rep-list p)) = 0
        using punit.lt-max ordered-powerprod-lin.leD by blast
    thus ?thesis unfolding punit.lc-def * by (simp add: q lookup-minus rep-list-minus
punit.lc-def)
qed
lemma sig-red-single-top-lt-rep-list:
    assumes sig-red-single sing-reg (=) pqft and rep-list q\not=0
    shows punit.lt (rep-list q) \prec punit.lt (rep-list p)
proof -
    from assms(1) have rep-list f}\not=0\mathrm{ and in-keys: t + punit.lt (rep-list f) }\in\mathrm{ keys
(rep-list p)
    and lt: t + punit.lt (rep-list f) = punit.lt (rep-list p)
```

and $q=p$ - monom-mult $(($ lookup $($ rep-list $p)(t+$ punit.lt $($ rep-list $f))) /$ punit.lc (rep-list f)) tf
by (rule sig-red-single $D$ ) +
from this(4) have $q: q=p+$ monom-mult ( - (lookup (rep-list $p)(t+$ punit.lt $($ rep-list f))) / punit.lc (rep-list f)) $t f$
(is $-=-+$ monom-mult ?c - -) by (simp add: monom-mult-uminus-left)
from $\langle$ rep-list $f \neq 0\rangle$ have punit.lc (rep-list $f) \neq 0$ by (rule punit.lc-not-0)
from $\operatorname{assms}(2)$ have $*$ : rep-list $p+$ punit.monom-mult ?c $t$ (rep-list $f) \neq 0$
by (simp add: q rep-list-plus rep-list-monom-mult)
from in-keys have lookup (rep-list p) $(t+$ punit.lt $($ rep-list $f)) \neq 0$
by (simp add: in-keys-iff)
moreover from $\langle r e p-l i s t f \neq 0\rangle$ have punit.lc (rep-list $f) \neq 0$ by (rule punit.lc-not-0)
ultimately have $? c \neq 0$ by $\operatorname{simp}$
hence punit.lt (punit.monom-mult ?c $t($ rep-list $f))=t+$ punit.lt (rep-list $f)$ using $\langle$ rep-list $f \neq 0\rangle$ by (rule lp-monom-mult)
hence punit.lt (punit.monom-mult ?c $t$ (rep-list f)) $=$ punit.lt (rep-list p) by (simp only: lt)
moreover have punit.lc (punit.monom-mult ?c $t$ (rep-list f)) $=-$ punit.lc (rep-list p)
by (simp add: lt punit.lc-def[symmetric] 〈punit.lc (rep-list f) $\neq 0\rangle$ )
ultimately show ?thesis unfolding rep-list-plus rep-list-monom-mult $q$ by (rule punit.lt-plus-lessI[OF *])
qed
lemma sig-red-single-monom-mult:
assumes sig-red-single sing-reg top-tail p $q f t$ and $c \neq 0$
shows sig-red-single sing-reg top-tail (monom-mult csp)(monom-mult cs q) f
$(s+t)$
proof -
from assms(1) have a: ord-term-lin.is-le-rel sing-reg and b: ordered-powerprod-lin.is-le-rel top-tail
by (rule sig-red-singleD4, rule sig-red-singleD5)
have eq1: $(s+t) \oplus l t f=s \oplus(t \oplus l t f)$ by (simp only: splus-assoc)
from assms (1) have $1: t+$ punit.lt (rep-list $f$ ) $\in$ keys (rep-list p) by (rule sig-red-singleD2)
hence rep-list $p \neq 0$ by auto
hence $p \neq 0$ by (auto simp: rep-list-zero)
with assms(2) have eq2:lt (monom-mult csp)=s $\oplus$ lt $p$ by (rule lt-monom-mult)
show ?thesis
proof (rule sig-red-singleI)
from $\operatorname{assms}(1)$ show rep-list $f \neq 0$ by (rule sig-red-singleD1)
next
show $s+t+$ punit.lt $($ rep-list $f) \in$ keys (rep-list (monom-mult csp))
by (auto simp: rep-list-monom-mult punit.keys-monom-mult[OF assms(2)]
ac-simps intro: 1)
next
from assms(1) have $q: q=p$ - monom-mult ((lookup (rep-list $p)(t+$ punit.lt (rep-list f))) / punit.lc (rep-list f)) tf
by (rule sig-red-singleD3)

```
    show monom-mult c s q}
            monom-mult c s p-
                monom-mult (lookup (rep-list (monom-mult c s p)) (s+t+ punit.lt
(rep-list f)) / punit.lc (rep-list f)) (s+t)f
    by (simp add: q monom-mult-dist-right-minus ac-simps rep-list-monom-mult
                punit.lookup-monom-mult-plus[simplified] monom-mult-assoc)
    next
        from assms(1) have sing-reg ( }t\oplusltf)(lt p) by (rule sig-red-singleD6
        thus sing-reg ((s+t)\opluslt f) (lt (monom-mult c s p))
            by (simp only: eq1 eq2 term-is-le-rel-canc-left[OF a])
    next
    from assms(1) have top-tail (t + punit.lt (rep-list f)) (punit.lt (rep-list p))
            by (rule sig-red-singleD7)
            thus top-tail (s+t+ punit.lt (rep-list f)) (punit.lt (rep-list (monom-mult c s
p)))
            by (simp add: rep-list-monom-mult punit.lt-monom-mult[OF assms(2) <rep-list
p\not=0>] add.assoc pp-is-le-rel-canc-left[OF b])
    qed (fact a, fact b)
qed
lemma sig-red-single-sing-reg-cases:
    sig-red-single ( }\mp@subsup{\Omega}{t}{})\mathrm{ top-tail p qft = (sig-red-single (=) top-tail p qft V sig-red-single
(}\mp@subsup{\prec}{t}{})\mathrm{ top-tail p q f t)
    by (auto simp: sig-red-single-def)
corollary sig-red-single-sing-regI:
    assumes sig-red-single sing-reg top-tail p qft
    shows sig-red-single ( }\mp@subsup{\preceq}{t}{})\mathrm{ top-tail p qft
proof -
    from assms have ord-term-lin.is-le-rel sing-reg by (rule sig-red-singleD)
    with assms show ?thesis unfolding ord-term-lin.is-le-rel-def
        by (auto simp: sig-red-single-sing-reg-cases)
qed
lemma sig-red-single-top-tail-cases:
    sig-red-single sing-reg (\preceq) pqft=(sig-red-single sing-reg (=) pqft\vee sig-red-single
sing-reg(\prec) pqft)
    by (auto simp: sig-red-single-def)
corollary sig-red-single-top-tailI:
    assumes sig-red-single sing-reg top-tail p qft
    shows sig-red-single sing-reg (\preceq) pqft
proof -
    from assms have ordered-powerprod-lin.is-le-rel top-tail by (rule sig-red-singleD)
    with assms show ?thesis unfolding ordered-powerprod-lin.is-le-rel-def
        by (auto simp: sig-red-single-top-tail-cases)
qed
lemma dgrad-max-set-closed-sig-red-single:
```

assumes dickson-grading $d$ and $p \in d g r a d-m a x-s e t ~ d ~ a n d ~ f \in d g r a d-m a x-s e t ~ d ~$ and sig-red-single sing-red top-tail $p q f t$
shows $q \in$ dgrad-max-set $d$
proof -
let ?f $=$ monom-mult (lookup (rep-list p) $(t+$ punit.lt (rep-list f)) / punit.lc (rep-list f)) tf
from assms(4) have $t: t+$ punit.lt (rep-list $f) \in$ keys (rep-list $p$ ) and $q: q=p$

- ?f
by (rule sig-red-singleD2, rule sig-red-singleD3)
from $\operatorname{assms}(1,2)$ have rep-list $p \in$ punit-dgrad-max-set $d$ by (rule dgrad-max-2)
show ?thesis unfolding $q$ using assms(2)
proof (rule dgrad-p-set-closed-minus)
from $\operatorname{assms}(1)-\operatorname{assms}(3)$ show ?f $\in$ dgrad-max-set d
proof (rule dgrad-p-set-closed-monom-mult)
from assms(1) have $d t \leq d(t+$ punit.lt (rep-list f)) by (simp add: dickson-gradingD1)
also from $\langle$ rep-list $p \in$ punit-dgrad-max-set $d\rangle t$ have $\ldots \leq$ dgrad-max $d$
by (rule punit.dgrad-p-setD[simplified $]$ )
finally show $d t \leq d g r a d-m a x d$.
qed
qed
qed
lemma sig-inv-set-closed-sig-red-single:
assumes $p \in$ sig-inv-set and $f \in$ sig-inv-set and sig-red-single sing-red top-tail
pqft
shows $q \in$ sig-inv-set
proof -
let ?f $=$ monom-mult (lookup (rep-list p) $(t+$ punit.lt (rep-list $f)) /$ punit.lc (rep-list f)) $t f$
from $\operatorname{assms}(3)$ have $t: t+$ punit.lt (rep-list $f) \in$ keys (rep-list $p$ ) and $q: q=p$ - ?f
by (rule sig-red-singleD2, rule sig-red-singleD3)
show ?thesis unfolding $q$ using assms(1)
proof (rule sig-inv-set-closed-minus)
from $\operatorname{assms}(2)$ show ?f $\in \operatorname{sig}$-inv-set by (rule sig-inv-set-closed-monom-mult)
qed
qed
corollary dgrad-sig-set-closed-sig-red-single:
assumes dickson-grading $d$ and $p \in d g r a d$-sig-set $d$ and $f \in d g r a d$-sig-set $d$
and sig-red-single sing-red top-tail $p q f t$
shows $q \in$ dgrad-sig-set $d$
using assms unfolding dgrad-sig-set'-def
by (auto intro: dgrad-max-set-closed-sig-red-single sig-inv-set-closed-sig-red-single)
lemma sig-red-regular-lt: sig-red $\left(\prec_{t}\right)$ top-tail $F p q \Longrightarrow l t q=l t p$
by (auto simp: sig-red-def intro: sig-red-single-regular-lt)
lemma sig-red-regular-lc: sig-red $\left(\prec_{t}\right)$ top-tail $F p q \Longrightarrow l c q=l c p$ by (auto simp: sig-red-def intro: sig-red-single-regular-lc)
lemma sig-red-lt: sig-red sing-reg top-tail $F p q \Longrightarrow l t ~ q \preceq_{t}$ lt $p$ by (auto simp: sig-red-def intro: sig-red-single-lt)
lemma sig-red-tail-lt-rep-list: sig-red sing-reg ( $\prec$ ) Fpq $q$ punit.lt (rep-list $q$ ) $=$ punit.lt (rep-list p)
by (auto simp: sig-red-def intro: sig-red-single-tail-lt-rep-list)
lemma sig-red-tail-lc-rep-list: sig-red sing-reg ( $\prec$ ) Fp $q \Longrightarrow$ punit.lc $($ rep-list $q)=$ punit.lc (rep-list p)
by (auto simp: sig-red-def intro: sig-red-single-tail-lc-rep-list)
lemma sig-red-top-lt-rep-list:
sig-red sing-reg $(=)$ F p $q \Longrightarrow$ rep-list $q \neq 0 \Longrightarrow$ punit.lt (rep-list $q) \prec$ punit.lt (rep-list p)
by (auto simp: sig-red-def intro: sig-red-single-top-lt-rep-list)
lemma sig-red-lt-rep-list: sig-red sing-reg top-tail Fp $q \Longrightarrow$ punit.lt (rep-list $q$ ) $\preceq$ punit.lt (rep-list p)
by (auto simp: sig-red-def intro: sig-red-single-lt-rep-list)
lemma sig-red-red: sig-red sing-reg top-tail $F p q \Longrightarrow$ punit.red (rep-list' $F$ ) (rep-list p) (rep-list q)
by (auto simp: sig-red-def punit.red-def dest: sig-red-single-red-single)
lemma sig-red-monom-mult:
sig-red sing-reg top-tail Fp $q \Longrightarrow c \neq 0 \Longrightarrow$ sig-red sing-reg top-tail $F$ (monom-mult $c s p)($ monom-mult $c s q)$
by (auto simp: sig-red-def punit.red-def dest: sig-red-single-monom-mult)
lemma sig-red-sing-reg-cases:
sig-red $\left(\preceq_{t}\right)$ top-tail F p $q=\left(\right.$ sig-red $(=)$ top-tail Fp $q \vee$ sig-red $\left(\prec_{t}\right)$ top-tail $F$ p q)
by (auto simp: sig-red-def sig-red-single-sing-reg-cases)
corollary sig-red-sing-regI: sig-red sing-reg top-tail Fpq sig-red $\left(\preceq_{t}\right)$ top-tail Fpq
by (auto simp: sig-red-def intro: sig-red-single-sing-regI)
lemma sig-red-top-tail-cases:
sig-red sing-reg ( $\preceq$ ) Fpq=(sig-red sing-reg (=) Fpqマ sig-red sing-reg ( $\prec$ ) $F$ p q)
by (auto simp: sig-red-def sig-red-single-top-tail-cases)
corollary sig-red-top-tailI: sig-red sing-reg top-tail Fpq sig-red sing-reg ( $\preceq$ ) Fpq
by (auto simp: sig-red-def intro: sig-red-single-top-tailI)

```
lemma sig-red-wf-dgrad-max-set:
    assumes dickson-grading d and F\subseteqdgrad-max-set d
    shows wfP(sig-red sing-reg top-tail F)}\mp@subsup{)}{}{-1-1
proof -
    from assms have rep-list' F\subseteq punit-dgrad-max-set d by (rule dgrad-max-3)
    with assms(1) have wfP (punit.red (rep-list' F))}\mp@subsup{)}{}{-1-1}\mathrm{ by (rule punit.red-wf-dgrad-p-set)
    hence *: #f.\foralli. (punit.red (rep-list'F))}\mp@subsup{)}{}{-1-1}(f(\mathrm{ Suc i)) (f i)
        by (simp add: wf-iff-no-infinite-down-chain[to-pred])
    show ?thesis unfolding wf-iff-no-infinite-down-chain[to-pred]
    proof (rule, elim exE)
        fix seq
        assume }\foralli.(\mathrm{ sig-red sing-reg top-tail F) -1-1 (seq (Suc i)) (seq i)
    hence sig-red sing-reg top-tail F (seq i) (seq (Suc i)) for i by simp
    hence punit.red (rep-list`F) ((rep-list \circ seq) i) ((rep-list ○ seq) (Suc i)) for i
        by (auto intro: sig-red-red)
    hence }\foralli.(\mathrm{ punit.red (rep-list'F))}\mp@subsup{)}{}{-1-1}((\mathrm{ rep-list ○ seq) (Suc i)) ((rep-list ○
seq) i) by simp
    hence }\existsf.\foralli.(\mathrm{ punit.red (rep-list'F))}\mp@subsup{)}{}{-1-1}(f(Suci))(fi)\mathrm{ by blast
    with * show False ..
    qed
qed
lemma dgrad-sig-set-closed-sig-red:
    assumes dickson-grading d and F\subseteqdgrad-sig-set d and p\indgrad-sig-set d
        and sig-red sing-red top-tail F p q
    shows q\indgrad-sig-set d
    using assms by (auto simp: sig-red-def intro: dgrad-sig-set-closed-sig-red-single)
lemma sig-red-mono: sig-red sing-reg top-tail Fpq\LongrightarrowF\subseteq\mp@subsup{F}{}{\prime}\Longrightarrow sig-red sing-reg
top-tail F' p q
    by (auto simp: sig-red-def)
lemma sig-red-Un:
    sig-red sing-reg top-tail (A\cupB) pq\longleftrightarrow (sig-red sing-reg top-tail A p q\vee sig-red
sing-reg top-tail B p q)
    by (auto simp: sig-red-def)
lemma sig-red-subset:
    assumes sig-red sing-reg top-tail F p q and sing-reg = (\preceq, ) \vee sing-reg = (\swarrow}
    shows sig-red sing-reg top-tail {f\inF. sing-reg (lt f) (lt p)} pq
proof -
    from assms(1) obtain ft where f}\inF\mathrm{ and *: sig-red-single sing-reg top-tail p
qft
        unfolding sig-red-def by blast
    have lt f=0 ¢lt f by (simp only: term-simps)
    also from zero-min have ... \preceq}\mp@subsup{\}{t}{}t\opluslt f\mathrm{ by (rule splus-mono-left)
    finally have 1:lt f}\mp@subsup{\preceq}{t}{}t\oplusltf
    from * have 2: sing-reg (t\opluslt f) (lt p) by (rule sig-red-singleD6)
```

```
    from assms(2) have sing-reg (lt f) (lt p)
    proof
        assume sing-reg=(\mp@subsup{\preceq}{t}{})
        with 12 show ?thesis by simp
    next
        assume sing-reg = (\mp@subsup{\prec}{t}{})
        with }12\mathrm{ show ?thesis by simp
    qed
    with}\langlef\inF\rangle\mathrm{ have }f\in{f\inF\mathrm{ . sing-reg (lt f) (lt p)} by simp
    thus ?thesis using * unfolding sig-red-def by blast
qed
lemma sig-red-regular-rtrancl-lt:
    assumes (sig-red ( }\mp@subsup{\prec}{t}{})\mathrm{ top-tail F)** pq
    shows lt q}=lt 
    using assms by (induct, auto dest: sig-red-regular-lt)
lemma sig-red-regular-rtrancl-lc:
    assumes (sig-red ( }\mp@subsup{\prec}{t}{})\mathrm{ top-tail F)** pq
    shows lc q=lc p
    using assms by (induct, auto dest: sig-red-regular-lc)
lemma sig-red-rtrancl-lt:
    assumes (sig-red sing-reg top-tail F)** p q
    shows lt q}\mp@subsup{\preceq}{t}{}lt 
    using assms by (induct, auto dest: sig-red-lt)
lemma sig-red-tail-rtrancl-lt-rep-list:
    assumes (sig-red sing-reg (\prec) F)** p q
    shows punit.lt (rep-list q) = punit.lt (rep-list p)
    using assms by (induct, auto dest: sig-red-tail-lt-rep-list)
lemma sig-red-tail-rtrancl-lc-rep-list:
    assumes (sig-red sing-reg (\prec) F)** p q
    shows punit.lc (rep-list q) = punit.lc (rep-list p)
    using assms by (induct, auto dest: sig-red-tail-lc-rep-list)
lemma sig-red-rtrancl-lt-rep-list:
    assumes (sig-red sing-reg top-tail F)** p q
    shows punit.lt (rep-list q) \preceq punit.lt (rep-list p)
    using assms by (induct, auto dest: sig-red-lt-rep-list)
lemma sig-red-red-rtrancl:
    assumes (sig-red sing-reg top-tail F)** p q
    shows (punit.red (rep-list`F))** (rep-list p) (rep-list q)
    using assms by (induct, auto dest: sig-red-red)
lemma sig-red-rtrancl-monom-mult:
    assumes (sig-red sing-reg top-tail F)** p q
```

```
    shows (sig-red sing-reg top-tail F)** (monom-mult c s p)(monom-mult c s q)
proof (cases c=0)
    case True
    thus ?thesis by simp
next
    case False
    from assms(1) show ?thesis
    proof induct
        case base
        show ?case ..
    next
    case (step y z)
            from step(2) False have sig-red sing-reg top-tail F (monom-mult cls y)
(monom-mult c s z)
            by (rule sig-red-monom-mult)
    with step(3) show ?case ..
    qed
qed
lemma sig-red-rtrancl-sing-regI:(sig-red sing-reg top-tail F)** p q\Longrightarrow(sig-red
(\preceq}\mp@subsup{t}{t}{)}\mathrm{ top-tail F)}\mp@subsup{)}{}{**}p
    by (induct rule: rtranclp-induct, auto dest: sig-red-sing-regI)
lemma sig-red-rtrancl-top-tailI:(sig-red sing-reg top-tail F)** p q\Longrightarrow(sig-red
sing-reg (\preceq) F)** p q
    by (induct rule: rtranclp-induct, auto dest: sig-red-top-tailI)
lemma dgrad-sig-set-closed-sig-red-rtrancl:
    assumes dickson-grading d and F\subseteqdgrad-sig-set d and p}\indgrad-sig-set d
        and (sig-red sing-red top-tail F)** p q
    shows q}\indgrad-sig-set d
    using assms(4, 1, 2, 3) by (induct, auto intro:dgrad-sig-set-closed-sig-red)
lemma sig-red-rtrancl-mono:
    assumes (sig-red sing-reg top-tail F)** pq and F\subseteqF'
    shows (sig-red sing-reg top-tail F}\mp@subsup{F}{}{\prime}\mp@subsup{)}{}{**}p
    using assms(1) by (induct rule: rtranclp-induct, auto dest: sig-red-mono[OF -
assms(2)])
lemma sig-red-rtrancl-subset:
    assumes (sig-red sing-reg top-tail F)** p q and sing-reg = (\preceq
(\prec}\mp@subsup{}{t}{}
    shows (sig-red sing-reg top-tail {f\inF. sing-reg (lt f) (lt p)})** p q
    using assms(1)
proof (induct rule: rtranclp-induct)
    case base
    show ?case by (fact rtranclp.rtrancl-refl)
next
    case (step y z)
```

```
    from step(2) assms(2) have sig-red sing-reg top-tail {f \inF. sing-reg (lt f)(lt
y)} yz
    by (rule sig-red-subset)
    moreover have {f\inF. sing-reg (lt f)(lt y)}\subseteq{f\inF. sing-reg (lt f)(lt p)}
    proof
    fix f
    assume f}\in{f\inF\mathrm{ . sing-reg(lt f) (lt y)}
    hence }f\inF\mathrm{ and 1: sing-reg (lt f)(lt y) by simp-all
    from step(1) have 2: lt }y\mp@subsup{\preceq}{t}{}\mathrm{ lt p by (rule sig-red-rtrancl-lt)
    from assms(2) have sing-reg (lt f) (lt p)
    proof
            assume sing-reg=(\preceq}
            with 12 show ?thesis by simp
    next
            assume sing-reg = (\mp@subsup{\prec}{t}{})
            with }12\mathrm{ show ?thesis by simp
    qed
    with}\langlef\inF\rangle\mathrm{ show }f\in{f\inF\mathrm{ . sing-reg (lt f) (lt p)} by simp
qed
ultimately have sig-red sing-reg top-tail {f f F. sing-reg (lt f) (lt p)} y z
    by (rule sig-red-mono)
    with step(3) show ?case ..
qed
lemma is-sig-red-is-red: is-sig-red sing-reg top-tail F p\Longrightarrow punit.is-red (rep-list`
F)(rep-list p)
    by (auto simp: is-sig-red-def punit.is-red-alt dest: sig-red-red)
lemma is-sig-red-monom-mult:
    assumes is-sig-red sing-reg top-tail Fp and c\not=0
    shows is-sig-red sing-reg top-tail F (monom-mult c s p)
proof -
    from assms(1) obtain q}\mathrm{ where sig-red sing-reg top-tail F p q unfolding is-sig-red-def
    hence sig-red sing-reg top-tail F (monom-mult c s p) (monom-mult c s q)
        using assms(2) by (rule sig-red-monom-mult)
    thus ?thesis unfolding is-sig-red-def ..
qed
lemma is-sig-red-sing-reg-cases:
    is-sig-red ( }\mp@subsup{\preceq}{t}{})\mathrm{ top-tail Fp=(is-sig-red (=) top-tail F p V is-sig-red ( }\mp@subsup{\prec}{t}{})\mathrm{ top-tail
F p)
    by (auto simp: is-sig-red-def sig-red-sing-reg-cases)
corollary is-sig-red-sing-regI: is-sig-red sing-reg top-tail F p\Longrightarrowis-sig-red (\preceq
top-tail F p
    by (auto simp: is-sig-red-def intro: sig-red-sing-regI)
lemma is-sig-red-top-tail-cases:
```

```
    is-sig-red sing-reg (\preceq) Fp=(is-sig-red sing-reg (=) Fp\vee is-sig-red sing-reg (\prec)
F p)
    by (auto simp: is-sig-red-def sig-red-top-tail-cases)
corollary is-sig-red-top-tailI: is-sig-red sing-reg top-tail F p\Longrightarrow is-sig-red sing-reg
(\preceq) Fp
    by (auto simp: is-sig-red-def intro: sig-red-top-tailI)
lemma is-sig-red-singletonI:
    assumes is-sig-red sing-reg top-tail Fr
    obtains f}\mathrm{ where f}\inF\mathrm{ and is-sig-red sing-reg top-tail {f}r
proof -
    from assms obtain r'where sig-red sing-reg top-tail Fr r' unfolding is-sig-red-def
    then obtain ft where f\inF and t: sig-red-single sing-reg top-tail r r'ft
        by (auto simp: sig-red-def)
    have is-sig-red sing-reg top-tail {f} r unfolding is-sig-red-def sig-red-def
    proof (intro exI bexI)
        show }f\in{f}\mathrm{ by simp
    qed fact
    with }\langlef\inF\rangle\mathrm{ show ?thesis ..
qed
lemma is-sig-red-singletonD:
    assumes is-sig-red sing-reg top-tail {f} r and f\inF
    shows is-sig-red sing-reg top-tail Fr
proof -
    from assms(1) obtain r' where sig-red sing-reg top-tail {f} r r' unfolding
is-sig-red-def ..
    then obtain t where sig-red-single sing-reg top-tail r r'ft by (auto simp:
sig-red-def)
    show ?thesis unfolding is-sig-red-def sig-red-def by (intro exI bexI, fact+)
qed
lemma is-sig-redD1:
    assumes is-sig-red sing-reg top-tail F p
    shows ord-term-lin.is-le-rel sing-reg
proof -
    from assms obtain q}\mathrm{ where sig-red sing-reg top-tail F p q unfolding is-sig-red-def
    then obtain fs where f}\inF\mathrm{ and sig-red-single sing-reg top-tail pqfs unfold-
ing sig-red-def by blast
    from this(2) show ?thesis by (rule sig-red-singleD)
qed
lemma is-sig-redD2:
    assumes is-sig-red sing-reg top-tail F p
    shows ordered-powerprod-lin.is-le-rel top-tail
proof -
```

from assms obtain $q$ where sig-red sing-reg top-tail $F p q$ unfolding is-sig-red-def
then obtain $f s$ where $f \in F$ and sig-red-single sing-reg top-tail $p q f s$ unfolding sig-red-def by blast
from this(2) show ?thesis by (rule sig-red-singleD)
qed
lemma is-sig-red-addsI:
assumes $f \in F$ and $t \in$ keys (rep-list $p$ ) and rep-list $f \neq 0$ and punit.lt (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 $(t \oplus l t f)$ (punit.lt (rep-list $f) \oplus l t p)$ and top-tail $t$ (punit.lt ( rep-list p))
shows is-sig-red sing-reg top-tail Fp
unfolding is-sig-red-def

## proof

let $? q=p-$ monom-mult $((l o o k u p($ rep-list $p) t) /$ punit.lc $($ rep-list $f))(t-$ punit.lt (rep-list f)) $f$
show sig-red sing-reg top-tail $F p$ ? $q$ unfolding sig-red-def
proof (intro bexI exI)
from assms(4) have eq: $(t-$ punit.lt (rep-list f) $)+$ punit.lt $($ rep-list $f)=t$
by (rule adds-minus)
from $\operatorname{assms}(4,5,7)$ have sing-reg $((t-$ punit.lt $($ rep-list $f)) \oplus l t f)(l t p)$
by (simp only: term-is-le-rel-minus)
thus sig-red-single sing-reg top-tail $p ? q f(t-$ punit.lt (rep-list $f))$
by (simp add: assms eq sig-red-singleI)
qed fact
qed
lemma is-sig-red-addsE:
assumes is-sig-red sing-reg top-tail $F p$
obtains $f t$ where $f \in F$ and $t \in$ keys (rep-list p) and rep-list $f \neq 0$
and punit.lt (rep-list f) adds $t$
and sing-reg $(t \oplus l t f)($ punit.lt $(r e p-l i s t ~ f) \oplus l t p)$ and top-tail $t$ (punit.lt (rep-list p))
proof -
from assms have $*$ : ord-term-lin.is-le-rel sing-reg by (rule is-sig-redD1)
from assms obtain $q$ where sig-red sing-reg top-tail Fp $q$ unfolding is-sig-red-def ..
then obtain $f s$ where $f \in F$ and sig-red-single sing-reg top-tail $p q f s$ unfolding sig-red-def by blast
from this(2) have 1: rep-list $f \neq 0$ and 2: s + punit.lt (rep-list $f$ ) $\in$ keys (rep-list p)
and 3: sing-reg $(s \oplus l t f)(l t p)$ and 4: top-tail $(s+$ punit.lt (rep-list f)) (punit.lt (rep-list p))
by (rule sig-red-single $D)+$
note $\langle f \in F\rangle 21$
moreover have punit.lt (rep-list f) adds $s+$ punit.lt (rep-list f) by simp
moreover from 3 have sing-reg $((s+$ punit.lt $($ rep-list f $)) \oplus l t f)$ (punit.lt

```
(rep-list f)\opluslt p)
    by (simp add: add.commute[of s] splus-assoc term-is-le-rel-canc-left[OF *])
    moreover from 4 have top-tail (s+ punit.lt (rep-list f)) (punit.lt (rep-list p))
by simp
    ultimately show ?thesis ..
qed
lemma is-sig-red-top-addsI:
    assumes f\inF and rep-list f}\not=0\mathrm{ and rep-list p}\not=
        and punit.lt (rep-list f) adds punit.lt (rep-list p) and ord-term-lin.is-le-rel
sing-reg
    and sing-reg (punit.lt (rep-list p)\opluslt f) (punit.lt (rep-list f) }\opluslt p
    shows is-sig-red sing-reg (=) F p
proof -
    note assms(1)
    moreover from assms(3) have punit.lt (rep-list p) \in keys (rep-list p) by (rule
punit.lt-in-keys)
    moreover note assms(2, 4, 5) ordered-powerprod-lin.is-le-relI(1) assms(6) refl
    ultimately show ?thesis by (rule is-sig-red-addsI)
qed
lemma is-sig-red-top-addsE:
    assumes is-sig-red sing-reg (=) F p
    obtains f}\mathrm{ where f}\inF\mathrm{ and rep-list f}\not=0\mathrm{ and rep-list p}\not=
        and punit.lt (rep-list f) adds punit.lt (rep-list p)
    and sing-reg (punit.lt (rep-list p)\opluslt f) (punit.lt (rep-list f)\oplus lt p)
proof -
    from assms obtain ft where 1:f\inF and 2: t\in keys (rep-list p) and 3:
rep-list f }=
    and 4: punit.lt (rep-list f) adds t
    and 5: sing-reg (t\opluslt f) (punit.lt (rep-list f)\opluslt p)
    and t: t = punit.lt (rep-list p) by (rule is-sig-red-addsE)
    note 1 3
    moreover from 2 have rep-list p}\not=0\mathrm{ by auto
    moreover from 4 have punit.lt (rep-list f) adds punit.lt (rep-list p) by (simp
only: t)
    moreover from 5 have sing-reg (punit.lt (rep-list p)\oplus lt f) (punit.lt (rep-list
f)}\opluslt p
    by (simp only: t)
    ultimately show ?thesis ..
qed
lemma is-sig-red-top-plusE:
    assumes is-sig-red sing-reg (=) Fp and is-sig-red sing-reg (=) Fq
    and lt p \preceq_t lt (p+q) and lt q\mp@subsup{\preceq}{t}{lt}(p+q) and sing-reg = (\preceq
=(\prec}\mp@subsup{)}{t}{
    assumes 1: is-sig-red sing-reg(=) F(p+q)\Longrightarrow thesis
    assumes 2: punit.lt (rep-list p) = punit.lt (rep-list q) \Longrightarrow punit.lc (rep-list p) +
punit.lc (rep-list q) = 0\Longrightarrow thesis
```

```
    shows thesis
proof -
    from assms(1) obtain f1 where f1 \inF and rep-list f1 }\not=0\mathrm{ and rep-list p}\not=
        and a: punit.lt (rep-list f1) adds punit.lt (rep-list p)
        and b: sing-reg (punit.lt (rep-list p)\opluslt f1) (punit.lt (rep-list f1) \oplus lt p)
    by (rule is-sig-red-top-addsE)
    from assms(\mathcal{O}) obtain f2 where f2 }\inF\mathrm{ and rep-list f2 }\not=0\mathrm{ and rep-list q}\not=
    and c: punit.lt (rep-list f2) adds punit.lt (rep-list q)
    and d: sing-reg (punit.lt (rep-list q) \oplus lt f2) (punit.lt (rep-list f2) }\opluslt q
    by (rule is-sig-red-top-addsE)
    show ?thesis
    proof (cases punit.lt (rep-list p) = punit.lt (rep-list q) ^ punit.lc (rep-list p) +
punit.lc (rep-list q) = 0)
    case True
    hence punit.lt (rep-list p) = punit.lt (rep-list q) and punit.lc (rep-list p) +
punit.lc (rep-list q) = 0
            by simp-all
    thus ?thesis by (rule 2)
next
    case False
    hence disj: punit.lt (rep-list p) f punit.lt (rep-list q) \vee punit.lc (rep-list p) +
punit.lc (rep-list q)}\not=
            by simp
    from assms(5) have ord-term-lin.is-le-rel sing-reg by (simp add:ord-term-lin.is-le-rel-def)
    have rep-list (p+q)}=0\mathrm{ unfolding rep-list-plus
    proof
        assume eq: rep-list p + rep-list q}=
        have eq2: punit.lt (rep-list p) = punit.lt (rep-list q)
        proof (rule ordered-powerprod-lin.linorder-cases)
            assume *: punit.lt (rep-list p)\prec punit.lt (rep-list q)
                    hence punit.lt (rep-list p + rep-list q) = punit.lt (rep-list q) by (rule
punit.lt-plus-eqI)
            with * zero-min[of punit.lt (rep-list p)] show ?thesis by (simp add: eq)
            next
            assume *: punit.lt (rep-list q) \prec punit.lt (rep-list p)
                    hence punit.lt (rep-list p + rep-list q) = punit.lt (rep-list p) by (rule
punit.lt-plus-eqI-2)
            with * zero-min[of punit.lt (rep-list q)] show ?thesis by (simp add: eq)
            qed
            with disj have punit.lc (rep-list p) + punit.lc (rep-list q)}=0\mathrm{ by simp
            thus False by (simp add: punit.lc-def eq2 lookup-add[symmetric] eq)
    qed
    have punit.lt (rep-list ( }p+q))=\mathrm{ ordered-powerprod-lin.max (punit.lt (rep-list
p))(punit.lt (rep-list q))
            unfolding rep-list-plus
    proof (rule punit.lt-plus-eq-maxI)
            assume punit.lt (rep-list p) = punit.lt (rep-list q)
            with disj show punit.lc (rep-list p) + punit.lc (rep-list q) =0 by simp
    qed
```

hence punit.lt $($ rep-list $(p+q))=$ punit.lt $($ rep-list $p) \vee$ punit.lt (rep-list $(p+$ $q))=$ punit.lt (rep-list $q$ )
by ( $\operatorname{simp}$ add: ordered-powerprod-lin.max-def)
thus ?thesis
proof
assume eq: punit.lt (rep-list $(p+q))=$ punit.lt (rep-list $p$ )
show ?thesis
proof (rule 1, rule is-sig-red-top-addsI)
from $a$ show punit.lt (rep-list f1) adds punit.lt (rep-list $(p+q)$ ) by (simp only: eq)
next
from $b$ have sing-reg (punit.lt (rep-list $(p+q)) \oplus l t f 1)$ (punit.lt (rep-list f1) $\oplus$ lt $p$ )
by (simp only: eq)
moreover from assms(3) have $\ldots \preceq_{t}$ punit.lt (rep-list f1) $\oplus l t(p+q)$ by (rule splus-mono)
ultimately show sing-reg (punit.lt (rep-list $(p+q)) \oplus l t$ f1) (punit.lt (rep-list f1) $\oplus l t(p+q))$
using assms(5) by auto
qed fact+
next
assume eq: punit.lt (rep-list $(p+q))=$ punit.lt (rep-list $q$ )
show ?thesis
proof (rule 1, rule is-sig-red-top-addsI)
from $c$ show punit.lt (rep-list f2) adds punit.lt (rep-list $(p+q)$ ) by (simp only: eq)
next
from $d$ have sing-reg (punit.lt (rep-list $(p+q)) \oplus l t$ f2) (punit.lt (rep-list f2) $\oplus(t q)$
by (simp only: eq)
moreover from $\operatorname{assms}(4)$ have $\ldots \preceq_{t}$ punit.lt (rep-list f2) $\oplus l t(p+q)$ by (rule splus-mono)
ultimately show sing-reg (punit.lt (rep-list $(p+q)) \oplus l t$ f2) (punit.lt
(rep-list f2) $\oplus l t(p+q))$
using assms(5) by auto
qed fact +
qed
qed
qed
lemma is-sig-red-singleton-monom-multD:
assumes is-sig-red sing-reg top-tail \{monom-mult ctf\} $p$
shows is-sig-red sing-reg top-tail $\{f\} p$
proof -
let ?f $=$ monom-mult $c t f$
from assms obtain $s$ where $s \in$ keys (rep-list p) and 2: rep-list ?f $\neq 0$
and 3: punit.lt (rep-list ?f) adds s
and 4: sing-reg $(s \oplus l t ? f)($ punit.lt $(r e p-l i s t ? f) \oplus l t p)$
and top-tail s (punit.lt (rep-list p))

```
    by (auto elim: is-sig-red-addsE)
    from 2 have c\not=0 and rep-list f}\not=
    by (simp-all add: rep-list-monom-mult punit.monom-mult-eq-zero-iff)
    hence f}\not=0\mathrm{ by (auto simp: rep-list-zero)
    with 〈c\not= 0\rangle have eq1:lt ?f = t\opluslt f by (simp add:lt-monom-mult)
    from}\langlec\not=0\rangle\langlerep-list f\not=0\rangle have eq2: punit.lt (rep-list ?f) =t + punit.lt
(rep-list f)
    by (simp add: rep-list-monom-mult punit.lt-monom-mult)
    from assms have *: ord-term-lin.is-le-rel sing-reg by (rule is-sig-redD1)
    show ?thesis
    proof (rule is-sig-red-addsI)
    show }f\in{f}\mathrm{ by simp
next
    have punit.lt (rep-list f) adds t + punit.lt (rep-list f) by (rule adds-triv-right)
    also from 3 have ... adds s by (simp only: eq2)
    finally show punit.lt (rep-list f) adds s.
next
    from4}4\mathrm{ have sing-reg (t }\oplus(s\opluslt f))(t\oplus(punit.lt (rep-list f)\opluslt p)
        by (simp add: eq1 eq2 splus-assoc splus-left-commute)
    with * show sing-reg (s\opluslt f) (punit.lt (rep-list f) }\oplus\mathrm{ lt p)
        by (simp add: term-is-le-rel-canc-left)
    next
    from assms show ordered-powerprod-lin.is-le-rel top-tail by (rule is-sig-redD2)
    qed fact+
qed
lemma is-sig-red-top-singleton-monom-multI:
    assumes is-sig-red sing-reg (=) {f} p and c\not=0
    and t adds punit.lt (rep-list p) - punit.lt (rep-list f)
    shows is-sig-red sing-reg (=) {monom-mult c tf} p
proof -
    let ?f = monom-mult ctf
    from assms have 2: rep-list f}\not=0\mathrm{ and rep-list p}\not=
        and 3: punit.lt (rep-list f) adds punit.lt (rep-list p)
        and 4: sing-reg (punit.lt (rep-list p)}\oplusltf)(punit.lt (rep-list f)\opluslt p
        by (auto elim: is-sig-red-top-addsE)
    hence f\not=0 by (auto simp: rep-list-zero)
    with }\langlec\not=0\rangle\mathrm{ have eq1:lt ?f = t }\oplusltf\mathrm{ by (simp add:lt-monom-mult)
    from}\langlec\not=0\rangle\langlerep-list f\not=0\rangle\mathrm{ have eq2: punit.lt (rep-list ?f) =t + punit.lt
(rep-list f)
    by (simp add: rep-list-monom-mult punit.lt-monom-mult)
from assms(1) have *: ord-term-lin.is-le-rel sing-reg by (rule is-sig-redD1)
show ?thesis
proof (rule is-sig-red-top-addsI)
    show ?f \in{?f} by simp
next
    from }\langlec\not=0\rangle\langlerep-list f\not=0\rangle\mathrm{ show rep-list ?f }\not=
            by (simp add: rep-list-monom-mult punit.monom-mult-eq-zero-iff)
next
```

```
        from assms(3) have t + punit.lt (rep-list f) adds
                    (punit.lt (rep-list p) - punit.lt (rep-list f)) + punit.lt (rep-list f)
        by (simp only: adds-canc)
        also from 3 have ... = punit.lt (rep-list p) by (rule adds-minus)
        finally show punit.lt (rep-list ?f) adds punit.lt (rep-list p) by (simp only: eq2)
    next
        from 4* show sing-reg (punit.lt (rep-list p)\opluslt ?f) (punit.lt (rep-list ?f) }
lt p)
            by (simp add: eq1 eq2 term-is-le-rel-canc-left splus-assoc splus-left-commute)
        qed fact+
qed
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
proof -
    from assms(1) have 1:ord-term-lin.is-le-rel sing-reg and 2: ordered-powerprod-lin.is-le-rel
top-tail
    by (rule is-sig-redD1, rule is-sig-redD2)
    from assms(1) obtain ft where f\inF and t\in keys (rep-list p) and rep-list f
# 0
    and punit.lt (rep-list f) adds t
    and sing-reg (t\opluslt f) (punit.lt (rep-list f) \opluslt p)
    and top-tail t (punit.lt (rep-list p)) by (rule is-sig-red-addsE)
    from this(1-4)12 this(5,6) show ?thesis unfolding assms(2, 3) by (rule
is-sig-red-addsI)
qed
lemma is-sig-red-cong:
    lt p=lt q\Longrightarrow rep-list p=rep-list q\Longrightarrow
        is-sig-red sing-reg top-tail F p\longleftrightarrow is-sig-red sing-reg top-tail F q
    by (auto intro: is-sig-red-cong')
lemma is-sig-red-top-cong:
    assumes is-sig-red sing-reg (=) F p and rep-list q\not=0 and lt p=lt q
        and punit.lt (rep-list p) = punit.lt (rep-list q)
    shows is-sig-red sing-reg (=) Fq
proof -
    from assms(1) have 1: ord-term-lin.is-le-rel sing-reg by (rule is-sig-redD1)
    from assms(1) obtain f}\mathrm{ where f}\inF\mathrm{ and rep-list f}\not=0\mathrm{ and rep-list p}\not=
        and punit.lt (rep-list f) adds punit.lt (rep-list p)
        and sing-reg (punit.lt (rep-list p)\opluslt f) (punit.lt (rep-list f) }\opluslt p
        by (rule is-sig-red-top-addsE)
    from this(1, 2) assms(2) this(4) 1 this(5) show ?thesis
        unfolding assms(3, 4) by (rule is-sig-red-top-addsI)
qed
lemma sig-irredE-dgrad-max-set:
```

```
    assumes dickson-grading d and F\subseteqdgrad-max-set d
    obtains q}\mathrm{ where (sig-red sing-reg top-tail F)** p q and }\neg\mathrm{ is-sig-red sing-reg
top-tail F q
proof -
    let ?Q = {q. (sig-red sing-reg top-tail F)** p q}
    from assms have wfP (sig-red sing-reg top-tail F)-1-1 by (rule sig-red-wf-dgrad-max-set)
    moreover have p\in?Q by simp
    ultimately obtain q}\mathrm{ where q}\in?Q\mathrm{ and }\bigwedgex.(\mathrm{ sig-red sing-reg top-tail F)}\mp@subsup{)}{}{-1-1
x q\Longrightarrowx\not\in?Q
    by (rule wfE-min[to-pred], blast)
    hence 1:(sig-red sing-reg top-tail F)** p q
    and 2: }\x\mathrm{ . sig-red sing-reg top-tail F q x }\Longrightarrow\mathrm{ ( sig-red sing-reg top-tail F)**
p x
    by simp-all
    show ?thesis
    proof
        show \neg is-sig-red sing-reg top-tail F q
        proof
            assume is-sig-red sing-reg top-tail Fq
            then obtain x where 3: sig-red sing-reg top-tail Fqx unfolding is-sig-red-def
..
            hence }\neg(\mathrm{ sig-red sing-reg top-tail F)** p x by (rule 2)
            moreover from 1 3 have (sig-red sing-reg top-tail F)** px ..
            ultimately show False ..
    qed
    qed fact
qed
lemma is-sig-red-mono:
    is-sig-red sing-reg top-tail F p\LongrightarrowF\subseteq F'\Longrightarrowis-sig-red sing-reg top-tail F'p
    by (auto simp: is-sig-red-def dest: sig-red-mono)
lemma is-sig-red-Un:
    is-sig-red sing-reg top-tail (A\cupB) p\longleftrightarrow (is-sig-red sing-reg top-tail A p}
is-sig-red sing-reg top-tail B p
    by (auto simp: is-sig-red-def sig-red-Un)
lemma is-sig-redD-lt:
    assumes is-sig-red ( }\mp@subsup{\preceq}{t}{})\mathrm{ top-tail {f} p
    shows lt f}\mp@subsup{\preceq}{t}{}lt 
proof -
    from assms obtain s where rep-list f}\not=0\mathrm{ and s k keys (rep-list p)
    and 1: punit.lt (rep-list f) adds s and 2: s\opluslt f}\mp@subsup{\preceq}{t punit.lt (rep-list f) }{\mathrm{ plt p}
    by (auto elim!: is-sig-red-addsE)
    from 1 obtain t where eq: s = punit.lt (rep-list f) +t by (rule addsE)
    hence punit.lt (rep-list f)}\oplus(t\oplusltf)=s\opluslt f by (simp add: splus-assoc)
    also note 2
    finally have t\opluslt f}\mp@subsup{\preceq}{t}{}lt p\mathrm{ by (rule ord-term-canc)
    have 0}\preceqt\mathrm{ by (fact zero-min)
```

```
    hence 0}\oplusltf\mp@subsup{\preceq}{t}{}t\oplusltf\mathrm{ by (rule splus-mono-left)
    hence lt f}\mp@subsup{\preceq}{t}{}t\oplusltf\mathrm{ by (simp add: term-simps)
    thus ?thesis using <t \opluslt f}\mp@subsup{\preceq}{t}{}lt p> by sim
qed
lemma is-sig-red-regularD-lt:
    assumes is-sig-red ( }\mp@subsup{\prec}{t}{})\mathrm{ top-tail {f} p
    shows lt f}\mp@subsup{\prec}{t}{lt p
proof -
    from assms obtain s where rep-list f}\not=0\mathrm{ and s}\in\mathrm{ keys (rep-list p)
    and 1: punit.lt (rep-list f) adds s and 2: s\opluslt f < t punit.lt (rep-list f) }\oplus\mathrm{ lt p
    by (auto elim!: is-sig-red-addsE)
    from 1 obtain t where eq:s= punit.lt (rep-list f) +t by (rule addsE)
    hence punit.lt (rep-list f)}\oplus(t\oplusltf)=s\opluslt f by (simp add: splus-assoc
    also note 2
    finally have t\opluslt f}\mp@subsup{\prec}{t}{}lt p by (rule ord-term-strict-canc
    have 0 \preceq t by (fact zero-min)
    hence 0}\oplusltf\mp@subsup{\preceq}{t}{}t\oplusltf\mathrm{ by (rule splus-mono-left)
    hence lt f}\mp@subsup{\preceq}{t}{}t\oplusltf\mathrm{ by (simp add: term-simps)
    thus ?thesis using <t \opluslt f \prec lt lt p> by (rule ord-term-lin.le-less-trans)
qed
lemma sig-irred-regular-self: ᄀ is-sig-red ( }\mp@subsup{\prec}{t}{})\mathrm{ top-tail {p} p
    by (auto dest: is-sig-red-regularD-lt)
```


### 4.2.2 Signature Gröbner Bases

definition sig-red-zero :: ( ${ }^{\prime} t \Rightarrow^{\prime} t \Rightarrow$ bool $) \Rightarrow\left({ }^{\prime} t \Rightarrow{ }_{0}{ }^{\prime} b\right)$ set $\Rightarrow\left({ }^{\prime} t \Rightarrow{ }_{0}{ }^{\prime} b\right) \Rightarrow$ bool where sig-red-zero sing-reg $F r \longleftrightarrow(\exists \text { s. (sig-red sing-reg }(\preceq) F)^{* *} r s \wedge$ rep-list $s=0$ )
definition $i s$-sig-GB-in :: (' $a \Rightarrow$ nat $) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ set $\Rightarrow{ }^{\prime} t \Rightarrow$ 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 $\left.\left(\preceq_{t}\right) G r\right)$
definition is-sig-GB-upt $::\left({ }^{\prime} a \Rightarrow n a t\right) \Rightarrow\left(' t \Rightarrow_{0}{ }^{\prime} b\right)$ set $\Rightarrow{ }^{\prime} t \Rightarrow$ bool where is-sig-GB-upt d $G u \longleftrightarrow$
$\left(G \subseteq\right.$ dgrad-sig-set $d \wedge\left(\forall v . v \prec_{t} u \longrightarrow d(p p-o f-t e r m ~ v) \leq d g r a d-m a x d\right.$ $\longrightarrow$ component-of-term $v<$ length $f s \longrightarrow i s-s i g-G B-i n$ $d G v)$ )
definition is-min-sig-GB :: (' $a \Rightarrow$ nat $) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ set $\Rightarrow$ bool where is-min-sig-GB $d G \longleftrightarrow G \subseteq d g r a d$-sig-set $d \wedge$
$(\forall u . d(p p-o f-t e r m u) \leq$ dgrad-max $d \longrightarrow$ component-of-term
$u<$ length $f s \longrightarrow$
is-sig-GB-in $d G u) \wedge$
$\left(\forall g \in G . \neg i s\right.$-sig-red $\left.\left(\preceq_{t}\right)(=)(G-\{g\}) g\right)$

```
definition is-syz-sig :: (' }a=>\mathrm{ nat) }=>\mp@subsup{}{}{\prime}t=>\mathrm{ bool
    where is-syz-sig d u \longleftrightarrow (\existss\indgrad-sig-set d.s\not=0^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 sing-reg F r
    unfolding sig-red-zero-def using assms by blast
lemma sig-red-zeroE:
    assumes sig-red-zero sing-reg F r
    obtains s where (sig-red sing-reg(\preceq) F)** rs and rep-list s = 0
    using assms unfolding sig-red-zero-def by blast
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 -
    from assms obtain s where (sig-red sing-reg (\preceq)F)** rs and rep-list s=0
        by (rule sig-red-zeroE)
    from this(1) have (sig-red sing-reg (\preceq) F)** (monom-mult c t r) (monom-mult
cts)
    by (rule sig-red-rtrancl-monom-mult)
    moreover have rep-list (monom-mult cts)=0 by (simp add: rep-list-monom-mult
<rep-list s=0`)
    ultimately show ?thesis by (rule sig-red-zeroI)
qed
lemma sig-red-zero-sing-regI:
    assumes sig-red-zero sing-reg G p
    shows sig-red-zero (\preceq}\mp@subsup{\}{t}{})G
proof -
    from assms obtain s where (sig-red sing-reg(\preceq)G)** ps and rep-list s=0
        by (rule sig-red-zeroE)
    from this(1) have (sig-red ( }\mp@subsup{\preceq}{t}{})(\preceq)G\mp@subsup{)}{}{**}ps\mathrm{ by (rule sig-red-rtrancl-sing-regI)
    thus ?thesis using <rep-list s=0\rangle}\mathrm{ by (rule sig-red-zeroI)
qed
lemma sig-red-zero-nonzero:
    assumes sig-red-zero sing-reg F r and rep-list r \not=0 and sing-reg =( }\mp@subsup{\preceq}{t}{})
sing-reg = (< }\mp@subsup{\iota}{t}{}
    shows is-sig-red sing-reg (=) Fr
proof -
    from assms(1) obtain s where (sig-red sing-reg(\preceq)F)**rs and rep-list s=
O
    by (rule sig-red-zeroE)
    from this(1) assms(2) show ?thesis
    proof (induct rule: converse-rtranclp-induct)
    case base
```

```
    thus ?case using <rep-list s = 0`..
next
    case (step y z)
    from step(1) obtain ft where f\inF and *: sig-red-single sing-reg (\preceq) yzft
        unfolding sig-red-def by blast
    from this(2) have 1: rep-list f}\not=0\mathrm{ and 2: t + punit.lt (rep-list f) }\in\mathrm{ keys
(rep-list y)
```

    and 3: \(z=y\) - monom-mult (lookup (rep-list \(y)(t+\) punit.lt (rep-list f) \() /\)
    punit.lc (rep-list f)) $t f$
and 4: ord-term-lin.is-le-rel sing-reg and 5: sing-reg ( $t \oplus l t f)$ (lt y)
by (rule sig-red-singleD)+
show? case
proof $($ cases $t+$ punit.lt $($ rep-list $f)=$ punit.lt $($ rep-list $y))$
case True
show ?thesis unfolding is-sig-red-def
proof
show sig-red sing-reg (=) F y z unfolding sig-red-def
proof (intro bexI exI)
from 1234 ordered-powerprod-lin.is-le-relI(1) 5 True
show sig-red-single sing-reg $(=) y z f t$ by (rule sig-red-singleI)
qed fact
qed
next
case False
from 2 have $t+$ punit.lt (rep-list $f$ ) $\preceq$ punit.lt (rep-list $y$ ) by (rule
punit.lt-max-keys)
with False have $t+$ punit.lt (rep-list $f$ ) $\prec$ punit.lt (rep-list y) by simp
with 1234 ordered-powerprod-lin.is-le-relI(3) 5 have sig-red-single sing-reg
$(\prec) y z f t$
by (rule sig-red-singleI)
hence punit.lt (rep-list $y$ ) $\in$ keys (rep-list $z$ )
and lt-z: punit.lt (rep-list z) $=$ punit.lt (rep-list y)
by (rule sig-red-single-tail-lt-in-keys-rep-list, rule sig-red-single-tail-lt-rep-list)
from this(1) have rep-list $z \neq 0$ by auto
hence is-sig-red sing-reg (=) Fz by (rule step(3))
then obtain $g$ where $g \in F$ and rep-list $g \neq 0$
and punit.lt (rep-list g) adds punit.lt (rep-list z)
and a: sing-reg (punit.lt (rep-list z) $\oplus$ lt g) (punit.lt $($ rep-list $g) \oplus l t z)$
by (rule is-sig-red-top-addsE)
from this(3) have punit.lt (rep-list g) adds punit.lt (rep-list y) by (simp only:
$l t-z)$
with $\langle g \in F\rangle\langle r e p-l i s t ~ g \neq 0\rangle$ step (4) show ?thesis
proof (rule is-sig-red-top-addsI)
from〈is-sig-red sing-reg (=) F z〉 show ord-term-lin.is-le-rel sing-reg by
(rule is-sig-redD1)
next
from $\langle$ sig-red-single sing-reg $(\prec) y z f t\rangle$ have $l t z \preceq_{t}$ lt $y$ by (rule
sig-red-single-lt)
from $\operatorname{assms}(3)$ show sing-reg (punit.lt (rep-list $y) \oplus$ lt g) (punit.lt (rep-list

```
g)}\opluslty
    proof
    assume sing-reg = (\mp@subsup{\preceq}{t}{})
    from a have punit.lt (rep-list y) \opluslt g}\mp@subsup{\preceq}{t punit.lt (rep-list g) \opluslt z}{
```



```
            also from <lt z \preceq́t lt y> have ... \preceq_ punit.lt (rep-list g) \oplus lt y by (rule
splus-mono)
            finally show ?thesis by (simp only: <sing-reg = (\preceq
        next
            assume sing-reg = ( }\mp@subsup{\prec}{t}{}
            from a have punit.lt (rep-list y) \opluslt g\mp@subsup{\prec}{t punit.lt (rep-list g) }{\mathrm{ plt z}}\mathbf{l}|
                by (simp only:lt-z<sing-reg = (\mp@subsup{\prec}{t}{})\rangle)
            also from <lt z \preceq\preceqt lt y> have ... \preceq_t punit.lt (rep-list g) }\oplus\mathrm{ lt y by (rule
splus-mono)
            finally show ?thesis by (simp only: <sing-reg = ( < }t)〉
            qed
        qed
    qed
    qed
qed
lemma sig-red-zero-mono: sig-red-zero sing-reg F p\LongrightarrowF\subseteq F'\Longrightarrowsig-red-zero
sing-reg F' p
    by (auto simp: sig-red-zero-def dest: sig-red-rtrancl-mono)
lemma sig-red-zero-subset:
    assumes sig-red-zero sing-reg F p and sing-reg = (\preceq_})\vee ving-reg = (\mp@subsup{\prec}{t}{}
    shows sig-red-zero sing-reg {f\inF. sing-reg (lt f) (lt p)} p
proof -
    from assms(1) obtain s where (sig-red sing-reg (\preceq) F)** ps and rep-list s=
O
    by (rule sig-red-zeroE)
    from this(1) assms(2) have (sig-red sing-reg (\preceq) {f\inF. sing-reg (lt f) (lt p)})**
p s
    by (rule sig-red-rtrancl-subset)
    thus ?thesis using <rep-list s=0` by (rule sig-red-zeroI)
qed
lemma sig-red-zero-idealI:
    assumes sig-red-zero sing-reg F p
    shows rep-list p}\in\mathrm{ ideal (rep-list' F)
proof -
    from assms obtain s where (sig-red sing-reg (\preceq) F)** ps and rep-list s=0
by (rule sig-red-zeroE)
    from this(1) have (punit.red (rep-list'F))** (rep-list p) (rep-list s) by (rule
sig-red-red-rtrancl)
    hence (punit.red (rep-list 'F))** (rep-list p) 0 by (simp only: <rep-list s = 0`)
    thus ?thesis by (rule punit.red-rtranclp-0-in-pmdl[simplified])
qed
```

```
lemma is-sig-GB-inI:
    assumes \r.lt r=u\Longrightarrowr\indgrad-sig-set d \Longrightarrow sig-red-zero ( }\mp@subsup{\preceq}{t}{})G
    shows is-sig-GB-in d Gu
    unfolding is-sig-GB-in-def using assms by blast
lemma is-sig-GB-inD:
    assumes is-sig-GB-in d Gu and r\indgrad-sig-set d and lt r=u
    shows sig-red-zero ( }\mp@subsup{\preceq}{t}{})G
    using assms unfolding is-sig-GB-in-def by blast
lemma is-sig-GB-inI-triv:
    assumes }\negd(pp\mathrm{ -of-term u)}\leqdgrad-max d \vee\neg component-of-term u<length
fs
    shows is-sig-GB-in d G u
proof (rule is-sig-GB-inI)
    fix r::'t = 0
    assume lt r=u and r\indgrad-sig-set d
    show sig-red-zero (\preceq}\mp@subsup{)}{}{\prime})G
    proof (cases r=0)
        case True
        hence rep-list r=0 by (simp only: rep-list-zero)
        with rtrancl-refl[to-pred] show ?thesis by (rule sig-red-zeroI)
    next
        case False
    from }\langler\indgrad-sig-set d> have d (lp r)\leqdgrad-max d by (rule dgrad-sig-setD-lp)
        moreover from <r \indgrad-sig-set d\rangle False have component-of-term (lt r)<
length fs
        by (rule dgrad-sig-setD-lt)
        ultimately show ?thesis using assms by (simp add: <lt r =u`)
    qed
qed
```



```
u
    by (auto simp: is-sig-GB-in-def dest: sig-red-zero-mono)
lemma is-sig-GB-uptI:
    assumes G\subseteqdgrad-sig-set d
        and }\v.v\mp@subsup{\prec}{t}{}u\Longrightarrowd(pp-of-term v)\leqdgrad-max d \Longrightarrow component-of-term
v< length fs \Longrightarrow
                is-sig-GB-in d Gv
    shows is-sig-GB-upt d Gu
    unfolding is-sig-GB-upt-def using assms by blast
lemma is-sig-GB-uptD1:
    assumes is-sig-GB-upt d Gu
    shows G\subseteqdgrad-sig-set d
    using assms unfolding is-sig-GB-upt-def by blast
```

```
lemma is-sig-GB-uptD2:
    assumes is-sig-GB-upt d Gu and v \prec}\mp@subsup{}{t}{}
    shows is-sig-GB-in d G v
    using assms is-sig-GB-inI-triv unfolding is-sig-GB-upt-def by blast
lemma is-sig-GB-uptD3:
    assumes is-sig-GB-upt d Gu and r dgrad-sig-set d and lt r }\mp@subsup{\prec}{t}{}
    shows sig-red-zero ( }\mp@subsup{\preceq}{t}{})G
    by (rule is-sig-GB-inD, rule is-sig-GB-uptD2, fact+, fact refl)
lemma is-sig-GB-upt-le:
    assumes is-sig-GB-upt d Gu and v \preceq.
    shows is-sig-GB-upt d Gv
proof (rule is-sig-GB-uptI)
    from assms(1) show G\subseteqdgrad-sig-set d by (rule is-sig-GB-uptD1)
next
    fix w
    assume w}\mp@subsup{\prec}{t}{}
    hence }w\mp@subsup{\prec}{t}{}u\mathrm{ using assms(2) by (rule ord-term-lin.less-le-trans)
    with assms(1) show is-sig-GB-in d G w by (rule is-sig-GB-uptD2)
qed
lemma is-sig-GB-upt-mono:
    is-sig-GB-upt d Gu\LongrightarrowG\subseteqG'\Longrightarrow ( ' 
G'u
    by (auto simp: is-sig-GB-upt-def dest!: is-sig-GB-in-mono)
lemma is-sig-GB-upt-is-Groebner-basis:
    assumes dickson-grading d and hom-grading d and G\subseteqdgrad-sig-set' jd
        and \u.component-of-term u<j\Longrightarrowis-sig-GB-in d Gu
    shows punit.is-Groebner-basis (rep-list' G)
    using assms(1)
proof (rule punit.weak-GB-is-strong-GB-dgrad-p-set[simplified])
    from assms(3) have G\subseteqdgrad-max-set d by (simp add: dgrad-sig-set'-def)
    with assms(1) show rep-list ' }G\subseteq\mathrm{ punit-dgrad-max-set d by (rule dgrad-max-3)
next
    fix f::'a = 0 'b
    assume f}\in\mathrm{ punit-dgrad-max-set d
    from assms(3) have G-sub: G\subseteq sig-inv-set' j by (simp add:dgrad-sig-set'-def)
    assume f}\in\mathrm{ ideal (rep-list' G)
    also from rep-list-subset-ideal-sig-inv-set[OF G-sub] have ... \subseteq ideal (set (take
j fs))
        by (rule ideal.span-subset-spanI)
    finally have f}\in\mathrm{ ideal (set (take j fs)).
    with assms(2)}<f\in\mathrm{ punit-dgrad-max-set d> obtain r where r f dgrad-sig-set d
        and r\indgrad-sig-set' jd and f:f=rep-list r
        by (rule in-idealE-rep-list-dgrad-sig-set-take)
    from this(2) have r sig-inv-set' j by (simp add:dgrad-sig-set'-def)
```

```
    show (punit.red (rep-list' G))** f 0
    proof (cases r=0)
    case True
    thus ?thesis by (simp add: f rep-list-zero)
    next
    case False
    hence lt r keys r by (rule lt-in-keys)
    with }\langler\in\mathrm{ sig-inv-set' j〉 have component-of-term (lt r) <j by (rule sig-inv-setD')
    hence is-sig-GB-in d G (lt r) by (rule assms(4))
    hence sig-red-zero ( }\mp@subsup{\preceq}{t}{})Gr\mathrm{ using <r < dgrad-sig-set d> refl by (rule is-sig-GB-inD)
    then obtain s}\mathrm{ where (sig-red ( }\mp@subsup{\preceq}{t}{})(\preceq)G\mp@subsup{)}{}{**}rs\mathrm{ and s: rep-list s=0 by (rule
sig-red-zeroE)
    from this(1) have (punit.red (rep-list'G))** (rep-list r) (rep-list s)
        by (rule sig-red-red-rtrancl)
    thus ?thesis by (simp only: fs)
    qed
qed
lemma is-sig-GB-is-Groebner-basis:
assumes dickson-grading \(d\) and hom-grading \(d\) and \(G \subseteq d g r a d\)-max-set \(d\) and \(\bigwedge u\). is-sig-GB-in d Gu
shows punit.is-Groebner-basis (rep-list' \(G\) )
using assms(1)
proof (rule punit.weak-GB-is-strong-GB-dgrad-p-set[simplified])
from \(\operatorname{assms}(1,3)\) show rep-list ' \(G \subseteq\) punit-dgrad-max-set \(d\) by (rule dgrad-max-3)
next
fix \(f::^{\prime} a \Rightarrow_{0}{ }^{\prime} b\)
assume \(f \in\) punit-dgrad-max-set \(d\)
assume \(f \in\) ideal (rep-list' \(G\) )
also from rep-list-subset-ideal have \(\ldots \subseteq\) ideal (set fs) by (rule ideal.span-subset-spanI)
finally have \(f \in\) ideal (set \(f s\) ) .
with assms(2) \(\langle f \in\) punit-dgrad-max-set d> obtain \(r\) where \(r \in\) dgrad-sig-set \(d\)
and \(f: f=\) rep-list \(r\)
by (rule in-idealE-rep-list-dgrad-sig-set)
from assms(4) this(1) refl have sig-red-zero \(\left(\preceq_{t}\right)\) Grby (rule is-sig-GB-inD)
then obtain \(s\) where \(\left(\operatorname{sig} \text {-red }\left(\preceq_{t}\right)(\preceq) G\right)^{* *} r s\) and \(s\) : rep-list \(s=0\) by (rule sig-red-zeroE)
from this (1) have (punit.red (rep-list' G) \()^{* *}\) (rep-list r) (rep-list s)
by (rule sig-red-red-rtrancl)
thus (punit.red (rep-list' \(G\) ) ) ** f 0 by (simp only: \(f s\) )
qed
lemma sig-red-zero-is-red:
assumes sig-red-zero sing-reg \(F r\) and rep-list \(r \neq 0\)
shows is-sig-red sing-reg ( \(\preceq\) ) Fr
proof -
from \(\operatorname{assms}(1)\) obtain \(s\) where \(*:(\text { sig-red sing-reg }(\preceq) F)^{* *} r s\) and rep-list \(s\) \(=0\)
by (rule sig-red-zeroE)
```

from this(2) assms(2) have $r \neq s$ by auto
with $*$ show ?thesis by (induct rule: converse-rtranclp-induct, auto simp: is-sig-red-def) qed
lemma is-sig-red-sing-top-is-red-zero:
assumes dickson-grading $d$ and is-sig-GB-upt $d G u$ and $a \in d g r a d-s i g-s e t ~ d ~$
and $l t a=u$
and is-sig-red $(=)(=) G a$ and $\neg i s$-sig-red $\left(\prec_{t}\right)(=) G a$
shows sig-red-zero $\left(\preceq_{t}\right) G a$
proof -
from $\operatorname{assms}(5)$ obtain $g$ where $g \in G$ and rep-list $g \neq 0$ and rep-list $a \neq 0$
and 1: punit.lt (rep-list g) adds punit.lt (rep-list a)
and 2: punit.lt (rep-list $a) \oplus l t g=$ punit.lt $($ rep-list $g) \oplus l t a$
by (rule is-sig-red-top-addsE)
from this (2, 3) have $g \neq 0$ and $a \neq 0$ by (auto simp: rep-list-zero)
hence $l c g \neq 0$ and $l c a \neq 0$ using lc-not-0 by blast+
from 1 have 3: (punit.lt (rep-list a) - punit.lt (rep-list g)) $\oplus$ lt $g=l t a$
by (simp add: term-is-le-rel-minus 2)
define $g^{\prime}$ where $g^{\prime}=$ monom-mult $(l c a / l c g)($ punit.lt (rep-list $a)-$ punit.lt $($ rep-list $g)) g$
from $\langle g \neq 0\rangle\langle l c$ a $\neq 0\rangle\langle l c g \neq 0\rangle$ have $l t-g^{\prime}: l t g^{\prime}=l t$ a by (simp add: $g^{\prime}$-def lt-monom-mult 3)
from $\langle l c g \neq 0\rangle$ have $l c-g^{\prime}: l c g^{\prime}=l c$ a by (simp add: $g^{\prime}$-def)
from $\operatorname{assms}(1)$ have $g^{\prime} \in d g r a d$-sig-set $d$ unfolding $g^{\prime}$-def
proof (rule dgrad-sig-set-closed-monom-mult)
from $\operatorname{assms}(1) 1$ have $d$ (punit.lt (rep-list a) - punit.lt $($ rep-list g)) $\leq d$ (punit.lt (rep-list a))
by (rule dickson-grading-minus)
also from $\operatorname{assms}(1,3)$ have $\ldots \leq d g r a d-m a x d$ by (rule dgrad-sig-setD-rep-list-lt)
finally show $d$ (punit.lt (rep-list a) - punit.lt (rep-list g)) $\leq$ dgrad-max $d$.
next

with $\langle g \in G\rangle$ show $g \in$ dgrad-sig-set $d$..
qed
with $\operatorname{assms}(3)$ have $b$-in: $a-g^{\prime} \in \operatorname{dgrad} d$-sig-set $d$ (is $? b \in-$ )
by (rule dgrad-sig-set-closed-minus)
from 1 have 4: punit.lt (rep-list a) - punit.lt (rep-list g) + punit.lt (rep-list g) $=$
punit.lt (rep-list a)
by (rule adds-minus)
show ?thesis
proof (cases lc a / lc g=punit.lc (rep-list a) / punit.lc (rep-list g))
case True
have sig-red-single $(=)(=) a ? b g$ (punit.lt (rep-list a) - punit.lt (rep-list g)) proof (rule sig-red-singleI)
show punit.lt (rep-list a) - punit.lt (rep-list g) + punit.lt (rep-list g) $\in$ keys (rep-list a)
unfolding 4 using 〈rep-list $a \neq 0\rangle$ by (rule punit.lt-in-keys)
next
show $? b=$
a－monom－mult
（lookup（rep－list a）（punit．lt（rep－list a）－punit．lt（rep－list g）＋punit．lt
$($ rep－list g））／
punit．lc（rep－list g））
（punit．lt（rep－list a）－punit．lt（rep－list g））g
by（simp add：$g^{\prime}$－def 4 punit．lc－def True）
qed（ simp－all add： 34 〈rep－list $g \neq 0$ 〉）
hence sig－red（＝）（＝）Ga？b unfolding sig－red－def using $\langle g \in G\rangle$ by blast
hence sig－red $\left(\preceq_{t}\right)\left(\preceq_{)} G a\right.$ ？b by（auto dest：sig－red－sing－regI sig－red－top－tailI）
hence 5：$\left(\operatorname{sig} \text {－red }\left(\preceq_{t}\right)(\preceq) G\right)^{* *} a$ ？b ．．
show ？thesis
proof（cases $? b=0$ ）
case True
hence rep－list ？$b=0$ by（simp only：rep－list－zero）
with 5 show ？thesis by（rule sig－red－zeroI）

## next

case False
hence $l t ? b \prec_{t}$ lt a using $l t-g^{\prime} l c-g^{\prime}$ by（rule lt－minus－lessI）
hence $l t ? b \prec_{t} u$ by（simp only：assms（4））
with assms（2）b－in have sig－red－zero $\left(\preceq_{t}\right) G ? b$ by（rule is－sig－GB－uptD3）
then obtain $s$ where $\left(\text { sig－red }\left(\preceq_{t}\right)(\preceq) G\right)^{* *} ? b s$ and rep－list $s=0$ by
（rule sig－red－zeroE）
from 5 this $(1)$ have $\left(\text { sig－red }\left(\preceq_{t}\right)(\preceq) G\right)^{* *} a s$ by（rule rtranclp－trans）
thus ？thesis using 〈rep－list $s=0$ 〉 by（rule sig－red－zeroI）
qed
next
case False
from 〈rep－list $g \neq 0\rangle\langle l c g \neq 0\rangle\langle l c a \neq 0\rangle$ have 5：punit．lt（rep－list $g^{\prime}$ ）$=$ punit．lt（rep－list a）
by（simp add：$g^{\prime}$－def rep－list－monom－mult punit．lt－monom－mult 4）
have 6：punit．lc（rep－list $\left.g^{\prime}\right)=\left(\begin{array}{ll}l c & a\end{array} / l c g\right) *$ punit．lc $($ rep－list $g)$
by（simp add：$g^{\prime}$－def rep－list－monom－mult）
also have 7：．．．$\neq$ punit．lc（rep－list a）
proof
assume lc a／lc g＊punit．lc（rep－list g）＝punit．lc（rep－list a）
moreover from 〈rep－list $g \neq 0\rangle$ have punit．lc（rep－list $g$ ）$\neq 0$ by（rule punit．lc－not－0）
ultimately have lc a／lc g＝punit．lc（rep－list a）／punit．lc（rep－list g）
by（simp add：field－simps）
with False show False ．．
qed
finally have punit．lc $\left(\right.$ rep－list $\left.g^{\prime}\right) \neq$ punit．lc（rep－list $\left.a\right)$ ．
with 5 have 8：punit．lt（rep－list ？b）$=$ punit．lt（rep－list a）unfolding rep－list－minus by（rule punit．lt－minus－eqI－3）
hence punit．lc（rep－list ？b）$=$ punit．lc $($ rep－list $a)-(l c a / l c ~ g) *$ punit．lc （rep－list g）
unfolding $6[$ symmetric $]$ by（simp only：punit．lc－def lookup－minus rep－list－minus
5)
also have $\ldots \neq 0$
proof
assume punit.lc (rep-list a) - lc a / lc $g *$ punit.lc (rep-list g) $=0$
hence lc a / lc g*punit.lc (rep-list g) = punit.lc (rep-list a) by simp
with 7 show False ..
qed
finally have rep-list $? b \neq 0$ by (simp add: punit.lc-eq-zero-iff)
hence ?b $\neq 0$ by (auto simp: rep-list-zero)
hence $l t$ ?b $\prec_{t}$ lt a using $l t-g^{\prime} l c-g^{\prime}$ by (rule lt-minus-lessI)
hence $l t ? b \prec_{t} u$ by (simp only: assms(4))
with assms(2) b-in have sig-red-zero $\left(\preceq_{t}\right) G ? b$ by (rule is-sig-GB-uptD3)
moreover note 〈rep-list ? $b \neq 0$ 〉
moreover have $\left(\preceq_{t}\right)=\left(\preceq_{t}\right) \vee\left(\preceq_{t}\right)=\left(\prec_{t}\right)$ by simp
ultimately have is-sig-red $\left(\preceq_{t}\right)(=) G$ ?b by (rule sig-red-zero-nonzero)
then obtain $g 0$ where $g 0 \in G$ and rep-list $g 0 \neq 0$
and 9: punit.lt (rep-list g0) adds punit.lt (rep-list ?b)
and 10: punit.lt (rep-list ?b) $\oplus l t ~ g 0 \preceq_{t}$ punit.lt (rep-list g0) $\oplus l t ? b$
by (rule is-sig-red-top-addsE)
from 9 have punit.lt (rep-list g0) adds punit.lt (rep-list a) by (simp only: 8)
from 10 have punit.lt (rep-list $a) \oplus l t g 0 \preceq_{t}$ punit.lt $($ rep-list g0) $\oplus l t ? b$ by (simp only: 8)
also from $\left\langle l t ? b \prec_{t} l t\right.$ a have $\ldots \prec_{t}$ punit.lt (rep-list g0) $\oplus l t$ a by (rule splus-mono-strict)
finally have punit.lt (rep-list a) $\oplus l t g 0 \prec_{t}$ punit.lt (rep-list g0) $\oplus l t a$.
have is-sig-red $\left(\prec_{t}\right)(=) G a$
proof (rule is-sig-red-top-addsI)
show ord-term-lin.is-le-rel $\left(\prec_{t}\right)$ by simp
qed fact+
with $\operatorname{assms}(6)$ show ?thesis ..
qed
qed
lemma sig-regular-reduced-unique:
assumes is-sig-GB-upt $d G(l t q)$ and $p \in d g r a d$-sig-set $d$ and $q \in d g r a d$-sig-set d
and lt $p=l t q$ and lc $p=l c q$ and $\neg i s$-sig-red $\left(\prec_{t}\right)(\preceq) G p$ and $\neg i s$-sig-red $\left(\prec_{t}\right)(\preceq) G q$
shows rep-list $p=$ rep-list $q$
proof (rule ccontr)
assume rep-list $p \neq$ rep-list $q$
hence rep-list $(p-q) \neq 0$ by (auto simp: rep-list-minus)
hence $p-q \neq 0$ by (auto simp: rep-list-zero)
hence $p+(-q) \neq 0$ by $\operatorname{simp}$
moreover from $\operatorname{assms}(4)$ have $l t(-q)=l t p$ by simp
moreover from $\operatorname{assms}(5)$ have $l c(-q)=-l c p$ by simp
ultimately have $l t(p+(-q)) \prec_{t} l t p$ by (rule lt-plus-lessI)
hence lt $(p-q) \prec_{t}$ lt $q$ using assms(4) by simp
with $\operatorname{assms}(1)$ have $i s$-sig-GB-in $d G(l t(p-q))$ by (rule is-sig-GB-uptD2)

```
    moreover from \(\operatorname{assms}(2,3)\) have \(p-q \in d g r a d\)-sig-set \(d\) by (rule dgrad-sig-set-closed-minus)
    ultimately have sig-red-zero \(\left(\preceq_{t}\right) G(p-q)\) using refl by (rule is-sig-GB-inD)
    hence is-sig-red \(\left(\preceq_{t}\right)(\preceq) G(p-q)\) using \(\langle r e p-l i s t ~(p-q) \neq 0\rangle\) by (rule
sig-red-zero-is-red)
    then obtain \(g t\) where \(g \in G\) and \(t: t \in\) keys (rep-list \((p-q))\) and rep-list \(g\)
\(\neq 0\)
    and adds: punit.lt (rep-list g) adds \(t\) and \(t \oplus l t g \preceq_{t}\) punit.lt (rep-list g) \(\oplus l t\)
( \(p-q\) )
    by (rule is-sig-red-addsE)
    note this(5)
    also from \(\left\langle l t(p-q) \prec_{t} l t q\right\rangle\) have punit.lt \((\) rep-list \(g) \oplus l t(p-q) \prec_{t}\) punit.lt
\((\) rep-list \(g) \oplus l t q\)
    by (rule splus-mono-strict)
    finally have \(1: t \oplus l t g \prec_{t}\) punit.lt (rep-list \(g\) ) \(\oplus l t q\).
    hence 2: \(t \oplus l t g \prec_{t}\) punit.lt (rep-list g) \(\oplus l t p\) by (simp only: assms(4))
    from \(t\) keys-minus have \(t \in\) keys (rep-list \(p\) ) \(\cup\) keys (rep-list \(q\) ) unfolding
rep-list-minus ..
    thus False
    proof
        assume \(t\)-in: \(t \in\) keys (rep-list \(p\) )
        hence \(t \preceq\) punit.lt (rep-list p) by (rule punit.lt-max-keys)
    with \(\langle g \in G\rangle\)-in \(\langle\) rep-list \(g \neq 0\rangle\) adds ord-term-lin.is-le-relI(3) ordered-powerprod-lin.is-le-relI(2)
2
    have is-sig-red \(\left(\prec_{t}\right)(\preceq) G p\) by (rule is-sig-red-addsI)
    with \(\operatorname{assms}(6)\) show False ..
    next
    assume \(t\)-in: \(t \in\) keys (rep-list q)
    hence \(t \preceq\) punit.lt (rep-list q) by (rule punit.lt-max-keys)
    with \(\langle g \in G\rangle\) t-in \(\langle\) rep-list \(g \neq 0\rangle\) adds ord-term-lin.is-le-relI(3) ordered-powerprod-lin.is-le-relI(2)
1
    have is-sig-red \(\left(\prec_{t}\right)(\preceq) G q\) by (rule is-sig-red-addsI)
    with \(\operatorname{assms}(7)\) show False ..
    qed
qed
corollary sig-regular-reduced-unique':
    assumes is-sig-GB-upt \(d G(l t q)\) and \(p \in d g r a d\)-sig-set \(d\) and \(q \in d g r a d\)-sig-set
\(d\)
    and lt \(p=l t q\) and \(\neg i s\)-sig-red \(\left(\prec_{t}\right)(\preceq) G p\) and \(\neg i s\)-sig-red \(\left(\prec_{t}\right)(\preceq) G q\)
    shows punit.monom-mult (lc q) 0 (rep-list \(p\) ) \(=\) punit.monom-mult (lc p) 0
(rep-list q)
proof (cases \(p=0 \vee q=0\) )
    case True
    thus ?thesis by (auto simp: rep-list-zero)
next
    case False
    hence \(p \neq 0\) and \(q \neq 0\) by simp-all
    hence lc \(p \neq 0\) and lc \(q \neq 0\) by (simp-all add: lc-not-0)
    let \(? p=\) monom-mult (lc q) \(0 p\)
```

```
    let \(? q=\) monom-mult (lc \(p\) ) \(0 q\)
    have \(l t ? q=l t q\) by (simp add: lt-monom-mult[OF \(\langle l c ~ p \neq 0\rangle\langle q \neq 0\rangle]\) splus-zero)
    with \(\operatorname{assms}(1)\) have \(i s\)-sig-GB-upt \(d G(l t ? q)\) by simp
```




```
    moreover from \(\langle l t ? q=l t ~ q\rangle\) have \(l t ? p=l t ? q\)
    by (simp add: lt-monom-mult \([O F\langle l c q \neq 0\rangle\langle p \neq 0\rangle]\) splus-zero \(\operatorname{assms}(4))\)
    moreover have \(l c ? p=l c\) ? \(q\) by simp
    moreover have \(\neg\) is-sig-red \(\left(\prec_{t}\right)(\preceq) G\) ?p
    proof
    assume is-sig-red \(\left(\prec_{t}\right)(\preceq) G ? p\)
    moreover from \(\langle l c q \neq 0\) 〉 have \(1 /(l c q) \neq 0\) by simp
    ultimately have is-sig-red \(\left(\prec_{t}\right)(\preceq) G\) (monom-mult \((1 / l c q) 0\) ?p) by (rule
is-sig-red-monom-mult)
    hence is-sig-red \(\left(\prec_{t}\right)(\preceq) G p\) by (simp add: monom-mult-assoc \(\langle l c q \neq 0\) ) \()\)
    with \(\operatorname{assms}(5)\) show False ..
    qed
    moreover have \(\neg\) is-sig-red \(\left(\prec_{t}\right)(\preceq) G ? q\)
    proof
    assume is-sig-red \(\left(\prec_{t}\right)(\preceq) G\) ? \(q\)
    moreover from \(\langle l c p \neq 0\) 〉 have \(1 /(l c p) \neq 0\) by simp
    ultimately have is-sig-red \(\left(\prec_{t}\right)(\preceq) G\) (monom-mult ( \(1 / l c p\) ) 0 ? \(q\) ) by (rule
is-sig-red-monom-mult)
    hence is-sig-red \(\left(\prec_{t}\right)(\preceq) G q\) by (simp add: monom-mult-assoc \(\left.\langle l c p \neq 0\rangle\right)\)
    with \(\operatorname{assms}(6)\) show False ..
    qed
    ultimately have rep-list ?p = rep-list ?q by (rule sig-regular-reduced-unique)
    thus ?thesis by (simp only: rep-list-monom-mult)
qed
lemma sig-regular-top-reduced-lt-lc-unique:
    assumes dickson-grading \(d\) and is-sig-GB-upt \(d G(l t ~ q)\) and \(p \in d g r a d\)-sig-set
\(d\) and \(q \in d g r a d-s i g-s e t ~ d ~ d\)
    and \(l t p=l t q\) and \((p=0) \longleftrightarrow(q=0)\) and \(\neg\) is-sig-red \(\left(\prec_{t}\right)(=) G p\) and
\(\neg i s\)-sig-red \(\left(\prec_{t}\right)(=) G q\)
    shows punit.lt \((\) rep-list \(p)=\) punit.lt \((\) rep-list \(q) \wedge l c q *\) punit.lc \((\) rep-list \(p)=\)
lc \(p\) * punit.lc (rep-list q)
proof (cases \(p=0\) )
    case True
    with \(\operatorname{assms}(6)\) have \(q=0\) by \(\operatorname{simp}\)
    thus ?thesis by (simp add: True)
next
    case False
    with \(\operatorname{assms}(6)\) have \(q \neq 0\) by \(\operatorname{simp}\)
    from False have lc \(p \neq 0\) by (rule lc-not-0)
    from \(\langle q \neq 0\rangle\) have \(l c q \neq 0\) by (rule lc-not-0)
```



```
    hence \(G \subseteq d g r a d-m a x-s e t ~ d\) by (simp add: dgrad-sig-set'-def)
    with assms (1) obtain \(p^{\prime}\) where \(p^{\prime}\)-red: \(\left(\operatorname{sig} \text {-red }\left(\prec_{t}\right)(\prec) G\right)^{* *} p p^{\prime}\) and \(\neg\)
```

```
is-sig-red ( }\mp@subsup{\prec}{t}{})(\prec)G\mp@subsup{p}{}{\prime
```

    by (rule sig-irredE-dgrad-max-set)
    from this(1) have \(l t-p^{\prime}: l t p^{\prime}=l t p\) and \(l t-p^{\prime \prime}:\) punit.lt (rep-list \(\left.p^{\prime}\right)=\) punit.lt
    (rep-list p)
and $l c-p^{\prime}: l c p^{\prime}=l c p$ and $l c-p^{\prime \prime}:$ punit.lc (rep-list $\left.p^{\prime}\right)=$ punit.lc (rep-list $\left.p\right)$
by (rule sig-red-regular-rtrancl-lt, rule sig-red-tail-rtrancl-lt-rep-list,
rule sig-red-regular-rtrancl-lc, rule sig-red-tail-rtrancl-lc-rep-list)
have $\neg$ is-sig-red $\left(\prec_{t}\right)(=) G p^{\prime}$
proof
assume $a$ : is-sig-red $\left(\prec_{t}\right)(=) G p^{\prime}$
hence rep-list $p^{\prime} \neq 0$ using is-sig-red-top-addsE by blast
hence rep-list $p \neq 0$ using $\left\langle\left(\operatorname{sig} \text {-red }\left(\prec_{t}\right)(\prec) G\right)^{* *} p p^{\prime}\right.$ 〉
by (auto simp: punit.rtrancl-0 dest!: sig-red-red-rtrancl)
with $a$ have is-sig-red $\left(\prec_{t}\right)(=) G p$ using $l t$ - $p^{\prime} l t-p^{\prime \prime}$ by (rule is-sig-red-top-cong)
with $\operatorname{assms}(7)$ show False ..
qed
with $\left\langle\neg\right.$ is-sig-red $\left(\prec_{t}\right)(\prec) G p^{\prime}>$ have 1 : $\neg$ is-sig-red $\left(\prec_{t}\right)(\preceq) G p^{\prime}$ by (simp
add: is-sig-red-top-tail-cases)
from $\operatorname{assms}(1)\langle G \subseteq$ dgrad-max-set $d\rangle$ obtain $q^{\prime}$ where $q^{\prime}$-red: $\left(\right.$ sig-red $\left(\prec_{t}\right)$
$(\prec) G)^{* *} q q^{\prime}$
and $\neg$ is-sig-red $\left(\prec_{t}\right)(\prec) G q^{\prime}$ by (rule sig-irredE-dgrad-max-set)
from this(1) have $l t-q^{\prime}: l t q^{\prime}=l t q$ and $l t-q^{\prime \prime}:$ punit.lt (rep-list $\left.q^{\prime}\right)=$ punit.lt
(rep-list q)
and $l c-q^{\prime}: l c q^{\prime}=l c \quad q$ and $l c-q^{\prime \prime}:$ punit.lc $\left(\right.$ rep-list $\left.q^{\prime}\right)=$ punit.lc $($ rep-list $q)$
by (rule sig-red-regular-rtrancl-lt, rule sig-red-tail-rtrancl-lt-rep-list,
rule sig-red-regular-rtrancl-lc, rule sig-red-tail-rtrancl-lc-rep-list)
have $\neg$ is-sig-red $\left(\prec_{t}\right)(=) G q^{\prime}$
proof
assume a: is-sig-red $\left(\prec_{t}\right)(=) G q^{\prime}$
hence rep-list $q^{\prime} \neq 0$ using is-sig-red-top-addsE by blast
hence rep-list $q \neq 0$ using $\left\langle\left(\text { sig-red }\left(\prec_{t}\right)(\prec) G\right)^{* *} q q^{\prime}\right\rangle$
by (auto simp: punit.rtrancl-0 dest!: sig-red-red-rtrancl)
with $a$ have is-sig-red $\left(\prec_{t}\right)(=) G q$ using $l t-q^{\prime} l t-q^{\prime \prime}$ by (rule is-sig-red-top-cong)
with $\operatorname{assms}(8)$ show False ..
qed
with $\left\langle\neg\right.$ is-sig-red $\left(\prec_{t}\right)(\prec) G q^{\prime}>$ have 2: $\neg$ is-sig-red $\left(\prec_{t}\right)(\preceq) G q^{\prime}$ by (simp
add: is-sig-red-top-tail-cases)
from assms(2) have is-sig-GB-upt d G (lt $q^{\prime}$ ) by (simp only: lt-q')
moreover from assms(1) G-sub assms(3) $p^{\prime}$-red have $p^{\prime} \in d g r a d$-sig-set $d$
by (rule dgrad-sig-set-closed-sig-red-rtrancl)
moreover from assms(1) G-sub assms(4) $q^{\prime}$-red have $q^{\prime} \in d g r a d$-sig-set $d$
by (rule dgrad-sig-set-closed-sig-red-rtrancl)
moreover have lt $p^{\prime}=l t q^{\prime}$ by (simp only: lt-p $p^{\prime} l t-q^{\prime} \operatorname{assms}(5)$ )
ultimately have eq: punit.monom-mult (lc $\left.q^{\prime}\right) 0\left(\right.$ rep-list $\left.p^{\prime}\right)=$ punit.monom-mult
(lc $p^{\prime}$ ) 0 (rep-list $q^{\prime}$ )
using 12 by (rule sig-regular-reduced-unique)
have lc $q$ * punit.lc (rep-list $p$ ) $=$ lc $q$ * punit.lc (rep-list $p^{\prime}$ ) by (simp only:
$\left.l c-p^{\prime \prime}\right)$

```
    also from «lc q\not=0` have ... = punit.lc (punit.monom-mult (lc q') 0 (rep-list
p'))
    by (simp add:lc-q')
    also have ... = punit.lc (punit.monom-mult (lc p') 0 (rep-list q')) by (simp only:
eq)
    also from «lc p\not=0\rangle have ... = lc p * punit.lc (rep-list q') by (simp add: lc-p')
    also have ... = lc p * punit.lc (rep-list q) by (simp only:lc-q'')
    finally have *:lc q* punit.lc (rep-list p)=lc p* punit.lc (rep-list q).
    have punit.lt (rep-list p) = punit.lt (rep-list p') by (simp only:lt-p'')
    also from <lc q\not=0〉 have ... = punit.lt (punit.monom-mult (lc q') 0 (rep-list
p'))
    by (simp add: lc-q' punit.lt-monom-mult-zero)
    also have ... = punit.lt (punit.monom-mult (lc p') 0 (rep-list q')) by (simp only:
eq)
    also from <lc p \not=0〉 have ... = punit.lt (rep-list q') by (simp add: lc-p'
punit.lt-monom-mult-zero)
    also have ... = punit.lt (rep-list q) by (fact lt-q'')
    finally show ?thesis using * ..
qed
corollary sig-regular-top-reduced-lt-unique:
    assumes dickson-grading d and is-sig-GB-upt d G (lt q) and p \indgrad-sig-set
d
            and q\indgrad-sig-set d and lt p=lt q and p\not=0 and q\not=0
            and }\neg\mathrm{ is-sig-red ( }\mp@subsup{\prec}{t}{})(=)Gp\mathrm{ and }\negis\mathrm{ -sig-red ( }\mp@subsup{\prec}{t}{})(=)G
    shows punit.lt (rep-list p) = punit.lt (rep-list q)
proof -
    from assms(6,7) have }(p=0)\longleftrightarrow(q=0) by sim
    with assms(1, 2, 3, 4, 5)
    have punit.lt (rep-list p)= punit.lt (rep-list q) ^ lc q* punit.lc (rep-list p) = lc
p * punit.lc (rep-list q)
            using assms(8,9) by (rule sig-regular-top-reduced-lt-lc-unique)
    thus ?thesis ..
qed
corollary sig-regular-top-reduced-lc-unique:
    assumes dickson-grading d and is-sig-GB-upt d G(lt q) and p d dgrad-sig-set
d and q}\indgrad-sig-set d
    and lt p=lt q and lc p=lc q and }\neg\mathrm{ is-sig-red ( (< ) (=)Gp and }\negis\mathrm{ -sig-red
(\mp@subsup{\prec}{t}{})(=)Gq
    shows punit.lc (rep-list p) = punit.lc (rep-list q)
proof (cases p=0)
    case True
    with assms(6) have q=0 by (simp add: lc-eq-zero-iff)
    with True show ?thesis by simp
next
    case False
    hence lc p\not=0 by (rule lc-not-0)
```

hence lc $q \neq 0$ by（simp add：assms（6））
hence $q \neq 0$ by（simp add：lc－eq－zero－iff）
with False have $(p=0) \longleftrightarrow(q=0)$ by simp
with $\operatorname{assms}(1,2,3,4,5)$
have punit．lt $($ rep－list $p)=$ punit．lt $($ rep－list $q) \wedge l c q *$ punit．lc $($ rep－list $p)=l c$ $p *$ punit．lc（rep－list $q$ ）
using $\operatorname{assms}(7,8)$ by（rule sig－regular－top－reduced－lt－lc－unique）
hence lc $q *$ punit．lc（rep－list $p$ ）$=l c p *$ punit．lc（rep－list $q$ ）．．
also have $\ldots=l c q *$ punit．lc（rep－list $q$ ）by（simp only：assms（ 6 ））
finally show ？thesis using 〈lc $q \neq 0$ 〉 by simp
qed
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 $G^{\prime} \subseteq d g r a d$－sig－set $d$
and $\bigwedge u$ ．$d($ pp－of－term $u) \leq$ dgrad－max $d \Longrightarrow$ component－of－term $u<$ length
$f s \Longrightarrow i s$－sig－GB－in d $G^{\prime} u$
and $g \in G$ and rep－list $g \neq 0$
obtains $g^{\prime}$ where $g^{\prime} \in G^{\prime}$ and rep－list $g^{\prime} \neq 0$ and $l t g^{\prime}=l t g$
and punit．lt $\left(\right.$ rep－list $\left.g^{\prime}\right)=$ punit．lt $($ rep－list $g)$
proof－
from $\operatorname{assms}(1)$ have $G \subseteq$ dgrad－sig－set $d$
and $1: \bigwedge u$ ．$d($ pp－of－term $u) \leq$ dgrad－max $d \Longrightarrow$ component－of－term $u<$ length
$f s \Longrightarrow i s$－sig－GB－in d Gu
and 2：$\bigwedge g 0 . g 0 \in G \Longrightarrow \neg i s$－sig－red $\left(\preceq_{t}\right)(=)(G-\{g 0\}) g 0$
by（simp－all add：is－min－sig－GB－def）
from $\operatorname{assms}(4)$ have 3：$\neg$ is－sig－red $\left(\preceq_{t}\right)(=)(G-\{g\}) g$ by（rule 2）
from $\operatorname{assms}(5)$ have $g \neq 0$ by（auto simp：rep－list－zero）
from $\operatorname{assms}(4)\langle G \subseteq$ dgrad－sig－set $d\rangle$ have $g \in$ dgrad－sig－set $d$ ．．
hence $d(l p g) \leq d g r a d-m a x ~ d$ and component－of－term（lt g）＜length $f s$
by（rule dgrad－sig－setD－lp，rule dgrad－sig－setD－lt［OF－$\langle g \neq 0\rangle]$ ）
hence $i s$－sig－GB－in $d G^{\prime}(l t g)$ by（rule assms（3））
hence sig－red－zero $\left(\preceq_{t}\right) G^{\prime} g$ using $\langle g \in d g r a d$－sig－set $d\rangle$ refl by（rule is－sig－GB－inD）
moreover note assms（5）
moreover have $\left(\preceq_{t}\right)=\left(\preceq_{t}\right) \vee\left(\preceq_{t}\right)=\left(\prec_{t}\right)$ by simp
ultimately have is－sig－red $\left(\preceq_{t}\right)(=) G^{\prime} g$ by（rule sig－red－zero－nonzero）
then obtain $g^{\prime}$ where $g^{\prime} \in G^{\prime}$ and rep－list $g^{\prime} \neq 0$
and adds1：punit．lt（rep－list $g^{\prime}$ ）adds punit．lt（rep－list g）
and le1：punit．lt $($ rep－list $g) \oplus l t g^{\prime} \preceq_{t}$ punit．lt $\left(\right.$ rep－list $\left.g^{\prime}\right) \oplus l t g$
by（rule is－sig－red－top－addsE）
from 〈rep－list $\left.g^{\prime} \neq 0\right\rangle$ have $g^{\prime} \neq 0$ by（auto simp：rep－list－zero）
from $\left\langle g^{\prime} \in G^{\prime}\right\rangle \operatorname{assms}(2)$ have $g^{\prime} \in d g r a d$－sig－set $d$ ．．
hence $d\left(l p g^{\prime}\right) \leq d g r a d-m a x d$ and component－of－term $\left(l t g^{\prime}\right)<l e n g t h f s$
by（rule dgrad－sig－setD－lp，rule dgrad－sig－setD－lt $\left.\left[O F-\left\langle g^{\prime} \neq 0\right\rangle\right]\right)$
hence $i s$－sig－GB－in d $G$（lt $g^{\prime}$ ）by（rule 1）
hence sig－red－zero $\left(\preceq_{t}\right) G g^{\prime}$ using $\left\langle g^{\prime} \in d g r a d\right.$－sig－set $\left.d\right\rangle$ refl by（rule is－sig－GB－inD）
moreover note $\left\langle\right.$ rep-list $\left.g^{\prime} \neq 0\right\rangle$
moreover have $\left(\preceq_{t}\right)=\left(\preceq_{t}\right) \vee\left(\preceq_{t}\right)=\left(\prec_{t}\right)$ by simp
ultimately have is-sig-red $\left(\preceq_{t}\right)(=) G g^{\prime}$ by (rule sig-red-zero-nonzero)
then obtain $g 0$ where $g 0 \in G$ and rep-list $g 0 \neq 0$
and adds2: punit.lt (rep-list g0) adds punit.lt (rep-list g')
and le2: punit.lt (rep-list $\left.g^{\prime}\right) \oplus l t ~ g 0 \preceq_{t}$ punit.lt (rep-list g0) $\oplus l t g^{\prime}$
by (rule is-sig-red-top-addsE)

```
have eq1: \(g 0=g\)
proof (rule ccontr)
    assume \(g 0 \neq g\)
    with \(\langle g 0 \in G\rangle\) have \(g 0 \in G-\{g\}\) by simp
    moreover note 〈rep-list \(g 0 \neq 0\) 〉 assms(5)
    moreover from adds2 adds1 have punit.lt (rep-list g0) adds punit.lt (rep-list
g)
        by (rule adds-trans)
    moreover have ord-term-lin.is-le-rel \(\left(\preceq_{t}\right)\) by simp
    moreover have punit.lt \((\) rep-list \(g) \oplus l t g 0 \preceq_{t}\) punit.lt (rep-list g0) \(\oplus l t g\)
    proof (rule ord-term-canc)
        have punit.lt \(\left(\right.\) rep-list \(\left.g^{\prime}\right) \oplus(\) punit.lt \((\) rep-list \(g) \oplus l t g 0)=\)
            punit.lt \((\) rep-list \(g) \oplus\left(\right.\) punit.lt \(\left(\right.\) rep-list \(\left.g^{\prime}\right) \oplus\) lt g0) by \((\) fact splus-left-commute \()\)
        also from le2 have \(\ldots \preceq_{t}\) punit.lt \((\) rep-list \(g) \oplus\left(\right.\) punit.lt \(\left(\right.\) rep-list g0) \(\left.\oplus l t g^{\prime}\right)\)
        by (rule splus-mono)
        also have \(\ldots=\) punit.lt \(\left(\right.\) rep-list g0) \(\oplus\left(\right.\) punit.lt \((\) rep-list \(\left.g) \oplus l t g^{\prime}\right)\)
            by (fact splus-left-commute)
        also from le1 have \(\ldots \preceq_{t}\) punit.lt (rep-list g0) \(\oplus\left(\right.\) punit.lt \(\left(\right.\) rep-list \(\left.g^{\prime}\right) \oplus l t\)
g)
                by (rule splus-mono)
        also have \(\ldots=\) punit.lt \(\left(\right.\) rep-list \(\left.g^{\prime}\right) \oplus(\) punit.lt \((\) rep-list \(g 0) \oplus l t g)\)
                by (fact splus-left-commute)
            finally show punit.lt \(\left(\right.\) rep-list \(\left.g^{\prime}\right) \oplus(\) punit.lt \((\) rep-list \(g) \oplus l t g 0) \preceq_{t}\)
                    punit.lt \(\left(\right.\) rep-list \(\left.g^{\prime}\right) \oplus(\) punit.lt \((\) rep-list \(g 0) \oplus l t g)\).
    qed
    ultimately have is-sig-red \(\left(\preceq_{t}\right)(=)(G-\{g\}) g\) by (rule is-sig-red-top-addsI)
    with 3 show False ..
qed
```

from adds2 adds1 have eq2: punit.lt (rep-list g') = punit.lt (rep-list g) by (simp add: eq1 adds-antisym)
with le1 le2 have punit.lt (rep-list g) $\oplus l t g^{\prime}=$ punit.lt (rep-list g) $\oplus l t g$ by (simp add: eq1)
hence lt $g^{\prime}=l t g$ by (simp only: splus-left-canc)
with $\left\langle g^{\prime} \in G^{\prime}\right\rangle\left\langle r e p-l i s t ~ g^{\prime} \neq 0\right\rangle$ show ?thesis using eq2 ..
qed
lemma sig-red-zero-regularI-adds:
assumes dickson-grading $d$ and is-sig-GB-upt $d G$ (lt q)
and $p \in d g r a d-s i g-s e t ~ d$ and $q \in d g r a d-s i g-s e t ~ d$ and $p \neq 0$ and sig-red-zero $\left(\prec_{t}\right) G p$
and $l t$ p addst $l t$ q
shows sig－red－zero $\left(\prec_{t}\right) G q$
proof（cases $q=0$ ）
case True
hence rep－list $q=0$ by（simp only：rep－list－zero）
with rtrancl－refl［to－pred］show ？thesis by（rule sig－red－zeroI）

## next

case False
hence lc $q \neq 0$ by（rule lc－not－0）
moreover from assms（5）have $l c p \neq 0$ by（rule lc－not－0）
ultimately have $l c q / l c p \neq 0$ by simp
from $\operatorname{assms}(7)$ have eq1：$(l p q-l p p) \oplus l t p=l t q$
by（metis add－diff－cancel－right＇adds－termE pp－of－term－splus）
from $\operatorname{assms}(7)$ have $l p p$ adds $l p q$ by（simp add：adds－term－def）
with assms（1）have $d(l p q-l p p) \leq d(l p q)$ by（rule dickson－grading－minus）
also from $\operatorname{assms}(4)$ have $\ldots \leq$ dgrad－max $d$ by（rule dgrad－sig－setD－lp）
finally have $d(l p q-l p p) \leq d g r a d-m a x d$ ．

hence $G \subseteq$ dgrad－max－set $d$ by（simp add：dgrad－sig－set＇－def）
let ？mult $=\lambda r$ ．monom－mult $(l c q /$ lc $p)(l p q-l p p) r$
from $\operatorname{assms}(6)$ obtain $p^{\prime}$ where $p$－red：$\left(\operatorname{sig} \text {－red }\left(\prec_{t}\right)(\preceq) G\right)^{* *} p p^{\prime}$ and rep－list $p^{\prime}=0$
by（rule sig－red－zeroE）
from $p$－red have $l t p^{\prime}=l t p$ and $l c p^{\prime}=l c p$
by（rule sig－red－regular－rtrancl－lt，rule sig－red－regular－rtrancl－lc）
hence $p^{\prime} \neq 0$ using $\langle l c p \neq 0\rangle$ by auto
with 〈lc $q / l c p \neq 0$ 〉 have ？mult $p^{\prime} \neq 0$ by（simp add：monom－mult－eq－zero－iff）
from 〈lc $q / l c p \neq 0\rangle\left\langle p^{\prime} \neq 0\right\rangle$ have $l t\left(? m u l t p^{\prime}\right)=l t q$
by（simp add：lt－monom－mult 〈lt $\left.p^{\prime}=l t p\right\rangle$ eq1）
from $\langle l c p \neq 0\rangle$ have $l c\left(\right.$ ？mult $\left.p^{\prime}\right)=l c q$ by（simp add：«lc $\left.\left.p^{\prime}=l c p\right\rangle\right)$
from $p$－red have mult－p－red：$\left(\text { sig－red }\left(\prec_{t}\right)(\preceq) G\right)^{* *}($ ？mult $p)\left(\right.$ ？mult $\left.p^{\prime}\right)$
by（rule sig－red－rtrancl－monom－mult）
have rep－list（？mult $\left.p^{\prime}\right)=0$ by（simp add：rep－list－monom－mult $\left\langle\right.$ rep－list $p^{\prime}=$ 0〉）
hence mult－p＇－irred：$\neg$ is－sig－red $\left(\prec_{t}\right)(\preceq) G$（？mult $\left.p^{\prime}\right)$
using is－sig－red－addsE by fastforce
from $\operatorname{assms}(1) G$－sub assms（3）$p$－red have $p^{\prime} \in d g r a d-s i g-s e t ~ d ~ d$ by（rule dgrad－sig－set－closed－sig－red－rtrancl）
with $\operatorname{assms}(1)\langle d(l p q-l p p) \leq d g r a d-m a x d\rangle$ have ？mult $p^{\prime} \in d g r a d$－sig－set $d$ by（rule dgrad－sig－set－closed－monom－mult）
from $\operatorname{assms}(1)\langle G \subseteq$ dgrad－max－set $d\rangle$ obtain $q^{\prime}$ where $q$－red：$\left(\right.$ sig－red $\left(\prec_{t}\right)(\preceq)$ $G)^{* *} q q^{\prime}$
and $q^{\prime}$－irred：$\neg$ is－sig－red $\left(\prec_{t}\right)(\preceq) G q^{\prime}$ by（rule sig－irredE－dgrad－max－set）
from $q$－red have $l t q^{\prime}=l t q$ and $l c q^{\prime}=l c q$
by（rule sig－red－regular－rtrancl－lt，rule sig－red－regular－rtrancl－lc）
hence $q^{\prime} \neq 0$ using $\langle l c q \neq 0\rangle$ by auto
from assms（2）have $i s$－sig－GB－upt d G（lt（？mult $\left.p^{\prime}\right)$ ）by（simp only：＜lt（？mult $\left.\left.\left.p^{\prime}\right)=l t q\right\rangle\right)$
moreover from $\operatorname{assms}(1) G$－sub assms（4）$q$－red have $q^{\prime} \in d g r a d$－sig－set $d$
by（rule dgrad－sig－set－closed－sig－red－rtrancl）
moreover note 〈？mult $p^{\prime} \in$ dgrad－sig－set d〉
moreover have $l t q^{\prime}=l t$（？mult $p^{\prime}$ ）by（simp only：〈lt（？mult $p^{\prime}$ ）$\left.=l t q\right\rangle\left\langle l t q^{\prime}\right.$ $=l t q\rangle)$
moreover have $l c q^{\prime}=l c\left(?\right.$ mult $\left.p^{\prime}\right)$ by（simp only：$\left\langle l c\left(?\right.\right.$ mult $\left.\left.p^{\prime}\right)=l c q\right\rangle\left\langle l c q^{\prime}\right.$ $=l c \quad q\rangle)$
ultimately have rep－list $q^{\prime}=$ rep－list（？mult $p^{\prime}$ ）using $q^{\prime}$－irred mult－$p^{\prime}$－irred
by（rule sig－regular－reduced－unique）
with $\left\langle\right.$ rep－list $\left(\right.$ ？mult $\left.\left.p^{\prime}\right)=0\right\rangle$ have rep－list $q^{\prime}=0$ by simp
with $q$－red show ？thesis by（rule sig－red－zeroI）
qed
lemma is－syz－sigI：
assumes $s \neq 0$ and $l t s=u$ and $s \in d g r a d$－sig－set $d$ and rep－list $s=0$
shows is－syz－sig $d u$
unfolding is－syz－sig－def using assms by blast
lemma is－syz－sigE：
assumes is－syz－sig d $u$
obtains $r$ where $r \neq 0$ and $l t r=u$ and $r \in d g r a d-$ sig－set $d$ and rep－list $r=$ 0
using assms unfolding is－syz－sig－def by blast
lemma is－syz－sig－adds：
assumes dickson－grading $d$ and is－syz－sig $d u$ and $u a d d s_{t} v$
and $d$（pp－of－term $v) \leq d g r a d-m a x d$
shows is－syz－sig dv
proof－
from assms（2）obtain $s$ where $s \neq 0$ and $l t s=u$ and $s \in d g r a d$－sig－set $d$ and rep－list $s=0$ by（rule is－syz－sigE）
from $\operatorname{assms}(3)$ obtain $t$ where $v: v=t \oplus u$ by（rule adds－termE）
show ？thesis
proof（rule is－syz－sigI）
from $\langle s \neq 0\rangle$ show monom－mult 1 t $s \neq 0$ by（simp add：monom－mult－eq－zero－iff）
next
from $\langle s \neq 0\rangle$ show $l t$（monom－mult $1 t s$ ）$=v$ by（simp add：lt－monom－mult $v<l t s=u\rangle)$
next
from $\operatorname{assms}(4)$ have $d(t+p p$－of－term $u) \leq d g r a d-m a x d$ by（simp add：$v$ term－simps）
with assms（1）have $d t \leq d g r a d-m a x d$ by（simp add：dickson－gradingD1）
with $\operatorname{assms}(1)$ show monom－mult $1 t s \in d g r a d-s i g-s e t ~ d u s i n g ~<s \in d g r a d-s i g-s e t$ d＞
by（rule dgrad－sig－set－closed－monom－mult）
next
show rep－list（monom－mult $1 t s)=0$ by（simp add：$\langle r e p-l i s t s=0\rangle$ rep－list－monom－mult）
qed
qed
lemma syzygy-crit:
assumes dickson-grading $d$ and $i s$-sig-GB-upt $d G u$ and $i s-s y z-s i g d u$
and $p \in d g r a d-s i g-s e t ~ d ~ a n d ~ l t ~ p=u$
shows sig-red-zero $\left(\prec_{t}\right) G p$
proof -
from assms(3) obtain $s$ where $s \neq 0$ and $l t s=u$ and $s \in d g r a d$-sig-set $d$ and rep-list $s=0$ by (rule is-syz-sigE)
note assms(1)
moreover from assms(2) have is-sig-GB-upt $d G$ (lt p) by (simp only: assms(5))
moreover note $\langle s \in d g r a d$-sig-set $d\rangle \operatorname{assms}(4)\langle s \neq 0\rangle$
moreover from rtranclp.rtrancl-refl $\langle$ rep-list $s=0\rangle$ have sig-red-zero $\left(\prec_{t}\right) G s$ by (rule sig-red-zeroI)
moreover have $l t s a d d s_{t} l t p$ by (simp only: assms(5) <lt $\left.s=u\right\rangle$ adds-term-refl)
ultimately show ?thesis by (rule sig-red-zero-regularI-adds)
qed
lemma lemma-21:
assumes dickson-grading $d$ and is-sig-GB-upt $d G$ (lt $p$ ) and $p \in d g r a d$-sig-set
$d$ and $g \in G$
and rep-list $p \neq 0$ and rep-list $g \neq 0$ and lt $g$ addst ${ }_{t}$ lt $p$
and punit.lt (rep-list g) adds punit.lt (rep-list p)
shows is-sig-red $\left(\preceq_{t}\right)(=) G p$
proof -
let $? l p=$ punit.lt (rep-list $p$ )
define $s$ where $s=$ ?lp - punit.lt (rep-list g)
from $\operatorname{assms}(8)$ have $s: ? l p=s+$ punit.lt (rep-list g) by (simp add: s-def mi-
nus-plus)
from $\operatorname{assms}(7)$ obtain $t$ where $l t-p$ : lt $p=t \oplus l t g$ by (rule adds-termE)
show ?thesis
proof (cases $s \oplus l t g \preceq_{t}$ lt $p$ )
case True
hence ?lp $\oplus$ lt $g \preceq_{t}$ punit.lt $($ rep-list $g) \oplus$ lt $p$
by (simp add: s splus-assoc splus-left-commute[of s] splus-mono)
with $\operatorname{assms}(4,6,5,8)$ ord-term-lin.is-le-relI(2) show ?thesis
by (rule is-sig-red-top-addsI)
next
case False
hence lt $p \prec_{t} s \oplus l t g$ by simp
hence $t \prec s$ by (simp add: lt-p ord-term-strict-canc-left)
hence $t+$ punit.lt (rep-list g) $\prec s+$ punit.lt (rep-list g) by (rule plus-monotone-strict)
hence $t+$ punit.lt (rep-list $g$ ) $\prec$ ?lp by (simp only: $s$ )
from $\operatorname{assms}(5)$ have $p \neq 0$ by (auto simp: rep-list-zero)
hence lc $p \neq 0$ by (rule lc-not-0)
from $\operatorname{assms}(6)$ have $g \neq 0$ by (auto simp: rep-list-zero)
hence lc $g \neq 0$ by (rule lc-not-0)
with 〈lc $p \neq 0$ 〉 have 1 : lc $p / l c g \neq 0$ by simp

```
    let ?g = monom-mult (lc p / lc g) tg
    from 1\langleg\not=0\rangle have lt ?g=lt p unfolding lt-p by (rule lt-monom-mult)
    from <lc g\not=0\rangle have lc ?g=lc p by simp
    have punit.lt (rep-list ?g) = t + punit.lt (rep-list g)
    unfolding rep-list-monom-mult using 1 assms(6) by (rule punit.lt-monom-mult[simplified])
    also have ... \prec?lp by fact
    finally have punit.lt (rep-list ?g) \prec?lp .
    hence lt-pg: punit.lt (rep-list (p-?g)) = ?lp and rep-list p}\not=\mathrm{ rep-list ?g
    by (auto simp: rep-list-minus punit.lt-minus-eqI-2)
    from this(2) have rep-list ( p-?g) \not=0 and p-?g\not=0
        by (auto simp: rep-list-minus rep-list-zero)
    from assms(2) have G\subseteqdgrad-sig-set d by (rule is-sig-GB-uptD1)
    note assms(1)
    moreover have dt\leqdgrad-max d
    proof (rule le-trans)
    have lp p=t+lpg by (simp add:lt-p term-simps)
    with assms(1) show dt\leqd (lp p) by (simp add:dickson-grading-adds-imp-le)
    next
        from assms(3) show d (lp p) \leqdgrad-max d by (rule dgrad-sig-setD-lp)
    qed
    moreover from assms(4)\langleG\subseteqdgrad-sig-set d> have g G dgrad-sig-set d ..
    ultimately have ?g \in dgrad-sig-set d by (rule dgrad-sig-set-closed-monom-mult)
    note assms(2)
    moreover from assms(3)<?g\indgrad-sig-set d> have p - ?g \indgrad-sig-set
d
    by (rule dgrad-sig-set-closed-minus)
    moreover from <p-?g\not=0\rangle\langlelt ?g=lt p\rangle\langlelc?g=lc p\rangle have lt (p-?g)
< lt p
        by (rule lt-minus-lessI)
    ultimately have sig-red-zero ( }\mp@subsup{\preceq}{t}{})G(p-?g
        by (rule is-sig-GB-uptD3)
    moreover note <rep-list (p-?g)}\not=0
```



```
    ultimately have is-sig-red ( }\mp@subsup{\preceq}{t}{})(=)G(p-?g) by (rule sig-red-zero-nonzero
    then obtain g1 where g1 \inG and rep-list g1 \not=0
        and 2: punit.lt (rep-list g1) adds punit.lt (rep-list (p - ?g))
        and 3: punit.lt (rep-list (p-?g))\opluslt g1 \preceq_ punit.lt (rep-list g1) }\opluslt (p
?g)
        by (rule is-sig-red-top-addsE)
    from }\langleg1\inG\rangle\langlerep-list g1 \not=0\rangle assms(5) show ?thesi
    proof (rule is-sig-red-top-addsI)
        from 2 show punit.lt (rep-list g1) adds punit.lt (rep-list p) by (simp only:
lt-pg)
    next
        have ?lp \oplus lt g1 = punit.lt (rep-list ( p - ?g)) \opluslt g1 by (simp only:lt-pg)
        also have ... \preceq́t punit.lt (rep-list g1) \opluslt (p-?g) by (fact 3)
```

```
            also from <lt (p-?g) < lt lt p> have ... < <t punit.lt (rep-list g1) \oplus lt p
            by (rule splus-mono-strict)
                            finally show ?lp }\opluslt g1 \mp@subsup{\preceq}{t punit.lt (rep-list g1) }{\mathrm{ f lt p by (rule ord-term-lin.less-imp-le)}
    qed simp
    qed
qed
```


### 4.2.3 Rewrite Bases

definition is-rewrite-ord $::\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\right.$ bool $) \Rightarrow$ bool where is-rewrite-ord rword $\longleftrightarrow$ (reflp rword $\wedge$ transp rword $\wedge(\forall a b$. rword ab $\vee$ rword $b$ a) $\wedge$
$(\forall a b$. rword $a b \longrightarrow$ rword $b a \longrightarrow f s t a=f s t b) \wedge$ $(\forall d G a b$. dickson-grading $d \longrightarrow i s$-sig-GB-upt $d G(l t$
b) $\longrightarrow$

$$
\begin{aligned}
a & \in G \longrightarrow b \in G \longrightarrow a \neq 0 \longrightarrow b \neq 0 \longrightarrow \text { lt } a \\
& \neg \text { is-sig-red }\left(\prec_{t}\right)(=) G b \longrightarrow \text { rword }(\text { spp-of } a)
\end{aligned}
$$

(spp-of b)))
definition is-canon-rewriter :: $\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow$ bool $) \Rightarrow$ ( $\left.{ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ set $\Rightarrow{ }^{\prime} t \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow$ bool
where is-canon-rewriter rword $A$ и $p \longleftrightarrow$
$\left(p \in A \wedge p \neq 0 \wedge l t p a d d s_{t} u \wedge\left(\forall a \in A . a \neq 0 \longrightarrow l t a \operatorname{adds} s_{t} u\right.\right.$ $\longrightarrow$ rword $($ spp-of $a)($ spp-of $p)))$
definition is-RB-in :: ('a nat) $\Rightarrow\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\right.$ bool $) \Rightarrow\left({ }^{\prime} t \Rightarrow{ }_{0}{ }^{\prime} b\right)$ set $\Rightarrow{ }^{\prime} t \Rightarrow$ bool
where $i s$ - RB-in d rword $G u \longleftrightarrow$
$\left(\left(\exists g\right.\right.$. is-canon-rewriter rword $G u g \wedge \neg i s$-sig-red $\left(\prec_{t}\right)(=) G$ (monom-mult $1(p p$-of-term $u-l p g) g)) \vee$

```
is-syz-sig d u)
```

definition is-RB-upt :: $\left({ }^{\prime} a \Rightarrow n a t\right) \Rightarrow\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\right.$ bool $) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ set $\Rightarrow{ }^{\prime} t \Rightarrow$ bool where is-RB-upt $d$ rword $G u \longleftrightarrow$
$\left(G \subseteq\right.$ dgrad-sig-set $d \wedge\left(\forall v . v \prec_{t} u \longrightarrow d(p p-o f\right.$-term $v) \leq d g r a d-m a x d$
$\qquad$ component-of-term $v<$ length $f s \longrightarrow i s-R B-i n d$ 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 $b a \Longrightarrow f s t a=f s t b$
and $\bigwedge d G a b$. dickson-grading $d \Longrightarrow$ is-sig-GB-upt $d G(l t b) \Longrightarrow a \in G \Longrightarrow$ $b \in G \Longrightarrow$

$$
a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow \text { lt } a \text { adds } s_{t} \text { lt } b \Longrightarrow \neg \text { is-sig-red }\left(\prec_{t}\right)(=) G b
$$

$\Longrightarrow$ rword (spp-of $a$ ) (spp-of b)
shows is-rewrite-ord rword
unfolding is-rewrite-ord-def using assms by blast
lemma is-rewrite-ordD1: is-rewrite-ord rword $\Longrightarrow$ rword a a by (simp add: is-rewrite-ord-def reflpD)
lemma is-rewrite-ordD2: is-rewrite-ord rword $\Longrightarrow$ rword $a b \Longrightarrow$ rword $b c \Longrightarrow$ rword a c
by (auto simp: is-rewrite-ord-def dest: transpD)
lemma is-rewrite-ordD3:
assumes is-rewrite-ord rword
and rword $a b \Longrightarrow$ thesis and $\neg$ rword $a b \Longrightarrow$ rword $b a \Longrightarrow$ thesis
shows thesis
proof -
from $\operatorname{assms}(1)$ have disj: rword $a b \vee$ rword $b a$ by (simp add: is-rewrite-ord-def del: split-paired-All)
show ?thesis
proof (cases rword a b)
case True
thus ?thesis by (rule assms(2))
next
case False
moreover from this disj have rword $b$ a by simp ultimately show ?thesis by (rule assms(3))
qed
qed
lemma is-rewrite-ordD4:
assumes is-rewrite-ord rword and rword $a b$ and rword $b a$
shows $f$ st $a=f s t b$
using assms unfolding is-rewrite-ord-def by blast
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 $l t a=l t b$
proof -
from assms have $f s t($ spp-of $a)=f s t(s p p-o f ~ b)$ by (rule is-rewrite-ordD4)
thus ?thesis by (simp add: spp-of-def)
qed
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 $l t a$ adds $s_{t} l t b$
and $\neg$ is-sig-red $\left(\prec_{t}\right)(=) G b$
shows rword (spp-of a) (spp-of b)
using assms unfolding is-rewrite-ord-def by blast
lemma is-canon-rewriterI:
assumes $p \in A$ and $p \neq 0$ and $l t p a d d s_{t} u$ and $\bigwedge a . a \in A \Longrightarrow a \neq 0 \Longrightarrow$ lt $a a d d s_{t} u \Longrightarrow$ rword (spp-of $a$ ) (spp-of $p$ )
shows is-canon-rewriter rword $A$ u $p$
unfolding is-canon-rewriter-def using assms by blast
lemma is-canon-rewriterD1: is-canon-rewriter rword $A u p \Longrightarrow p \in A$
by (simp add: is-canon-rewriter-def)
lemma is-canon-rewriterD2: is-canon-rewriter rword $A u p \Longrightarrow p \neq 0$
by (simp add: is-canon-rewriter-def)
lemma is-canon-rewriterD3: is-canon-rewriter rword $A u p \Longrightarrow l t p a d d s_{t} u$ by (simp add: is-canon-rewriter-def)
lemma is-canon-rewriterD4:
is-canon-rewriter rword $A$ up $\Longrightarrow a \in A \Longrightarrow a \neq 0 \Longrightarrow l t$ a addst $u \Longrightarrow$ rword
(spp-of a) (spp-of p)
by (simp add: is-canon-rewriter-def)
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 $l t a a d d s_{t}$ u
obtains $p$ where $i s$-canon-rewriter rword $A$ up
proof -
let $? A=\left\{x . x \in A \wedge x \neq 0 \wedge l t x\right.$ adds $\left.s_{t} u\right\}$
let ${ }^{\text {? rel }}=\lambda x y$.strict rword $($ spp-of $y)($ spp-of $x)$
have finite? A
proof (rule finite-subset)
show ? $A \subseteq A$ by blast
qed fact
moreover have $? A \neq\{ \}$
proof
from $\operatorname{assms}(3,4,5)$ have $a \in ? A$ by $\operatorname{simp}$
also assume ? $A=\{ \}$
finally show False by simp
qed
moreover have irreflp? rel
proof -
from assms(1) have reflp rword by (simp add: is-rewrite-ord-def)
thus ?thesis by (simp add: reflp-def irreflp-def)
qed
moreover have transp ? rel
proof -
from assms(1) have transp rword by (simp add: is-rewrite-ord-def)
thus ?thesis by (auto simp: transp-def simp del: split-paired-All)
qed
ultimately obtain $p$ where $p \in ? A$ and $*: \bigwedge b$. ?rel $b p \Longrightarrow b \notin ? A$ by (rule finite-minimalE, blast)
from this(1) have $p \in A$ and $p \neq 0$ and $l t p$ adds $s_{t} u$ by simp-all
show ?thesis
proof (rule, rule is-canon-rewriterI)
fix $q$
assume $q \in A$ and $q \neq 0$ and lt $q a d d s_{t} u$
hence $q \in$ ? A by simp
with $*$ have $\neg$ ? rel $q p$ by blast
hence disj: $\neg$ rword (spp-of p) (spp-of $q) \vee$ rword (spp-of $q$ ) (spp-of $p$ ) by simp
from $\operatorname{assms}(1)$ show rword (spp-of q) (spp-of p)
proof (rule is-rewrite-ordD3)
assume $\neg$ rword $(s p p-o f q)(s p p-o f p)$ and rword (spp-of $p)(s p p-o f q)$
with disj show ?thesis by simp
qed
qed fact+
qed
lemma is-rewrite-ord-canon-rewriterD1:
assumes is-rewrite-ord rword and is-canon-rewriter rword Aup and is-canon-rewriter
rword $A v q$
and $l t p a d d s_{t} v$ and $l t q a d d s_{t} u$
shows $l t p=l t q$
proof -
from $\operatorname{assms}(2)$ have $p \in A$ and $p \neq 0$
and 1: $\bigwedge a . a \in A \Longrightarrow a \neq 0 \Longrightarrow l t a a d d s_{t} u \Longrightarrow$ rword (spp-of $a$ ) (spp-of p)
by (rule is-canon-rewriterD) +
from $\operatorname{assms}(3)$ have $q \in A$ and $q \neq 0$
and 2: $\wedge a . a \in A \Longrightarrow a \neq 0 \Longrightarrow l t a a d d s_{t} v \Longrightarrow$ rword $(s p p-o f a)(s p p-o f q)$
by (rule is-canon-rewriterD)+
note assms (1)
moreover from $\langle p \in A\rangle\langle p \neq 0\rangle \operatorname{assms}(4)$ have rword (spp-of $p$ ) (spp-of $q$ ) by (rule 2)
moreover from $\langle q \in A\rangle\langle q \neq 0\rangle \operatorname{assms}(5)$ have rword (spp-of $q$ ) (spp-of $p$ ) by (rule 1)
ultimately show ?thesis by (rule is-rewrite-ordD4')
qed
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 $p=l t q$
using assms
proof (rule is-rewrite-ord-canon-rewriterD1)
from $\operatorname{assms(2)}$ show lt $p a d d s_{t} u$ by (rule is-canon-rewriterD)
next
from assms(3) show $l t q a d d s_{t} u$ by (rule is-canon-rewriterD)

## qed

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 $l t a a d d s_{t} u$ and is-sig-GB-upt $d A$ (lt a)
and lt paddst lt a and $\neg$ is-sig-red $\left(\prec_{t}\right)(=) A$ a
shows $l t p=l t a$
proof -
note assms (1)
moreover from $\operatorname{assms}(1,2,7)-\operatorname{assms}(4)-\operatorname{assms}(5,8,9)$ have rword (spp-of p) $(s p p-o f a)$
proof (rule is-rewrite-ordD5)
from $\operatorname{assms}(3)$ show $p \in A$ and $p \neq 0$ by (rule is-canon-rewriter $D)+$
qed
moreover from $\operatorname{assms}(3,4,5,6)$ have rword (spp-of a) (spp-of p) by (rule is-canon-rewriterD4)
ultimately show ?thesis by (rule is-rewrite-ordD4')
qed
lemma is-RB-inI1:
assumes is-canon-rewriter rword $G u g$ and $\neg$ is-sig-red $\left(\prec_{t}\right)(=) G$ (monom-mult 1 (pp-of-term $u-l p g) g$ ) shows is-RB-in d rword $G u$
unfolding is-RB-in-def using assms is-canon-rewriterD1 by blast
lemma is-RB-inI2:
assumes is-syz-sig du $u$
shows is-RB-in d rword $G u$
unfolding is-RB-in-def Let-def using assms by blast
lemma $i s-R B-i n E$ :
assumes is-RB-in d rword $G u$
and is-syz-sig $d u \Longrightarrow$ thesis
and $\bigwedge g$. $\neg i s$-syz-sig $d u \Longrightarrow i s$-canon-rewriter rword $G u g \Longrightarrow$
$\neg$ is-sig-red $\left(\prec_{t}\right)(=) G($ monom-mult $1(p p$-of-term $u-l p g) g) \Longrightarrow$
thesis
shows thesis
using assms unfolding is-RB-in-def by blast
lemma is-RB-inD:
assumes dickson-grading $d$ and $G \subseteq d g r a d$-sig-set $d$ and $i s$ - $R B$-in $d$ rword $G u$ and $\neg i s$-syz-sig $d u$ and $d(p p-o f$-term $u) \leq d g r a d$-max $d$
and is-canon-rewriter rword $G u g$
shows rep-list $g \neq 0$
proof
assume $a$ : rep-list $g=0$
from $\operatorname{assms}(1)$ have is-syz-sig d $u$
proof (rule is-syz-sig-adds)

```
    show is-syz-sig d (lt g)
    proof (rule is-syz-sigI)
        from assms(6) show g\not=0 by (rule is-canon-rewriterD2)
    next
        from assms(6) have g\inG by (rule is-canon-rewriterD1)
        thus g\indgrad-sig-set d using assms(2) ..
    qed (fact refl, fact a)
    next
    from assms(6) show lt g addst u by (rule is-canon-rewriterD3)
    qed fact
    with assms(4) show False ..
qed
lemma is-RB-uptI:
    assumes G\subseteqdgrad-sig-set d
    and }\v.v\mp@subsup{\prec}{t}{}u\Longrightarrowd(pp-of-term v)\leqdgrad-max d \Longrightarrow component-of-term
v<length fs \Longrightarrow
    is-RB-in d canon G v
    shows is-RB-upt d canon Gu
    unfolding is-RB-upt-def using assms by blast
lemma is-RB-uptD1:
    assumes is-RB-upt d canon Gu
    shows }G\subseteqdgrad-sig-set d 
    using assms unfolding is-RB-upt-def by blast
lemma is-RB-uptD2:
    assumes is-RB-upt d canon Gu and v}\mp@subsup{\prec}{t}{}u\mathrm{ and d (pp-of-term v) }\leqdgrad-max
d
    and component-of-term v<length fs
    shows is-RB-in d canon Gv
    using assms unfolding is-RB-upt-def by blast
lemma is-RB-in-UnI:
    assumes is-RB-in d rword Gu and }\bigwedgeh.h\inH\Longrightarrowu\mp@subsup{\prec}{t}{}lt
    shows is-RB-in d rword (H\cupG)u
    using assms(1)
proof (rule is-RB-inE)
    assume is-syz-sig d u
    thus is-RB-in d rword (H\cupG)u by (rule is-RB-inI2)
next
    fix g}\mp@subsup{g}{}{\prime
    assume crw: is-canon-rewriter rword Gu g'
        and irred: \neg is-sig-red ( }\mp@subsup{\prec}{t}{})(=)G(monom-mult 1 (pp-of-term u - lp g') g'
    from crw have g}\mp@subsup{g}{}{\prime}\inG\mathrm{ and }\mp@subsup{g}{}{\prime}\not=0\mathrm{ and lt g'addst u
        and max: \bigwedgea. }\\inG\Longrightarrowa\not=0\Longrightarrowlt a addst u\Longrightarrow rword (spp-of a)(spp-of
g')
    by (rule is-canon-rewriterD)+
    show is-RB-in d rword (H\cupG)u
```

```
proof (rule is-RB-inI1)
    show is-canon-rewriter rword (H\cupG) u g'
    proof (rule is-canon-rewriterI)
        from }\langle\mp@subsup{g}{}{\prime}\inG\rangle\mathrm{ show }\mp@subsup{g}{}{\prime}\inH\cupG\mathrm{ by simp
    next
        fix }
        assume }a\inH\cupG\mathrm{ and }a\not=0\mathrm{ and lt a addst u
        from this(1) show rword (spp-of a) (spp-of g')
        proof
            assume a }\in
            with <lt a addst u> have lt a addst u by simp
            hence lt a \preceq. 
            moreover from <a\inH\rangle have }u\mp@subsup{\prec}{t}{}lt a by (rule assms(2)
            ultimately show ?thesis by simp
        next
            assume a \inG
            thus ?thesis using <a\not=0\rangle\langlelt a addst u> by (rule max)
        qed
    qed fact+
next
    show }\neg\mathrm{ is-sig-red ( }\mp@subsup{\prec}{t}{})(=)(H\cupG)(monom-mult 1 (pp-of-term u - lp g') g'
        (is \negis-sig-red - - ?g)
    proof
        assume is-sig-red ( }\mp@subsup{\prec}{t}{})(=)(H\cupG) ?
        with irred have is-sig-red ( }\mp@subsup{\prec}{t}{})(=)H\mathrm{ ?g by (simp add: is-sig-red-Un del:
Un-insert-left)
            then obtain h}\mathrm{ where h}\inH\mathrm{ and is-sig-red (}\mp@subsup{\prec}{t}{})(=){h} ?g by (rule
is-sig-red-singletonI)
            from this(2) have lt h}\mp@subsup{\prec}{t}{}\mathrm{ lt ?g by (rule is-sig-red-regularD-lt)
            also from }\langle\mp@subsup{g}{}{\prime}\not=0\rangle\langlelt \mp@subsup{g}{}{\prime}add\mp@subsup{s}{t}{}u\rangle\mathrm{ have ...=u
            by (auto simp:lt-monom-mult adds-term-alt pp-of-term-splus)
        finally have lt h}\mp@subsup{\prec}{t}{}u\mathrm{ .
        moreover from <h\inH\rangle have }u\mp@subsup{\prec}{t}{}lth\mathrm{ by (rule assms(2))
        ultimately show False by simp
    qed
    qed
qed
corollary is-RB-in-insertI:
    assumes is-RB-in d rword Gu and }u\mp@subsup{\prec}{t}{}lt
    shows is-RB-in d rword (insert gG)u
proof -
    from assms(1) have is-RB-in d rword ({g}\cupG)u
    proof (rule is-RB-in-UnI)
            fix }
            assume h\in{g}
            with assms(2) show }u\mp@subsup{\prec}{t}{}\mathrm{ lt }h\mathrm{ by simp
qed
thus ?thesis by simp
```

corollary is-RB-upt-UnI:
assumes is-RB-upt d rword $G u$ and $H \subseteq d g r a d$-sig-set $d$ and $\bigwedge h . h \in H \Longrightarrow$ $u \preceq_{t}$ lt $h$
shows is-RB-upt d rword $(H \cup G) u$
proof (rule is-RB-uptI)
from $\operatorname{assms}(1)$ have $G \subseteq d g r a d$-sig-set $d$ by (rule is-RB-uptD1)
with assms(2) show $H \cup G \subseteq d g r a d$-sig-set $d$ by (rule Un-least)
next
fix $v$
assume $v \prec_{t} u$ and $d(p p$-of-term $v) \leq d g r a d-m a x d$ and component-of-term $v$
$<$ length fs
with $\operatorname{assms}(1)$ have $i s-R B$-in $d$ rword $G v$ by (rule is- $R B$-uptD2)
moreover from $\left\langle v \prec_{t} u\right\rangle \operatorname{assms}(3)$ have $\wedge h . h \in H \Longrightarrow v \prec_{t}$ lt $h$ by (rule ord-term-lin.less-le-trans)
ultimately show is-RB-in d rword $(H \cup G) v$ by (rule is-RB-in-UnI)
qed
corollary is-RB-upt-insertI:
assumes is-RB-upt d rword $G u$ and $g \in d g r a d$-sig-set $d$ and $u \preceq_{t} l t g$
shows is-RB-upt d rword (insert $g G$ ) u
proof -
from $\operatorname{assms}(1)$ have is-RB-upt $d$ rword $(\{g\} \cup G) u$
proof (rule is-RB-upt-UnI)
from $\operatorname{assms}(2)$ show $\{g\} \subseteq d g r a d-s i g-s e t ~ d ~ b y ~ s i m p ~$
next
fix $h$
assume $h \in\{g\}$
with assms(3) show $u \preceq_{t}$ lt $h$ by simp
qed
thus ?thesis by simp
qed
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 (rule ccontr)
let ? $Q=\left\{v . v \prec_{t} u \wedge d(p p\right.$-of-term $v) \leq$ dgrad-max $d \wedge$ component-of-term $v$
$<$ length $f s \wedge \neg$ is-sig-GB-in $d G v\}$
have $Q$-sub: pp-of-term '? $Q \subseteq$ dgrad-set $d$ (dgrad-max d) by blast
from $\operatorname{assms}$ (2) have $G$-sub: $G \subseteq d g r a d$-sig-set $d$ by (rule is-RB-uptD1)
hence $G \subseteq$ dgrad-max-set $d$ by (simp add: dgrad-sig-set'-def)
assume $\neg$ is-sig-GB-upt $d G u$
with $G$-sub obtain $v^{\prime}$ where $v^{\prime} \in$ ? $Q$ unfolding is-sig-GB-upt-def by blast
with $\operatorname{assms}(1)$ obtain $v$ where $v \in ? Q$ and min: $\bigwedge y . y \prec_{t} v \Longrightarrow y \notin ? Q$ using Q-sub
by (rule ord-term-minimum-dgrad-set, blast)
from $\langle v \in ? Q\rangle$ have $v \prec_{t} u$ and $d$ (pp-of-term $\left.v\right) \leq d g r a d-m a x d$ and compo-

```
nent-of-term v < length fs
    and \negis-sig-GB-in d Gv by simp-all
    from assms(2) this(1, 2, 3) have is-RB-in d rword G v by (rule is-RB-uptD2)
    from }\neg\mathrm{ is-sig-GB-in d G v` obtain r where lt r =v and r fdgrad-sig-set d
and }\neg\mathrm{ sig-red-zero ( }\mp@subsup{\preceq}{t}{})G
    unfolding is-sig-GB-in-def by blast
    from this(3) have rep-list r}\not=0\mathrm{ by (auto simp: sig-red-zero-def)
    hence }r\not=0\mathrm{ by (auto simp: rep-list-zero)
    hence lc r\not=0 by (rule lc-not-0)
    from G-sub have is-sig-GB-upt d G v
    proof (rule is-sig-GB-uptI)
    fix w
    assume dw:d (pp-of-term w) \leqdgrad-max d and cp:component-of-term w<
length fs
    assume w}\mp@subsup{\prec}{t}{}
    hence }w\not\in??Q by (rule min)
    hence }\negw\mp@subsup{\prec}{t}{}u\veeis-sig-GB-in d G w by (simp add:dw cp
    thus is-sig-GB-in d G w
    proof
        assume }\negw\mp@subsup{\prec}{t}{}
        moreover from }\langlew\mp@subsup{\prec}{t}{}v\rangle\langlev\mp@subsup{\prec}{t}{}u\rangle\mathrm{ have w < <t u}\mathrm{ by (rule ord-term-lin.less-trans)
        ultimately show ?thesis ..
        qed
qed
from <is-RB-in d rword G v` have sig-red-zero ( }\mp@subsup{\preceq}{t}{})G
proof (rule is-RB-inE)
    assume is-syz-sig d v
    have sig-red-zero ( }\mp@subsup{\prec}{t}{})Gr\mathrm{ by (rule syzygy-crit, fact+)
    thus ?thesis by (rule sig-red-zero-sing-regI)
next
    fix g
    assume a:\negis-sig-red ( }\mp@subsup{\prec}{t}{})(=)G(monom-mult 1 (pp-of-term v-lpg)g
    assume is-canon-rewriter rword G vg
    hence }g\inG\mathrm{ and }g\not=0\mathrm{ and lt gaddst v by (rule is-canon-rewriterD)+
    assume }\negis-syz-sig d 
    from }\langleg\inG`G\mathrm{ -sub have g}\indgrad-sig-set d ..
    from }\langleg\not=0\rangle\mathrm{ have lc g#0 by (rule lc-not-0)
    with «lc r\not=0\rangle have lc r / lc g\not=0 by simp
    from <lt g addst v> have lt g addst lt r by (simp only: <lt r = v>)
    hence eq1: (lp r - lp g) \opluslt g=lt r by (metis add-implies-diff adds-termE
pp-of-term-splus)
    let ?h = monom-mult (lc r / lc g) (lpr - lp g)g
    from <lc g\not=0\rangle\langlelcr\not=0\rangle\langleg\not=0\rangle have ?h\not=0 by (simp add: monom-mult-eq-zero-iff)
    have h-irred: \neg is-sig-red ( }\mp@subsup{\prec}{t}{})(=)G?
    proof
        assume is-sig-red ( }\mp@subsup{\prec}{t}{})(=)G?
```

moreover from 〈lc $g \neq 0\rangle\langle l c r \neq 0\rangle$ have $l c g / l c r \neq 0$ by simp
ultimately have is－sig－red $\left(\prec_{t}\right)(=) G$（monom－mult（lc g／lc r） 0 ？h）by （rule is－sig－red－monom－mult）
with $\langle l c g \neq 0\rangle\left\langle l c r \neq 0\right.$ 〉have is－sig－red $\left(\prec_{t}\right)(=) G$（monom－mult 1 （pp－of－term $v-l p g) g$ ）
by（simp add：monom－mult－assoc $\langle l t r=v\rangle$ ）
with $a$ show False ．．
qed
from $\langle l c r / l c g \neq 0\rangle\langle g \neq 0\rangle$ have $l t ? h=l t r$ by（simp add：lt－monom－mult eq1）
hence $l t ? h=v$ by（simp only：$\langle l t r=v\rangle$ ）
from $\langle l c g \neq 0\rangle$ have $l c ? h=l c r$ by simp
from $\operatorname{assms}(1)-\langle g \in d g r a d-s i g-s e t ~ d\rangle$ have $? h \in d g r a d$－sig－set $d$
proof（rule dgrad－sig－set－closed－monom－mult）
from $\left\langle l t ~ g a d d s_{t} l t r\right\rangle$ have $l p g$ adds lp r by（simp add：adds－term－def）
with assms（1）have $d(l p r-l p g) \leq d(l p r)$ by（rule dickson－grading－minus）
also from $\langle r \in d g r a d$－sig－set $d\rangle$ have $\ldots \leq d g r a d$－max $d$ by（rule dgrad－sig－setD－lp）
finally show $d(l p r-l p g) \leq d g r a d-m a x d$ ．
qed
have rep－list $? h \neq 0$
proof
assume rep－list $? h=0$
with $\langle ? h \neq 0\rangle\langle l t ? h=v\rangle\langle ? h \in d g r a d$－sig－set $d\rangle$ have $i s$－syz－sig $d v$ by（rule is－syz－sigI）
with $\langle\neg$ is－syz－sig $d v$ show False ．．
qed
hence rep－list $g \neq 0$ by（simp add：rep－list－monom－mult punit．monom－mult－eq－zero－iff）
hence punit．lc（rep－list g）$\neq 0$ by（rule punit．lc－not－0）
from $\operatorname{assms}(1)\langle G \subseteq$ dgrad－max－set $d\rangle$ obtain $s$ where s－red：$\left(\operatorname{sig}\right.$－red $\left(\prec_{t}\right)$
$(\preceq) G)^{* *} r s$
and s－irred：$\neg$ is－sig－red $\left(\prec_{t}\right)(\preceq) G s$ by（rule sig－irredE－dgrad－max－set）
from $s$－red have $s$－red ${ }^{\prime}:\left(\text { sig－red }\left(\preceq_{t}\right)(\preceq) G\right)^{* *} r s$ by（rule sig－red－rtrancl－sing－regI）
have rep－list $s \neq 0$
proof
assume rep－list $s=0$
with s－red ${ }^{\prime}$ have sig－red－zero $\left(\preceq_{t}\right) G r$ by（rule sig－red－zeroI）
with $\left\langle\neg\right.$ sig－red－zero $\left.\left(\preceq_{t}\right) G r\right\rangle$ show False ．．
qed
from $\operatorname{assms}(1) G$－sub $\langle r \in d g r a d-s i g-s e t ~ d\rangle s$－red have $s \in d g r a d$－sig－set $d$
by（rule dgrad－sig－set－closed－sig－red－rtrancl）
from $s$－red have $l t s=l t r$ and $l c s=l c r$
by（rule sig－red－regular－rtrancl－lt，rule sig－red－regular－rtrancl－lc）
hence $l t ? h=l t s$ and $l c ? h=l c s$ and $s \neq 0$
using $\langle l c \quad r \neq 0\rangle$ by（auto simp：〈lt ？$h=l t r\rangle\langle l c ? h=l c r\rangle \operatorname{simp}$ del： lc－monom－mult）
from $s$－irred have $\neg i s$－sig－red $\left(\prec_{t}\right)(=) G s$ by（simp add：is－sig－red－top－tail－cases）
from 〈is－sig－GB－upt $d G v\rangle$ have is－sig－GB－upt $d G$（lt s）by（simp only：＜lt $s$ $=l t r\rangle\langle l t r=v\rangle)$
have punit．lt（rep－list ？h）$=$ punit．lt（rep－list s）

```
        by (rule sig-regular-top-reduced-lt-unique, fact+)
    hence eq2:lp r - lp g + punit.lt (rep-list g) = punit.lt (rep-list s)
        using <lc r / lc g\not=0\rangle\langlerep-list g\not=0\rangle by (simp add: rep-list-monom-mult
punit.lt-monom-mult)
    have punit.lc (rep-list ?h) = punit.lc (rep-list s)
        by (rule sig-regular-top-reduced-lc-unique, fact+)
    hence eq3:lc r / lc g= punit.lc (rep-list s) / punit.lc (rep-list g)
    using <punit.lc (rep-list g)}\not=0\\mathrm{ by (simp add: rep-list-monom-mult field-simps)
    have sig-red-single (=) (=) s(s-?h) g (lp r - lp g)
        by (rule sig-red-singleI, auto simp: eq1 eq2 eq3 punit.lc-def[symmetric]<lt s
= lt r>
            <rep-list g\not=0\rangle\langlerep-list s\not=0\rangle intro!: punit.lt-in-keys)
        with }\langleg\inG\rangle\mathrm{ have sig-red (=)(=)Gs(s-?h) unfolding sig-red-def by
blast
    hence sig-red (\mp@subsup{\preceq}{t}{})(\preceq)Gs(s - ?h) by (auto dest: sig-red-sing-regI sig-red-top-tailI)
    with s-red' have r-red: (sig-red (\preceq}\mp@subsup{\}{}{\prime})(\preceq)G\mp@subsup{)}{}{**}r(s-?h) ..
    show ?thesis
    proof (cases s - ?h=0)
        case True
        hence rep-list (s - ?h) = 0 by (simp only: rep-list-zero)
        with r-red show ?thesis by (rule sig-red-zeroI)
    next
        case False
        note <is-sig-GB-upt d G (lt s)\rangle
        moreover from <s \indgrad-sig-set d\rangle\langle?h }\in\mathrm{ dgrad-sig-set d> have s-?h 
dgrad-sig-set d
            by (rule dgrad-sig-set-closed-minus)
            moreover from False<lt ?h = lt s\rangle\langlelc ?h = lc s> have lt (s-?h) \prec}\mp@subsup{\}{t}{lt s
by (rule lt-minus-lessI)
            ultimately have sig-red-zero ( }\mp@subsup{\preceq}{t}{})G(s-?h) by (rule is-sig-GB-uptD3)
            then obtain s' where (sig-red ( }\mp@subsup{\preceq}{t}{})(\preceq)G\mp@subsup{)}{}{**}(s-?h) s' and rep-list s'=
            by (rule sig-red-zeroE)
            from r-red this(1) have (sig-red (\mp@subsup{\preceq}{t}{})(\preceq)G)** r s' by simp
            thus ?thesis using <rep-list s'}=0\rangle\mathrm{ by (rule sig-red-zeroI)
        qed
    qed
    with «\neg sig-red-zero ( }\mp@subsup{\preceq}{t}{})\mathrm{ G r` show False ..
qed
corollary is-RB-upt-is-syz-sigD:
    assumes dickson-grading d and is-RB-upt d rword Gu
        and is-syz-sig d u and p\indgrad-sig-set d and lt p=u
    shows sig-red-zero ( }\mp@subsup{\prec}{t}{})\mathrm{ G p
proof -
    note assms(1)
    moreover from assms(1, 2) have is-sig-GB-upt d Gu by (rule is-RB-upt-is-sig-GB-upt)
    ultimately show ?thesis using assms(3, 4, 5) by (rule syzygy-crit)
qed
```


### 4.2.4 S-Pairs

```
definition spair :: \(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\)
    where spair p \(q=(\) let \(t 1=\) punit.lt (rep-list \(p) ;\) t2 \(=\) punit.lt (rep-list \(q) ; l=\)
lcs t1 t2 in
```

```
(monom-mult (1 / punit.lc (rep-list p)) (l - t1) p) -
```

(monom-mult (1 / punit.lc (rep-list p)) (l - t1) p) -
(monom-mult (1 / punit.lc (rep-list q)) (l - t2) q))
(monom-mult (1 / punit.lc (rep-list q)) (l - t2) q))
definition is-regular-spair :: ( $\left.' t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow$ bool where is-regular-spair $p \quad q \longleftrightarrow$ (rep-list $p \neq 0 \wedge$ rep-list $q \neq 0 \wedge$ (let t1 $=$ punit.lt $($ rep-list $p) ;$ t2 $=$ punit.lt $($ rep-list $q) ; l=$ lcs t1
t2 in

$$
(l-t 1) \oplus l t p \neq(l-t 2) \oplus l t q))
$$

lemma rep-list-spair: rep-list (spair p $q$ ) = punit.spoly (rep-list p) (rep-list $q$ ) by (simp add: spair-def punit.spoly-def Let-def rep-list-minus rep-list-monom-mult punit.lc-def)
lemma spair-comm: spair $p q=-\operatorname{spair} q p$
by (simp add: spair-def Let-def lcs-comm)
lemma dgrad-sig-set-closed-spair:
assumes dickson-grading $d$ and $p \in d g r a d-s i g-s e t ~ d ~ a n d ~ q \in d g r a d-s i g-s e t ~ d ~$
shows spair $p q \in d g r a d$-sig-set $d$
proof -
define $t 1$ where $t 1=$ punit.lt (rep-list p)
define t2 where $12=$ punit.lt $($ rep-list $q)$
let ?l = lcs t1 t2
have $d t 1 \leq$ dgrad-max $d$
proof (cases rep-list $p=0$ )
case True
show ?thesis by (simp add: t1-def True dgrad-max-0)
next
case False

```

```

with $\operatorname{assms}(1)$ have rep-list $p \in$ punit-dgrad-max-set $d$ by (rule dgrad-max-2)
thus ?thesis unfolding $t 1$-def using False by (rule punit.dgrad- $p$-setD-lp[simplified])
qed
moreover have $d$ t2 $\leq$ dgrad-max $d$
proof (cases rep-list $q=0$ )
case True
show ?thesis by (simp add: t2-def True dgrad-max-0)
next
case False

```

``` with \(\operatorname{assms}(1)\) have rep-list \(q \in\) punit-dgrad-max-set \(d\) by (rule dgrad-max-2)
thus ?thesis unfolding t2-def using False by (rule punit.dgrad-p-setD-lp[simplified])
qed
ultimately have ord-class.max \((d t 1)(d\) t2 \() \leq d g r a d-m a x d\) by \(\operatorname{simp}\)
```

moreover from $\operatorname{assms}(1)$ have $d ? l \leq$ ord-class.max $(d t 1)$ ( $d$ t2) by (rule dickson-grading-lcs)
ultimately have $*: d ? l \leq d g r a d-m a x d$ by auto
thm dickson-grading-minus
show ?thesis
proof (simp add: spair-def Let-def t1-def[symmetric] t2-def[symmetric], intro dgrad-sig-set-closed-minus dgrad-sig-set-closed-monom-mult[OF assms(1)])
from $\operatorname{assms}(1)$ adds-lcs have $d(? l-t 1) \leq d ? l$ by (rule dickson-grading-minus)
thus $d(? l-t 1) \leq d g r a d-m a x ~ d$ using $*$ by (rule le-trans)
next
from $\operatorname{assms}(1)$ adds-lcs-2 have $d(? l-t 2) \leq d ? l$ by (rule dickson-grading-minus)
thus $d(? l-t 2) \leq$ dgrad-max $d$ using $*$ by (rule le-trans)
qed fact+
qed
lemma $l t$-spair:
assumes rep-list $p \neq 0$ and punit.lt $($ rep-list $p) \oplus l t q \prec_{t}$ punit.lt $($ rep-list $q) \oplus$ lt $p$
shows lt $($ spair $p q)=(l c s($ punit.lt $($ rep-list $p))($ punit.lt $($ rep-list $q))-$ punit.lt $($ rep-list $p)) \oplus l t p$
proof -
define $l$ where $l=l$ cs $($ punit.lt (rep-list $p))($ punit.lt (rep-list $q))$
have 1: punit.lt (rep-list p) adds $l$ and 2: punit.lt (rep-list q) adds $l$
unfolding l-def by (rule adds-lcs, rule adds-lcs-2)
have eq1: spair $p q=($ monom-mult $(1 /$ punit.lc (rep-list $p))(l-p u n i t . l t$ $($ rep-list $p)) p)+$
(monom-mult (-1/punit.lc (rep-list q)) (l-punit.lt (rep-list
q)) $q$ )
by (simp add: spair-def Let-def l-def monom-mult-uminus-left)
from $\operatorname{assms}(1)$ have punit.lc (rep-list $p) \neq 0$ by (rule punit.lc-not-0)
hence 1 / punit.lc (rep-list $p$ ) $\neq 0$ by simp
moreover from assms(1) have $p \neq 0$ by (auto simp: rep-list-zero)
ultimately have eq2: lt (monom-mult (1 / punit.lc (rep-list p)) (l - punit.lt $($ rep-list $p)) p)=$

$$
(l-\text { punit.lt }(\text { rep-list } p)) \oplus l t p
$$

by (rule lt-monom-mult)
have lt (monom-mult (- $1 /$ punit.lc (rep-list q)) $(l-$ punit.lt $($ rep-list q)) q) $\preceq_{t}$

$$
(l-\text { punit.lt }(\text { rep-list } q)) \oplus l t q
$$

by (fact lt-monom-mult-le)
also from $\operatorname{assms}(2)$ have $\ldots \prec_{t}(l-$ punit.lt $($ rep-list $p)) \oplus$ lt $p$
by (simp add: term-is-le-rel-minus-minus[OF - 2 1])
finally show ?thesis unfolding eq2[symmetric] eq1 l-def[symmetric] by (rule lt-plus-eqI-2)
qed
lemma $l t$-spair ${ }^{\prime}$ :
assumes rep-list $p \neq 0$ and $a+$ punit.lt (rep-list $p$ ) $=b+$ punit.lt (rep-list $q$ ) and $b \oplus l t q \prec_{t} a \oplus l t p$

```
    shows lt (spair p q) =(a-gcs a b) \oplus lt p
proof -
    from assms(3) have punit.lt (rep-list p)\oplus(b\opluslt q) \prect punit.lt (rep-list p) }
(a\opluslt p)
    by (fact splus-mono-strict)
    hence (b + punit.lt (rep-list p))\opluslt q\mp@subsup{\prec}{t}{}(b+\mathrm{ punit.lt (rep-list q)) }\opluslt p
    by (simp only: splus-assoc[symmetric] add.commute assms(2))
    hence punit.lt (rep-list p)\opluslt q}\mp@subsup{\prec}{t}{}\mathrm{ punit.lt (rep-list q) }\oplus\mathrm{ lt p
    by (simp only: splus-assoc ord-term-strict-canc)
    with assms(1)
    have lt (spair p q) = (lcs (punit.lt (rep-list p)) (punit.lt (rep-list q)) - punit.lt
(rep-list p)) \opluslt p
    by (rule lt-spair)
    with assms(2) show ?thesis by (simp add: lcs-minus-1)
qed
lemma lt-rep-list-spair:
    assumes rep-list p\not=0 and rep-list q}\not=0\mathrm{ and rep-list (spair p q) }=
    and a + punit.lt (rep-list p)=b+ punit.lt (rep-list q)
    shows punit.lt (rep-list (spair p q)) \prec(a-gcs a b) + punit.lt (rep-list p)
proof -
    from assms(1) have 1: punit.lc (rep-list p) f=0 by (rule punit.lc-not-0)
    from assms(2) have 2: punit.lc (rep-list q)}\not=0\mathrm{ by (rule punit.lc-not-0)
    define l where l=lcs (punit.lt (rep-list p))(punit.lt (rep-list q))
    have eq: rep-list (spair p q) = (punit.monom-mult (1 / punit.lc (rep-list p)) (l
- punit.lt (rep-list p)) (rep-list p)) +
                            (punit.monom-mult (- 1 / punit.lc (rep-list q)) (l -
punit.lt (rep-list q)) (rep-list q))
            (is - = ?a + ?b)
    by (simp add: spair-def Let-def rep-list-minus rep-list-monom-mult punit.monom-mult-uminus-left
l-def)
    have ?a + ?b # = 0 unfolding eq[symmetric] by (fact assms(3))
    moreover from 1 2 assms(1, 2) have punit.lt ?b = punit.lt ?a
        by (simp add: lp-monom-mult l-def minus-plus adds-lcs adds-lcs-2)
    moreover have punit.lc ?b = - punit.lc ?a by (simp add: 1 2)
    ultimately have punit.lt (rep-list (spair p q)) \prec punit.lt ?a unfolding eq by
(rule punit.lt-plus-lessI)
    also from 1 assms(1) have ... = (l - punit.lt (rep-list p)) + punit.lt (rep-list p)
by (simp add: lp-monom-mult)
    also have ... =l by (simp add:l-def minus-plus adds-lcs)
    also have ... = (a+ punit.lt (rep-list p)) - gcs a b unfolding l-def using
assms(4) by (rule lcs-alt-1)
    also have ... = (a-gcs a b) + punit.lt (rep-list p) by (simp add: minus-plus
gcs-adds)
    finally show ?thesis.
qed
lemma is-regular-spair-sym: is-regular-spair p q \Longrightarrow is-regular-spair q p
    by (auto simp: is-regular-spair-def Let-def lcs-comm)
```

```
lemma is-regular-spairI:
    assumes rep-list p\not=0 and rep-list q}\not=
    and punit.lt (rep-list q)\opluslt p = punit.lt (rep-list p)\oplus lt q
    shows is-regular-spair p q
proof -
    have *: (lcs (punit.lt (rep-list p)) (punit.lt (rep-list q)) - punit.lt (rep-list p))\oplus
lt p\not=
                    (lcs (punit.lt (rep-list p)) (punit.lt (rep-list q)) - punit.lt (rep-list q)) }
lt q
            (is ?l l ? ?r)
    proof
            assume ?l = ?r
            hence punit.lt (rep-list q)\opluslt p = punit.lt (rep-list p)\oplus lt q
            by (simp add: term-is-le-rel-minus-minus adds-lcs adds-lcs-2)
            with assms(3) show False ..
    qed
    with assms(1, 2) show ?thesis by (simp add: is-regular-spair-def)
qed
lemma is-regular-spairI':
    assumes rep-list p\not=0 and rep-list q}\not=
    and a+ punit.lt (rep-list p)=b+ punit.lt (rep-list q) and a \opluslt p\not=b\opluslt q
    shows is-regular-spair p q
proof -
    have punit.lt (rep-list q) \oplus lt p}\not=\mathrm{ punit.lt (rep-list p) }\opluslt 
    proof
    assume punit.lt (rep-list q) \opluslt p= punit.lt (rep-list p) \opluslt q
    hence }a\oplus(\mathrm{ punit.lt (rep-list q)}\opluslt p)=a\oplus(punit.lt (rep-list p)\opluslt q) by
(simp only:)
    hence (a + punit.lt (rep-list q)) \opluslt p=(b + punit.lt (rep-list q)) \oplus lt q
            by (simp add: splus-assoc[symmetric] assms(3))
    hence punit.lt (rep-list q)\oplus(a\opluslt p)= punit.lt (rep-list q)}\oplus(b\opluslt q
            by (simp only: add.commute[of - punit.lt (rep-list q)] splus-assoc)
    hence }a\opluslt p=b\oplusltq\mathrm{ by (simp only: splus-left-canc)
    with assms(4) show False ..
    qed
    with assms(1, 2) show ?thesis by (rule is-regular-spairI)
qed
lemma is-regular-spairD1: is-regular-spair p q\Longrightarrow rep-list p\not=0
    by (simp add: is-regular-spair-def)
lemma is-regular-spairD2: is-regular-spair p q \Longrightarrow rep-list q}=
    by (simp add: is-regular-spair-def)
lemma is-regular-spairD3:
    fixes pq
    defines t1 \equiv punit.lt (rep-list p)
```

```
    defines t2 \(\equiv\) punit.lt (rep-list q)
    assumes is-regular-spair \(p q\)
    shows \(t 2 \oplus l t p \neq t 1 \oplus l t q\) (is ?thesis1)
    and \(l t\) (monom-mult \((1 /\) punit.lc \((\) rep-list \(p))(\) lcs t1 t2 - t1) \(p) \neq\)
        lt (monom-mult \((1 /\) punit.lc \((\) rep-list q) \()(l c s t 1\) t2 \(-t 2) q)(\) is \(? l \neq ? r)\)
proof -
    from \(\operatorname{assms}(3)\) have rep-list \(p \neq 0\) by (rule is-regular-spairD1)
    hence punit.lc (rep-list \(p) \neq 0\) and \(p \neq 0\) by (auto simp: rep-list-zero punit.lc-eq-zero-iff)
    from assms(3) have rep-list \(q \neq 0\) by (rule is-regular-spairD2)
    hence punit.lc (rep-list \(q\) ) \(=0\) and \(q \neq 0\) by (auto simp: rep-list-zero punit.lc-eq-zero-iff)
    have \(? l=(l\) cs \(t 1 t 2-t 1) \oplus l t p\)
    using \(\langle\) punit.lc (rep-list \(p) \neq 0\rangle\langle p \neq 0\rangle\) by (simp add: lt-monom-mult)
    also from \(\operatorname{assms}(3)\) have \(*: \ldots \neq(\) lcs t1 t2 - t2 \() \oplus\) lt \(q\)
    by ( simp add: is-regular-spair-def t1-def t2-def Let-def)
    also have (lcs t1 t2 - t2) \(\oplus l t q=\) ? \(r\)
    using \(\langle\) punit.lc (rep-list \(q) \neq 0\rangle\langle q \neq 0\rangle\) by (simp add: lt-monom-mult)
    finally show ?l \(\neq ?\) ? .
    show ?thesis1
    proof
    assume \(t 2 \oplus l t p=t 1 \oplus l t q\)
    hence (lcs t1 t2 - t1) \(\oplus l t p=(l c s t 1 t 2-t 2) \oplus l t q\)
            by (simp add: term-is-le-rel-minus-minus adds-lcs adds-lcs-2)
    with \(*\) show False ..
    qed
qed
lemma is-regular-spair-nonzero: is-regular-spair p \(q \Longrightarrow\) spair \(p q \neq 0\)
    by (auto simp: spair-def Let-def dest: is-regular-spairD3)
lemma is-regular-spair-lt:
    assumes is-regular-spair \(p q\)
    shows lt (spair \(p q\) ) \(=\) ord-term-lin.max
            \(((l c s(\) punit.lt \((\) rep-list \(p))(\) punit.lt \((\) rep-list \(q))-\) punit.lt (rep-list p) \()\)
\(\oplus\) lt p)
            \(((l c s(\) punit.lt \((\) rep-list \(p))(\) punit.lt \((\) rep-list \(q))-p u n i t . l t(r e p-l i s t ~ q)) ~\)
\(\oplus l t q)\)
proof -
    let \(?\) t1 \(=\) punit.lt \((\) rep-list \(p)\)
    let \(? t 2=\) punit.lt \((\) rep-list \(q)\)
    let ?l = lcs ? t1 ? t2
    show ?thesis
    proof (rule ord-term-lin.linorder-cases)
    assume \(a\) : ? t2 \(\oplus l t p \prec_{t}\) ? \(t 1 \oplus\) lt \(q\)
    hence \((? l-? t 1) \oplus l t p \prec_{t}(? l-? t 2) \oplus l t q\)
        by (simp add: term-is-le-rel-minus-minus adds-lcs adds-lcs-2)
    hence \(l e:(? l-? t 1) \oplus l t p \preceq_{t}(? l-? t 2) \oplus l t q\) by (rule ord-term-lin.less-imp-le)
    from assms have rep-list \(q \neq 0\) by (rule is-regular-spairD2)
```

```
    have lt (spair p q) = lt (spair q p) by (simp add: spair-comm[of p])
    also from〈rep-list q}\not=0\mathrm{ \ a have ... = (lcs ?t2 ?t1 - ?t2) }\opluslt q by (rule
lt-spair)
    also have ... = (?l - ?t2) }\opluslt q by (simp only:lcs-comm)
    finally show ?thesis using le by (simp add: ord-term-lin.max-def)
    next
    assume a: ?t1 \opluslt q}\mp@subsup{\prec}{t}{}\mathrm{ ?t2 }\oplus\mathrm{ lt p
    hence (?l - ?t2) \opluslt q \prec-t (?l - ?t1) }\oplus\mathrm{ lt p
            by (simp add: term-is-le-rel-minus-minus adds-lcs adds-lcs-2)
    hence le:\neg ((?l - ?t1) }\opluslt p\mp@subsup{\preceq}{t}{}(?l - ?t2) \oplus lt q) by sim
    from assms have rep-list p\not=0 by (rule is-regular-spairD1)
    hence lt (spair p q) = (lcs ?t1 ?t2 - ?t1) \oplus lt p using a by (rule lt-spair)
    thus?thesis using le by (simp add: ord-term-lin.max-def)
    next
    from assms have ?t2 \opluslt p\not=?t1 \opluslt q by (rule is-regular-spairD3)
    moreover assume ?t2 \oplus lt p = ?t1 \oplus lt q
    ultimately show ?thesis ..
    qed
qed
lemma is-regular-spair-lt-ge-1:
    assumes is-regular-spair p q
    shows lt p \preceq_ lt (spair p q)
proof -
    have lt p=0 \oplus lt p by (simp only: term-simps)
    also from zero-min have ... \preceq́t (lcs (punit.lt (rep-list p)) (punit.lt (rep-list q))
- punit.lt (rep-list p)) \opluslt p
    by (rule splus-mono-left)
    also have ... }\mp@subsup{\preceq}{t}{}\mathrm{ ord-term-lin.max
                            ((lcs (punit.lt (rep-list p)) (punit.lt (rep-list q)) - punit.lt (rep-list p))
\oplus lt p)
                            ((lcs (punit.lt (rep-list p)) (punit.lt (rep-list q)) - punit.lt (rep-list q))
\oplus lt q)
    by (rule ord-term-lin.max.cobounded1)
    also from assms have ... =lt (spair p q) by (simp only: is-regular-spair-lt)
    finally show ?thesis.
qed
corollary is-regular-spair-lt-ge-2:
    assumes is-regular-spair p q
    shows lt q \preceq́t lt (spair p q)
proof -
    from assms have is-regular-spair q p by (rule is-regular-spair-sym)
    hence lt q \preceq_t lt (spair q p) by (rule is-regular-spair-lt-ge-1)
    also have ... = lt (spair p q) by (simp add: spair-comm[of q])
    finally show ?thesis.
qed
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)
proof (rule ord-term-lin.linorder-cases)
    from assms have rep-list q}=0\mathrm{ by (rule is-regular-spairD2)
    moreover assume punit.lt (rep-list q)\oplus lt p < punit.lt (rep-list p) \oplus lt q
    ultimately have lt (spair q p)=(lcs (punit.lt (rep-list q)) (punit.lt (rep-list p))
- punit.lt (rep-list q)) \opluslt q
        by (rule lt-spair)
    thus ?thesis by (simp add: spair-comm[of p] term-simps)
next
    from assms have rep-list p\not=0 by (rule is-regular-spairD1)
    moreover assume punit.lt (rep-list p)\opluslt q \prec t punit.lt (rep-list q) \oplus lt p
    ultimately have lt (spair p q) = (lcs (punit.lt (rep-list p)) (punit.lt (rep-list q))
- punit.lt (rep-list p)) \opluslt p
    by (rule lt-spair)
    thus ?thesis by (simp add: term-simps)
next
    from assms have punit.lt (rep-list q) \oplus lt p\not= punit.lt (rep-list p) }\opluslt 
        by (rule is-regular-spairD3)
    moreover assume punit.lt (rep-list q) \oplus lt p = punit.lt (rep-list p)\oplus lt q
    ultimately show ?thesis ..
qed
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 \negis-syz-sig d u and is-canon-rewriter rword Gug1 and
h\inG
        and is-sig-red (}\mp@subsup{\prec}{t}{})(=){h}(monom-mult 1(pp-of-term u - lp g1) g1)
        and d (pp-of-term u)\leqdgrad-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)
proof -
    from assms(8) have rep-list (monom-mult 1 (pp-of-term u - lp g1) g1)}\not=
        using is-sig-red-top-addsE by fastforce
    hence rep-list g1 \not=0 by (simp add: rep-list-monom-mult punit.monom-mult-eq-zero-iff)
    hence g1 =0 by (auto simp: rep-list-zero)
    from assms(6) have g1 \inG and lt g1 addst u by (rule is-canon-rewriterD)+
    from assms(3) have G\subseteqdgrad-sig-set d by (rule is-RB-uptD1)
    with }\langleg1\inG\rangle\mathrm{ have g1 Єdgrad-sig-set d ..
    hence component-of-term (lt g1) < length fs using <g1 =0\rangle by (rule dgrad-sig-setD-lt)
        with <lt g1 addst u〉 have component-of-term u < length fs by (simp add:
adds-term-def)
```

from 〈lt g1 addst $u\rangle$ obtain $a$ where $u: u=a \oplus l t g 1$ by（rule adds－termE）
hence $a$ ：$a=p p$－of－term $u-l p$ g1 by（simp add：term－simps）
from $\operatorname{assms}(8)$ have is－sig－red $\left(\prec_{t}\right)(=)\{h\}$（monom－mult 1 a g1）by（simp only：a）
hence rep－list $h \neq 0$ and rep－list（monom－mult 1 a g1）$\neq 0$ and
2：punit．lt（rep－list h）adds punit．lt（rep－list（monom－mult 1 a g1））and
3：punit．lt（rep－list（monom－mult 1 a g1））$\oplus l t h \prec_{t}$ punit．lt $($ rep－list $h) \oplus l t$
（monom－mult 1 a g1）
by（auto elim：is－sig－red－top－addsE）
from this（2）have rep－list $g 1 \neq 0$ by（simp add：rep－list－monom－mult punit．monom－mult－eq－zero－iff）
hence g1 $\neq 0$ by（auto simp：rep－list－zero）
from 〈rep－list $h \neq 0\rangle$ have $h \neq 0$ by（auto simp：rep－list－zero）
from $2\langle$ rep－list $g 1 \neq 0$ 〉 have punit．lt（rep－list h）adds a + punit．lt（rep－list g1）
by（simp add：rep－list－monom－mult punit．lt－monom－mult）
then obtain $b$ where eq1：$a+$ punit．lt（rep－list g1）$=b+$ punit．lt（rep－list $h$ ）
by（auto elim：addsE simp：add．commute）
hence $b: b=(($ pp－of－term $u-l p g 1)+$ punit．lt $($ rep－list g1 $))-$ punit．lt（rep－list h）
by（ $\operatorname{simp}$ add：a）
define $g$ where $g=g c s a b$
have $g=0$
proof（rule ccontr）
assume $g \neq 0$
have $g$ adds a unfolding $g$－def by（fact gcs－adds）
also have ．．．adds $s_{p} u$ unfolding $u$ by（fact adds－pp－triv）
finally obtain $v$ where $u 2: u=g \oplus v$ by（rule adds－ppE）
hence $v: v=u \ominus g$ by（simp add：term－simps）
from $u 2$ have $v a d d s_{t} u$ by（rule adds－termI）
hence $v \preceq_{t} u$ by（rule ord－adds－term）
moreover have $v \neq u$
proof
assume $v=u$
hence $g \oplus v=0 \oplus v$ by（simp add：u2 term－simps）
hence $g=0$ by（simp only：splus－right－canc）
with $\langle g \neq 0\rangle$ show False ．．
qed
ultimately have $v \prec_{t} u$ by $\operatorname{simp}$
note $\operatorname{assms}(3)\left\langle v \prec_{t} u\right\rangle$
moreover have $d$（pp－of－term $v) \leq d g r a d-m a x d$
proof（rule le－trans）
from $\operatorname{assms}(1)$ show $d(p p$－of－term $v) \leq d(p p$－of－term $u)$
by（simp add：u2 term－simps dickson－gradingD1）
qed fact
moreover from＜component－of－term $u<$ length $f_{s}$ 〉 have component－of－term $v$
$<$ length fs
by（simp only：v term－simps）
ultimately have $i s-R B$－in d rword $G v$ by（rule is－RB－uptD2）

## thus False

proof（rule is－RB－inE）
assume $i s$－syz－sig $d v$
with $\operatorname{assms}(1)$ have $i s$－syz－sig $d u$ using $\left\langle v a d d s_{t} u\right\rangle \operatorname{assms}(9)$ by（rule is－syz－sig－adds）
with assms（5）show False ．．

## next

fix $g 2$
assume $*: \neg$ is－sig－red $\left(\prec_{t}\right)(=) G$（monom－mult 1 （pp－of－term $\left.v-l p g 2\right)$ g2）
assume is－canon－rewriter rword $G v g 2$
hence $g 2 \in G$ and $g 2 \neq 0$ and $l t g 2 a d d s_{t} v$ by（rule is－canon－rewriterD）＋
assume $\neg i s-s y z-s i g d v$
note $\operatorname{assms}(2)\langle i s$－canon－rewriter rword $G v g 2\rangle \operatorname{assms}(6)$
moreover from＜lt g2 addst $v\rangle\left\langle v a d d s_{t} u\right\rangle$ have $l t ~ g 2$ addst $u$ by（rule adds－term－trans）
moreover from $\left\langle g\right.$ adds a〉 have lt $g 1$ adds $s_{t} v$ by（simp add：v u mi－ nus－splus［symmetric］adds－termI）
ultimately have lt $g 2=$ lt $g 1$ by（rule is－rewrite－ord－canon－rewriterD1）
with $\operatorname{assms}(4)$ have $g 2=g 1$ using $\langle g 2 \in G\rangle\langle g 1 \in G\rangle$ by（rule inj－onD）
have pp－of－term $v-l p g 1=a-g$ by（simp add：uv term－simps diff－diff－add）
have is－sig－red $\left(\prec_{t}\right)(=) G$（monom－mult 1 （pp－of－term $\left.\left.v-l p g 2\right) g 2\right)$ unfolding $\langle g 2=g 1\rangle\langle p p$－of－term $v-l p g 1=a-g\rangle \operatorname{using} \operatorname{assms}(7)$ $\langle$ rep－list $h \neq 0\rangle$
proof（rule is－sig－red－top－addsI）
from $\langle$ rep－list $g 1 \neq 0\rangle$ show rep－list（monom－mult $1(a-g) g 1) \neq 0$ by（simp add：rep－list－monom－mult punit．monom－mult－eq－zero－iff）
next
have eq3：$(a-g)+$ punit．lt（rep－list g1）$=$ lcs（punit．lt（rep－list g1）） （punit．lt（rep－list h）） by（simp add：g－def lcs－minus－1［OF eq1，symmetric］adds－minus adds－lcs） from 〈rep－list g1 $\neq 0$ 〉
show punit．lt（rep－list h）adds punit．lt（rep－list（monom－mult $1(a-g)$ g1）） by（simp add：rep－list－monom－mult punit．lt－monom－mult eq3 adds－lcs－2）
next
from $3\langle$ rep－list $g 1 \neq 0\rangle\langle g 1 \neq 0\rangle$
show punit．lt（rep－list（monom－mult $1(a-g)$ g1））$\oplus l t h \prec_{t}$ punit．lt（rep－list h）$\oplus l t$（monom－mult $1(a-g)$ g1）
by（auto simp：rep－list－monom－mult punit．lt－monom－mult lt－monom－mult splus－assoc splus－left－commute
dest！：ord－term－strict－canc intro：splus－mono－strict）
next
show ord－term－lin．is－le－rel $\left(\prec_{t}\right)$ by（fact ord－term－lin．is－le－relI）
qed
with $*$ show False ．．
qed
qed
thus ？thesis1 and ？thesis2 by（simp－all add：a blcs－minus－1［OF eq1］lcs－minus－2［OF

```
eq1] g-def)
    hence eq3: spair g1 h = monom-mult (1 / punit.lc (rep-list g1)) a g1 -
                                    monom-mult (1 / punit.lc (rep-list h)) b h
    by (simp add: spair-def Let-def a b)
    from 3 <rep-list g1 \not= 0\rangle\langleg1 f 0\rangle have b \oplus lt h \prec }\mp@subsup{t}{}{\prime}a\opluslt g
    by (auto simp: rep-list-monom-mult punit.lt-monom-mult lt-monom-mult eq1
splus-assoc
            splus-left-commute[of b] dest!: ord-term-strict-canc)
    hence a @ lt g1 \not=b\opluslt h by simp
    with <rep-list g1 \not= 0〉\langlerep-list h\not= 0\rangle eq1 show ?thesis3
    by (rule is-regular-spairI')
    have lt (monom-mult (1 / punit.lc (rep-list h)) b h)=b\opluslt h
    proof (rule lt-monom-mult)
            from <rep-list h\not=0\rangle show 1 / punit.lc (rep-list h) \not=0 by (simp add:
punit.lc-eq-zero-iff)
    qed fact
    also have ... \prec}\mp@subsup{\prec}{}{a}a\opluslt g1 by fac
    also have ... = lt (monom-mult (1 / punit.lc (rep-list g1)) a g1)
    proof (rule HOL.sym, rule lt-monom-mult)
        from <rep-list g1 \not= 0〉 show 1 / punit.lc (rep-list g1) \not= 0 by (simp add:
punit.lc-eq-zero-iff)
    qed fact
    finally have lt (spair g1 h)=lt (monom-mult (1 / punit.lc (rep-list g1)) a g1)
    unfolding eq3 by (rule lt-minus-eqI-2)
    also have ... = a \opluslt g1 by (rule HOL.sym, fact)
    finally show ?thesis4 by (simp only:u)
qed
lemma is-RB-upt-finite:
    assumes dickson-grading d and is-rewrite-ord rword and G\subseteqdgrad-sig-set d
and inj-on lt G
    and finite G
    and \g1 g2.g1 \inG\Longrightarrowg2 
g2) }\mp@subsup{\prec}{t}{}u
                        is-RB-in d rword G (lt (spair g1 g2))
        and }\i.i< length fs \Longrightarrowterm-of-pair (0,i)\mp@subsup{\prec}{t}{}u\Longrightarrow is-RB-in d rword G
(term-of-pair (0,i))
    shows is-RB-upt d rword Gu
proof (rule ccontr)
    let ?Q = {v.v \prec < u ^d (pp-of-term v) \leqdgrad-max d ^ component-of-term v
< ~ l e n g t h ~ f s ~ \wedge \neg ~ i s - R B - i n ~ d ~ r w o r d ~ G v \}
    have Q-sub:pp-of-term' ?Q \subseteq dgrad-set d (dgrad-max d) by blast
    from assms(3) have G\subseteqdgrad-max-set d by (simp add: dgrad-sig-set'-def)
    assume }\neg\mathrm{ is-RB-upt d rword Gu
    with assms(3) obtain v}\mp@subsup{v}{}{\prime}\mathrm{ where }\mp@subsup{v}{}{\prime}\in?Q\mathrm{ unfolding is-RB-upt-def by blast
    with assms(1) obtain v}\mathrm{ where v}v\in?Q and min: \y. y\mp@subsup{\prec}{t}{}v\Longrightarrowy\not\in?Q usin
Q-sub
```

```
    by (rule ord-term-minimum-dgrad-set, blast)
    from }\langlev\in?Q\rangle\mathrm{ have }v\mp@subsup{\prec}{t}{}u\mathrm{ and d (pp-of-term v)}\leqdgrad-max d and compo
nent-of-term v<length fs
    and }\neg\mathrm{ is-RB-in d rword Gv by simp-all
    from this(4)
    have impl: \bigwedgeg.g\inG\Longrightarrow is-canon-rewriter rword G v g\Longrightarrow
                    is-sig-red (}\mp@subsup{\prec}{t}{})(=)G(monom-mult 1 (pp-of-term v-lpg)g
    and \negis-syz-sig d v by (simp-all add:is-RB-in-def Let-def)
    from assms(3) have is-RB-upt d rword G v
    proof (rule is-RB-uptI)
    fix w
    assume dw:d (pp-of-term w) \leqdgrad-max d and cp:component-of-term w<
length fs
    assume w}\mp@subsup{\prec}{t}{}
    hence w}\not\in?Q by (rule min)
    hence }\negw\mp@subsup{\prec}{t}{}u\veeis-RB-in d rword G w by (simp add: dw cp
    thus is-RB-in d rword Gw
    proof
        assume }\negw\mp@subsup{\prec}{t}{}
    moreover from }\langlew\mp@subsup{\prec}{t}{}v\rangle\langlev\mp@subsup{\prec}{t}{}u\rangle\mathrm{ have }w\mp@subsup{\prec}{t}{}u\mathrm{ by (rule ord-term-lin.less-trans)
        ultimately show ?thesis ..
    qed
qed
show False
proof (cases \existsg\inG.g\not=0\wedgelt gaddst v)
    case False
    hence }x:\g.g\inG\Longrightarrowlt gaddstv\Longrightarrowg=0 by blas
    let ?w = term-of-pair (0, component-of-term v)
    have ?w addst v by (simp add: adds-term-def term-simps)
    hence ? w}\mp@subsup{\preceq}{t}{}v\mathrm{ by (rule ord-adds-term)
    also have ... }\mp@subsup{\prec}{t}{}u\mathrm{ by fact
    finally have ?w }\mp@subsup{\prec}{t}{}u\mathrm{ .
    with «component-of-term v< length fs` have is-RB-in d rword G ?w by (rule
assms(7))
    thus ?thesis
    proof (rule is-RB-inE)
        assume is-syz-sig d ?w
        with assms(1) have is-syz-sig d v using <?w addstst v〉\langled (pp-of-term v)\leq
dgrad-max d>
            by (rule is-syz-sig-adds)
        with «\neg is-syz-sig d v` show ?thesis ..
    next
        fix g1
        assume is-canon-rewriter rword G ?w g1
    hence g1\not=0 and g1\inG and lt g1 addst ?w by (rule is-canon-rewriterD)+
    from this(3) have lt g1 addst v using <?w addst v> by (rule adds-term-trans)
    with }\langleg1\inG\rangle\mathrm{ have g1 = 0 by (rule }x\mathrm{ )
```

with $\langle g 1 \neq 0\rangle$ show ?thesis ..
qed
next
case True
then obtain $g^{\prime}$ where $g^{\prime} \in G$ and $g^{\prime} \neq 0$ and $l t g^{\prime} a d d s_{t} v$ by blast
with $\operatorname{assms}(2,5)$ obtain $g 1$ where crw: is-canon-rewriter rword $G v g 1$ by (rule is-rewrite-ord-finite-canon-rewriterE)
hence $g 1 \in G$ by (rule is-canon-rewriterD1)
hence is-sig-red $\left(\prec_{t}\right)(=) G$ (monom-mult 1 (pp-of-term $v$-lpg1) g1) using crw by (rule impl)
then obtain $h$ where $h \in G$ and is-sig-red $\left(\prec_{t}\right)(=)\{h\}$ (monom-mult 1
(pp-of-term $v-l p g 1) ~ g 1)$
by (rule is-sig-red-singletonI)
with $\operatorname{assms}(1,2)\langle i s-R B$-upt $d$ rword $G v\rangle \operatorname{assms}(4)\langle\neg i s$-syz-sig d $v\rangle c r w$
have is-regular-spair g1 $h$ and eq: lt (spair g1 $h$ ) $=v$
using $\langle d(p p$-of-term $v) \leq$ dgrad-max $d\rangle$ by (rule lemma-9) +
from $\left\langle v \prec_{t} u\right\rangle$ have lt (spair g1 $h$ ) $\prec_{t} u$ by (simp only: eq)
with $\langle g 1 \in G\rangle\langle h \in G\rangle\langle i s$-regular-spair $g 1 h\rangle$ have is-RB-in d rword $G$ (lt (spair g1 h))
by (rule assms(6))
hence is-RB-in d rword $G v$ by (simp only: eq)
with $\langle\neg i s-R B$-in d rword $G$ v〉 show ?thesis ..
qed
qed
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 $p \in d g r a d$-sig-set $d$ and is-sig-red $\left(\prec_{t}\right)(=) G p$
obtains $u g$ where $u \prec_{t} l t p$ and $d$ (pp-of-term $\left.u\right) \leq d g r a d$-max $d$ and compo-nent-of-term $u<$ length $f s$
and $\neg$ is-syz-sig $d u$ and is-canon-rewriter rword $G u g$
and $u=($ punit.lt $($ rep-list $p)-$ punit.lt $($ rep-list $g)) \oplus l t g$ and is-sig-red $\left(\prec_{t}\right)$
(=) $\{g\} p$
proof -
from $\operatorname{assms}(3)$ have $G$-sub: $G \subseteq d g r a d$-sig-set $d$ by (rule is-RB-uptD1)
from $\operatorname{assms}(5)$ have rep-list $p \neq 0$ using is-sig-red-addsE by fastforce
hence $p \neq 0$ by (auto simp: rep-list-zero)
let ?lc $=$ punit.lc $($ rep-list $p)$
let ?lp $=$ punit.lt $($ rep-list $p)$
from 〈rep-list $p \neq 0\rangle$ have ?lc $\neq 0$ by (rule punit.lc-not-0)

from $\operatorname{assms}(4)$ have $d(l p p) \leq d g r a d-m a x ~ d$ by (rule dgrad-sig-setD-lp)
from $\operatorname{assms}(4)\langle p \neq 0\rangle$ have component-of-term (lt $p$ ) $<$ length $f s$ by (rule dgrad-sig-setD-lt)
from $\operatorname{assms}(1)\langle p \in$ dgrad-max-set $d\rangle$ have rep-list $p \in$ punit-dgrad-max-set d by (rule dgrad-max-2)
hence $d ? l p \leq d g r a d-m a x d$ using $\langle r e p-l i s t ~ p \neq 0\rangle$ by (rule punit.dgrad-p-setD-lp[simplified])
from $\operatorname{assms}(5)$ obtain $g 0$ where $g 0 \in G$ and is-sig-red $\left(\prec_{t}\right)(=)\{g 0\} p$
by (rule is-sig-red-singletonI)
from $\langle g 0 \in G\rangle G$-sub have $g 0 \in$ dgrad-sig-set $d$..
let ?g0 $=$ monom-mult $(? l c /$ punit.lc $($ rep-list g0) $)(? l p-$ punit.lt $($ rep-list g0) $)$ g0
define $M$ where $M=\{$ monom-mult (?lc / punit.lc (rep-list g)) (?lp - punit.lt $($ rep-list $g$ )) $g \mid$
g. $g \in$ dgrad-sig-set $d \wedge$ is-sig-red $\left.\left(\prec_{t}\right)(=)\{g\} p\right\}$
from $\langle g 0 \in d g r a d$-sig-set $d\rangle\left\langle\right.$ is-sig-red $\left.\left(\prec_{t}\right)(=)\{g 0\} p\right\rangle$ have ? $g 0 \in M$ by (auto simp: $M$-def)
have $0 \notin$ rep-list ' $M$
proof
assume $0 \in$ rep-list ' $M$
then obtain $g$ where 1 : is-sig-red $\left(\prec_{t}\right)(=)\{g\} p$
and 2: rep-list (monom-mult (?lc / punit.lc (rep-list g)) (?lp - punit.lt (rep-list
g)) $g)=0$
unfolding $M$-def by fastforce
from 1 have rep-list $g \neq 0$ using is-sig-red-addsE by fastforce
moreover from this have punit.lc (rep-list $g$ ) $\neq 0$ by (rule punit.lc-not-0)
ultimately have rep-list (monom-mult (?lc / punit.lc (rep-list g)) (?lp punit.lt (rep-list g)) g) $\neq 0$
using $\langle ? l c \neq 0\rangle$ by (simp add: rep-list-monom-mult punit.monom-mult-eq-zero-iff)
thus False using 2 ..
qed
with rep-list-zero have $0 \notin M$ by auto
have $M \subseteq d g r a d$-sig-set $d$
proof
fix $m$
assume $m \in M$
then obtain $g$ where $g \in$ dgrad-sig-set $d$ and 1: is-sig-red $\left(\prec_{t}\right)(=)\{g\} p$ and $m: m=$ monom-mult (?lc / punit.lc (rep-list g)) (?lp - punit.lt (rep-list g)) $g$ unfolding $M$-def by fastforce
from 1 have punit.lt (rep-list g) adds ?lp using is-sig-red-top-addsE by fastforce
note $\operatorname{assms}(1)$
thm dickson-grading-minus
moreover have $d(? l p-$ punit.lt $($ rep-list $g)) \leq$ dgrad-max $d$
by (rule le-trans, rule dickson-grading-minus, fact+)
ultimately show $m \in$ dgrad-sig-set $d$ unfolding $m$ using $\langle g \in$ dgrad-sig-set d>
by (rule dgrad-sig-set-closed-monom-mult)
qed
hence $M \subseteq$ sig-inv-set by (simp add: dgrad-sig-set'-def)
let $? M=l t$ ‘ $M$
note $\operatorname{assms}(1)$
moreover from $\langle ? g 0 \in M\rangle$ have $l t ? g 0 \in ? M$ by (rule imageI)
moreover from $\langle M \subseteq d g r a d$-sig-set $d\rangle$ have pp-of-term' ? $M \subseteq$ dgrad-set $d$ (dgrad-max d)
by (auto intro!: dgrad-sig-setD-lp)
ultimately obtain $u$ where $u \in ? M$ and $\min : \Lambda v . v \prec_{t} u \Longrightarrow v \notin ? M$
by (rule ord-term-minimum-dgrad-set, blast)
from $\langle u \in ? M\rangle$ obtain $m$ where $m \in M$ and $u^{\prime}: u=l t m$..
from this(1) obtain g1 where g1 $\in d g r a d$-sig-set d and 1: is-sig-red $\left(\prec_{t}\right)(=)$ $\{g 1\} p$
and $m$ : $m=$ monom-mult (?lc / punit.lc (rep-list g1)) (?lp - punit.lt (rep-list g1)) $g 1$
unfolding $M$-def by fastforce
from 1 have adds: punit.lt (rep-list g1) adds ?lp and ?lp $\oplus$ lt g1 $\prec_{t}$ punit.lt (rep-list g1) $\oplus$ lt $p$
and rep-list g1 $\neq 0$ using is-sig-red-top-addsE by fastforce+
from this(3) have lc-g1: punit.lc (rep-list g1) $\neq 0$ by (rule punit.lc-not-0)
from $\langle m \in M\rangle\langle 0 \notin$ rep-list ' $M\rangle$ have rep-list $m \neq 0$ by fastforce
from $\langle m \in M\rangle\langle 0 \notin M\rangle$ have $m \neq 0$ by blast
hence $l c$ $m \neq 0$ by (rule lc-not-0)
from lc-g1 have eq0: punit.lc (rep-list $m$ ) $=$ ?lc by (simp add: $m$ rep-list-monom-mult)
from $\langle ? l c \neq 0\rangle\langle r e p-l i s t ~ g 1 \neq 0\rangle$ adds have eq1: punit.lt (rep-list $m$ ) $=$ ?lp by (simp add: m rep-list-monom-mult punit.lt-monom-mult punit.lc-eq-zero-iff adds-minus)
from $\langle m \in M\rangle\langle M \subseteq$ dgrad-sig-set $d\rangle$ have $m \in$ dgrad-sig-set $d$..
from $\langle r e p-l i s t ~ g 1 \neq 0\rangle$ have punit.lc (rep-list g1) $\neq 0$ and $g 1 \neq 0$
by (auto simp: rep-list-zero punit.lc-eq-zero-iff)
with $\langle ? l c \neq 0\rangle$ have $u: u=(? l p-$ punit.lt $($ rep-list g1 $)) \oplus l t ~ g 1$
by (simp add: $u^{\prime} m$ lt-monom-mult lc-eq-zero-iff)
hence punit.lt (rep-list g1 $) \oplus u=$ punit.lt (rep-list g1 $) \oplus((? l p-p u n i t . l t$ (rep-list $g 1)) \oplus(t$ g1)
by $\operatorname{simp}$
also from $a d d s$ have $\ldots=? l p \oplus l t ~ g 1$
by (simp only: splus-assoc[symmetric], metis add.commute adds-minus)
also have $\ldots \prec_{t}$ punit.lt (rep-list g1) $\oplus$ lt $p$ by fact
finally have $u \prec_{t}$ lt $p$ by (rule ord-term-strict-canc)
from $\langle u \in ? M\rangle$ have $p p$-of-term $u \in p p$-of-term' ? $M$ by (rule imageI)
also have $\ldots \subseteq$ dgrad-set $d$ (dgrad-max $d$ ) by fact
finally have $d$ (pp-of-term $u) \leq d g r a d-m a x ~ d ~ b y ~(r u l e ~ d g r a d-s e t D) ~$
from $\langle u \in$ ? $M$ 〉 have component-of-term $u \in$ component-of-term' ?M by (rule imageI)
also from $\langle M \subseteq$ sig-inv-set $\langle 0 \notin M\rangle$ sig-inv-set $D$-lt have $\ldots \subseteq\{0 . .<$ length $f s\}$ by fastforce
finally have component-of-term $u<l e n g t h ~ f s ~ b y ~ s i m p ~$
have $\neg i s-s y z-s i g d u$

## proof

assume $i s$－syz－sig $d u$
then obtain $s$ where $s \neq 0$ and $l t s=u$ and $s \in d g r a d$－sig－set $d$ and rep－list $s=0$
by（rule is－syz－sigE）
let ？s＝monom－mult（lc m／lc s） 0 s
have rep－list ？s $=0$ by（simp add：rep－list－monom－mult 〈rep－list $s=0\rangle$ ）
from $\langle s \neq 0\rangle$ have $l c s \neq 0$ by（rule lc－not－0）
hence $l c m / l c s \neq 0$ using «lc $m \neq 0$ 〉 by simp
have $m-$ ？$s \neq 0$
proof
assume $m-$ ？$s=0$
hence $m=$ ？s by simp
with $\langle$ rep－list ？$s=0\rangle$ have rep－list $m=0$ by simp
with $\langle r e p-l i s t ~ m \neq 0\rangle$ show False ．．
qed
moreover from $\langle l c m / l c s \neq 0\rangle$ have $l t ? s=l t m$ by（simp add：lt－monom－mult－zero〈lt $s=u\rangle u^{\prime}$ ）
moreover from 〈lc $s \neq 0$ 〉 have $l c$ ？$s=l c m$ by simp
ultimately have $l t$（ $m-$ ？s）$\prec_{t} u$ unfolding $u^{\prime}$ by（rule lt－minus－lessI）
hence $l t(m-$ ？$s) \notin ? M$ by（rule min）
hence $m-$ ？s $\notin M$ by blast
moreover have $m-$ ？$s \in M$
proof－
have ？s＝monom－mult（？lc／lc s） 0 （monom－mult（lc g1／punit．lc（rep－list g1）） $0 s$
by（simp add：m monom－mult－assoc mult．commute）
define $m^{\prime}$ where $m^{\prime}=m-$ ？s
have eq：rep－list $m^{\prime}=$ rep－list $m$ by（simp add：$m^{\prime}$－def rep－list－minus $\langle r e p-l i s t$ ？$s=0\rangle$ ）
from $\langle ? l c \neq 0\rangle$ have $m^{\prime}=$ monom－mult $\left(? l c /\right.$ punit．lc $\left(\right.$ rep－list $\left.\left.m^{\prime}\right)\right)(? l p-$ punit．lt（rep－list $\left.m^{\prime}\right)$ ）$m^{\prime}$
by（simp add：eq eq0 eq1）
also have $\ldots \in M$ unfolding $M$－def
proof（rule，intro exI conjI）
from $\langle s \in d g r a d-$－sig－set $d\rangle$ have $? s \in d g r a d$－sig－set $d$
by（rule dgrad－sig－set－closed－monom－mult－zero）
with $\langle m \in$ dgrad－sig－set $d\rangle$ show $m^{\prime} \in d g r a d$－sig－set $d$ unfolding $m^{\prime}$－def by（rule dgrad－sig－set－closed－minus）
next
show is－sig－red $\left(\prec_{t}\right)(=)\left\{m^{\prime}\right\} p$
proof（rule is－sig－red－top－addsI）
show $m^{\prime} \in\left\{m^{\prime}\right\}$ by simp
next
from $\langle r e p-l i s t ~ m \neq 0\rangle$ show rep－list $m^{\prime} \neq 0$ by（simp add：eq）
next
show punit．lt（rep－list $m^{\prime}$ ）adds punit．lt（rep－list p）by（simp add：eq eq1） next
have punit．lt $($ rep－list $p) \oplus l t m^{\prime} \prec_{t}$ punit．lt $($ rep－list $p) \oplus u$

```
                    by (rule splus-mono-strict, simp only: m'-def <lt (m - ?s) < <t u〉)
                    also have ... }\mp@subsup{\prec}{t}{}\mathrm{ punit.lt (rep-list m') }\oplus\mathrm{ lt p
                    unfolding eq eq1 using <u < < lt p> by (rule splus-mono-strict)
                    finally show punit.lt (rep-list p)\opluslt m' }\mp@subsup{\prec}{t}{}\mathrm{ punit.lt (rep-list m') }\oplus\mathrm{ lt p .
            next
                show ord-term-lin.is-le-rel (}\mp@subsup{\prec}{t}{})\mathrm{ by simp
            qed fact
        qed (fact refl)
        finally show ?thesis by (simp only: m'-def)
    qed
    ultimately show False ..
qed
have is-RB-in d rword Gu by (rule is-RB-uptD2, fact+)
thus ?thesis
proof (rule is-RB-inE)
    assume is-syz-sig d u
    with 〈\neg is-syz-sig d u` show ?thesis ..
next
    fix g
    assume is-canon-rewriter rword G ug
    hence g}\inG\mathrm{ and }g\not=0\mathrm{ and adds': lt g addst u by (rule is-canon-rewriterD)+
    assume irred: ᄀ is-sig-red ( }\mp@subsup{\prec}{t}{})(=)G(monom-mult 1 (pp-of-term u - lp g)
g)
    define b}\mathrm{ where b= monom-mult 1(pp-of-term u-lpg)g
    note assms(1)
    moreover have is-sig-GB-upt d G (lt m) unfolding u'[symmetric]
    by (rule is-sig-GB-upt-le, rule is-RB-upt-is-sig-GB-upt, fact+, rule ord-term-lin.less-imp-le,
fact)
    moreover from assms(1) have b\indgrad-sig-set d unfolding b-def
    proof (rule dgrad-sig-set-closed-monom-mult)
            from adds' have lp g adds pp-of-term u by (simp add: adds-term-def)
            with assms(1) have d (pp-of-term u-lpg)\leqd (pp-of-term u) by (rule
dickson-grading-minus)
            thus d (pp-of-term u - lp g) \leqdgrad-max d using<d (pp-of-term u)\leq
dgrad-max d>
            by (rule le-trans)
        next
            from <g\inG`G-sub show g}\indgrad-sig-set d ..
    qed
    moreover note <m \indgrad-sig-set d>
    moreover from }\langleg\not=0\rangle\mathrm{ have lt b=lt m
    by (simp add: b-def u'[symmetric] lt-monom-mult,
        metis adds' add-diff-cancel-right' adds-termE pp-of-term-splus)
    moreover from }\langleg\not=0\rangle\mathrm{ have }b\not=0\mathrm{ by (simp add: b-def monom-mult-eq-zero-iff)
    moreover note < m}\not=0\mathrm{ \
    moreover from irred have }\neg\mathrm{ is-sig-red ( }\mp@subsup{\prec}{t}{})(=)Gb\mathrm{ by (simp add: b-def)
    moreover have }\neg\mathrm{ is-sig-red ( }\mp@subsup{\prec}{t}{})(=)G
```


## proof

assume is-sig-red $\left(\prec_{t}\right)(=) G m$
then obtain $g_{2}$ where 1:g2 $\in G$ and 2: rep-list $g 2 \neq 0$
and 3: punit.lt (rep-list g2) adds punit.lt (rep-list m)
and 4: punit.lt $($ rep-list $m) \oplus l t$ g2 $\prec_{t}$ punit.lt $($ rep-list g2) $\oplus l t m$
by (rule is-sig-red-top-addsE)
from 2 have $g 2 \neq 0$ and punit.lc (rep-list g2) $\neq 0$ by (auto simp: rep-list-zero punit.lc-eq-zero-iff)
with 3 \& have $l t$ (monom-mult (?lc / punit.lc (rep-list g2)) (?lp - punit.lt
(rep-list g2)) $g$ 2) $\prec_{t} u$
(is $l t ? g 2 \prec_{t} u$ )
using 〈?lc $\neq 0$ 〉 by (simp add: term-is-le-rel-minus u' eq1 lt-monom-mult)
hence $l t ? g 2 \notin ? M$ by (rule min)
hence ? $g 2 \notin M$ by blast
hence $g 2 \notin d g r a d$-sig-set $d \vee \neg$ is-sig-red $\left(\prec_{t}\right)(=)\{g 2\} p$ by (simp add:
M-def)
thus False
proof
assume $g 2 \notin d g r a d$-sig-set $d$
moreover from $\langle g 2 \in G\rangle G$-sub have $g 2 \in d g r a d$-sig-set $d .$.
ultimately show ?thesis ..
next
assume $\neg$ is-sig-red $\left(\prec_{t}\right)(=)\{g 2\} p$
moreover have is-sig-red $\left(\prec_{t}\right)(=)\{g 2\} p$
proof (rule is-sig-red-top-addsI)
show $g 2 \in\{g 2\}$ by $\operatorname{simp}$
next
from 3 show punit.lt (rep-list g2) adds punit.lt (rep-list p) by (simp only: eq1)
next
from 4 have ?lp $\oplus$ lt g2 $_{2} \prec_{t}$ punit.lt (rep-list g2) $\oplus u$ by (simp only: eq1 $u^{\prime}$ )
also from $\left\langle u \prec_{t}\right.$ lt $\left.p\right\rangle$ have $\ldots \prec_{t}$ punit.lt (rep-list g2) $\oplus$ lt $p$ by (rule splus-mono-strict)
finally show ?lp $\oplus$ lt g2 $\prec_{t}$ punit.lt (rep-list g2) $\oplus$ lt $p$.
next
show ord-term-lin.is-le-rel $\left(\prec_{t}\right)$ by simp
qed fact+
ultimately show ?thesis ..
qed
qed
ultimately have eq2: punit.lt (rep-list b) = punit.lt (rep-list m)
by (rule sig-regular-top-reduced-lt-unique)
have rep-list $g \neq 0$ by (rule is- $R B$-inD, fact+)
moreover from $a d d s^{\prime}$ have $l p g$ adds pp-of-term $u$ and component-of-term (lt $g)=$ component-of-term u
by (simp-all add: adds-term-def)
ultimately have $u=(? l p-$ punit.lt $($ rep-list $g)) \oplus$ lt $g$
by (simp add: eq1[symmetric] eq2[symmetric] b-def rep-list-monom-mult

```
punit.lt-monom-mult
                splus-def adds-minus term-simps)
    have is-sig-red \(\left(\prec_{t}\right)(=)\{b\} p\)
    proof (rule is-sig-red-top-addsI)
        show \(b \in\{b\}\) by simp
    next
        from \(\langle\) rep-list \(g \neq 0\rangle\) show rep-list \(b \neq 0\)
            by (simp add: b-def rep-list-monom-mult punit.monom-mult-eq-zero-iff)
    next
        show punit.lt (rep-list b) adds punit.lt (rep-list p) by (simp add: eq1 eq2)
    next
        show punit.lt \((\) rep-list \(p) \oplus l t b \prec_{t}\) punit.lt (rep-list b) \(\oplus\) lt \(p\)
        by (simp add: eq1 eq2 «lt \(b=l t m\rangle u^{\prime}[\) symmetric \(]\left\langle u \prec_{t}\right.\) lt \(\left.p\right\rangle\) splus-mono-strict \()\)
    next
        show ord-term-lin.is-le-rel \(\left(\prec_{t}\right)\) by simp
    qed fact
    hence is-sig-red \(\left(\prec_{t}\right)(=)\{g\} p\) unfolding \(b\)-def by (rule is-sig-red-singleton-monom-mult \(D\) )
    show ?thesis by (rule, fact+)
    qed
qed
```


### 4.2.5 Termination

definition term-pp-rel :: $\left({ }^{\prime} t \Rightarrow{ }^{\prime} t \Rightarrow\right.$ bool $) \Rightarrow\left({ }^{\prime} t \times{ }^{\prime} a\right) \Rightarrow\left({ }^{\prime} t \times{ }^{\prime} a\right) \Rightarrow$ bool where term-pp-rel $r$ a $b \longleftrightarrow r($ snd $b \oplus f s t a)($ snd $a \oplus f s t b)$
definition canon-term-pp-pair :: (' $\left.t \times{ }^{\prime} a\right) \Rightarrow$ bool where canon-term-pp-pair $a \longleftrightarrow($ gcs $($ pp-of-term $(f s t a))($ snd $a)=0)$
definition cancel-term-pp-pair :: ( $\left.t \times{ }^{\prime} a\right) \Rightarrow\left({ }^{\prime} t \times{ }^{\prime} a\right)$
where cancel-term-pp-pair $a=(f s t a \ominus(g c s(p p-o f-t e r m(f s t ~ a))($ snd $a))$, snd $a$

- (gcs (pp-of-term $($ fst $a))($ snd $a)))$
lemma term-pp-rel-refl: reflp $r \Longrightarrow$ term-pp-rel raa
by (simp add: term-pp-rel-def reflp-def)
lemma term-pp-rel-irrefl: irreflp $r \Longrightarrow$ $\Longrightarrow$ term-pp-rel $r$ a a
by (simp add: term-pp-rel-def irreflp-def)
lemma term-pp-rel-sym: symp $r \Longrightarrow$ term-pp-rel rablerm-pp-rel r ba by (auto simp: term-pp-rel-def symp-def)
lemma term-pp-rel-trans:
assumes ord-term-lin.is-le-rel $r$ and term-pp-rel $r a b$ and term-pp-rel rbc
shows term-pp-rel r a c
proof -
from $\operatorname{assms}(1)$ have transp $r$ by (rule ord-term-lin.is-le-relE, auto)
from $\operatorname{assms}(2)$ have 1: $r$ (snd $b \oplus f s t a)$ (snd $a \oplus f s t b$ ) by (simp only:
term-pp-rel-def)

```
    from assms(3) have 2: r (snd c\oplusfst b) (snd b \oplus fst c) by (simp only:
term-pp-rel-def)
    have snd b\oplus(snd c\oplusfst a)=snd c\oplus(snd b\oplusfst a) by (rule splus-left-commute)
    also from assms(1) 1 have r ... (snd a\oplus(snd c\oplusfst b))
        by (simp add: splus-left-commute[of snd a] term-is-le-rel-canc-left)
    also from assms(1) 2 have r ... (snd b \oplus (snd a @ fst c))
    by (simp add: splus-left-commute[of snd b] term-is-le-rel-canc-left)
    finally(transpD[OF <transp r>]) show ?thesis using assms(1)
    by (simp only: term-pp-rel-def term-is-le-rel-canc-left)
qed
lemma term-pp-rel-trans-eq-left:
    assumes ord-term-lin.is-le-rel r and term-pp-rel (=) ab and term-pp-rel rb c
    shows term-pp-rel r a c
proof -
    from assms(1) have transp r by (rule ord-term-lin.is-le-relE, auto)
    from assms(2) have 1: snd b\oplus fst a=snd a }\oplus\mathrm{ fst b by (simp only: term-pp-rel-def)
    from assms(3) have 2: r (snd c\oplusfst b) (snd b \oplus fst c) by (simp only:
term-pp-rel-def)
    have snd b\oplus(snd c\oplusfst a)=snd c\oplus(snd b\oplusfst a) by (rule splus-left-commute)
    also from assms(1) 1 have ... = (snd a \oplus(snd c\oplusfst b))
            by (simp add: splus-left-commute[of snd a])
    finally have eq: snd b\oplus(snd c\oplus fst a)= snd a }\oplus(\mathrm{ snd c }\oplus\mathrm{ fst b).
    from assms(1) 2 have r (snd b \oplus (snd c\oplusfst a)) (snd b\oplus (snd a\oplus fst c))
    unfolding eq by (simp add: splus-left-commute[of snd b] term-is-le-rel-canc-left)
    thus ?thesis using assms(1) by (simp only: term-pp-rel-def term-is-le-rel-canc-left)
qed
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
proof -
    from assms(1) have transp r by (rule ord-term-lin.is-le-relE, auto)
    from assms(2) have 1:r (snd b \oplus fst a) (snd a \oplus fst b) by (simp only:
term-pp-rel-def)
    from assms(3) have 2: snd c\oplus fst b=snd b}\oplus\mathrm{ fst c by (simp only: term-pp-rel-def)
    have snd b}\oplus(snd a\oplusfstc)=snd a\oplus(snd b\oplusfst c) by (rule splus-left-commute
    also from assms(1) 2 have ... = (snd a \oplus(snd c\oplus fst b))
            by (simp add: splus-left-commute[of snd a])
    finally have eq: snd b \oplus(snd a\oplus fst c)= snd a }\oplus(\mathrm{ snd c }\oplus\mathrm{ fst b).
    from assms(1) 1 have r (snd b \oplus(snd c\oplusfst a)) (snd b \oplus (snd a \oplus fst c))
    unfolding eq by (simp add: splus-left-commute[of - snd c] term-is-le-rel-canc-left)
    thus ?thesis using assms(1) by (simp only: term-pp-rel-def term-is-le-rel-canc-left)
qed
lemma canon-term-pp-cancel: canon-term-pp-pair (cancel-term-pp-pair a)
    by (simp add: cancel-term-pp-pair-def canon-term-pp-pair-def gcs-minus-gcs term-simps)
lemma term-pp-rel-cancel:
```

```
    assumes reflp r
    shows term-pp-rel r a (cancel-term-pp-pair a)
proof -
    obtain us where a: a=(u,s) by (rule prod.exhaust)
    show ?thesis
    proof (simp add: a cancel-term-pp-pair-def)
    let ?g=gcs (pp-of-term u)s
    have ?g adds s by (fact gcs-adds-2)
    hence (s-?g)\oplus(u\ominus0)=s\oplusu\ominus(?g+0) using zero-adds-pp
        by (rule minus-splus-sminus)
    also have ... =s\oplus(u\ominus?g)
        by (metis add.left-neutral add.right-neutral adds-pp-def diff-zero gcs-adds-2
gcs-comm
                minus-splus-sminus zero-adds)
            finally have r ((s-?g)\oplusu) (s\oplus(u\ominus ?g)) using assms by (simp add:
term-simps reflp-def)
            thus term-pp-rel r (u,s) (u\ominus?g,s - ?g) by (simp add: a term-pp-rel-def)
        qed
qed
lemma canon-term-pp-rel-id:
    assumes term-pp-rel (=) ab and canon-term-pp-pair a and canon-term-pp-pair
b
    shows }a=
proof -
    obtain us where a: a= (u,s) by (rule prod.exhaust)
    obtain vt where b: b=(v,t) by (rule prod.exhaust)
    from assms(1) have t\oplusu=s\oplusv by (simp add: term-pp-rel-def a b)
    hence 1:t+pp-of-term u=s+pp-of-term v by (metis pp-of-term-splus)
    from assms(2) have 2: gcs (pp-of-term u) s=0 by (simp add: canon-term-pp-pair-def
a)
    from assms(3) have 3: gcs (pp-of-term v) t=0 by (simp add: canon-term-pp-pair-def
b)
    have t=t +gcs (pp-of-term u)s by (simp add: 2)
    also have ... = gcs (t+pp-of-term u) (t+s) by (simp only: gcs-plus-left)
    also have \ldots.. gcs (s+pp-of-term v) (s+t) by (simp only: 1 add.commute)
    also have ... =s + gcs (pp-of-term v) t by (simp only: gcs-plus-left)
    also have ... =s by (simp add: 3)
    finally have }t=s\mathrm{ .
    moreover from }\langlet\oplusu=s\oplusv\rangle\mathrm{ have }u=v\mathrm{ by (simp only: <t=s` splus-left-canc)
    ultimately show ?thesis by (simp add: a b)
qed
lemma min-set-finite:
    fixes seq :: nat => ('t =>0 'b::field )
    assumes dickson-grading d and range seq }\subseteqdgrad-sig-set d and 0 # rep-list '
range seq
    and \ij.i<j\Longrightarrowlt (seq i) \prec}\mp@subsup{}{t}{lt}(seqj
    shows finite {i.\neg(\existsj<i.lt (seq j) addst lt (seq i)^
```

proof -
have $i n j(\lambda i$. lt (seq $i))$
proof
fix $i j$
assume eq: lt $(s e q i)=l t(s e q j)$
show $i=j$
proof (rule linorder-cases)
assume $i<j$
hence lt (seq i) $\prec_{t} l t(s e q j)$ by (rule assms(4))
thus ?thesis by (simp add: eq)
next
assume $j<i$
hence $l t(s e q j) \prec_{t} l t(s e q i)$ by (rule assms(4))
thus ?thesis by (simp add: eq)
qed
qed
hence inj seq unfolding comp-def[symmetric] by (rule inj-on-imageI2)
let $? P 1=\lambda p q$. lt $p$ addst lt $q$
let ?P2 $=\lambda p$ q. punit.lt $($ rep-list p) adds punit.lt $($ rep-list $q)$
let $? P=\lambda p q$. ?P1 $p q \wedge$ ?P2 $p q$
have reflp ?P by (simp add: reflp-def adds-term-refl)
have almost-full-on ?P1 (range seq)
proof (rule almost-full-on-map)
let $? B=\{t$. pp-of-term $t \in d g r a d$-set $d($ dgrad-max $d) \wedge$ component-of-term $t$
$\in\{0 . .<$ length $f s\}\}$
from assms(1) finite-atLeastLessThan show almost-full-on (addst) ?B by (rule
Dickson-term)
show lt'range seq $\subseteq$ ?B
proof
fix $v$
assume $v \in l t$ ' range seq
then obtain $p$ where $p \in$ range seq and $v: v=l t p$..
from this(1) assms(3) have rep-list $p \neq 0$ by auto
hence $p \neq 0$ by (auto simp: rep-list-zero)
from $\langle p \in$ range seq〉assms(2) have $p \in$ dgrad-sig-set $d$..
hence $d(l p \quad p) \leq d g r a d-m a x ~ d$ by (rule dgrad-sig-setD-lp)
hence $l p$ p dgrad-set $d$ (dgrad-max d) by (simp add: dgrad-set-def)
moreover from $\langle p \in d g r a d$-sig-set $d\rangle\langle p \neq 0\rangle$ have component-of-term (lt
$p)<$ length $f s$
by (rule dgrad-sig-setD-lt)
ultimately show $v \in ? B$ by ( $\operatorname{simp} a d d: v$ )
qed
qed
moreover have almost-full-on ?P2 (range seq)
proof (rule almost-full-on-map)
let $? B=$ dgrad-set $d$ (dgrad-max d)
from $\operatorname{assms}(1)$ show almost-full-on (adds) ?B by (rule dickson-gradingD-dgrad-set)

```
    show ( }\lambda\mathrm{ p. punit.lt (rep-list p))'range seq }\subseteq\mathrm{ ? B
    proof
        fix }
        assume t ( }\lambda\mathrm{ p. punit.lt (rep-list p))' range seq
        then obtain p where p\in range seq and t: t= punit.lt (rep-list p)..
        from this(1) assms(3) have rep-list p\not=0 by auto
        from }\langlep\in\mathrm{ range seq> assms(2) have p}\in\mathrm{ dgrad-sig-set d ..
        hence }p\indgrad-max-set d by (simp add: dgrad-sig-set'-def
        with assms(1) have rep-list p f punit-dgrad-max-set d by (rule dgrad-max-2)
            from this <rep-list p\not=0\rangle have d (punit.lt (rep-list p)) \leq dgrad-max d
            by (rule punit.dgrad-p-setD-lp[simplified])
            thus t\in?B by (simp add: t dgrad-set-def)
        qed
    qed
    ultimately have almost-full-on ?P (range seq) by (rule almost-full-on-same)
    with\langlereflp?P\rangle obtain T where finite T and T\subseteq range seq and *: \p.p\in
range seq\Longrightarrow(\existsq\inT.?P q p)
    by (rule almost-full-on-finite-subsetE, blast)
    from <T\subseteq range seq\rangle obtain I where T:T = seq'I by (meson sub-
set-image-iff)
    have {i.\neg(\existsj<i. ?P (seq j) (seq i))}\subseteqI
    proof
        fix i
        assume }i\in{i.\neg(\existsj<i.?PP(seq j)(seq i))
        hence }x\mathrm{ : }\neg(\existsj<i. ?P (seq j) (seq i)) by sim
        obtain j where j\inI and ?P (seq j) (seq i)
        proof -
            have seq i f range seq by simp
            hence }\existsq\inT\mathrm{ . ?P q (seq i) by (rule *)
            then obtain q where q\inT and ?P q (seq i)..
            from this(1) obtain j where j\inI and q=seq j unfolding T ..
            from this(1)〈?P q (seq i)\rangle show ?thesis unfolding <q = seq j〉 ..
        qed
        from this(2) x have i\leqj by auto
        moreover have }\negi<
        proof
            assume i<j
            hence lt (seq i) \prec}\mp@subsup{}{t}{lt (seq j) by (rule assms(4))
            hence \neg ?P1 (seq j) (seq i) using ord-adds-term ord-term-lin.leD by blast
            with «?P (seq j) (seq i)` show False by simp
            qed
            ultimately show }i\inI\mathrm{ using <j 
    qed
    moreover from〈inj seq〉<finite T〉 have finite I by (simp add: finite-image-iff
inj-on-subset T)
    ultimately show ?thesis by (rule finite-subset)
qed
lemma rb-termination:
```

```
    fixes seq :: nat \(\Rightarrow\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) field \()\)
    assumes dickson-grading \(d\) and range seq \(\subseteq\) dgrad-sig-set \(d\) and \(0 \notin\) rep-list •
range seq
    and \(\bigwedge i j . i<j \Longrightarrow l t(\) seq \(i) \prec_{t} l t(\) seq \(j)\)
    and \(\bigwedge i\). \(\neg i s\)-sig-red \(\left(\prec_{t}\right)(\preceq)(\) seq ' \(\{0 . .<i\})(\) seq \(i)\)
    and \(\bigwedge i .(\exists j<\) length \(f\). lt \((\) seq \(i)=l t\) (monomial \((1:: ' b)(\) term-of-pair \((0, j)))\)
\(\wedge\)
            punit.lt (rep-list (seq i)) \(\preceq\) punit.lt (rep-list (monomial 1 (term-of-pair
\((0, j))))) \vee\)
            \((\exists j k . i s\)-regular-spair \((\operatorname{seq} j)(\) seq \(k) \wedge\) rep-list \((\operatorname{spair}(\operatorname{seq} j)(\) seq \(k)) \neq\)
\(0 \wedge\)
                    \(l t(\) seq \(i)=l t(\) spair \((\) seq \(j)(\) seq \(k)) \wedge\)
                    punit.lt (rep-list (seq i)) \(\preceq\) punit.lt (rep-list \((\operatorname{spair}(\operatorname{seq} j)(\) seq \(k)))\) )
    and \(\bigwedge i\) is-sig-GB-upt \(d(\) seq' \(\{0 . .<i\})(l t(\) seq \(i))\)
    shows thesis
proof -
    from assms(3) have \(0 \notin\) range seq using rep-list-zero by auto
    have ord-term-lin.is-le-rel \((=)\) and ord-term-lin.is-le-rel \(\left(\prec_{t}\right)\) by (rule ord-term-lin.is-le-relI) +
    have reflp (=) and symp (=) by (simp-all add: symp-def)
    have irreflp \(\left(\prec_{t}\right)\) by (simp add: irreflp-def)
    have inj ( \(\lambda i\). lt (seq i))
    proof
        fix \(i j\)
        assume eq: lt \((\) seq \(i)=l t(s e q j)\)
        show \(i=j\)
        proof (rule linorder-cases)
            assume \(i<j\)
            hence \(l t(\) seq \(i) \prec_{t} l t(s e q j)\) by (rule assms(4))
            thus ?thesis by (simp add: eq)
    next
            assume \(j<i\)
            hence lt (seq \(j) \prec_{t} l t\) (seq i) by (rule assms(4))
            thus ?thesis by (simp add: eq)
        qed
    qed
    hence inj seq unfolding comp-def[symmetric] by (rule inj-on-imageI2)
    define \(R\) where \(R=(\lambda x\). \(\{i\). term-pp-rel \((=)(l t\) (seq \(i)\), punit.lt (rep-list (seq
i))) \(x\}\) )
    let ?A \(=\{x\). canon-term-pp-pair \(x \wedge R x \neq\{ \}\}\)
    have finite ?A
    proof -
    define min-set where min-set \(=\{i . \neg(\exists j<i\).lt \((\) seq \(j)\) addst \(l t(s e q i) \wedge\)
                            punit.lt (rep-list (seq j)) adds punit.lt (rep-list (seq
i))) \(\}\)
    have \(? A \subseteq(\lambda i\). cancel-term-pp-pair \((l t(\) seq \(i)\), punit.lt (rep-list \((\) seq \(i))))\) '
min-set
    proof
```

fix $u t$
assume $(u, t) \in ? A$
hence canon－term－pp－pair $(u, t)$ and $R(u, t) \neq\{ \}$ by simp－all
from this（2）obtain $i$ where $x$ ：term－pp－rel（＝）（lt（seq i），punit．lt（rep－list （seq $i))$ ）（ $u, t$ ）
by（auto simp：R－def）
let ？equiv $=(\lambda i j$ ．term－pp－rel $(=)($ lt $($ seq i），punit．lt $($ rep－list $($ seq $i)))(l t$ （seq j），punit．lt（rep－list（seq j））））
obtain $j$ where $j \in$ min－set and ？equiv $j i$
proof（cases $i \in$ min－set）
case True
moreover have ？equiv $i$ i by（simp add：term－pp－rel－refl）
ultimately show ？thesis ．．
next
case False
let ？$Q=\{$ seq $j \mid j . j<i \wedge i s$－sig－red $(=)(=)\{$ seq $j\}($ seq $i)\}$
have ？$Q \subseteq$ range seq by blast
also have $\ldots \subseteq$ dgrad－sig－set $d$ by（fact assms（2））
finally have ？$Q \subseteq$ dgrad－max－set d by（simp add：dgrad－sig－set＇－def）
moreover from 〈？$Q \subseteq$ range seq〉〈 $\notin$ range seq〉 have $0 \notin ? Q$ by blast
ultimately have $Q$－sub：pp－of－term＇lt＇？$Q \subseteq$ dgrad－set $d$（dgrad－max d）
unfolding image－image by（smt CollectI dgrad－p－setD－lp dgrad－set－def image－subset－iff subsetCE）
have $*: \exists g \in$ seq＇$\{0 . .<k\}$ ．is－sig－red $(=)(=)\{g\}($ seq $k)$ if $k \notin$ min－set for $k$
proof－
from that obtain $j$ where $j<k$ and $a$ ：lt $(s e q j) a d d s_{t} l t(s e q k)$
and b：punit．lt（rep－list（seq j））adds punit．lt（rep－list（seq $k$ ））by（auto simp：min－set－def）
note $\operatorname{assms}(1,7)$
moreover from assms（2）have seq $k \in d g r a d$－sig－set $d$ by fastforce
moreover from $\langle j<k\rangle$ have $\operatorname{seq} j \in \operatorname{seq}{ }^{\prime}\{0 . .<k\}$ by simp
moreover from $\operatorname{assms}(3)$ have rep－list $($ seq $k) \neq 0$ and rep－list（seq j）
$\neq 0$ by fastforce +
ultimately have is－sig－red $\left(\preceq_{t}\right)(=)(s e q '\{0 . .<k\})(s e q k)$ using $a b$ by （rule lemma－21）
moreover from $\operatorname{assms}(5)[$ of $k]$ have $\neg$ is－sig－red $\left(\prec_{t}\right)(=)(s e q ‘\{0 . .<k\})$ （seq $k$ ）
by（simp add：is－sig－red－top－tail－cases）
ultimately have is－sig－red $(=)(=)($ seq＇$\{0 . .<k\})($ seq $k)$
by（simp add：is－sig－red－sing－reg－cases）
then obtain $g 0$ where $g 0 \in$ seq＇$\{0 . .<k\}$ and is－sig－red $(=)(=)\{g 0\}$ （seq k）
by（rule is－sig－red－singletonI）
thus ？thesis ．．
qed
from this［OF False］obtain $g 0$ where $g 0 \in s e q$＇$\{0 . .<i\}$ and is－sig－red $(=)(=)\{g 0\}($ seq $i) .$.

```
    hence g0 \in?Q by fastforce
    hence lt g0 \inlt '?Q by (rule imageI)
```



```
&lt '?Q
            using Q-sub by (rule ord-term-minimum-dgrad-set, blast)
            from this(1) obtain j where j<i and is-sig-red (=)(=){seq j} (seq i)
            and v:v=lt (seq j) by fastforce
    hence 1: punit.lt (rep-list (seq j)) adds punit.lt (rep-list (seq i))
    and 2: punit.lt (rep-list (seq i)) \opluslt (seq j) = punit.lt (rep-list (seq j)) }
lt (seq i)
            by (auto elim: is-sig-red-top-addsE)
            show ?thesis
    proof
        show ?equiv j i by (simp add: term-pp-rel-def 2)
    next
        show j\in min-set
        proof (rule ccontr)
            assume j\not\in min-set
            from *[OF this] obtain g1 where g1\in seq'{ {0..<j} and red:is-sig-red
(=)(=){g1} (seq j)..
            from this(1) obtain j0 where j0<j and g1 = seq j0 by fastforce+
            from red have 3: punit.lt (rep-list (seq j0)) adds punit.lt (rep-list (seq
j))
                            and 4: punit.lt (rep-list (seq j)) \opluslt (seq j0) = punit.lt (rep-list (seq
j0))}\opluslt(seq j
                            by (auto simp: <g1 = seq j0> elim: is-sig-red-top-addsE)
            from }\langlej0<j\rangle<j<i\rangle have j0<i by sim
            from <j0<j\rangle have lt (seq j0) \prec}\mp@subsup{t}{t}{}v\mathrm{ unfolding v by (rule assms(4))
            hence lt (seq j0) #lt '?Q by (rule min)
            with}\langlej0<i\rangle\mathrm{ have }\negis\mathrm{ -sig-red (=) (=) {seq j0} (seq i) by blast
            moreover have is-sig-red (=) (=) {seq j0} (seq i)
            proof (rule is-sig-red-top-addsI)
                from assms(3) show rep-list (seq j0) \not=0 by fastforce
            next
                from assms(3) show rep-list (seq i)}\not=0\mathrm{ by fastforce
            next
            from 3 1 show punit.lt (rep-list (seq j0)) adds punit.lt (rep-list (seq i))
                by (rule adds-trans)
            next
                from 4 have ?equiv j0 j by (simp add: term-pp-rel-def)
                also from 2 have ?equiv j i by (simp add: term-pp-rel-def)
                finally(term-pp-rel-trans[OF <ord-term-lin.is-le-rel (=)>])
                    show punit.lt (rep-list (seq i)) \opluslt (seq j0) = punit.lt (rep-list (seq
j0))\opluslt (seq i)
            by (simp add: term-pp-rel-def)
            next
                show ord-term-lin.is-le-rel (=) by simp
```

qed simp－all
ultimately show False ．．
qed
qed
qed
have term－pp－rel（ $=$ ）（cancel－term－pp－pair（lt（seq j），punit．lt（rep－list（seq
$j))$ ）（lt（seq j），punit．lt（rep－list（seq j）））
by（rule term－pp－rel－sym，fact «symp（＝）〉，rule term－pp－rel－cancel，fact 〈reflp （＝）＞）
also note 〈？equiv $j$ i〉
also（term－pp－rel－trans $[$ OF 〈ord－term－lin．is－le－rel（＝）$)]$ ）note $x$
finally（term－pp－rel－trans $[$ OF＜ord－term－lin．is－le－rel（＝）〉］）
have term－pp－rel $(=)$（cancel－term－pp－pair（lt（seq j），punit．lt（rep－list（seq $j))$ ）（u，t）．
with $\langle\operatorname{symp}(=)$ ）have term－pp－rel $(=)(u, t)$（cancel－term－pp－pair（lt（seq j）， punit．lt（rep－list（seq j））））
by（rule term－pp－rel－sym）
hence $(u, t)=$ cancel－term－pp－pair（lt（seq j），punit．lt（rep－list（seq j）））
using 〈canon－term－pp－pair（ $u, t$ ）〉 canon－term－pp－cancel by（rule canon－term－pp－rel－id）
with $\langle j \in$ min－set $\rangle$ show $(u, t) \in(\lambda i$ ．cancel－term－pp－pair（lt（seq $i)$ ，punit．lt （rep－list（seq i））））＇min－set
by fastforce
qed
moreover have finite（ $(\lambda i$ ．cancel－term－pp－pair（lt（seq i），punit．lt（rep－list （seq i））））＇min－set）
proof（rule finite－imageI）
show finite min－set unfolding min－set－def using assms（1－4）by（rule min－set－finite）
qed
ultimately show ？thesis by（rule finite－subset）
qed
have range seq $\subseteq$ seq＇$(\bigcup(R$＇？$A))$
proof（rule image－mono，rule）
fix $i$
show $i \in(\bigcup(R$＇？$A))$
proof
show $i \in R($ cancel－term－pp－pair（lt（seq i），punit．lt（rep－list（seq $i)))$ ）
by（simp add：R－def term－pp－rel－cancel）
thus cancel－term－pp－pair（lt（seq i），punit．lt（rep－list（seq i）））$\in$ ？A using canon－term－pp－cancel by blast
qed
qed
moreover from 〈inj seq〉 have infinite（range seq）by（rule range－inj－infinite）
ultimately have infinite（seq＇$(\bigcup(R ‘ ? A))$ ）by（rule infinite－super）
moreover have finite（seq＇$(\bigcup(R ' ? A)))$
proof（rule finite－imageI，rule finite－UN－I）
fix $x$
assume $x \in$ ？$A$

```
    let ?rel = term-pp-rel ( }\mp@subsup{\prec}{t}{}
    have irreflp?rel by (rule irreflpI, rule term-pp-rel-irrefl, fact)
    moreover have transp ?rel by (rule transpI, drule term-pp-rel-trans[OF
<ord-term-lin.is-le-rel ( }\mp@subsup{\prec}{t}{})\mathrm{ \])
    ultimately have wfp-on ?rel ?A using <finite ?A> by (rule wfp-on-finite)
    thus finite ( }Rx\mathrm{ ) using <x < ?A>
    proof (induct rule: wfp-on-induct)
        case (less x)
    from less(1) have canon-term-pp-pair x by simp
    define R' where }\mp@subsup{R}{}{\prime}=\bigcup(R`({x.canon-term-pp-pair x ^R x\not={}}\cap{z
term-pp-rel ( }\mp@subsup{\prec}{t}{})\mathrm{ z x}))
    define red-set where red-set = (\lambdap::'t =\mp@subsup{O}{0}{\prime}'b.{k.lt (seq k)=lt p}
                                    punit.lt (rep-list (seq k))\preceq punit.lt (rep-list p)})
    have finite-red-set: finite (red-set p) for p
    proof (cases red-set p={})
        case True
        thus ?thesis by simp
    next
        case False
        then obtain k where lt-k:lt (seq k)=lt p by (auto simp: red-set-def)
        have red-set p\subseteq{k}
        proof
            fix }\mp@subsup{k}{}{\prime
            assume }\mp@subsup{k}{}{\prime}\inred-set 
            hence lt (seq k') = lt p by (simp add: red-set-def)
            hence lt (seq k') =lt (seq k) by (simp only:lt-k)
            with <inj ( \lambdai.lt (seq i))> have }\mp@subsup{k}{}{\prime}=k\mathrm{ by (rule injD)
            thus }\mp@subsup{k}{}{\prime}\in{k}\mathrm{ by simp
        qed
        thus ?thesis using infinite-super by auto
    qed
    have R x\subseteq(\bigcupi\in\mp@subsup{R}{}{\prime}.\cupj\in\mp@subsup{R}{}{\prime}.
                            (\bigcupj\in{0..<length fs}.red-set (monomial 1 (term-of-pair (0,j))))
        (is - \subseteq? B\cup?C)
    proof
        fix i
        assume i 
        hence i-x: term-pp-rel (=) (lt (seq i), punit.lt (rep-list (seq i))) x
        by (simp add: R-def term-pp-rel-def)
        from assms(6)[of i] show i\in?B\cup?C
        proof (elim disjE exE conjE)
            fix }
            assume j < length fs
            hence j\in{0..<length fs } by simp
            assume lt (seq i) =lt (monomial (1::'b) (term-of-pair ( 0, j)))
        and punit.lt (rep-list (seq i))\preceq punit.lt (rep-list (monomial 1 (term-of-pair
(0, j))))
            hence i fred-set (monomial 1 (term-of-pair ( }0,j))\mathrm{ ) by (simp add:
```

```
red-set-def)
    with <j\in{0..<length fs}> have i\in?C ..
    thus ?thesis ..
    next
        fix jk
    let ?li = punit.lt (rep-list (seq i))
    let ?lj = punit.lt (rep-list (seq j))
    let ?lk = punit.lt (rep-list (seq k))
    assume lt-i:lt (seq i)=lt (spair (seq j) (seq k))
        and lt-i': ?li \preceq punit.lt (rep-list (spair (seq j) (seq k)))
        and spair-0: rep-list (spair (seq j) (seq k)) = 0
    hence i f red-set (spair (seq j) (seq k)) by (simp add: red-set-def)
    from assms(3) have i-0: rep-list (seq i)}\not=0\mathrm{ and j-0: rep-list (seq j)}\not=
        and k-0: rep-list (seq k)}=0\mathrm{ by fastforce+
    have R'I: a \in R' if term-pp-rel ( }\mp@subsup{\prec}{t}{})(lt (seq a), punit.lt (rep-list (seq a)))
x for a
    proof -
        let ?x = cancel-term-pp-pair (lt (seq a), punit.lt (rep-list (seq a)))
        show ?thesis unfolding R'-def
        proof (rule UN-I, simp, intro conjI)
            show }a\inR\mathrm{ ?x by (simp add: R-def term-pp-rel-cancel)
            thus R?x\not={} by blast
        next
            note <ord-term-lin.is-le-rel ( }\mp@subsup{\prec}{t}{})\mathrm{ 〉
                moreover have term-pp-rel (=) ?x (lt (seq a), punit.lt (rep-list (seq
a)))
                by (rule term-pp-rel-sym, fact, rule term-pp-rel-cancel, fact)
                    ultimately show term-pp-rel (}\mp@subsup{\prec}{t}{})\mathrm{ ? x x using that by (rule
term-pp-rel-trans-eq-left)
            qed (fact canon-term-pp-cancel)
    qed
    assume is-regular-spair (seq j) (seq k)
    hence ?lk \opluslt (seq j) \not=?lj\opluslt (seq k) by (rule is-regular-spairD3)
    hence term-pp-rel ( }\mp@subsup{\prec}{t}{})(lt (seq j),?lj) x ^ term-pp-rel ( < < ) (lt (seq k)
?lk) }
    proof (rule ord-term-lin.neqE)
        assume c: ?lk \opluslt (seq j) \prec}\mp@subsup{\}{t}{l}lj\opluslt (seq k
        hence j-k: term-pp-rel (\mp@subsup{\prec}{t}{})(lt (seq j), ?lj) (lt (seq k),?lk)
            by (simp add: term-pp-rel-def)
        note <ord-term-lin.is-le-rel (}\mp@subsup{\prec}{t}{})\mathrm{ 〉
        moreover have term-pp-rel ( }\mp@subsup{\prec}{t}{})(lt (seq k), ?lk) (lt (seq i), ?li)
        proof (simp add: term-pp-rel-def)
            from lt-i' have ?li \opluslt (seq k) \preceq.t
                        punit.lt (rep-list (spair (seq j) (seq k))) \opluslt (seq k)
                by (rule splus-mono-left)
        also have ... }\mp@subsup{\prec}{t}{}(?lk - gcs ?lk ?lj + ?lj)\oplus lt (seq k
            by (rule splus-mono-strict-left, rule lt-rep-list-spair, fact+, simp only:
```

add.commute)
also have $\ldots=((? l k+? l j)-g c s ? l j ? l k) \oplus l t(s e q k)$
by (simp add: minus-plus gcs-adds-2 gcs-comm)
also have $\ldots=? l k \oplus((? l j-g c s ? l j ? l k) \oplus l t(s e q k))$
by (simp add: minus-plus' gcs-adds splus-assoc[symmetric])
also have $\ldots=? l k \oplus l t($ seq $i)$
by (simp add: lt-spair ' $[$ OF $k-0-c]$ add.commute spair-comm[of seq $j]$
$l t-i)$
finally show ? ${ }^{2} \oplus l t(\operatorname{seq} k) \prec_{t} ? l k \oplus l t(\operatorname{seq} i) \cdot$
qed
ultimately have term-pp-rel $\left(\prec_{t}\right)$ (lt (seq $k$ ), ?lk) $x$ using $i$-x
by (rule term-pp-rel-trans-eq-right)
moreover from <ord-term-lin.is-le-rel $\left.\left(\prec_{t}\right)\right\rangle j$-k this
have term-pp-rel $\left(\prec_{t}\right)$ (lt (seq $j$ ), ?lj) $x$ by (rule term-pp-rel-trans)
ultimately show? thesis by simp
next
assume $c: ? l j \oplus l t(s e q k) \prec_{t} ? l k \oplus l t(s e q j)$
hence $j$-k: term-pp-rel $\left(\prec_{t}\right)(l t(s e q k)$, ?lk) (lt (seq j), ?lj)
by (simp add: term-pp-rel-def)
note $\left\langle\right.$ ord-term-lin.is-le-rel $\left(\prec_{t}\right)$ 〉
moreover have term-pp-rel $\left(\prec_{t}\right)(l t(s e q j)$,?lj) (lt (seq i), ?li)
proof (simp add: term-pp-rel-def)
from $l t-i^{\prime}$ have ? $l i \oplus l t(s e q j) \preceq_{t}$ punit.lt (rep-list $(\operatorname{spair}(\operatorname{seq} j)(\operatorname{seq} k))) \oplus l t($ seq $j)$
by (rule splus-mono-left)
thm lt-rep-list-spair
also have $\ldots \prec_{t}(? l k-g c s ? l k ? l j+? l j) \oplus l t(s e q j)$
by (rule splus-mono-strict-left, rule lt-rep-list-spair, fact+, simp only:
add.commute)
also have $\ldots=((? l k+? l j)-g c s ? l k ? l j) \oplus l t(s e q j)$
by (simp add: minus-plus gcs-adds-2 gcs-comm)
also have $\ldots=? l j \oplus((? l k-g c s ? l k ? l j) \oplus l t(s e q j))$
by (simp add: minus-plus' gcs-adds splus-assoc[symmetric] add.commute) also have $\ldots=? l j \oplus l t(s e q ~ i)$ by (simp add: lt-spair ${ }^{\prime}[O F j-0-c] l t-i$ add.commute)
finally show ? $l i \oplus l t(s e q j) \prec_{t} ? l j \oplus l t(s e q i)$.
qed
ultimately have term-pp-rel $\left(\prec_{t}\right)(l t(s e q j), ? l j) x$ using $i$-x by (rule term-pp-rel-trans-eq-right)
moreover from <ord-term-lin.is-le-rel $\left.\left(\prec_{t}\right)\right\rangle j$-k this
have term-pp-rel $\left(\prec_{t}\right)$ (lt (seq $k$ ), ?lk) $x$ by (rule term-pp-rel-trans)
ultimately show?thesis by simp
qed
with $\langle i \in$ red-set (spair $(\operatorname{seq} j)($ seq $k))\rangle$ have $i \in ? B$ using $R^{\prime} I$ by blast
thus ?thesis ..
qed
qed
moreover have finite $(? B \cup ? C)$
proof (rule finite-UnI)

```
            have finite R' unfolding R'-def
            proof (rule finite-UN-I)
            from〈finite?A〉 show finite (?A \cap {z. term-pp-rel (< }\mp@subsup{\prec}{t}{})zx})\mathrm{ by simp
            next
                    fix y
                    assume y \in?A\cap{z. term-pp-rel (}\mp@subsup{\prec}{t}{})zx
                    hence }y\in?A\mathrm{ and term-pp-rel (}\mp@subsup{\prec}{t}{})\mathrm{ y }x\mathrm{ by simp-all
                    thus finite (R y) by (rule less(2))
            qed
            show finite ?B by (intro finite-UN-I〈finite R'〉 finite-red-set)
        next
            show finite ?C by (intro finite-UN-I finite-atLeastLessThan finite-red-set)
        qed
        ultimately show ?case by (rule finite-subset)
    qed
    qed fact
    ultimately show ?thesis ..
qed
```


## 4．2．6 Concrete Rewrite Orders

```
definition is-strict-rewrite-ord :: \(\left(\left(^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\right.\) bool \()\)
\(\Rightarrow\) bool
    where \(i s\)-strict-rewrite-ord rel \(\longleftrightarrow i s\)-rewrite-ord \((\lambda x y . \neg\) rel \(y x)\)
lemma is-strict-rewrite-ordI: is-rewrite-ord \((\lambda x y . \neg\) rel \(y x) \Longrightarrow i s\)-strict-rewrite-ord
rel
    unfolding is-strict-rewrite-ord-def by blast
lemma is-strict-rewrite-ordD: is-strict-rewrite-ord rel \(\Longrightarrow\) is-rewrite-ord \((\lambda x y . \neg\)
rel \(y x\) )
    unfolding is-strict-rewrite-ord-def by blast
lemma is-strict-rewrite-ord-antisym:
    assumes is-strict-rewrite-ord rel and \(\neg\) rel \(x y\) and \(\neg\) rel \(y x\)
    shows \(f\) st \(x=f s t y\)
    by (rule is-rewrite-ordD4, rule is-strict-rewrite-ord \(D\), fact+)
lemma is-strict-rewrite-ord-asym:
    assumes is-strict-rewrite-ord rel and rel \(x y\)
    shows \(\neg\) rel \(y x\)
proof -
    from \(\operatorname{assms}(1)\) have \(i s\)-rewrite-ord \((\lambda x y . \neg\) rel \(y x)\) by (rule is-strict-rewrite-ord \(D)\)
    thus ?thesis
    proof (rule is-rewrite-ordD3)
        assume \(\neg \neg\) rel \(y x\)
        assume \(\neg\) rel \(x y\)
        thus ?thesis using 〈rel \(x y\) 〉..
    qed
```


## qed

lemma is-strict-rewrite-ord-irrefl: is-strict-rewrite-ord rel $\Longrightarrow \neg$ rel $x x$
using is-strict-rewrite-ord-asym by blast
definition rw-rat :: $\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow$ bool
where rw-rat $p q \longleftrightarrow$ (let $u=$ punit.lt $($ snd $q) \oplus$ fst $p ; v=$ punit.lt $($ snd $p) \oplus$ fst $q$ in

$$
\left.u \prec_{t} v \vee\left(u=v \wedge f_{s t} p \preceq_{t} f s t q\right)\right)
$$

definition rw-rat-strict : : $\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow$ bool
where rw-rat-strict $p q \longleftrightarrow$ (let $u=$ punit.lt $($ snd $q) \oplus$ fst $p ; v=$ punit.lt (snd $p) \oplus f s t q$ in

$$
\left.u \prec_{t} v \vee\left(u=v \wedge f s t p \prec_{t} f s t q\right)\right)
$$

definition $r w-a d d::\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow$ bool
where rw-add $p q \longleftrightarrow\left(\right.$ fst $^{p} \preceq_{t}$ fst $\left.q\right)$
definition rw-add-strict :: $\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow$ bool where $r w$-add-strict $p q \longleftrightarrow\left(\right.$ fst $p \prec_{t}$ fst $\left.q\right)$
lemma rw-rat-alt: rw-rat $=\left(\begin{array}{ll}\lambda p q . ~ & \text { rw-rat-strict } q\end{array}\right)$
by (intro ext, auto simp: rw-rat-def rw-rat-strict-def Let-def)
lemma rw-rat-is-rewrite-ord: is-rewrite-ord rw-rat
proof (rule is-rewrite-ordI)
show reflp rw-rat by (simp add: reflp-def rw-rat-def)
next
have 1: ord-term-lin.is-le-rel $\left(\prec_{t}\right)$ and 2: ord-term-lin.is-le-rel (=)
by (rule ord-term-lin.is-le-relI)+
have rw-rat p $q \longleftrightarrow$ (term-pp-rel $\left(\prec_{t}\right)($ fst $p$, punit.lt (snd $p$ )) (fst $q$, punit.lt $($ snd q) ) $\vee$

$$
(\text { term-pp-rel }(=)(f s t p, \text { punit.lt }(\text { snd } p))(f \text { st } q, \text { punit.lt }(\text { snd } q))
$$

$\wedge$

$$
\left.\left.f_{s t} p \preceq_{t} f s t q\right)\right)
$$

for $p q$
by (simp add: rw-rat-def term-pp-rel-def Let-def)
thus transp rw-rat
by (auto simp: transp-def dest: term-pp-rel-trans[OF 1] term-pp-rel-trans-eq-left[OF 1]
term-pp-rel-trans-eq-right[OF 1] term-pp-rel-trans[OF 2])
next
fix $p q$
show rw-rat $p q \vee r w-r a t q u$ by (auto simp: rw-rat-def Let-def)
next
fix $p q$
assume rw-rat $p q$ and $r w$-rat $q p$
thus $f$ st $p=f s t q$ by (auto simp: rw-rat-def Let-def)
next
fix $d G p q$
assume $d$ : dickson-grading $d$ and $g b:$ is-sig-GB-upt $d G(l t q)$ and $p \in G$ and $q \in G$
and $p \neq 0$ and $q \neq 0$ and $l t p a d d s_{t}$ lt $q$ and $\neg i s$-sig-red $\left(\prec_{t}\right)(=) G q$
let $? u=$ punit.lt $($ rep-list $q) \oplus$ lt $p$
let $? v=$ punit.lt (rep-list $p) \oplus l t q$
from $\left\langle l t p\right.$ adds $\left.s_{t} l t q\right\rangle$ obtain $t$ where $l t-q: l t ~ q=t \oplus l t p$ by (rule adds-termE)
from $g b$ have $G \subseteq d g r a d$-sig-set $d$ by (rule is-sig-GB-uptD1)
hence $G \subseteq d g r a d-m a x-s e t ~ d ~ b y ~(s i m p ~ a d d: ~ d g r a d-s i g-s e t '-d e f)$
with $d$ obtain $p^{\prime}$ where red: $\left(\text { sig-red }\left(\prec_{t}\right)(=) G\right)^{* *}($ monom-mult $1 t p) p^{\prime}$
and $\neg i s$-sig-red $\left(\prec_{t}\right)(=) G p^{\prime}$ by (rule sig-irredE-dgrad-max-set)
from red have $l t p^{\prime}=l t($ monom-mult $1 t p)$ and $l c p^{\prime}=l c($ monom-mult $1 t p)$ and 2: punit.lt (rep-list $\left.p^{\prime}\right) \preceq$ punit.lt (rep-list (monom-mult 1 t p))
by (rule sig-red-regular-rtrancl-lt, rule sig-red-regular-rtrancl-lc, rule sig-red-rtrancl-lt-rep-list)
with $\langle p \neq 0\rangle$ have $l t p^{\prime}=l t q$ and $l c p^{\prime}=l c p$ by (simp-all add: lt-q lt-monom-mult)
from 2 punit.lt-monom-mult-le[simplified] have 3: punit.lt (rep-list $p^{\prime}$ ) $\preceq t+$ punit.lt (rep-list p)
unfolding rep-list-monom-mult by (rule ordered-powerprod-lin.order-trans)
have punit.lt (rep-list $\left.p^{\prime}\right)=$ punit.lt $($ rep-list q)
proof (rule sig-regular-top-reduced-lt-unique)
show $p^{\prime} \in d g r a d-s i g-s e t ~ d$
proof (rule dgrad-sig-set-closed-sig-red-rtrancl)
note $d$
moreover have $d t \leq d g r a d-m a x d$
proof (rule le-trans)
have $t$ adds $l p q$ by (simp add: lt- $q$ term-simps)
with $d$ show $d t \leq d$ (lp $q$ ) by (rule dickson-grading-adds-imp-le)
next
from $\langle q \in G\rangle\langle G \subseteq$ dgrad-max-set $d\rangle$ have $q \in$ dgrad-max-set $d .$.
thus $d(l p q) \leq d g r a d-m a x d$ using $\langle q \neq 0\rangle$ by (rule dgrad- $p$-setD-lp)
qed
moreover from $\langle p \in G\rangle\langle G \subseteq d g r a d$-sig-set $d\rangle$ have $p \in$ dgrad-sig-set $d$..
ultimately show monom-mult 1 t $p \in d g r a d$-sig-set $d$ by (rule dgrad-sig-set-closed-monom-mult)
qed fact+
next
from $\langle q \in G\rangle\langle G \subseteq d g r a d$-sig-set $d\rangle$ show $q \in$ dgrad-sig-set $d .$.
next
from $\langle p \neq 0\rangle\left\langle l c p^{\prime}=l c p\right\rangle$ show $p^{\prime} \neq 0$ by (auto simp: lc-eq-zero-iff)
qed fact+
with 3 have punit.lt (rep-list $q$ ) $\preceq t+$ punit.lt (rep-list p) by simp
hence ? $u \preceq_{t}(t+$ punit.lt (rep-list $\left.p)\right) \oplus l t p$ by (rule splus-mono-left)
also have $\ldots=$ ?v by (simp add: lt-q splus-assoc splus-left-commute)
finally have ? $u \preceq_{t}$ ? $v$ by (simp only: rel-def)
moreover from $\left\langle l t p a d d s_{t} l t q\right\rangle$ have $l t p \preceq_{t} l t q$ by (rule ord-adds-term)
ultimately show rw-rat (spp-of p) (spp-of q) by (auto simp: rw-rat-def Let-def
spp-of-def)
qed
lemma rw-rat-strict-is-strict-rewrite-ord: is-strict-rewrite-ord rw-rat-strict

```
proof (rule is-strict-rewrite-ordI)
    show is-rewrite-ord ( }\lambdaxy.\neg\mathrm{ rw-rat-strict y x)
        unfolding rw-rat-alt[symmetric] by (fact rw-rat-is-rewrite-ord)
qed
lemma rw-add-alt:rw-add = ( }\lambdap\mathrm{ q. }\neg rw-add-strict q p)
    by (intro ext, auto simp: rw-add-def rw-add-strict-def)
lemma rw-add-is-rewrite-ord: is-rewrite-ord rw-add
proof (rule is-rewrite-ordI)
    show reflp rw-add by (simp add: reflp-def rw-add-def)
next
    show transp rw-add by (auto simp: transp-def rw-add-def)
next
    fix pq
    show rw-add p q\vee rw-add q p by (simp only: rw-add-def ord-term-lin.linear)
next
    fix pq
    assume rw-add pq and rw-add q p
    thus fst p=fst q unfolding rw-add-def
            by simp
next
    fix p q:: 't }\mp@subsup{=>}{0}{\prime}\mp@subsup{}{}{\prime}
    assume lt p addst lt q
    thus rw-add (spp-of p) (spp-of q) unfolding rw-add-def spp-of-def fst-conv by
(rule ord-adds-term)
qed
lemma rw-add-strict-is-strict-rewrite-ord: is-strict-rewrite-ord rw-add-strict
proof (rule is-strict-rewrite-ordI)
    show is-rewrite-ord ( }\lambdaxy.\neg rw-add-strict y x)
    unfolding rw-add-alt[symmetric] by (fact rw-add-is-rewrite-ord)
qed
```


### 4.2.7 Preparations for Sig-Poly-Pairs

```
context
fixes dgrad \(::{ }^{\prime} a \Rightarrow n a t\)
begin
definition spp-rel :: \(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\) bool
where spp-rel sp \(r \longleftrightarrow(r \neq 0 \wedge r \in\) dgrad-sig-set dgrad \(\wedge l t r=f s t ~ s p \wedge r e p-l i s t\) \(r=s n d s p\) )
definition spp-inv :: ('t \(\left.\times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\) bool
where \(s p p-i n v s p \longleftrightarrow E x\) (spp-rel \(s p\) )
definition vec-of :: ( \(\left.' t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\)
where vec-of \(s p=(\) if \(s p p-i n v\) sp then Eps (spp-rel sp) else 0)
```

```
lemma spp-inv-spp-of:
    assumes r\not=0 and r\indgrad-sig-set dgrad
    shows spp-inv (spp-of r)
    unfolding spp-inv-def spp-rel-def
proof (intro exI conjI)
    show lt r = fst (spp-of r) by (simp add: spp-of-def)
next
    show rep-list r = snd (spp-of r) by (simp add: spp-of-def)
qed fact+
context
    fixes sp :: 't > (' }a\not=
    assumes spi: spp-inv sp
begin
lemma sig-poly-rel-vec-of: spp-rel sp (vec-of sp)
proof -
    from spi have eq: vec-of sp = Eps (spp-rel sp) by (simp add: vec-of-def)
    from spi show ?thesis unfolding eq spp-inv-def by (rule someI-ex)
qed
lemma vec-of-nonzero: vec-of sp}\not=
    using sig-poly-rel-vec-of by (simp add: spp-rel-def)
lemma lt-vec-of:lt (vec-of sp) = fst sp
    using sig-poly-rel-vec-of by (simp add: spp-rel-def)
lemma rep-list-vec-of:rep-list (vec-of sp) = snd sp
    using sig-poly-rel-vec-of by (simp add: spp-rel-def)
lemma spp-of-vec-of: spp-of (vec-of sp) = sp
    by (simp add: spp-of-def lt-vec-of rep-list-vec-of)
end
lemma map-spp-of-vec-of:
    assumes list-all spp-inv sps
    shows map (spp-of ○ vec-of) sps = sps
proof (rule map-idI)
    fix sp
    assume sp \in set sps
    with assms have spp-inv sp by (simp add: list-all-def)
    hence spp-of (vec-of sp) = sp by (rule spp-of-vec-of)
    thus (spp-of ○ vec-of) sp = sp by simp
qed
lemma vec-of-dgrad-sig-set: vec-of sp \in dgrad-sig-set dgrad
proof (cases spp-inv sp)
```

```
    case True
    hence spp-rel sp (vec-of sp) by (rule sig-poly-rel-vec-of)
    thus ?thesis by (simp add: spp-rel-def)
next
    case False
    moreover have 0 f dgrad-sig-set dgrad unfolding dgrad-sig-set'-def
    proof
        show 0 \in dgrad-max-set dgrad by (rule dgrad-p-setI) simp
    next
        show 0 \in sig-inv-set by (rule sig-inv-setI) (simp add: term-simps)
    qed
    ultimately show ?thesis by (simp add: vec-of-def)
qed
lemma spp-invD-fst:
    assumes spp-inv sp
    shows dgrad (pp-of-term (fst sp)) \leq dgrad-max dgrad and component-of-term
(fst sp) < length fs
proof -
    from vec-of-dgrad-sig-set have dgrad (lp (vec-of sp)) \leqdgrad-max dgrad by (rule
dgrad-sig-setD-lp)
    with assms show dgrad (pp-of-term (fst sp)) \leq dgrad-max dgrad by (simp add:
lt-vec-of)
    from vec-of-dgrad-sig-set vec-of-nonzero[OF assms] have component-of-term (lt
(vec-of sp)) < length fs
    by (rule dgrad-sig-setD-lt)
    with assms show component-of-term (fst sp) < length fs by (simp add:lt-vec-of)
qed
lemma spp-invD-snd:
    assumes dickson-grading dgrad and spp-inv sp
    shows snd sp \in punit-dgrad-max-set dgrad
proof -
    from vec-of-dgrad-sig-set[of sp] have vec-of sp \in dgrad-max-set dgrad by (simp
add: dgrad-sig-set'-def)
    with assms(1) have rep-list (vec-of sp) \in punit-dgrad-max-set dgrad by (rule
dgrad-max-2)
    with assms(2) show ?thesis by (simp add: rep-list-vec-of)
qed
lemma vec-of-inj:
    assumes spp-inv sp and vec-of sp = vec-of sp'
    shows }sp=s\mp@subsup{p}{}{\prime
proof -
    from assms(1) have vec-of sp}\not=0\mathrm{ by (rule vec-of-nonzero)
    hence vec-of sp
    hence spp-inv sp' by (simp add: vec-of-def split: if-split-asm)
    from assms(1) have sp = spp-of (vec-of sp) by (simp only: spp-of-vec-of)
    also have ... = spp-of (vec-of sp') by (simp only:assms(2))
```

```
    also from 〈spp-inv sp'〉 have ... = sp' by (rule spp-of-vec-of)
    finally show ?thesis.
qed
lemma spp-inv-alt: spp-inv sp}\longleftrightarrow(vec-of sp\not=0
proof -
    have spp-inv sp if vec-of sp}\not=
    proof (rule ccontr)
        assume }\neg spp-inv s
        hence vec-of sp = 0 by (simp add: vec-of-def)
        with that show False ..
    qed
    thus ?thesis by (auto dest: vec-of-nonzero)
qed
lemma spp-of-vec-of-spp-of:
    assumes p d dgrad-sig-set dgrad
    shows spp-of (vec-of (spp-of p))}=\mathrm{ spp-of p
proof (cases p=0)
    case True
    show ?thesis
    proof (cases spp-inv (spp-of p))
        case True
        thus ?thesis by (rule spp-of-vec-of)
    next
        case False
        hence vec-of (spp-of p)=0 by (simp add: spp-inv-alt)
        thus ?thesis by (simp only: True)
    qed
next
    case False
    have spp-inv (spp-of p) unfolding spp-inv-def
    proof
    from False assms show spp-rel (spp-of p) p by (simp add: spp-rel-def spp-of-def)
    qed
    thus ?thesis by (rule spp-of-vec-of)
qed
```


### 4.2.8 Total Reduction

```
primrec find-sig-reducer : : ('t \(\left.\times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\) list \(\Rightarrow^{\prime} t \Rightarrow{ }^{\prime} a \Rightarrow\) nat \(\Rightarrow\) nat option where
    find-sig-reducer [] -- = None |
    find-sig-reducer (b # bs) u ti=
        (if snd b}\not=0\wedge punit.lt (snd b) adds t\wedge(t-punit.lt (snd b)) \oplus fst b \prec. <
u then Some i
    else find-sig-reducer bs u t (Suc i))
lemma find-sig-reducer-SomeD-aux:
```

```
    assumes find-sig-reducer bs u t i=Some j
    shows }i\leqj\mathrm{ and j -i< length bs
proof -
    from assms have i\leqj^j-i< length bs
    proof (induct bs arbitrary: i)
        case Nil
        thus ?case by simp
    next
        case (Cons b bs)
        from Cons(2) show ?case
        proof (simp split: if-split-asm)
            assume find-sig-reducer bs u t (Suc i)=Some j
            hence Suc i\leqj^j-Suc i< length bs by (rule Cons(1))
            thus i\leqj^j-i<Suc (length bs) by auto
    qed
    qed
    thus i\leqj and j-i< length bs by simp-all
qed
lemma find-sig-reducer-SomeD':
    assumes find-sig-reducer bs u ti=Some j and b=bs! (j - i)
    shows b\in set bs and snd b}=0\mathrm{ and punit.lt (snd b) adds t and (t - punit.lt
(snd b)) }\oplus\mathrm{ fst b < }\mp@subsup{}{t}{}
proof -
    from assms(1) have j-i< length bs by (rule find-sig-reducer-SomeD-aux)
    thus b\in set bs unfolding assms(2) by (rule nth-mem)
next
    from assms have snd b}=0\wedge punit.lt (snd b) adds t ^(t - punit.lt (snd b)) 
fst b}\mp@subsup{\prec}{t}{}
    proof (induct bs arbitrary: i)
        case Nil
        from Nil(1) show ?case by simp
    next
        case (Cons a bs)
        from Cons(2) show ?case
        proof (simp split: if-split-asm)
            assume i=j
            with Cons(3) have b=a by simp
            moreover assume snd a\not=0 and punit.lt (snd a) adds t and (t - punit.lt
(snd a)) \oplus fst a \prec}\mp@subsup{t}{}{u
            ultimately show ?case by simp
    next
            assume *: find-sig-reducer bs u t (Suc i)=Some j
            hence Suc i\leqj by (rule find-sig-reducer-SomeD-aux)
            note Cons(3)
            also from <Suc i\leqj` have (a# #s)! (j - i)=bs! (j - Suc i) by simp
            finally have b=bs! (j-Suc i).
            with * show ?case by (rule Cons(1))
            qed
```

qed
thus snd $b \neq 0$ and punit.lt (snd b) adds $t$ and $(t-p u n i t . l t($ snd $b)) \oplus$ fst $b$ $\prec_{t} u$ by simp-all
qed
corollary find-sig-reducer-SomeD:
assumes find-sig-reducer (map spp-of bs) ut $0=$ Some $i$
shows $i<$ length bs and rep-list (bs $!i) \neq 0$ and punit.lt (rep-list (bs $!i)$ ) adds $t$
and $(t-$ punit.lt $(r e p-l i s t(b s!i))) \oplus l t(b s!i) \prec_{t} u$
proof -
from assms have $i-0<$ length (map spp-of bs) by (rule find-sig-reducer-SomeD-aux)
thus $i<$ length bs by simp
hence spp-of $(b s!i)=($ map spp-of bs $)!(i-0)$ by simp
with assms have snd $(\operatorname{spp-of}(b s!i)) \neq 0$ and punit.lt (snd $(\operatorname{spp-of}(b s!i)))$
adds t
and $(t$ - punit.lt $($ snd $($ spp-of $(b s!i)))) \oplus f s t($ spp-of $(b s!i)) \prec_{t} u$
by (rule find-sig-reducer-Some $D^{\prime}$ )+
thus rep-list $(b s!i) \neq 0$ and punit.lt (rep-list $(b s!i))$ adds $t$
and $(t-$ punit.lt $(r e p-l i s t(b s!i))) \oplus l t(b s!i) \prec_{t} u$ by (simp-all add: fst-spp-of snd-spp-of)
qed
lemma find-sig-reducer-NoneE:
assumes find-sig-reducer bs $u t i=N o n e$ and $b \in$ set $b s$
assumes snd $b=0 \Longrightarrow$ thesis and snd $b \neq 0 \Longrightarrow \neg$ punit.lt (snd b) adds $t \Longrightarrow$ thesis
and snd $b \neq 0 \Longrightarrow$ punit.lt $($ snd $b)$ adds $t \Longrightarrow \neg(t-$ punit.lt $($ snd $b)) \oplus$ fst $b$ $\prec_{t} u \Longrightarrow$ thesis
shows thesis
using assms
proof (induct bs arbitrary: thesis $i$ )
case Nil
from $\operatorname{Nil(2)}$ show ?case by simp
next
case (Cons a bs)
from Cons(2) have 1: snd $a=0 \vee \neg$ punit.lt (snd a) adds $t \vee \neg(t-$ punit.lt
$(s n d a)) \oplus f s t a \prec_{t} u$
and eq: find-sig-reducer bs ut (Suc $i)=$ None by (simp-all split: if-splits)
from Cons(3) have $b=a \vee b \in$ set bs by simp
thus? case
proof
assume $b=a$
show ?thesis
proof (cases snd $a=0$ )
case True
show ?thesis by (rule Cons(4), simp add: $\langle b=a\rangle$ True)
next
case False
with 1 have 2: $\neg$ punit.lt $($ snd $a)$ adds $t \vee \neg(t-$ punit.lt $($ snd $a)) \oplus$ fst $a$ $\prec_{t} u$ by $\operatorname{simp}$
show ?thesis
proof (cases punit.lt (snd a) adds t)
case True
with 2 have 3: $\neg(t-$ punit.lt $($ snd $a)) \oplus f s t a \prec_{t} u$ by simp
show ?thesis by (rule Cons(6), simp-all add: $\langle b=a\rangle\langle s n d a \neq 0\rangle$ True 3)
next
case False
show ?thesis by (rule Cons(5), simp-all add: $\langle b=a\rangle\langle s n d a \neq 0\rangle$ False)
qed
qed
next
assume $b \in$ set $b s$
with eq show ?thesis
proof (rule Cons(1))
assume snd $b=0$
thus ?thesis by (rule Cons(4))
next
assume snd $b \neq 0$ and $\neg$ punit.lt (snd b) adds $t$
thus ?thesis by (rule Cons(5))
next
assume snd $b \neq 0$ and punit.lt (snd b) adds $t$ and $\neg(t-$ punit.lt (snd b))
$\oplus f s t b \prec_{t} u$
thus ?thesis by (rule $\operatorname{Cons}(6)$ )
qed
qed
qed
lemma find-sig-reducer-SomeD-red-single:
assumes $t \in$ keys (rep-list p) and find-sig-reducer (map spp-of bs) (lt p) t $0=$ Some $i$
shows sig-red-single $\left(\prec_{t}\right)(\preceq) p(p-$ monom-mult (lookup (rep-list $p) t / p u n i t . l c$ (rep-list $(b s!i))$ ) $(t-$ punit.lt $(r e p-l i s t(b s!i)))(b s!i))(b s!i)(t-p u n i t . l t(r e p-l i s t(b s$ ! i) ))
proof -
from assms(2) have punit.lt (rep-list (bs!i)) adds $t$ and 1: rep-list (bs!i) $\neq$ 0
and 2: $(t$ - punit.lt $($ rep-list $(b s!i))) \oplus l t(b s!i) \prec_{t} l t p$
by (rule find-sig-reducer-SomeD)+
from this(1) have eq: $t$ - punit.lt (rep-list (bs!i)) + punit.lt (rep-list (bs!i)) $=t$
by (rule adds-minus)
from $\operatorname{assms}(1)$ have 3: $t \preceq$ punit.lt (rep-list p) by (rule punit.lt-max-keys)
show ?thesis by (rule sig-red-singleI, simp-all add: eq 123 assms(1))
qed
corollary find-sig-reducer-SomeD-red:
assumes $t \in$ keys (rep-list p) and find-sig-reducer (map spp-of bs) (lt p) to= Some $i$
shows sig-red $\left(\prec_{t}\right)(\preceq)($ set bs) $p(p-$ monom-mult (lookup (rep-list p) $t /$ punit.lc (rep-list (bs!i))) $(t-$ punit.lt $($ rep-list $(b s!i)))(b s!i))$
unfolding sig-red-def
proof (intro bexI exI, rule find-sig-reducer-SomeD-red-single)
from $\operatorname{assms}(2)$ have $i-0<$ length (map spp-of bs) by (rule find-sig-reducer-SomeD-aux)
hence $i<$ length bs by simp
thus $b s!i \in$ set bs by (rule nth-mem)
qed fact+

## context

fixes bs :: ( $\left.{ }^{\prime} t \nRightarrow_{0}{ }^{\prime} b\right)$ list
begin
definition sig-trd-term :: ('a $\Rightarrow$ nat $) \Rightarrow\left(\left({ }^{\prime} a \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} a \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)$ set
where sig-trd-term $d=\{(x, y)$. punit.dgrad-p-set-le d \{rep-list (snd $x)\}$
(insert (rep-list (snd y)) (rep-list' set bs)) $\wedge$
fst $x \in$ keys (rep-list $($ snd $x)) \wedge$ fst $y \in$ keys (rep-list
$($ snd $y)) \wedge$

$$
\text { fst } x \prec f s t y\}
$$

lemma sig-trd-term-wf:
assumes dickson-grading $d$
shows wf (sig-trd-term d)
proof (rule wfI-min)
fix $x::{ }^{\prime} a \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ and $Q$
assume $x \in Q$
show $\exists z \in Q . \forall y .(y, z) \in$ sig-trd-term $d \longrightarrow y \notin Q$
proof (cases fst $x \in$ keys (rep-list (snd $x)$ ))
case True
define $X$ where $X=$ rep-list'set bs
let ? $A=$ insert $($ rep-list $($ snd $x)) X$
have finite $X$ unfolding $X$-def by simp
hence finite ? A by (simp only: finite-insert)
then obtain $m$ where $A: ? A \subseteq$ punit.dgrad-p-set $d m$ by (rule punit.dgrad-p-set-exhaust)
hence $x$ : rep-list $($ snd $x) \in$ punit.dgrad- $p$-set $d m$ and $X: X \subseteq$ punit.dgrad-p-set

## d $m$

by simp-all
let ? $Q=\{q \in Q$. rep-list $($ snd $q) \in$ punit.dgrad- $p$-set $d m \wedge f$ st $q \in$ keys (rep-list (snd q)) \}
from $\langle x \in Q\rangle x$ True have $x \in ? Q$ by simp
have $\forall Q x . x \in Q \wedge Q \subseteq\{q . d q \leq m\} \longrightarrow(\exists z \in Q . \forall y . y \prec z \longrightarrow y \notin Q)$
by (rule wfp-on-imp-minimal, rule wfp-on-ord-strict, fact assms)
hence 1: fst $x \in f_{s t}$ ' ? $Q \Longrightarrow f s t$ ' ? $Q \subseteq\{q . d q \leq m\} \Longrightarrow(\exists z \in f s t$ ' ? $Q . \forall y$.
$y \prec z \longrightarrow y \notin f s t$ '? $Q$ )
by meson

```
    have fst x\in fst'?Q by (rule, fact refl, fact)
    moreover have fst'?Q\subseteq{q. d q}\leqm
    proof -
    {
        fix q
            assume a: rep-list (snd q) \in punit.dgrad-p-set d m and b: fst q \in keys
(rep-list (snd q))
            from a have keys (rep-list (snd q)) \subseteq dgrad-set d m by (simp add:
punit.dgrad-p-set-def)
            with b have fst q \in dgrad-set d m ..
            hence d (fst q) \leqm by (simp add:dgrad-set-def)
        }
        thus ?thesis by auto
    qed
    ultimately have }\existsz\infst`?Q. \forally.y\precz\longrightarrowy\not\infst'?Q by (rule 1
    then obtain z0 where z0\infst'?Q and 2: \y. y\precz0\Longrightarrowy\not\infst'?Q by
blast
    from this(1) obtain z}\mathrm{ where z ? ?Q and z0:z0 = fst z ..
    hence z}\inQ\mathrm{ and z: rep-list (snd z) f punit.dgrad-p-set d m by simp-all
    from this(1) show }\existsz\inQ.\forally.(y,z)\in\mathrm{ sig-trd-term d }\longrightarrowy\not\in
    proof
        show }\forally.(y,z)\in\mathrm{ sig-trd-term d }\longrightarrowy\not\in
        proof (intro allI impI)
            fix }
            assume (y,z)\in sig-trd-term d
            hence 3: punit.dgrad-p-set-le d {rep-list (snd y)} (insert (rep-list (snd z))
X)
            and 4: fst }y\in\mathrm{ keys (rep-list (snd y)) and fst y}\precz
            by (simp-all add: sig-trd-term-def X-def z0)
            from this(3) have fst y \not\infst'??Q by (rule 2)
            hence y}\not\inQ\vee rep-list (snd y) \not二 punit.dgrad-p-set d m\vee fst y & key
(rep-list (snd y))
            by auto
            thus y\not\inQ
            proof (elim disjE)
            assume 5: rep-list (snd y)\not\in punit.dgrad-p-set d m
                    from z X have insert (rep-list (snd z)) X\subseteq punit.dgrad-p-set d m by
simp
                    with }3\mathrm{ have {rep-list (snd y)}}\subseteq\mathrm{ punit.dgrad-p-set d m by (rule
punit.dgrad-p-set-le-dgrad-p-set)
            hence rep-list (snd y) \in punit.dgrad-p-set d m by simp
            with 5 show ?thesis ..
            next
                assume fst y & keys (rep-list (snd y))
            thus ?thesis using 4 ..
        qed
    qed
    qed
```

```
    next
        case False
        from }\langlex\inQ\rangle\mathrm{ show ?thesis
        proof
        show }\forally.(y,x)\in\mathrm{ sig-trd-term d }\longrightarrowy\not\in
        proof (intro allI impI)
            fix }
            assume (y,x)\in sig-trd-term d
            hence fst }x\in\mathrm{ keys (rep-list (snd x)) by (simp add: sig-trd-term-def)
            with False show y }\not\inQ.
        qed
    qed
    qed
qed
```



```
    sig-trd-aux (t, p)=
        (let p' =
            (case find-sig-reducer (map spp-of bs) (lt p) t 0 of
                None }=>
            | Some i m p - monom-mult (lookup (rep-list p) t | punit.lc (rep-list (bs !
i)))
                                    (t - punit.lt (rep-list (bs!i))) (bs!i));
        p\prime\prime}=\mathrm{ punit.lower (rep-list p')t in
    if p}\mp@subsup{p}{}{\prime\prime}=0\mathrm{ then p' else sig-trd-aux (punit.lt p',}\mp@subsup{p}{}{\prime})
    by auto
lemma sig-trd-aux-domI:
    assumes fst args0 \in keys (rep-list (snd argsO))
    shows sig-trd-aux-dom args0
proof -
    from ex-hgrad obtain d::'a m nat where dickson-grading d ^hom-grading d ..
    hence dg: dickson-grading d ..
    hence wf (sig-trd-term d) by (rule sig-trd-term-wf)
    thus ?thesis using assms
    proof (induct args0)
        case (less args)
        obtain t p where args: args = (t,p) using prod.exhaust by blast
    with less(1) have 1: \bigwedgesq. ((s,q),(t,p)) \in sig-trd-term d\Longrightarrows\in keys (rep-list
q)\Longrightarrow sig-trd-aux-dom (s,q)
        using prod.exhaust by auto
    from less(2) have t\in keys (rep-list p) by (simp add: args)
    show ?case unfolding args
    proof (rule sig-trd-aux.domintros)
        define }\mp@subsup{p}{}{\prime}\mathrm{ where }\mp@subsup{p}{}{\prime}=(\mathrm{ case find-sig-reducer (map spp-of bs) (lt p) t 0 of
                        None }=>
                                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))
```

define $p^{\prime \prime}$ where $p^{\prime \prime}=$ punit.lower (rep-list $p^{\prime}$ ) $t$
assume $p^{\prime \prime} \neq 0$
from $\left\langle p^{\prime \prime} \neq 0\right\rangle$ have punit.lt $p^{\prime \prime} \in$ keys $p^{\prime \prime}$ by (rule punit.lt-in-keys)
also have $\ldots \subseteq$ keys (rep-list $p^{\prime}$ ) by (auto simp: $p^{\prime \prime}$-def punit.keys-lower)
finally have punit.lt $p^{\prime \prime} \in$ keys (rep-list $p^{\prime}$ ).
with - show sig-trd-aux-dom (punit.lt $p^{\prime \prime}, p^{\prime}$ )
proof (rule 1)
have punit.dgrad-p-set-le d \{rep-list p'\} (insert (rep-list p) (rep-list'set bs))
proof (cases find-sig-reducer (map spp-of bs) (lt p) t 0)
case None
hence $p^{\prime}=p$ by (simp add: $p^{\prime}$-def)
hence $\left\{\right.$ rep-list $\left.p^{\prime}\right\} \subseteq$ insert (rep-list $p$ ) (rep-list'set bs) by simp
thus ?thesis by (rule punit.dgrad-p-set-le-subset)
next
case (Some $i$ )
hence $p^{\prime}: p^{\prime}=p$ - monom-mult (lookup (rep-list p) t/punit.lc (rep-list (bs!i)))
$(t-$ punit.lt $(r e p-l i s t(b s!i)))(b s!i)$ by (simp add:
$\left.p^{\prime}-d e f\right)$
have sig-red $\left(\prec_{t}\right)(\preceq)\left(\right.$ set bs) p $p^{\prime}$ unfolding $p^{\prime}$ using $\langle t \in$ keys (rep-list p)> Some
by (rule find-sig-reducer-SomeD-red)
hence punit.red (rep-list'set bs) (rep-list p) (rep-list p') by (rule sig-red-red)
with $d g$ show ?thesis by (rule punit.dgrad-p-set-le-red)
qed
moreover note $\left\langle\right.$ punit.lt $p^{\prime \prime} \in$ keys (rep-list $\left.\left.p^{\prime}\right)\right\rangle\langle t \in$ keys (rep-list p) $\rangle$
moreover from $\left\langle p^{\prime \prime} \neq 0\right\rangle$ have punit.lt $p^{\prime \prime} \prec t$ unfolding $p^{\prime \prime}$-def by (rule punit.lt-lower-less)
ultimately show ((punit.lt $\left.\left.p^{\prime \prime}, p^{\prime}\right), t, p\right) \in$ sig-trd-term $d$ by (simp add: sig-trd-term-def)
qed
qed
qed
qed
definition sig-trd :: (' $\left.t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$
where sig-trd $p=($ if rep-list $p=0$ then $p$ else sig-trd-aux (punit.lt (rep-list $p)$,
p))
lemma sig-trd-aux-red-rtrancl:
assumes fst args0 $\in$ keys (rep-list (snd args0))
shows $\left(\text { sig-red }\left(\prec_{t}\right)(\preceq)(\text { set bs })\right)^{* *}($ snd args0) $($ sig-trd-aux args0)
proof -
from assms have sig-trd-aux-dom args0 by (rule sig-trd-aux-domI)
thus ?thesis using assms
proof (induct args0 rule: sig-trd-aux.pinduct)
case (1tp)
define $p^{\prime}$ where $p^{\prime}=($ case find-sig-reducer (map spp-of bs) (lt p) t 0 of

$$
\begin{aligned}
& \text { None } \Rightarrow p \\
& \mid \text { Some } i \Rightarrow p- \\
& \text { monom-mult (lookup (rep-list p) } t / \text { punit.lc (rep-list (bs }
\end{aligned}
$$

! i) )
$(t-$ punit.lt $($ rep-list $(b s!i)))(b s!i))$
define $p^{\prime \prime}$ where $p^{\prime \prime}=$ punit.lower (rep-list $p^{\prime}$ ) $t$
from 1(3) have $t \in$ keys (rep-list $p$ ) by simp
have $*:\left(\operatorname{sig} \text {-red }\left(\prec_{t}\right)(\preceq)(\text { set bs) })\right)^{* *} p p^{\prime}$
proof (cases find-sig-reducer (map spp-of bs) (lt p) t 0)
case None
hence $p^{\prime}=p$ by (simp add: $p^{\prime}$-def)
thus ?thesis by simp
next
case (Some i)
hence $p^{\prime}: p^{\prime}=p$ - monom-mult (lookup (rep-list p) $t /$ punit.lc (rep-list (bs ! $i)$ )
$(t-$ punit.lt $($ rep-list $(b s!i)))(b s!i)$ by $(s i m p$ add $:$
$\left.p^{\prime}-d e f\right)$
have sig-red $\left(\prec_{t}\right)(\preceq)\left(\right.$ set bs) $p p^{\prime}$ unfolding $p^{\prime}$ using $\langle t \in$ keys (rep-list $p$ ) >
Some
by (rule find-sig-reducer-SomeD-red)
thus ?thesis ..
qed
show ?case
proof (simp add: sig-trd-aux.psimps[OF 1(1)] Let-def $p^{\prime}$-def[symmetric] $p^{\prime \prime}$-def[symmetric] *, intro impI)
assume $p^{\prime \prime} \neq 0$
from $*$ have $\left(\text { sig-red }\left(\prec_{t}\right)(\preceq)(\text { set bs })\right)^{* *} p\left(\right.$ snd $\left(\right.$ punit.lt $\left.\left.p^{\prime \prime}, p^{\prime}\right)\right)$ by (simp only: snd-conv)
moreover have $\left(\operatorname{sig}-r e d\left(\prec_{t}\right)(\preceq)(\text { set bs) })\right)^{* *}\left(\right.$ snd $\left(\right.$ punit.lt $\left.\left.p^{\prime \prime}, p^{\prime}\right)\right)($ sig-trd-aux (punit.lt $\left.p^{\prime \prime}, p^{\prime}\right)$ )
using $p^{\prime}$-def $p^{\prime \prime}$-def $\left\langle p^{\prime \prime} \neq 0\right\rangle$
proof (rule 1(2))
from $\left\langle p^{\prime \prime} \neq 0\right\rangle$ have punit.lt $p^{\prime \prime} \in$ keys $p^{\prime \prime}$ by (rule punit.lt-in-keys)
also have $\ldots \subseteq$ keys (rep-list $p^{\prime}$ ) by (auto simp: $p^{\prime \prime}$-def punit.keys-lower)
finally show $f$ st (punit.lt $\left.p^{\prime \prime}, p^{\prime}\right) \in$ keys (rep-list (snd (punit.lt $\left.\left.p^{\prime \prime}, p^{\prime}\right)\right)$ ) by
simp
qed
ultimately show $\left(\text { sig-red }\left(\prec_{t}\right)(\preceq)(\text { set bs })\right)^{* *} p\left(\right.$ sig-trd-aux $\left(\right.$ punit.lt $\left.\left.p^{\prime \prime}, p^{\prime}\right)\right)$
by (rule rtranclp-trans)
qed
qed
qed
corollary sig-trd-red-rtrancl: $\left(\text { sig-red }\left(\prec_{t}\right)(\preceq)(\text { set bs })\right)^{* *} p($ sig-trd $p)$
unfolding sig-trd-def
proof (split if-split, intro conjI impI rtranclp.rtrancl-refl)
let ?args $=($ punit.lt $($ rep-list $p), p)$
assume rep-list $p \neq 0$
hence punit.lt (rep-list p) keys (rep-list p) by (rule punit.lt-in-keys)
hence fst (punit.lt (rep-list p), p) keys (rep-list (snd (punit.lt (rep-list p), p)))
by (simp only: fst-conv snd-conv)
hence $\left(\right.$ sig-red $\left(\prec_{t}\right)(\preceq)(\text { set bs))})^{* *}$ (snd ?args) (sig-trd-aux ?args) by (rule sig-trd-aux-red-rtrancl)
thus $\left(\right.$ sig-red $\left(\prec_{t}\right)(\preceq)($ set bs) $){ }^{* *} p($ sig-trd-aux $($ punit.lt $($ rep-list $p), p))$ by (simp only: snd-conv)
qed
lemma sig-trd-aux-irred:
assumes fst args0 $\in$ keys (rep-list (snd args0))
and $\bigwedge b s . b \in$ set $b s \Longrightarrow$ rep-list $b \neq 0 \Longrightarrow$ fst args0 $\prec s+$ punit.lt (rep-list b) $\Longrightarrow$

$$
s \oplus l t b \prec_{t} l t(\operatorname{snd}(\operatorname{args} 0)) \Longrightarrow \text { lookup }(\text { rep-list }(\text { snd } \operatorname{args} 0))(s+
$$ punit.lt (rep-list b)) $=0$

shows $\neg$ is-sig-red $\left(\prec_{t}\right)(\preceq)($ set bs) (sig-trd-aux args0)
proof -
from $\operatorname{assms}(1)$ have sig-trd-aux-dom args0 by (rule sig-trd-aux-domI)
thus ?thesis using assms
proof (induct args0 rule: sig-trd-aux.pinduct)
case (1tp)
define $p^{\prime}$ where $p^{\prime}=($ case find-sig-reducer (map spp-of bs) (lt p) t 0 of
None $\Rightarrow p$
| Some $i \Rightarrow p-$ monom-mult (lookup (rep-list p) $t /$ punit.lc (rep-list (bs
! i) )

$$
(t-\text { punit.lt }(\text { rep-list }(b s!i)))(b s!i))
$$

define $p^{\prime \prime}$ where $p^{\prime \prime}=$ punit.lower (rep-list $p^{\prime}$ ) $t$
from 1 (3) have $t \in$ keys (rep-list $p$ ) by simp
from 1 (4) have $a: b \in$ set $b s \Longrightarrow$ rep-list $b \neq 0 \Longrightarrow t \prec s+$ punit.lt (rep-list b) $\Longrightarrow$

$$
\left.s \oplus l t b \prec_{t} \text { lt } p \Longrightarrow \text { lookup (rep-list } p\right)(s+\text { punit.lt (rep-list b)) }
$$

$=0$
for $b s$ by (simp only: fst-conv snd-conv)
have lt $p^{\prime}=$ lt $p \wedge\left(\forall s . t \prec s \longrightarrow\right.$ lookup (rep-list $\left.p^{\prime}\right) s=$ lookup (rep-list p) s)
proof (cases find-sig-reducer (map spp-of bs) (lt p) t 0)
case None
thus ?thesis by (simp add: $p^{\prime}$-def)
next
case (Some i)
hence $p^{\prime}: p^{\prime}=p-$ monom-mult (lookup (rep-list $p$ ) $t /$ punit.lc (rep-list (bs ! $i)$ )
$(t-p u n i t . l t(r e p-l i s t(b s!i)))(b s!i)$ by (simp add:
$\left.p^{\prime}-d e f\right)$
have sig-red-single $\left(\prec_{t}\right)(\preceq) p p^{\prime}(b s!i)(t-$ punit.lt (rep-list $\left.(b s!i))\right)$
unfolding $p^{\prime}$ using $\langle t \in$ keys (rep-list $p$ ) >Some by (rule find-sig-reducer-SomeD-red-single)
hence $r$ : punit.red-single (rep-list p) (rep-list $\left.p^{\prime}\right)($ rep-list $(b s!i))(t-p u n i t . l t$ (rep-list (bs!i)))
and $l t p^{\prime}=l t p$ by (rule sig-red-single-red-single, rule sig-red-single-regular-lt)

```
    have }\foralls.t\precs\longrightarrowlookup (rep-list p')s=lookup(rep-list p)
proof (intro allI impI)
fix }
assume t}\prec
from Some have punit.lt (rep-list (bs!i)) adds t by (rule find-sig-reducer-SomeD)
    hence eq0:(t - punit.lt (rep-list (bs!i))) + punit.lt (rep-list (bs!i)) =t
(is ? }t=t\mathrm{ )
            by (rule adds-minus)
    from }\langlet\precs\rangle\mathrm{ have lookup (rep-list p')s=lookup (punit.higher (rep-list p')
?t) s
            by (simp add: eq0 punit.lookup-higher-when)
    also from r have ... = lookup (punit.higher (rep-list p) ?t) s
            by (simp add: punit.red-single-higher[simplified])
            also from \langlet \prec s\rangle have ... = lookup (rep-list p) s by (simp add: eq0
punit.lookup-higher-when)
            finally show lookup (rep-list p') s= lookup (rep-list p)s.
            qed
            with <lt p' = lt p\rangle show ?thesis ..
    qed
    hence lt-p':lt p' = lt p and b: \s.t \prec s\Longrightarrowlookup (rep-list p') s=lookup
(rep-list p) s
        by blast+
    have c:lookup (rep-list p') (s+ punit.lt (rep-list b)) = 0
        if b\in set bs and rep-list b}=0\mathrm{ and }t\preceqs+\mathrm{ punit.lt (rep-list b) and s}\oplusl
b}\mp@subsup{\prec}{t}{lt p}\mp@subsup{p}{}{\prime}\mathrm{ for bs
    proof (cases t}\precs+\mathrm{ punit.lt (rep-list b))
        case True
        hence lookup (rep-list p})(s+\mathrm{ punit.lt (rep-list b)) =
            lookup (rep-list p) (s+ punit.lt (rep-list b)) by (rule b)
        also from that(1, 2) True that(4) have ... = 0 unfolding lt-p' by (rule a)
        finally show ?thesis.
    next
        case False
        with that(3) have t:t=s+ punit.lt (rep-list b) by simp
        show ?thesis
        proof (cases find-sig-reducer (map spp-of bs) (lt p) t 0)
            case None
            from that(1) have spp-of b\in set (map spp-of bs) by fastforce
            with None show ?thesis
            proof (rule find-sig-reducer-NoneE)
                assume snd (spp-of b)=0
                with that(2) show ?thesis by (simp add: snd-spp-of)
            next
                assume \neg punit.lt (snd (spp-of b)) adds t
                    thus ?thesis by (simp add: snd-spp-of t)
            next
                assume }\neg(t-punit.lt (snd (spp-of b))) \oplusfst (spp-of b) < < lt 
                with that(4) show ?thesis by (simp add: fst-spp-of snd-spp-of t lt-p')
            qed
```


## next

case (Some i)
hence $p^{\prime}: p^{\prime}=p$ - monom-mult (lookup (rep-list p) $t /$ punit.lc (rep-list ( $b s!i))$ )
$(t-p u n i t . l t(r e p-l i s t(b s!i)))(b s!i)$ by $($ simp add:

$$
\left.p^{\prime}-d e f\right)
$$

have sig-red-single $\left(\prec_{t}\right)(\preceq) p p^{\prime}(b s!i)(t-p u n i t . l t(r e p-l i s t(b s!i)))$
unfolding $p^{\prime}$ using $\langle t \in$ keys (rep-list p) > Some by (rule find-sig-reducer-SomeD-red-single)
hence $r$ : punit.red-single (rep-list $p)\left(\right.$ rep-list $\left.p^{\prime}\right)($ rep-list $(b s!i))(t-$ punit.lt (rep-list (bs!i)))
by (rule sig-red-single-red-single)
from Some have punit.lt (rep-list (bs ! i)) adds $t$ by (rule find-sig-reducer-SomeD)
hence eq $0:(t-$ punit.lt $($ rep-list $(b s!i)))+$ punit.lt $($ rep-list $(b s!i))=t$ (is ? $t=t$ )
by (rule adds-minus)
from $r$ have lookup (rep-list $\left.p^{\prime}\right)((t-$ punit.lt $($ rep-list $(b s!i)))+$ punit.lt $($ rep-list $(b s!i)))=0$
by (rule punit.red-single-lookup [simplified])
thus ?thesis by (simp only: eq0 t[symmetric])
qed
qed
show ?case
proof (simp add: sig-trd-aux.psimps[OF 1(1)] Let-def $p^{\prime}$-def[symmetric] $p^{\prime \prime}$-def[symmetric], intro conjI impI)
assume $p^{\prime \prime}=0$
show $\neg i s$-sig-red $\left(\prec_{t}\right)(\preceq)\left(\right.$ set bs) $p^{\prime}$
proof
assume is-sig-red $\left(\prec_{t}\right)(\preceq)\left(\right.$ set bs) $p^{\prime}$
then obtain $b s$ where $b \in$ set $b s$ and $s \in$ keys (rep-list $p^{\prime}$ ) and rep-list $b$ $\neq 0$
and adds: punit.lt (rep-list b) adds s and $s \oplus l t b \prec_{t}$ punit.lt (rep-list b) $\oplus l t p^{\prime}$
by (rule is-sig-red-addsE)
let ?s $=s$ - punit.lt (rep-list b)
from adds have eq0: ?s + punit.lt (rep-list b) $=s$ by (simp add: adds-minus)
show False
proof (cases $t \preceq s$ )
case True
note $\langle b \in$ set $b s\rangle\langle$ rep-list $b \neq 0\rangle$
moreover from True have $t \preceq ? s+$ punit.lt (rep-list b) by (simp only:
eq0)
moreover from adds $\left\langle s \oplus l t b \prec_{t}\right.$ punit.lt (rep-list b) $\left.\oplus l t p^{\prime}\right\rangle$ have ?s $\oplus$ lt $b \prec_{t}$ lt $p^{\prime}$
by (simp add: term-is-le-rel-minus)
ultimately have lookup (rep-list p') $($ ?s + punit.lt (rep-list b) $)=0$ by (rule c)
hence $s \notin$ keys (rep-list $p^{\prime}$ ) by (simp add: eq0 in-keys-iff)
thus ?thesis using $\langle s \in$ keys (rep-list $p$ ) 〉..
next

```
            case False
            hence s}\prect\mathrm{ by simp
            hence lookup (rep-list p')s=lookup(punit.lower (rep-list p') t) s
                    by (simp add: punit.lookup-lower-when)
            also from < < '' = 0〉 have ... = 0 by (simp add: p '''def)
            finally have s\not\in keys (rep-list p') by (simp add: in-keys-iff)
            thus ?thesis using <s \in keys (rep-list p}\mp@subsup{p}{}{\prime})\rangle.
            qed
        qed
    next
        assume p"}\not=
        with p'-def p}\mp@subsup{}{}{\prime\prime}\mathrm{ -def show }\neg\mathrm{ is-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq)(set bs) (sig-trd-aux (punit.lt
p\prime\prime, p')
    proof (rule 1(2))
        from }\langle\mp@subsup{p}{}{\prime\prime}\not=0\rangle\mathrm{ have punit.lt p" }\in\mathrm{ keys p'' by (rule punit.lt-in-keys)
        also have ...\subseteq keys (rep-list p') by (auto simp: p''-def punit.keys-lower)
        finally show fst (punit.lt p'\prime, p') \in keys (rep-list (snd (punit.lt p", p'))) by
simp
    next
        fix bs
        assume b\in set bs and rep-list b}=
        assume fst (punit.lt p'\prime, p')\precs+ punit.lt (rep-list b)
            and s\opluslt b}\mp@subsup{\prec}{t}{lt (snd (punit.lt p'\prime, p'))
    hence punit.lt p\prime\prime}\precs+\mathrm{ punit.lt (rep-list b) and s}\oplusltb\prec\mp@subsup{\prec}{t}{lt p' by simp-all
        have lookup (rep-list p')}(s+\mathrm{ punit.lt (rep-list b)) = 0
        proof (cases t\preceqs+ punit.lt (rep-list b))
            case True
            with }\langleb\in\mathrm{ set bs`<rep-list b}=0\mathrm{ \ show ?thesis using <s }\opluslt b < <t lt p'>
by (rule c)
    next
                case False
            hence s + punit.lt (rep-list b) \prect by simp
            hence lookup (rep-list p') (s + punit.lt (rep-list b)) =
                    lookup (punit.lower (rep-list p') t) (s + punit.lt (rep-list b))
                    by (simp add: punit.lookup-lower-when)
            also have ... = 0
            proof (rule ccontr)
                assume lookup (punit.lower (rep-list p') t) (s+ punit.lt (rep-list b)) }=
                    hence s + punit.lt (rep-list b) \preceq punit.lt (punit.lower (rep-list p') t)
                    by (rule punit.lt-max)
                    also have ... = punit.lt p'\prime by (simp only: p}\mp@subsup{}{}{\prime\prime}-def
                    finally show False using<punit.lt p'l}\precs+\mathrm{ punit.lt (rep-list b)〉 by
simp
            qed
            finally show ?thesis .
            qed
            thus lookup (rep-list (snd (punit.lt p'\prime},\mp@subsup{p}{}{\prime})))(s+\mathrm{ punit.lt (rep-list b)) =0
                by (simp only: snd-conv)
    qed
```

```
        qed
    qed
qed
corollary sig-trd-irred: ᄀ is-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq)(set bs) (sig-trd p
    unfolding sig-trd-def
proof (split if-split, intro conjI impI)
    assume rep-list p=0
    show \neg is-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq)(set bs)
    proof
        assume is-sig-red (\mp@subsup{\prec}{t}{})(\preceq) (set bs)p
        then obtain t where t\inkeys (rep-list p) by (rule is-sig-red-addsE)
        thus False by (simp add: <rep-list p = 0`)
    qed
next
    assume rep-list p\not=0
    show }\neg\mathrm{ is-sig-red (}\mp@subsup{\prec}{t}{})(\preceq)(\mathrm{ set bs) (sig-trd-aux (punit.lt (rep-list p), p))
    proof (rule sig-trd-aux-irred)
        from 〈rep-list p\not=0\rangle have punit.lt (rep-list p) \in keys (rep-list p) by (rule
punit.lt-in-keys)
    thus fst (punit.lt (rep-list p), p) \in keys (rep-list (snd (punit.lt (rep-list p), p)))
by simp
    next
        fix bs
        assume fst (punit.lt (rep-list p), p)\precs+ punit.lt (rep-list b)
        thus lookup (rep-list (snd (punit.lt (rep-list p), p))) (s + punit.lt (rep-list b))
= 0
            using punit.lt-max by force
        qed
qed
end
context
    fixes bs:: ('t \times ('a =>0 'b)) list
begin
context
    fixes v :: 't
begin
```



```
where
    sig-trd-spp-body ( }p,r)
        (case find-sig-reducer bs v (punit.lt p) 0 of
            None }=>\mathrm{ (punit.tail p,r + monomial (punit.lc p)(punit.lt p))
            | Some i 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 :: $\left(\left(^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)$
where sig-trd-spp-aux-def [code del]: sig-trd-spp-aux $=$ tailrec.fun $(\lambda x$.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 $(p$, $r$ ))
by (simp add: sig-trd-spp-aux-def tailrec.simps)
end
fun sig-trd-spp :: $\left.{ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)$ 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 $f s$ of scalar polynomials is a syzygy of the form $f s$ $!i \odot$ monomial $\left(1:^{\prime} b\right)\left(\right.$ term-of-pair $\left.\left(0::^{\prime} a, j\right)\right)-f s!j \odot$ monomial $\left(1:^{\prime} b\right)$ (term-of-pair $\left(0::^{\prime} a, i\right)$ ), for $i<j$ and $j<$ length $f s$.
primrec Koszul-syz-sigs-aux :: ( $\left.{ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow$ nat $\Rightarrow$ 't list where
Koszul-syz-sigs-aux [] $i=[] \mid$
Koszul-syz-sigs-aux (b \# bs) $i=$
map-idx $\left(\lambda b^{\prime} j\right.$. ord-term-lin.max (term-of-pair (punit.lt b, $j$ )) (term-of-pair (punit.lt $\left.\left.b^{\prime}, i\right)\right)$ )bs (Suc i) @

Koszul-syz-sigs-aux bs (Suc i)
definition Koszul-syz-sigs :: ( $\left.{ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow$ 't list
where Koszul-syz-sigs bs $=$ filter-min $\left(a d d s_{t}\right)($ Koszul-syz-sigs-aux bs 0)
fun new-syz-sigs :: 't list $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow\left(\left(' t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+$ nat $\Rightarrow$ 't list
where

```
        new-syz-sigs ss bs \((\operatorname{Inl}(a, b))=s s \mid\)
        new-syz-sigs ss bs (Inr \(j\) ) \(=\)
            (if is-pot-ord then
                filter-min-append \(\left(a d d s_{t}\right)\) ss (filter-min \(\left(a d d s_{t}\right)\) (map ( \(\lambda\) b. term-of-pair
(punit.lt (rep-list b), j)) bs))
        else ss)
fun new-syz-sigs-spp :: 't list \(\Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\) list \(\Rightarrow\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t\right.\right.\)
\(\left.\left.\times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+\) nat \(\Rightarrow\) 't list
    where
```

new-syz-sigs-spp ss bs $(\operatorname{Inl}(a, b))=s s$
new-syz-sigs-spp ss bs $(\operatorname{Inr} j)=$
(if is-pot-ord then
filter-min-append $\left(a d d s_{t}\right)$ ss (filter-min $\left(a d d s_{t}\right)$ (map ( $\lambda$ b. term-of-pair (punit.lt (snd b), j)) bs))
else ss)
lemma Koszul-syz-sigs-auxI:
assumes $i<j$ and $j<$ length $b s$
shows ord-term-lin.max (term-of-pair (punit.lt (bs ! i), k+j)) (term-of-pair
(punit.lt $(b s!j), k+i)) \in$
set (Koszul-syz-sigs-aux bs $k$ )
using assms
proof (induct bs arbitrary: ijk)
case Nil
from $\operatorname{Nil(2)~show~?case~by~simp~}$
next
case (Cons bbs)
from Cons(2) obtain $j 0$ where $j: j=$ Suc $j 0$ by (meson lessE)
from Cons(3) have $j 0<$ length bs by (simp add: $j$ )
let ? $A=(\lambda j$. ord-term-lin.max (term-of-pair (punit.lt b, Suc $(j+k))$ ) (term-of-pair (punit.lt $(b s!j), k))$ )'
$\{0 . .<$ length $b s\}$
let ${ }^{\text {? } B=}=$ set $($ Koszul-syz-sigs-aux bs $($ Suc $k))$
show ?case
proof (cases i)
case 0
from $\langle j 0<$ length $b s\rangle$ have $j 0 \in\{0 . .<$ length $b s\}$ by simp
hence ord-term-lin.max (term-of-pair (punit.lt b, Suc $(j 0+k))$ )
(term-of-pair (punit.lt $(b s!j 0), k)) \in ? A$ by (rule imageI)
thus ?thesis by (simp add: $\langle i=0\rangle j$ set-map-idx ac-simps)
next
case (Suc i0)
from Cons(2) have $i 0<j 0$ by (simp add: $\langle i=$ Suc $i 0\rangle j$ )
hence ord-term-lin.max (term-of-pair (punit.lt (bs!i0), Suc $k+j 0)$ )
(term-of-pair (punit.lt (bs!j0), Suc $k+i 0)) \in ? B$ using $\langle j 0<$ length $b s\rangle$ by (rule Cons(1))
thus ?thesis by (simp add: $\langle i=S u c i 0\rangle j$ set-map-idx ac-simps)
qed
qed
lemma Koszul-syz-sigs-auxE:
assumes $v \in$ set (Koszul-syz-sigs-aux bs $k$ )
obtains $i j$ where $i<j$ and $j<$ length $b s$
and $v=$ ord-term-lin.max (term-of-pair (punit.lt $(b s!i), k+j)$ ) (term-of-pair (punit.lt (bs!j),k+i))
using assms
proof (induct bs arbitrary: $k$ thesis)
case Nil

```
    from Nil(2) show ?case by simp
next
    case (Cons b bs)
    have}v\in(\lambdaj. ord-term-lin.max (term-of-pair (punit.lt b, Suc (j+k))) (term-of-pair
(punit.lt (bs!j),k)))`
                            {0..<length bs} \cup set (Koszul-syz-sigs-aux bs (Suc k)) (is v\in?A\cup
?B)
    using Cons(3) by (simp add: set-map-idx)
    thus ?case
    proof
        assume v\in?A
    then obtain }j\mathrm{ where j}\in{0..<length bs 
        and v:v=ord-term-lin.max (term-of-pair (punit.lt b, Suc (j+k)))
                            (term-of-pair (punit.lt (bs!j),k)) ..
    from this(1) have j<length bs by simp
    show ?thesis
    proof (rule Cons(2))
        show 0<Suc j by simp
    next
        from <j < length bs` show Suc j < length (b # bs) by simp
    next
        show v =ord-term-lin.max (term-of-pair (punit.lt ((b # bs)!0), k+Suc
j))
                    (term-of-pair (punit.lt ((b # bs)!Suc j), k+0))
        by (simp add: v ac-simps)
    qed
    next
    assume v\in?B
    obtain ij where i<j and j< length bs
        and v:v =ord-term-lin.max (term-of-pair (punit.lt (bs!i), Suc k+j))
                            (term-of-pair (punit.lt (bs!j),Suc k+i))
        by (rule Cons(1), assumption, rule }\langlev\in?B>
    show ?thesis
    proof (rule Cons(2))
        from <i<j〉 show Suc i<Suc j by simp
    next
        from <j< length bs` show Suc j< length (b # bs) by simp
    next
        show v = ord-term-lin.max (term-of-pair (punit.lt ((b # bs)!Suc i),k+
Suc j))
                                    (term-of-pair (punit.lt ((b # bs)!Suc j),k+Suc i))
            by (simp add: v)
        qed
    qed
qed
lemma lt-Koszul-syz-comp:
    assumes 0 # set fs and i< length fs
    shows lt ((fs !i)\odot monomial 1 (term-of-pair (0,j))) = term-of-pair (punit.lt
```

```
(fs!i),j)
proof -
    from assms(2) have fs !i\in set fs by (rule nth-mem)
    with assms(1) have fs!i\not=0 by auto
    thus ?thesis by (simp add: lt-mult-scalar-monomial-right splus-def term-simps)
qed
lemma Koszul-syz-nonzero-lt:
    assumes rep-list a\not=0 and rep-list b\not=0 and component-of-term (lt a) <
component-of-term (lt b)
    shows rep-list a\odotb-rep-list b \odota\not=0 (is ?p - ?q}\not=0
        and lt (rep-list a \odotb-rep-list b \odota)=
            ord-term-lin.max (punit.lt (rep-list a)}\opluslt b)(punit.lt (rep-list b)\opluslt a
(is - = ?r)
proof -
    from assms(2) have b\not=0 by (auto simp: rep-list-zero)
    with assms(1) have lt-p:lt ? p = punit.lt (rep-list a) \opluslt b by (rule lt-mult-scalar)
    from assms(1) have a\not=0 by (auto simp: rep-list-zero)
    with assms(2) have lt-q:lt ?q = punit.lt (rep-list b) }\opluslt a by (rule lt-mult-scalar)
    from assms(3) have component-of-term (lt ?p) \not= component-of-term (lt ?q)
        by (simp add: lt-p lt-q component-of-term-splus)
    hence lt ?p \not=lt?q by auto
    hence lt (?p - ?q) = ord-term-lin.max (lt ?p) (lt ?q) by (rule lt-minus-distinct-eq-max)
    also have ... = ?r by (simp only: lt-p lt-q)
    finally show lt (?p - ?q) = ?r .
    from <lt ?p plt ?q> show ?p - ?q }\not=0\mathrm{ by auto
qed
lemma Koszul-syz-is-syz:rep-list (rep-list a \odotb-rep-list b \odota)=0
    by (simp add: rep-list-minus rep-list-mult-scalar)
lemma dgrad-sig-set-closed-Koszul-syz:
```



``` dgrad
shows rep-list \(a \odot b-\) rep-list \(b \odot a \in\) dgrad-sig-set dgrad proof -
from \(\operatorname{assms}(2,3)\) have 1: \(a \in\) dgrad-max-set dgrad and 2: \(b \in\) dgrad-max-set dgrad
by (simp-all add: dgrad-sig-set'-def)
show ?thesis
by (intro dgrad-sig-set-closed-minus dgrad-sig-set-closed-mult-scalar dgrad-max-2 assms 1 2)
qed
corollary Koszul-syz-is-syz-sig:
```



``` dgrad
and rep-list \(a \neq 0\) and rep-list \(b \neq 0\) and component-of-term (lt \(a\) ) <compo-
```

```
nent-of-term (lt b)
    shows is-syz-sig dgrad (ord-term-lin.max (punit.lt (rep-list a) \oplus lt b) (punit.lt
(rep-list b) }\opluslt a)
proof (rule is-syz-sigI)
    from assms(4-6) show rep-list a \odotb - rep-list b \odota\not=0
    and lt (rep-list a \odotb - rep-list b \odota) =
                ord-term-lin.max (punit.lt (rep-list a)}\opluslt b)(punit.lt (rep-list b)\opluslt a)
    by (rule Koszul-syz-nonzero-lt)+
next
    from assms(1-3) show rep-list a \odotb - rep-list b \odota\indgrad-sig-set dgrad
        by (rule dgrad-sig-set-closed-Koszul-syz)
qed (fact Koszul-syz-is-syz)
corollary lt-Koszul-syz-in-Koszul-syz-sigs-aux:
    assumes distinct fs and 0}\not=\mathrm{ set fs and i<j and j< length fs
    shows lt ((fs!i)\odot monomial 1 (term-of-pair (0,j)) - (fs!j)\odot monomial 1
(term-of-pair (0,i)))\in
    set (Koszul-syz-sigs-aux fs 0) (is ?l \in ?K)
proof -
    let ?a = monomial (1::'b) (term-of-pair (0,i))
    let ?b = monomial (1::'b) (term-of-pair ( 0, j))
    from assms(3, 4) have i< length fs by simp
    with assms(1) have a: rep-list ?a = fs ! i by (simp add: rep-list-monomial
term-simps)
    from assms(1, 4) have b: rep-list ?b = fs ! j by (simp add: rep-list-monomial
term-simps)
    have ?l = lt (rep-list ?a \odot ?b - rep-list ?b \odot ?a) by (simp only: a b)
    also have ... = ord-term-lin.max (punit.lt (rep-list ?a) \opluslt ?b) (punit.lt (rep-list
    ?b) }\opluslt ?a
    proof (rule Koszul-syz-nonzero-lt)
        from «i< length fs` have fs ! i\in set fs by (rule nth-mem)
        with assms(2) show rep-list ?a }\not=0\mathrm{ by (auto simp: a)
    next
        from assms(4) have fs! j\in set fs by (rule nth-mem)
        with assms(2) show rep-list ?b}\not=0\mathrm{ by (auto simp: b)
    next
        from assms(3) show component-of-term (lt ?a) < component-of-term (lt ?b)
            by (simp add:lt-monomial component-of-term-of-pair)
    qed
    also have ... = ord-term-lin.max (term-of-pair (punit.lt (fs!i), 0 +j)) (term-of-pair
(punit.lt (fs!j), 0 + i))
    by (simp add: a b lt-monomial splus-def term-simps)
    also from assms(3, 4) have .. \in?K by (rule Koszul-syz-sigs-auxI)
    thm Koszul-syz-sigs-auxI[OF assms(3, 4)]
    finally show ?thesis.
qed
corollary lt-Koszul-syz-in-Koszul-syz-sigs:
    assumes \negis-pot-ord and distinct fs and 0 & set fs and i<j and j< length
```

```
fs
    obtains v where v\in set (Koszul-syz-sigs fs)
    and vaddst lt ((fs!i)\odot monomial 1 (term-of-pair (0,j))-(fs!j)\odot monomial
    1 (term-of-pair (0,i)))
proof -
    have transp (addst) by (rule transpI, drule adds-term-trans)
    moreover have lt ((fs ! i) \odot monomial 1 (term-of-pair (0, j)) - (fs ! j) \odot
monomial 1 (term-of-pair (0, i))) \in
                                    set (Koszul-syz-sigs-aux fs 0) (is ?l \in set ?ks)
        using assms(2-5) by (rule lt-Koszul-syz-in-Koszul-syz-sigs-aux)
    ultimately show ?thesis
    proof (rule filter-min-cases)
        assume ?l \in set (filter-min (addst) ?ks)
        hence ?l \in set (Koszul-syz-sigs fs) by (simp add: Koszul-syz-sigs-def assms(1))
        thus ?thesis using adds-term-refl ..
    next
        fix v
        assume v}\in\operatorname{set (filter-min (adds}\mp@subsup{)}{t}{})\mathrm{ ?ks)
        hence }v\in\mathrm{ set (Koszul-syz-sigs fs) by (simp add: Koszul-syz-sigs-def assms(1))
        moreover assume v addst ?l
        ultimately show ?thesis ..
    qed
qed
lemma lt-Koszul-syz-init:
    assumes 0}\not\in\mathrm{ set fs and i<j and j< length fs
    shows lt ((fs ! i) \odot monomial 1 (term-of-pair (0,j)) - (fs ! j) \odot monomial 1
(term-of-pair (0,i))) =
        ord-term-lin.max (term-of-pair (punit.lt (fs !i),j)) (term-of-pair (punit.lt
(fs!j),i))
            (is lt (?p-?q) =?r)
proof -
    from assms(2, 3) have i< length fs by simp
        with assms(1) have lt-i:lt ?p = term-of-pair (punit.lt (fs ! i), j) by (rule
lt-Koszul-syz-comp)
    from assms(1, 3) have lt-j:lt ? q = term-of-pair (punit.lt (fs ! j), i) by (rule
lt-Koszul-syz-comp)
    from assms(2) have component-of-term (lt ?p) = component-of-term (lt ?q)
        by (simp add: lt-i lt-j component-of-term-of-pair)
    hence lt ?p \not=lt ?q by auto
    hence lt (?p - ?q) = ord-term-lin.max (lt ?p) (lt ?q) by (rule lt-minus-distinct-eq-max)
    also have ... = ?r by (simp only: lt-i lt-j)
    finally show ?thesis.
qed
corollary Koszul-syz-sigs-auxE-lt-Koszul-syz:
    assumes 0& set fs and v\in set (Koszul-syz-sigs-aux fs 0)
    obtains ij where i<j and j< length fs
        and v}=lt((\mp@subsup{f}{s}{}!i)\odot monomial 1 (term-of-pair (0, j))-(fs!j)\odot monomial
```

```
1(term-of-pair (0,i)))
proof -
    from assms(2) obtain ij where i<j and j< length fs
        and v =ord-term-lin.max (term-of-pair (punit.lt (fs!i),0 + j))
                        (term-of-pair (punit.lt (fs!j),0 + i))
        by (rule Koszul-syz-sigs-auxE)
    with assms(1) have v=lt ((fs!i)\odot monomial 1 (term-of-pair (0,j)) -
                        (fs!j) \odot monomial 1 (term-of-pair (0,i)))
    by (simp add: lt-Koszul-syz-init)
    with }\langlei<j\rangle\langlej< length fs` show ?thesis ..
qed
corollary Koszul-syz-sigs-is-syz-sig:
    assumes dickson-grading dgrad and distinct fs and 0}#\mathrm{ set fs and v set
(Koszul-syz-sigs fs)
    shows is-syz-sig dgrad v
proof -
    from assms(4) have v\in set (Koszul-syz-sigs-aux fs 0)
        using filter-min-subset by (fastforce simp:Koszul-syz-sigs-def)
    with assms(3) obtain ij where i<j and j< length fs
    and v}\mp@subsup{v}{}{\prime}:v=lt((fs!i)\odot monomial 1 (term-of-pair (0,j))- (fs!j)\odot monomial
1 (term-of-pair (0,i)))
                (is v=lt (?p - ?q))
    by (rule Koszul-syz-sigs-auxE-lt-Koszul-syz)
    let ?a = monomial (1::'b) (term-of-pair (0, i))
    let ?b = monomial (1::'b) (term-of-pair (0,j))
    from }\langlei<j\rangle\langlej< length fs\rangle have i< length fs by sim
    with assms(2) have a: rep-list ?a = fs ! i by (simp add: rep-list-monomial
term-simps)
    from assms(2) <j < length fs` have b: rep-list ?b = fs ! j by (simp add:
rep-list-monomial term-simps)
    note }\mp@subsup{v}{}{\prime
    also have lt (?p - ?q) = ord-term-lin.max (term-of-pair (punit.lt (fs ! i), j))
(term-of-pair (punit.lt (fs ! j), i))
    using assms(3)<i<j\rangle<j< length fs> by (rule lt-Koszul-syz-init)
    also have ... = ord-term-lin.max (punit.lt (rep-list ?a) \opluslt ?b) (punit.lt (rep-list
?b) }\opluslt ?a
    by (simp add: a b lt-monomial splus-def term-simps)
    finally have v:v=ord-term-lin.max (punit.lt (rep-list ?a) \oplus lt ?b) (punit.lt
(rep-list ?b) }\opluslt ?a)
    show ?thesis unfolding v using assms(1)
    proof (rule Koszul-syz-is-syz-sig)
    show ?a \in dgrad-sig-set dgrad
            by (rule dgrad-sig-set-closed-monomial, simp-all add: term-simps dgrad-max-0
< < length fs>)
    next
        show ?b \in dgrad-sig-set dgrad
            by (rule dgrad-sig-set-closed-monomial, simp-all add: term-simps dgrad-max-0
<j < length fs`)
```

```
    next
        from <i< length fs` have fs ! i\in set fs by (rule nth-mem)
    with assms(3) show rep-list ?a}\not=0\mathrm{ by (fastforce simp: a)
    next
    from <j< length fs` have fs ! j\in set fs by (rule nth-mem)
    with assms(3) show rep-list ?b}\not=0\mathrm{ by (fastforce simp: b)
    next
    from <i< j\rangle show component-of-term (lt ?a) < component-of-term (lt ?b)
        by (simp add:lt-monomial component-of-term-of-pair)
    qed
qed
lemma Koszul-syz-sigs-minimal:
    assumes }u\in\operatorname{set (Koszul-syz-sigs fs) and v\in set (Koszul-syz-sigs fs) and u
addst v
    shows }u=
proof -
    from assms(1, 2) have u\in\operatorname{set (filter-min (addst) (Koszul-syz-sigs-aux fs 0))}
        and v\in set (filter-min (addst) (Koszul-syz-sigs-aux fs 0)) by (simp-all add:
Koszul-syz-sigs-def)
    with - show ?thesis using assms(3)
    proof (rule filter-min-minimal)
        show transp (addst) by (rule transpI, drule adds-term-trans)
    qed
qed
lemma Koszul-syz-sigs-distinct: distinct (Koszul-syz-sigs fs)
proof -
    from adds-term-refl have reflp (adds}\mp@subsup{)}{t}{})\mathrm{ by (rule reflpI)
    thus ?thesis by (simp add: Koszul-syz-sigs-def filter-min-distinct)
qed
```


### 4.2.10 Algorithms

definition spair-spp :: $\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)$ where spair-spp p $q=($ let $t 1=$ punit.lt (snd $p) ;$ t2 $=$ punit.lt (snd $q) ; l=$ lcs t1 t2 in

$$
\begin{aligned}
& \text { (ord-term-lin.max }((l-t 1) \oplus \text { fst p) }((l-t 2) \oplus \text { fst } q) \text {, } \\
& \text { punit.monom-mult }(1 / \text { punit.lc }(\text { snd p }))(l-t 1)(\text { snd p })- \\
& \text { punit.monom-mult }(1 / \text { punit.lc }(\text { snd } q))(l-t 2)(\text { snd } q)))
\end{aligned}
$$

definition is-regular-spair-spp :: (' $\left.t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow$ bool where is-regular-spair-spp $p \quad q \longleftrightarrow$ (snd $p \neq 0 \wedge$ snd $q \neq 0 \wedge$ punit.lt (snd $q) \oplus$ fst $p \neq$ punit.lt (snd $p) \oplus f s t q)$
definition spair-sigs :: $\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \times{ }^{\prime} t\right)$
where spair-sigs p $q=$
$($ let $t 1=$ punit.lt $($ rep-list $p) ;$ t2 $=$ punit.lt $($ rep-list $q) ; l=l$ cs $t 1$ t2

$$
((l-t 1) \oplus l t p,(l-t 2) \oplus l t q))
$$

definition spair-sigs-spp :: $\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times{ }^{\prime} t\right)$ where spair-sigs-spp $p q=$

$$
(\text { let } t 1=\text { punit.lt }(\text { snd } p) ; \text { t2 }=\text { punit.lt }(\text { snd } q) ; l=\text { lcs t1 t2 in }
$$

$$
((l-t 1) \oplus \text { fst } p,(l-t 2) \oplus \text { fst } q))
$$

```
fun poly-of-pair :: \(\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+\right.\) nat \() \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\)
    where
    poly-of-pair \((\operatorname{Inl}(p, q))=\operatorname{spair} p q \mid\)
    poly-of-pair \((\operatorname{Inr} j)=\) monomial 1 (term-of-pair \((0, j))\)
```

fun spp-of-pair :: $\left.\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+n a t\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}\right.\right.$
'b))
where
$\operatorname{spp}$-of-pair $(\operatorname{Inl}(p, q))=\operatorname{spair-spp} p q \mid$
spp-of-pair $(\operatorname{Inr} j)=($ term-of-pair $(0, j), f s!j)$
fun sig-of-pair :: $\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+\right.$ nat $) \Rightarrow{ }^{\prime} t$
where
sig-of-pair $(\operatorname{Inl}(p, q))=($ let $(u, v)=$ spair-sigs $p q$ in ord-term-lin.max $u v) \mid$
sig-of-pair $(\operatorname{Inr} j)=$ term-of-pair $(0, j)$
fun sig-of-pair-spp :: $\left.\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+n a t\right) \Rightarrow{ }^{\prime} t$
where
sig-of-pair-spp $(\operatorname{Inl}(p, q))=($ let $(u, v)=$ spair-sigs-spp p q in ord-term-lin.max
$u v)$ |
sig-of-pair-spp $(\operatorname{Inr} j)=$ term-of-pair $(0, j)$
definition pair-ord :: $\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+n a t\right) \Rightarrow\left(\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}\right.\right.\right.$
$\left.\left.\left.{ }^{\prime} b\right)\right)+n a t\right) \Rightarrow$ bool
where pair-ord $x y \longleftrightarrow$ (sig-of-pair $x \preceq_{t}$ sig-of-pair $\left.y\right)$
definition pair-ord-spp :: $\left.\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+n a t\right) \Rightarrow$
$\left(\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+\right.$ nat $) \Rightarrow$ bool
where pair-ord-spp $x$ y (sig-of-pair-spp $x \preceq_{t}$ sig-of-pair-spp $\left.y\right)$
primrec new-spairs :: $\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)\right.$

+ nat) list where
new-spairs [] $p=[] \mid$
new-spairs ( $b \# b s$ ) $p=$
(if is-regular-spair $p$ b then insort-wrt pair-ord $(\operatorname{Inl}(p, b))$ (new-spairs bs $p$ ) else
new-spairs bs $p$ )
primrec new-spairs-spp :: (' $\left.t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)$ list $\Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow$
$\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+$ nat $)$ list where
new-spairs-spp [] $p=[] \mid$
new-spairs-spp (b \# bs) $p=$
(if is-regular-spair-spp $p b$ then
insort-wrt pair-ord-spp ( $\operatorname{Inl}(p, b))$ (new-spairs-spp bs $p)$
else new-spairs-spp bs p)
definition add-spairs :: $\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+$ nat $)$ list $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow$ $\left.{ }^{\prime} t \Rightarrow{ }_{0}{ }^{\prime} b\right) \Rightarrow$

$$
\left(\left(\left({ }^{\prime} t \Rightarrow_{0}^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}^{\prime} b\right)\right)+\text { nat }\right) \text { list }
$$

where add-spairs ps bs $p=$ merge-wrt pair-ord (new-spairs bs $p$ ) ps
definition add-spairs-spp :: $\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+$ nat $)$ list $\Rightarrow$ $\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)$ list $\left.\Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right) \Rightarrow$
$\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+$ 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 $p$ $q))-l p p) p-$
monom-mult (1 / punit.lc (rep-list q)) (pp-of-term (snd (spair-sigs p
$q))-l p q) q$
by (simp add: spair-def spair-sigs-def Let-def term-simps)
lemma sig-of-spair:
assumes is-regular-spair $p q$
shows sig-of-pair $(\operatorname{Inl}(p, q))=l t(\operatorname{spair} p q)$
proof -
from assms have rep-list $p \neq 0$ by (rule is-regular-spairD1)
hence 1: punit.lc (rep-list $p$ ) $\neq 0$ and $p \neq 0$ by (rule punit.lc-not- 0 , auto simp: rep-list-zero)
from assms have rep-list $q \neq 0$ by (rule is-regular-spairD2)
hence 2: punit.lc (rep-list $q$ ) $\neq 0$ and $q \neq 0$ by (rule punit.lc-not-0, auto simp: rep-list-zero)
let ? $\mathrm{t} 1=$ punit.lt $($ rep-list $p)$
let ? t 2 $=$ punit.lt (rep-list $q$ )
let ? $l=l c s$ ? $t 1$ ? $t 2$
from assms have lt (monom-mult (1 / punit.lc (rep-list p)) (?l - ?t1) p) $\neq$ lt (monom-mult (1 / punit.lc (rep-list q)) (?l - ?t2) q)
by (rule is-regular-spairD3)
hence *: lt (monom-mult (1 / punit.lc (rep-list p)) (pp-of-term (fst (spair-sigs p $q))-l p p) p) \neq$
lt (monom-mult (1 / punit.lc (rep-list q)) (pp-of-term (snd (spair-sigs p
$q))-l p q) q$ )
by (simp add: spair-sigs-def Let-def term-simps)
from $12\langle p \neq 0\rangle\langle q \neq 0\rangle$ show ?thesis
by (simp add: spair-alt-spair-sigs lt-monom-mult lt-minus-distinct-eq-max[OF *],
simp add: spair-sigs-def Let-def term-simps)
qed
lemma sig-of-spair-commute: sig-of-pair $(\operatorname{Inl}(p, q))=\operatorname{sig-of-pair}(\operatorname{Inl}(q, p))$

```
    by (simp add: spair-sigs-def Let-def lcs-comm ord-term-lin.max.commute)
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)
    using assms(1)
proof (induct bs)
    case Nil
    thus ?case by simp
next
    case (Cons a bs)
    from Cons(2) have b=a\veeb\in set bs by simp
    thus?case
    proof
        assume b=a
        from assms(2) show ?thesis by (simp add: <b = a〉)
    next
        assume b f set bs
        hence Inl (p,b)\in set (new-spairs bs p) by (rule Cons(1))
        thus ?thesis by simp
    qed
qed
lemma in-new-spairsD:
    assumes Inl (a,b)\in set (new-spairs bs p)
    shows }a=p\mathrm{ and }b\in\mathrm{ set bs and is-regular-spair pb
proof -
    from assms have a=p^b\in set bs ^is-regular-spair pb
    proof (induct bs)
    case Nil
    thus ?case by simp
    next
        case (Cons c bs)
        from Cons(2) have (is-regular-spair p c ^ Inl (a,b) = Inl (p,c))\vee Inl (a,b)
set (new-spairs bs p)
            by (simp split: if-split-asm)
        thus ?case
        proof
            assume is-regular-spair p c ^ Inl (a,b) = Inl ( p, c)
            hence is-regular-spair p c and a=p and b=c by simp-all
            thus ?thesis by simp
        next
            assume Inl (a,b) \in set (new-spairs bs p)
            hence }a=p\wedgeb\in\mathrm{ set bs ^ is-regular-spair p b by (rule Cons(1))
            thus ?thesis by simp
        qed
    qed
    thus }a=p\mathrm{ and }b\in\mathrm{ set bs and is-regular-spair p b by simp-all
qed
```

```
corollary in-new-spairs-iff:
    Inl (p,b) \in set (new-spairs bs p)\longleftrightarrow(b\in set bs ^ is-regular-spair p b)
    by (auto intro: in-new-spairsI dest: in-new-spairsD)
lemma Inr-not-in-new-spairs: Inr j & set (new-spairs bs p)
    by (induct bs, simp-all)
lemma sum-prodE:
    assumes \ab.p=\operatorname{Inl (a,b)\Longrightarrow thesis and }\j.p=Inr j\Longrightarrow thesis
    shows thesis
    using - assms(2)
proof (rule sumE)
    fix }
    assume p=Inl x
    moreover obtain a b where x=(a,b) by fastforce
    ultimately have p=Inl (a,b) by simp
    thus ?thesis by (rule assms(1))
qed
corollary in-new-spairsE:
    assumes q\in set (new-spairs bs p)
    obtains b}\mathrm{ where b set bs and is-regular-spair pb and q=Inl (p,b)
proof (rule sum-prodE)
    fix ab
    assume q: q = Inl (a,b)
    from assms have a=p and b\in set bs and is-regular-spair p b
        unfolding q by (rule in-new-spairsD)+
    note this(2, 3)
    moreover have q=Inl (p,b) by (simp only: q<a = p〉)
    ultimately show ?thesis ..
next
    fix }
    assume q=Inr j
    with assms show ?thesis by (simp add: Inr-not-in-new-spairs)
qed
lemma new-spairs-sorted: sorted-wrt pair-ord (new-spairs bs p)
proof (induct bs)
    case Nil
    show ?case by simp
next
    case (Cons a bs)
    moreover have transp pair-ord by (rule transpI, simp add: pair-ord-def)
    moreover have pair-ord x y v pair-ord y x for x y by (simp add: pair-ord-def
ord-term-lin.linear)
    ultimately show ?case by (simp add: sorted-wrt-insort-wrt)
qed
```


## lemma sorted-add-spairs

assumes sorted-wrt pair-ord ps
shows sorted-wrt pair-ord (add-spairs ps bs p)
unfolding add-spairs-def using - new-spairs-sorted assms
proof (rule sorted-merge-wrt)
show transp pair-ord by (rule transpI, simp add: pair-ord-def)
next
fix $x y$
show pair-ord $x y \vee$ pair-ord $y x$ by (simp add: pair-ord-def ord-term-lin.linear)
qed

## context

fixes rword-strict :: $\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow$ bool - Must be a strict rewrite order.
begin
qualified definition rword $::\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow$ 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 $u=\left(\exists v \in\right.$ set ss. $\left.v a d d s_{t} u\right)$
definition is-rewritable :: (' $\left.t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow{ }^{\prime} t \Rightarrow$ bool
where is-rewritable bs $p u=\left(\exists b \in\right.$ set bs. $b \neq 0 \wedge l t b a d d s_{t} u \wedge$ rword-strict (spp-of p) (spp-of b))
definition is-rewritable-spp :: (' $\left.t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)$ list $\Rightarrow\left(' t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow{ }^{\prime} t \Rightarrow$ bool
where is-rewritable-spp bs p $u=\left(\exists b \in\right.$ set bs. fst $b a d d s_{t} u \wedge$ rword-strict $\left.p b\right)$
fun sig-crit :: (' $\left.t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow{ }^{\prime} t$ list $\Rightarrow\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+\right.$ nat $) \Rightarrow$ bool where
sig-crit bs ss $(\operatorname{Inl}(p, q))=$
(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 $(\operatorname{Inr} j)=i s$-pred-syz ss (term-of-pair $(0, j))$
fun sig-crit' $::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+\right.$ nat $) \Rightarrow$ bool
where
sig-crit' bs $(\operatorname{Inl}(p, q))=$
(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 $(\operatorname{Inr} j)=i s$-syz-sig dgrad (term-of-pair $(0, j))$
fun sig-crit-spp :: $\left(^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)$ list $\Rightarrow{ }^{\prime} t$ list $\Rightarrow\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left(^{\prime} a\right.\right.\right.$ $\left.\left.\Rightarrow_{0}{ }^{\prime} b\right)\right)$ ) + 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\mathrm{ is-pred-syz ss }v\vee\mathrm{ is-rewritable-spp bs puマ is-rewritable-spp
bs qv)|
    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 \(r b-s p p-b o d y::\)
    \(\left(\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right.\right.\) list \(\times{ }^{\prime} t\) list \(\times\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+\)
nat) list) \(\times\) nat \() \Rightarrow\)
    \(\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right.\) list \(\times{ }^{\prime} t\) list \(\times\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)\)
+ nat) list) \(\times\) nat)
    where
    rb-spp-body \(((b s, s s,[]), z)=((b s, s s,[]), z) \mid\)
rb-spp-body \(((b s, s s, p \# p s), z)=\)
    (let \(s s^{\prime}=\) new-syz-sigs-spp ss bs \(p\) in
            if sig-crit-spp bs ss' \(p\) then
                    \(\left(\left(b s, s s^{\prime}, p s\right), z\right)\)
            else
                    let \(p^{\prime}=\) sig-trd-spp bs (spp-of-pair \(\left.p\right)\) in
                    if snd \(p^{\prime}=0\) then
                    \(\left(\left(b s, f s t p^{\prime} \# s s^{\prime}, p s\right)\right.\), Suc \(\left.z\right)\)
                        else
                            \(\left(\left(p^{\prime} \# b s, s s^{\prime}\right.\right.\), add-spairs-spp ps bs \(\left.\left.\left.p^{\prime}\right), z\right)\right)\)
```

definition rb-spp-aux ::
$\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right.$ list $\times{ }^{\prime} t$ list $\times\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+$
nat) list) $\times$ nat $) \Rightarrow$
$\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right.$ list $\times{ }^{\prime} t$ list $\times\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)$

+ nat) list) $\times$ nat)
where rb-spp-aux-def [code del]: rb-spp-aux $=$ tailrec.fun $(\lambda x$. snd $($ snd $(f s t x))$
$=[])(\lambda x . x)$ rb-spp-body
lemma rb-spp-aux-Nil [code]: rb-spp-aux ((bs,ss, []), z) $=((b s, s s,[]), z)$
by (simp add: rb-spp-aux-def tailrec.simps)
lemma rb-spp-aux-Cons [code]:
rb-spp-aux $((b s, s s, p \# p s), z)=r b-s p p-a u x(r b-s p p-b o d y((b s, s s, p \# p s), z))$
by (simp add: rb-spp-aux-def tailrec.simps)

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

## context

assumes rword-is-strict-rewrite-ord: is-strict-rewrite-ord rword-strict
assumes dgrad: dickson-grading dgrad
begin
lemma rword: is-rewrite-ord rword
unfolding rword-def using rword-is-strict-rewrite-ord by (rule is-strict-rewrite-ordD)

```
lemma sig-crit'-sym: sig-crit' bs (Inl ( }p,q))\Longrightarrow\mp@subsup{\operatorname{sig}}{}{\prime}\mathrm{ -crit' bs (Inl ( }q,p)
    by (auto simp: spair-sigs-def Let-def lcs-comm)
lemma is-rewritable-ConsD:
    assumes is-rewritable (b# bs)pu and u}\mp@subsup{\prec}{t}{lt b
    shows is-rewritable bs pu
proof -
    from assms(1) obtain b}\mp@subsup{b}{}{\prime}\mathrm{ where }\mp@subsup{b}{}{\prime}\in\operatorname{set}(b#bs)\mathrm{ and }\mp@subsup{b}{}{\prime}\not=0\mathrm{ and lt b}\mp@subsup{b}{}{\prime}add\mp@subsup{s}{t}{}
        and rword-strict (spp-of p) (spp-of b}\mathrm{ ) unfolding is-rewritable-def by blast
    from this(3) have lt b}\mp@subsup{b}{}{\prime}\mp@subsup{\preceq}{t}{}u\mathrm{ by (rule ord-adds-term)
    with assms(2) have }\mp@subsup{b}{}{\prime}\not=b\mathrm{ by auto
    with }\langle\mp@subsup{b}{}{\prime}\in\operatorname{set}(b#bs)\rangle\mathrm{ have }\mp@subsup{b}{}{\prime}\in\mathrm{ set bs by simp
    with }\langle\mp@subsup{b}{}{\prime}\not=0\rangle\langlelt \mp@subsup{b}{}{\prime}add\mp@subsup{s}{t}{}u\rangle\langlerword-strict (spp-of p) (spp-of b)\rangle show ?thesi
        by (auto simp: is-rewritable-def)
qed
lemma sig-crit'-ConsD:
    assumes sig-crit' (b # bs) p and sig-of-pair p \prec}\mp@subsup{}{t}{lt b
    shows sig-crit' bs p
proof (rule sum-prodE)
    fix }x
    assume p:p=\operatorname{Inl (x,y)}
    define }u\mathrm{ where }u=\mathrm{ fst (spair-sigs x y)
    define v}\mathrm{ where v=snd (spair-sigs x y)
    have sigs: spair-sigs x y = (u,v) by (simp add: u-def v-def)
    have}u\mp@subsup{\preceq}{t}{}\mathrm{ sig-of-pair p and v }\mp@subsup{\preceq}{t}{}\mathrm{ sig-of-pair p by (simp-all add: p sigs)
    hence }u\mp@subsup{\prec}{t}{lt b}\mathrm{ and v}\mp@subsup{\prec}{t}{lt b using assms(2) by simp-all
    with assms(1) show ?thesis by (auto simp: p sigs dest: is-rewritable-ConsD)
next
    fix }
    assume p:p=Inr j
    from assms show ?thesis by (simp add: p)
qed
definition rb-aux-inv1 :: ('t =>0 'b) list }=>\mathrm{ bool
    where rb-aux-inv1 bs =
                            (set bs\subseteqdgrad-sig-set dgrad ^0 # rep-list'set bs ^
                            sorted-wrt ( }\lambdaxy\mathrm{ y. lt }y\mp@subsup{\prec}{t}{}\mathrm{ lt }x\mathrm{ ) bs }
                    (\foralli<length bs. ᄀis-sig-red (}\mp@subsup{\prec}{t}{})(\preceq)(set (drop (Suc i) bs)) (bs!i))
                        ( }\forall<l<length bs
        ((\existsj<length fs.lt (bs ! i) =lt (monomial (1::'b) (term-of-pair ( 0, j))) ^
            punit.lt (rep-list (bs!i))\preceq punit.lt (rep-list (monomial 1 (term-of-pair
(0,j)))))\vee
            (\existsp\inset bs. \existsq\inset bs. is-regular-spair p q^ rep-list (spair p q) \not=0^
                    lt (bs!i)=lt (spair p q)^ punit.lt (rep-list (bs!i))\preceq punit.lt (rep-list
(spair p q))))) ^
                                    (\forall i<length bs. is-RB-upt dgrad rword (set (drop (Suc i) bs)) (lt (bs !
```

$i))$ )
fun rb-aux-inv :: (('t $\left.\Rightarrow_{0}{ }^{\prime} b\right)$ list $\times{ }^{\prime} t$ list $\times\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+\right.$ nat $)$ list $)$ $\Rightarrow$ bool
where rb-aux-inv (bs, ss, ps) $=$
(rb-aux-inv1 bs $\wedge$ $(\forall u \in$ set ss. is-syz-sig dgrad $u) \wedge$
$(\forall p$. Inl $(p, q) \in$ set $p s \longrightarrow(i s$-regular-spair $p q \wedge p \in$ set $b s \wedge q \in$ set bs)) $\wedge$
( $\forall j$. Inr $j \in$ set $p s \longrightarrow\left(j<\right.$ length $f s \wedge\left(\forall b \in\right.$ set bs. lt $b \prec_{t}$ term-of-pair $(0, j))) \wedge$
length $($ filter $(\lambda q$. sig-of-pair $q=$ term-of-pair $(0, j)) p s)$
$\leq 1) \wedge$
(sorted-wrt pair-ord ps) $\wedge$
( $\forall p \in$ set ps. $(\forall b 1 \in$ set bs. $\forall b 2 \in$ set bs. is-regular-spair b1 b2 $\longrightarrow$
sig-of-pair $p \prec_{t} l t($ spair b1 b2) $\longrightarrow(\operatorname{Inl}(b 1, b 2) \in$ set $p s \vee$ Inl $(b 2, b 1) \in \operatorname{set} p s)) \wedge$
$\left(\forall j<\right.$ length $f$ s. sig-of-pair $p \prec_{t}$ term-of-pair $(0, j) \longrightarrow \operatorname{Inr} j \in$ set $p s)) \wedge$
$\left(\forall b \in\right.$ set bs. $\forall p \in$ set ps. lt $b \preceq_{t}$ sig-of-pair $\left.p\right) \wedge$
( $\forall a \in$ set bs. $\forall b \in$ set bs. is-regular-spair $a b \longrightarrow \operatorname{Inl}(a, b) \notin$ set $p s \longrightarrow$ Inl $(b, a) \notin$ set $p s \longrightarrow$

$$
\neg \text { is-RB-in dgrad rword }(\text { set bs) }(\text { lt }(\text { spair a b) }) \longrightarrow
$$ $(\exists p \in$ set ps. sig-of-pair $p=l t($ spair a $b) \wedge \neg$ sig-crit' bs $p)) \wedge$ ( $\forall j<$ length $f$ s. Inr $j \notin$ set $p s \longrightarrow(i s-R B$-in dgrad rword (set bs) (term-of-pair $(0, j)) \wedge$

rep-list (monomial (1::'b) (term-of-pair $(0, j))) \in$ ideal (rep-list'set $b s))$ ))
lemmas $[$ simp del $]=$ rb-aux-inv.simps
lemma rb-aux-inv1-D1: rb-aux-inv1 bs $\Longrightarrow$ set bs $\subseteq$ dgrad-sig-set dgrad
by (simp add: rb-aux-inv1-def)
lemma rb-aux-inv1-D2: rb-aux-inv1 bs $\Longrightarrow 0 \notin$ rep-list'set bs
by (simp add: rb-aux-inv1-def)
lemma rb-aux-inv1-D3: rb-aux-inv1 bs $\Longrightarrow$ sorted-wrt $\left(\lambda x y\right.$. lt $y \prec_{t}$ lt $\left.x\right)$ bs
by (simp add: rb-aux-inv1-def)
lemma rb-aux-inv1-D4:
rb-aux-inv1 bs $\Longrightarrow i<$ length $b s \Longrightarrow \neg \operatorname{is-sig-red~}\left(\prec_{t}\right)(\preceq)($ set $($ drop $($ Suc $i) b s))$ (bs!i)
by (simp add: rb-aux-inv1-def)
lemma rb-aux-inv1-D5:
rb-aux-inv1 bs $\Longrightarrow i<$ length bs $\Longrightarrow$ is-RB-upt dgrad rword (set (drop (Suc i) $b s))(l t(b s!i))$
by (simp add: rb-aux-inv1-def)
lemma rb-aux-inv1-E:
assumes rb-aux-inv1 bs and $i<$ length bs
and $\bigwedge j . j<$ length $f s \Longrightarrow l t(b s!i)=l t$ (monomial $\left(1::{ }^{\prime} b\right)$ (term-of-pair ( 0 , $j))) \Longrightarrow$
punit.lt (rep-list (bs!i)) $\preceq$ punit.lt (rep-list (monomial 1 (term-of-pair $(0, j)))) \Longrightarrow$ thesis
and $\bigwedge p q . p \in$ set $b s \Longrightarrow q \in$ set $b s \Longrightarrow$ is-regular-spair $p q \Longrightarrow$ rep-list (spair $p q) \neq 0 \Longrightarrow$ $l t(b s!i)=l t(s p a i r p q) \Longrightarrow$ punit.lt (rep-list $(b s!i)) \preceq$ punit.lt (rep-list
$($ spair $p q)) \Longrightarrow$ thesis
shows thesis
using assms unfolding rb-aux-inv1-def by blast
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 (rule distinct-sorted-wrt-irrefl)
show irreflp $\left(\succ_{t}\right)$ by (simp add: irreflp-def)
next
show transp $\left(\succ_{t}\right)$ by (auto simp: transp-def)
next
from assms show sorted-wrt $\left(\succ_{t}\right)$ (map lt bs)
unfolding sorted-wrt-map conversep-iff by (rule rb-aux-inv1-D3)
qed
corollary rb-aux-inv1-lt-inj-on:
assumes rb-aux-inv1 bs
shows inj-on lt (set bs)
proof
fix $a b$
assume $a \in$ set $b s$
then obtain $i$ where $i: i<l e n g t h b s$ and $a: a=b s!i$ by (metis in-set-conv-nth)
assume $b \in$ set $b s$
then obtain $j$ where $j: j<$ length $b s$ and $b: b=b s!j$ by (metis in-set-conv-nth)
assume $l t a=l t b$
with $i j$ have (map lt bs) ! $i=($ map lt bs) ! $j$ by (simp add: a b)
moreover from assms have distinct (map lt bs) by (rule rb-aux-inv1-distinct-lt)
moreover from $i$ have $i<$ length (map lt bs) by simp
moreover from $j$ have $j<$ length (map lt bs) by simp
ultimately have $i=j$ by (simp only: nth-eq-iff-index-eq)
thus $a=b$ by (simp add: $a b$ )
qed
lemma canon-rewriter-unique:
assumes rb-aux-inv1 bs and is-canon-rewriter rword (set bs) ua
and $i s$-canon-rewriter rword (set bs) $u b$
shows $a=b$
proof -
from assms(1) have inj-on lt (set bs) by (rule rb-aux-inv1-lt-inj-on)
moreover from rword(1) assms(2, 3) have lt $a=l t b$ by (rule is-rewrite-ord-canon-rewriterD2)
moreover from assms(2) have $a \in$ set bs by (rule is-canon-rewriterD1)
moreover from $\operatorname{assms}(3)$ have $b \in$ set bs by (rule is-canon-rewriterD1)
ultimately show?thesis by (rule inj-onD)
qed
lemma rb-aux-inv-D1: rb-aux-inv $(b s, s s, p s) \Longrightarrow r b-a u x-i n v 1 b s$ by (simp add: rb-aux-inv.simps)
lemma rb-aux-inv-D2: rb-aux-inv $(b s, s s, p s) \Longrightarrow u \in$ set ss $\Longrightarrow$ is-syz-sig dgrad u
by (simp add: rb-aux-inv.simps)
lemma rb-aux-inv-D3:
assumes rb-aux-inv (bs, ss, ps) and $\operatorname{Inl}(p, q) \in$ set ps
shows $p \in$ set bs and $q \in$ set bs and is-regular-spair $p q$
using assms by (simp-all add: rb-aux-inv.simps)
lemma rb-aux-inv-D4:
assumes rb-aux-inv (bs, ss, ps) and Inr $j \in$ set ps
shows $j<$ length $f s$ and $\bigwedge b . b \in$ set $b s \Longrightarrow l t b \prec_{t}$ term-of-pair $(0, j)$
and length (filter ( $\lambda$ q. sig-of-pair $q=$ term-of-pair $(0, j)) p s) \leq 1$
using assms by (simp-all add: rb-aux-inv.simps)
lemma rb-aux-inv-D5: rb-aux-inv (bs, ss, ps) $\Longrightarrow$ sorted-wrt pair-ord $p s$ by (simp add: rb-aux-inv.simps)
lemma rb-aux-inv-D6-1:
assumes rb-aux-inv (bs, ss, ps) and $p \in$ set $p s$ and $b 1 \in$ set $b s$ and $b 2 \in$ set bs and is-regular-spair b1 b2 and sig-of-pair $p \prec_{t}$ lt (spair b1 b2)
obtains Inl $(b 1, b 2) \in$ set $p s \mid \operatorname{Inl}(b 2, b 1) \in$ set $p s$
using assms unfolding rb-aux-inv.simps by blast
lemma rb-aux-inv-D6-2:
rb-aux-inv $(b s, s s, p s) \Longrightarrow p \in$ set $p s \Longrightarrow j<$ length $f s \Longrightarrow$ sig-of-pair $p \prec_{t}$
term-of-pair $(0, j) \Longrightarrow$
Inr $j \in$ set $p s$
by (simp add: rb-aux-inv.simps)
lemma rb-aux-inv-D7: rb-aux-inv $(b s, s s, p s) \Longrightarrow b \in$ set $b s \Longrightarrow p \in$ set $p s \Longrightarrow$ lt $b \preceq_{t}$ sig-of-pair $p$
by (simp add: rb-aux-inv.simps)
lemma rb-aux-inv-D8:
assumes rb-aux-inv (bs,ss, ps) and $a \in$ set $b s$ and $b \in s e t$ bs and is-regular-spair
$a b$
and $\operatorname{Inl}(a, b) \notin$ set $p s$ and $\operatorname{Inl}(b, a) \notin$ set $p s$ and $\neg i s$-RB-in dgrad rword (set bs) (lt (spair ab))
obtains $p$ where $p \in$ set $p s$ and sig-of-pair $p=l t($ spair $a b)$ and $\neg$ sig-crit ${ }^{\prime}$ bs $p$
using assms unfolding rb-aux-inv.simps by meson
lemma rb-aux-inv-D9:
assumes rb-aux-inv (bs, ss, ps) and $j<$ length fs and Inr $j \notin$ set ps
shows is-RB-in dgrad rword (set bs) (term-of-pair ( $0, j$ ))
and rep-list (monomial ( $\left.1::{ }^{\prime} b\right)$ (term-of-pair $\left.\left.(0, j)\right)\right) \in$ ideal (rep-list' set bs)
using assms by (simp-all add: rb-aux-inv.simps)
lemma rb-aux-inv-is-RB-upt:
assumes $r b$-aux-inv $(b s, s s, p s)$ and $\bigwedge p . p \in$ set $p s \Longrightarrow u \preceq_{t}$ sig-of-pair $p$
shows is-RB-upt dgrad rword (set bs) u
proof -
from $\operatorname{assms}(1)$ have inv1: rb-aux-inv1 bs by (rule rb-aux-inv-D1)
from dgrad rword(1) show ?thesis
proof (rule is-RB-upt-finite)
from inv1 show set bs $\subseteq$ dgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
next
from inv1 show inj-on lt (set bs) by (rule rb-aux-inv1-lt-inj-on)
next
show finite (set bs) by (fact finite-set)
next
fix $g 1 g 2$
assume 1: g1 $\in$ set bs and 2: g2 $\in$ set bs and 3: is-regular-spair g1 g2
and 4: lt (spair g1 g2) $\prec_{t} u$
have 5: $p \notin$ set $p s$ if sig-of-pair $p=l t$ (spair g1 g2) for $p$
proof
assume $p \in$ set $p s$
hence $u \preceq_{t}$ sig-of-pair $p$ by (rule assms(2))
also have $\ldots \prec_{t} u$ unfolding that by (fact 4)
finally show False ..
qed
show is-RB-in dgrad rword (set bs) (lt (spair g1 g2))
proof (rule ccontr)
note $\operatorname{assms}(1) 123$
moreover have Inl $(g 1, g 2) \notin$ set ps by (rule 5, rule sig-of-spair, fact 3)
moreover have Inl $(g 2, g 1) \notin$ set $p s$
by (rule 5, simp only: sig-of-spair-commute, rule sig-of-spair, fact 3)
moreover assume $\neg i s-R B$-in dgrad rword (set bs) (lt (spair g1 g2))
ultimately obtain $p$ where $p \in$ set $p s$ and sig-of-pair $p=l t(s p a i r ~ g 1 ~ g 2) ~$
by (rule rb-aux-inv-D8)
from this(2) have $p \notin$ set $p s$ by (rule 5)
thus False using $\langle p \in$ set $p s\rangle$..
qed
next
fix $j$
assume 1: term-of-pair $(0, j) \prec_{t} u$
note assms(1)
moreover assume $j<$ length fs
moreover have Inr $j \notin$ set ps
proof
assume Inr $j \in$ set ps
hence $u \preceq_{t}$ sig-of-pair (Inr $j$ ) by (rule assms(2))
also have $\ldots \prec_{t} u$ by (simp add: 1)
finally show False ..
qed
ultimately show is-RB-in dgrad rword (set bs) (term-of-pair $(0, j)$ ) by (rule rb-aux-inv-D9)
qed
qed
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)
using assms
proof (rule rb-aux-inv-is-RB-upt)
fix $q$
assume $q \in \operatorname{set}(p \# p s)$
hence $q=p \vee q \in$ set $p$ s by simp
thus sig-of-pair $p \preceq_{t}$ sig-of-pair $q$
proof
assume $q=p$
thus ?thesis by simp
next
assume $q \in$ set $p s$
moreover from assms have sorted-wrt pair-ord ( $p \# p s$ ) by (rule rb-aux-inv-D5)
ultimately show ?thesis by (simp add: pair-ord-def)
qed
qed
lemma Inr-in-tailD:
assumes rb-aux-inv (bs, ss, p\#ps) and Inr $j \in$ set ps
shows sig-of-pair $p \neq$ term-of-pair $(0, j)$
proof
assume eq: sig-of-pair $p=$ term-of-pair $(0, j)$
from $\operatorname{assms}$ (2) have $\operatorname{Inr} j \in \operatorname{set}(p \# p s)$ by simp
let $? P=\lambda q$. sig-of-pair $q=$ term-of-pair $(0, j)$
from $\operatorname{assms}(2)$ obtain $i 1$ where $i 1<$ length ps and Inrj: Inr $j=p s!i 1$
by (metis in-set-conv-nth)
from $\operatorname{assms}(1)\langle\operatorname{Inr} j \in \operatorname{set}(p \# p s)\rangle$ have length $($ filter ? $P(p \# p s)) \leq 1$ by (rule rb-aux-inv-D4)
moreover from 〈i1<length ps〉 have Suc i1 < length ( $p \# p s$ ) by simp
moreover have $0<$ length ( $p \# p s$ ) by simp
moreover have ?P (( $p$ \# ps)! Suc i1) by (simp add: Inrj[symmetric])

```
    moreover have ?P (( \(p\) \# ps)!0) by (simp add: eq)
    ultimately have Suc i1 \(=0\) by (rule length-filter-le-1)
    thus False ..
qed
lemma pair-list-aux:
    assumes rb-aux-inv (bs, ss, ps) and \(p \in\) set ps
    shows sig-of-pair \(p=l t\) (poly-of-pair \(p) \wedge\) poly-of-pair \(p \neq 0 \wedge\) poly-of-pair \(p \in\)
dgrad-sig-set dgrad
proof (rule sum-prodE)
    fix \(a b\)
    assume \(p: p=\operatorname{Inl}(a, b)\)
    from \(\operatorname{assms}(1)\) have rb-aux-inv1 bs by (rule rb-aux-inv-D1)
    hence bs-sub: set bs \(\subseteq\) dgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
    from assms have is-regular-spair a \(b\) unfolding \(p\) by (rule rb-aux-inv-D3)
    hence sig-of-pair \(p=l t\) (poly-of-pair \(p\) ) and poly-of-pair \(p \neq 0\)
    unfolding \(p\) poly-of-pair.simps by (rule sig-of-spair, rule is-regular-spair-nonzero)
    moreover from dgrad have poly-of-pair \(p \in d g r a d\)-sig-set dgrad unfolding \(p\)
poly-of-pair.simps
    proof (rule dgrad-sig-set-closed-spair)
    from assms have \(a \in\) set bs unfolding \(p\) by (rule rb-aux-inv-D3)
    thus \(a \in d g r a d\)-sig-set dgrad using \(b s\)-sub ..
    next
            from assms have \(b \in\) set bs unfolding \(p\) by (rule rb-aux-inv-D3)
            thus \(b \in d g r a d\)-sig-set dgrad using bs-sub ..
    qed
    ultimately show ?thesis by simp
next
    fix \(j\)
    assume \(p=\operatorname{Inr} j\)
    from assms have \(j<\) length fs unfolding \(\langle p=\operatorname{Inr} j\rangle\) by (rule rb-aux-inv-D4)
    have monomial 1 (term-of-pair \((0, j)) \in\) dgrad-sig-set dgrad
    by (rule dgrad-sig-set-closed-monomial, simp add: pp-of-term-of-pair dgrad-max-0,
                simp add: component-of-term-of-pair \(\langle j<\) length fs〉)
    thus ?thesis by (simp add: \(\langle p=\) Inr \(j\rangle\) lt-monomial monomial-0-iff)
qed
corollary pair-list-sig-of-pair:
    rb-aux-inv (bs, ss, ps) \(\Longrightarrow p \in\) set \(p s \Longrightarrow\) sig-of-pair \(p=l t\) (poly-of-pair \(p\) )
    by (simp add: pair-list-aux)
corollary pair-list-nonzero: rb-aux-inv (bs, ss, ps) \(\Longrightarrow p \in\) set \(p s \Longrightarrow\) poly-of-pair
\(p \neq 0\)
    by (simp add: pair-list-aux)
corollary pair-list-dgrad-sig-set:
    rb-aux-inv (bs, ss, ps) \(\Longrightarrow p \in\) set \(p s \Longrightarrow\) poly-of-pair \(p \in\) dgrad-sig-set dgrad
    by (simp add: pair-list-aux)
```

```
lemma is-rewritableI-is-canon-rewriter:
    assumes rb-aux-inv1 bs and b\in set bs and b\not=0 and lt b addst
        and \negis-canon-rewriter rword (set bs) ub
    shows is-rewritable bs b u
proof -
    from assms(2-5) obtain }\mp@subsup{b}{}{\prime}\mathrm{ where }\mp@subsup{b}{}{\prime}\in\mathrm{ set bs and }\mp@subsup{b}{}{\prime}\not=0\mathrm{ and lt b}\mp@subsup{b}{}{\prime}add\mp@subsup{s}{t}{}
        and 1: ᄀ rword (spp-of b
    show ?thesis unfolding is-rewritable-def
    proof (intro bexI conjI)
        from rword(1) have 2: rword (spp-of b) (spp-of b
        proof (rule is-rewrite-ordD3)
            assume rword (spp-of b}\mathrm{ ) (spp-of b)
            with 1 show ?thesis ..
        qed
        from rword(1) 1 have b\not= b' by (auto dest: is-rewrite-ordD1)
        have lt b}\not=lt b
        proof
            assume lt b=lt b
            with rb-aux-inv1-lt-inj-on[OF assms(1)] have b= b' using assms(2) < 'b
set bs>
            by (rule inj-onD)
            with }<b\not=\mp@subsup{b}{}{\prime}\rangle\mathrm{ show False ..
        qed
        hence fst (spp-of b)}\not=fst (spp-of b') by (simp add: spp-of-def
        with rword-is-strict-rewrite-ord 2 show rword-strict (spp-of b) (spp-of b')
            by (auto simp: rword-def dest: is-strict-rewrite-ord-antisym)
    qed fact+
qed
lemma is-rewritableD-is-canon-rewriter:
    assumes rb-aux-inv1 bs and is-rewritable bs bu
    shows \neg is-canon-rewriter rword (set bs) u b
proof
    assume is-canon-rewriter rword (set bs) ub
    hence b\in set bs and b}=0\mathrm{ and lt baddst u
        and 1: \a.a\in set bs\Longrightarrowa\not=0\Longrightarrowlt a addst u\Longrightarrowrword (spp-of a) (spp-of
b)
    by (rule is-canon-rewriterD)+
    from assms(2) obtain b}\mp@subsup{b}{}{\prime}\mathrm{ where }\mp@subsup{b}{}{\prime}\in\mathrm{ set bs and }\mp@subsup{b}{}{\prime}\not=0\mathrm{ and lt b}\mp@subsup{b}{}{\prime}add\mp@subsup{s}{t}{}
        and 2: rword-strict (spp-of b) (spp-of b') unfolding is-rewritable-def by blast
    from this(1, 2, 3) have rword (spp-of b') (spp-of b) by (rule 1)
    moreover from rword-is-strict-rewrite-ord 2 have rword (spp-of b) (spp-of b
        unfolding rword-def by (rule is-strict-rewrite-ord-asym)
    ultimately have fst (spp-of b}\mp@subsup{b}{}{\prime})=fst (spp-of b) by (rule is-rewrite-ordD4[OF
rword])
    hence lt b}\mp@subsup{b}{}{\prime}=lt b by (simp add: spp-of-def
    with rb-aux-inv1-lt-inj-on[OF assms(1)] have }\mp@subsup{b}{}{\prime}=b\mathrm{ using }\langle\mp@subsup{b}{}{\prime}\in\mathrm{ set bs><b}
set bs>
        by (rule inj-onD)
```

from rword-is-strict-rewrite-ord have $\neg$ rword-strict (spp-of b) (spp-of b')
unfolding $\left\langle b^{\prime}=b\right\rangle$ by (rule is-strict-rewrite-ord-irrefl)
thus False using 2 ..
qed
lemma lemma-12:
assumes rb-aux-inv (bs, ss, ps) and is-RB-upt dgrad rword (set bs) u
and dgrad (pp-of-term $u$ ) $\leq$ dgrad-max dgrad and is-canon-rewriter rword (set bs) $u a$
and $\neg$ is-syz-sig dgrad $u$ and is-sig-red $\left(\prec_{t}\right)(=)($ set bs) (monom-mult 1
(pp-of-term $u-l p a) a)$
obtains $p q$ where $p \in$ set bs and $q \in$ set bs and is-regular-spair $p q$ and $l t$ $($ spair $p q)=u$
and $\neg \operatorname{sig}$-crit ${ }^{\prime}$ bs $(\operatorname{Inl}(p, q))$
proof -
from $\operatorname{assms}(1)$ have inv1: rb-aux-inv1 bs by (rule rb-aux-inv-D1)
hence inj: inj-on lt (set bs) by (rule rb-aux-inv1-lt-inj-on)
from $\operatorname{assms}(4)$ have $l t$ a adds $s_{t} u$ by (rule is-canon-rewriterD3)
hence $l p$ a adds pp-of-term $u$ and comp-a: component-of-term (lt a) =compo-nent-of-term u
by (simp-all add: adds-term-def)
let ?s $=p p$-of-term $u-l p a$
let $? a=$ monom-mult 1 ?s a
from $\operatorname{assms}(4)$ have $a \in$ set bs by (rule is-canon-rewriterD1)
from $\operatorname{assms}(6)$ have rep-list $? a \neq 0$ using is-sig-red-top-addsE by blast
hence rep-list $a \neq 0$ by (auto simp: rep-list-monom-mult)
hence $a \neq 0$ by (auto simp: rep-list-zero)
hence $l t ? a=$ ?s $\oplus l t$ a by (simp add: lt-monom-mult)
also from $\langle l p$ a adds pp-of-term $u\rangle$ have eq0: ... $=u$
by (simp add: splus-def comp-a adds-minus term-simps)
finally have $l t ? a=u$.
note dgrad rword(1)
moreover from $\operatorname{assms}(2)$ have $i s$ - $R B$-upt dgrad rword (set bs) (lt ?a) by (simp only: $\langle l t ? a=u\rangle)$
moreover from dgrad have ?a $\in d g r a d$-sig-set dgrad
proof (rule dgrad-sig-set-closed-monom-mult)
from dgrad $\langle l p$ a adds pp-of-term $u\rangle$ have dgrad (pp-of-term $u-l p a) \leq d g r a d$ (pp-of-term u)
by (rule dickson-grading-minus)
 le-trans)
next
from inv1 have set bs $\subseteq$ dgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
with $\langle a \in$ set $b s\rangle$ show $a \in$ dgrad-sig-set dgrad ..
qed
ultimately obtain $v b$ where $v \prec_{t} l t ? a$ and dgrad (pp-of-term $\left.v\right) \leq d g r a d-m a x$ dgrad
and component-of-term $v<l e n g t h ~ f s$ and $n s: \neg i s-s y z-s i g$ dgrad $v$
and $v: v=($ punit.lt $($ rep-list ? $a)-$ punit.lt $($ rep-list $b)) \oplus l t b$
and cr: is-canon-rewriter rword (set bs) vb and is-sig-red $\left(\prec_{t}\right)(=)\{b\} ? a$ using assms(6) by (rule lemma-11)
from this(6) have $b \in$ set bs by (rule is-canon-rewriterD1)
with $\langle a \in$ set $b s\rangle$ show ?thesis
proof
from dgrad rword(1) assms(2) inj assms (5, 4) $\left\langle b \in\right.$ set bs〉〈is-sig-red $\left(\prec_{t}\right)(=)$
\{b\} ? a ${ }^{\text {b }}$ assms (3)
show is-regular-spair a by (rule lemma-9(3))
next
from dgrad rword(1) assms(2) inj $\operatorname{assms}(5,4)\langle b \in \operatorname{set} b s\rangle\left\langle i s-\operatorname{sig}-r e d\left(\prec_{t}\right)(=)\right.$
$\{b\}$ ? a> assms (3)
show $l t$ (spair a $b$ ) $=u$ by (rule lemma-9(4))
next
from $\langle$ rep-list $a \neq 0\rangle$ have $v^{\prime}: v=(? s+$ punit.lt (rep-list a) - punit.lt (rep-list b)) $\oplus l t b$
by (simp add: v rep-list-monom-mult punit.lt-monom-mult)
moreover from dgrad rword(1) assms(2) inj assms(5, 4) $\langle b \in$ set bs $\langle i s$-sig-red $\left(\prec_{t}\right)(=)\{b\}$ ?a> assms(3)
have lcs $($ punit.lt $($ rep-list a) $)($ punit.lt $($ rep-list b) $)-$ punit.lt $($ rep-list $a)=$ ?s and lcs $($ punit.lt $($ rep-list a) $)($ punit.lt $($ rep-list b) $)-$ punit.lt $($ rep-list $b)=$ ?s + punit.lt (rep-list a) - punit.lt (rep-list b)
by (rule lemma-9)+
ultimately have eq1: spair-sigs a $b=(u, v)$ by (simp add: spair-sigs-def eq0)
show $\neg$ sig-crit' bs $(\operatorname{Inl}(a, b))$
proof (simp add: eq1 assms(5) ns, intro conjI notI)
assume is-rewritable bs a u
with inv1 have $\neg i s$-canon-rewriter rword (set bs) uaby (rule is-rewritableD-is-canon-rewriter)
thus False using assms(4)..
next
assume is-rewritable bs $b v$
with inv1 have $\neg i s$-canon-rewriter rword (set bs) vb by (rule is-rewritableD-is-canon-rewriter) thus False using cr ..
qed
qed
qed
lemma is-canon-rewriterI-eq-sig:
assumes rb-aux-inv1 bs and $b \in$ set bs
shows is-canon-rewriter rword (set bs) (lt b) $b$
proof -
from $\operatorname{assms}(2)$ have rep-list $b \in$ rep-list'set bs by (rule imageI)
moreover from assms(1) have $0 \notin$ rep-list'set bs by (rule rb-aux-inv1-D2)
ultimately have $b \neq 0$ by (auto simp: rep-list-zero)
with assms(2) show ?thesis
proof (rule is-canon-rewriterI)
fix $a$
assume $a \in$ set $b s$ and $a \neq 0$ and $l t a a d d s_{t} l t b$
from assms(2) obtain $i$ where $i<l e n g t h ~ b s$ and $b: b=b s!i$ by (metis in-set-conv-nth)
from $\operatorname{assms}(1)$ this(1) have is-RB-upt dgrad rword (set (drop (Suc i)bs)) (lt ( $b s!i)$ ) by (rule rb-aux-inv1-D5)
with dgrad have is-sig-GB-upt dgrad (set (drop (Suc i)bs)) (lt (bs!i)) by (rule is-RB-upt-is-sig-GB-upt)
hence is-sig-GB-upt dgrad (set (drop (Suc i) bs)) (lt b) by (simp only: b)
moreover have set (drop (Suc i) bs) $\subseteq$ set bs by (rule set-drop-subset)
moreover from assms (1) have set bs $\subseteq$ dgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
ultimately have is-sig-GB-upt dgrad (set bs) (lt b) by (rule is-sig-GB-upt-mono)
with rword (1) dgrad show rword (spp-of a) (spp-of b)
proof (rule is-rewrite-ordD5)
from $\operatorname{assms}(1)\left\langle i<\right.$ length bs〉 have $\neg$ is-sig-red $\left(\prec_{t}\right)(\preceq)($ set (drop (Suc i) $b s))(b s!i)$
by (rule rb-aux-inv1-D4)
hence $\neg$ is-sig-red $\left(\prec_{t}\right)(=)($ set $(\operatorname{drop}(S u c i) b s)) b$ by (simp add: b is-sig-red-top-tail-cases)
moreover have $\neg i s$-sig-red $\left(\prec_{t}\right)(=)(\operatorname{set}($ take $($ Suc i)bs)) b
proof
assume $i s$-sig-red $\left(\prec_{t}\right)(=)($ set $($ take $(S u c i) b s)) b$
then obtain $f$ where $f$-in: $f \in \operatorname{set}($ take $(S u c i) b s)$ and is-sig-red $\left(\prec_{t}\right)$
(=) $\{f\} b$
by (rule is-sig-red-singletonI)
from this(2) have lt $f \prec_{t} l t b$ by (rule is-sig-red-regular $D$-lt)
from $\langle i<$ length $b s\rangle$ have take-eq: take $(S u c i) b s=($ take $i b s) @[b]$ unfolding $b$ by (rule take-Suc-conv-app-nth)
from assms(1) have sorted-wrt ( $\lambda x$ y. lt $y \prec_{t}$ lt $\left.x\right)$ ((take (Suc i)bs) @
(drop (Suc i) bs))
unfolding append-take-drop-id by (rule rb-aux-inv1-D3)
hence 1: $\bigwedge y . y \in \operatorname{set}($ take $i b s) \Longrightarrow l t b \prec_{t}$ lt $y$
by (simp add: sorted-wrt-append take-eq del: append-take-drop-id)
from $f$-in have $f=b \vee f \in$ set (take $i$ bs) by (simp add: take-eq)
hence $l t b \preceq_{t} l t f$
proof
assume $f \in$ set (take $i$ bs)
hence lt $b \prec_{t}$ lt $f$ by (rule 1)
thus?thesis by simp
qed simp
with $\left\langle l t f \prec_{t} l t b\right\rangle$ show False by simp
qed
ultimately have $\neg$ is-sig-red $\left(\prec_{t}\right)(=)($ set (take $($ Suc i) bs) $\cup$ set (drop (Suc
i) $b s$ )) $b$
by (simp add: is-sig-red-Un)
thus $\neg$ is-sig-red $\left(\prec_{t}\right)(=)($ set bs) b by (metis append-take-drop-id set-append)
qed fact+
qed (simp add: term-simps)
qed
lemma not-sig-crit:
assumes rb-aux-inv (bs, ss, p\#ps) and $\neg$ sig-crit bs (new-syz-sigs ss bs p) p
and $b \in$ set $b s$
shows $l t b \prec_{t}$ sig-of-pair $p$
proof (rule sum-prodE)
fix $x y$
assume $p: p=\operatorname{Inl}(x, y)$
have $p \in \operatorname{set}(p \# p s)$ by $\operatorname{simp}$
hence Inl $(x, y) \in \operatorname{set}(p \# p s)$ by (simp only: $p$ )
define $t 1$ where $t 1=$ punit.lt (rep-list $x$ )
define t2 where t2 $=$ punit.lt (rep-list y)
define $u$ where $u=$ fst (spair-sigs $x y$ )
define $v$ where $v=$ snd (spair-sigs $x y$ )
have $u: u=($ lcs t1 t2 - t1 $) \oplus l t x$ by (simp add: $u$-def spair-sigs-def t1-def t2-def Let-def)
have $v: v=($ lcs t1 t2 - t2 $) \oplus l t y$ by (simp add: v-def spair-sigs-def t1-def t2-def Let-def)
have spair-sigs: spair-sigs $x y=(u, v)$ by (simp add: u-def $v$-def)
with assms(2) have $\neg$ is-rewritable bs $x u$ and $\neg i s$-rewritable bs y $v$ by (simp-all add: p)
from $\operatorname{assms}(1)\langle\operatorname{Inl}(x, y) \in \operatorname{set}(p \# p s)\rangle$ have $x$-in: $x \in$ set bs and $y$-in: $y \in$ set bs and is-regular-spair $x$ y by (rule rb-aux-inv-D3)+
from $\operatorname{assms}(1)$ have inv1: rb-aux-inv1 bs by (rule rb-aux-inv-D1)
from inv1 have $0 \notin$ rep-list' set bs by (rule rb-aux-inv1-D2)
with $x$-in $y$-in have rep-list $x \neq 0$ and rep-list $y \neq 0$ by auto
hence $x \neq 0$ and $y \neq 0$ by (auto simp: rep-list-zero)
from inv1 have sorted: sorted-wrt $\left(\lambda x y\right.$. lt $y \prec_{t}$ lt $\left.x\right)$ bs by (rule rb-aux-inv1-D3)
from $x$-in obtain $i 1$ where $i 1<l e n g t h ~ b s$ and $x: x=b s$ ! i1 by (metis in-set-conv-nth)
from $y$-in obtain $i 2$ where $i 2<$ length $b s$ and $y: y=b s!i 2$ by (metis in-set-conv-nth)
have lt $b \neq$ sig-of-pair $p$
proof
assume $l t$-b: lt $b=$ sig-of-pair $p$
from inv1 have crw: is-canon-rewriter rword (set bs) (lt b) busing assms(3) by (rule is-canon-rewriterI-eq-sig)
show False
proof (rule ord-term-lin.linorder-cases)
assume $u \prec_{t} v$
hence $l t b=v$ by (auto simp: lt-b $p$ spair-sigs ord-term-lin.max-def)
with crw have crw-b: is-canon-rewriter rword (set bs) vb by simp
from $v$ have $l t y a d d s_{t} v$ by (rule adds-termI)
hence is-canon-rewriter rword (set bs) $v y$
using inv1 $y$-in $\langle y \neq 0\rangle\langle\neg i s$-rewritable bs $y v\rangle$ is-rewritableI-is-canon-rewriter
by blast
with inv1 crw-b have $b=y$ by (rule canon-rewriter-unique)
with $\langle l t b=v\rangle$ have $l t y=v$ by $\operatorname{simp}$
from inv1 $\langle i 2<$ length $b s\rangle$ have $\neg$ is-sig-red $\left(\prec_{t}\right)(\preceq)$ (set (drop (Suc i2)
bs)) (bs!i2)
by (rule rb-aux-inv1-D4)

```
moreover have is-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq) (set (drop (Suc i2) bs)) (bs!i2
proof (rule is-sig-red-singletonD)
    have is-sig-red ( }\mp@subsup{\prec}{t}{})(=){x}
    proof (rule is-sig-red-top-addsI)
        from <lt y = v> have (lcs t1 t2 - t2) \oplus lt y = lt y by (simp only: v)
        also have ... = 0 \oplus lt y by (simp only: term-simps)
    finally have lcs t1 t2 - t2 = 0 by (simp only: splus-right-canc)
            hence lcs t1 t2 = t2 by (metis (full-types) add.left-neutral adds-minus
adds-lcs-2)
            with adds-lcs[of t1 t2] show punit.lt (rep-list x) adds punit.lt (rep-list y)
                by (simp only: t1-def t2-def)
    next
            from }\langleu\mp@subsup{\prec}{t}{}v\rangle\mathrm{ show punit.lt (rep-list y) }\opluslt x\mp@subsup{\prec}{t}{}\mathrm{ punit.lt (rep-list x)}
lt y
                    by (simp add: t1-def t2-def u v term-is-le-rel-minus-minus adds-lcs
adds-lcs-2)
    qed (simp|fact)+
    thus is-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq){x}(bs!iQ) by (simp add:y is-sig-red-top-tail-cases
    next
        have lt }x\mp@subsup{\preceq}{t}{}0\opluslt x by (simp only: term-simps
        also have ... \preceq}t u unfolding u using zero-min by (rule splus-mono-left
        also have ... }\mp@subsup{\prec}{t}{}v\mathrm{ by fact
            finally have *:lt (bs!i1) \prec}\mp@subsup{\mp@code{t}}{l}{lt}(bs!i2) by (simp only:<lt y = v>x
y[symmetric])
        have i2 < i1
        proof (rule linorder-cases)
            assume i1<i2
            with sorted have lt (bs!i2) \prec}\mp@subsup{}{t}{lt}(bs!i1) using<i2 < length bs>
            by (rule sorted-wrt-nth-less)
            with * show ?thesis by simp
        next
            assume i1 = i2
            with * show ?thesis by simp
        qed
        hence Suc i2 \leqi1 by simp
            thus }x\in\mathrm{ set (drop (Suc i2) bs) unfolding x using <i1 < length bs` by
(rule nth-in-set-dropI)
    qed
    ultimately show ?thesis ..
    next
        assume v}\mp@subsup{\prec}{t}{}
        hence lt b=u by (auto simp:lt-b p spair-sigs ord-term-lin.max-def)
        with crw have crw-b: is-canon-rewriter rword (set bs) u b by simp
    from }u\mathrm{ have lt x addst }u\mathrm{ by (rule adds-termI)
    hence is-canon-rewriter rword (set bs) ux
    using inv1 x-in <x\not=0\rangle\langle\neg is-rewritable bs x u〉 is-rewritableI-is-canon-rewriter
by blast
    with inv1 crw-b have b=x by (rule canon-rewriter-unique)
    with <lt b = u` have lt x = u by simp
```

```
    from inv1 <i1 < length bs` have ᄀ is-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq) (set (drop (Suc i1
bs))(bs!i1)
    by (rule rb-aux-inv1-D4)
    moreover have is-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq)(set (drop (Suc i1) bs)) (bs!i1
    proof (rule is-sig-red-singletonD)
    have is-sig-red ( }\mp@subsup{\prec}{t}{})(=){y}
    proof (rule is-sig-red-top-addsI)
        from <lt x = u` have (lcs t1 t2 - t1) \oplus lt x=lt x by (simp only:u)
        also have ... = 0 \opluslt x by (simp only: term-simps)
        finally have lcs t1 t2 - t1 = 0 by (simp only: splus-right-canc)
            hence lcs t1 t2 = t1 by (metis (full-types) add.left-neutral adds-minus
adds-lcs)
            with adds-lcs-2[of t2 t1] show punit.lt (rep-list y) adds punit.lt (rep-list
x)
            by (simp only: t1-def t2-def)
        next
            from }\langlev\mp@subsup{\prec}{t}{}u\rangle\mathrm{ show punit.lt (rep-list x) }\oplus\mathrm{ lt }y\mp@subsup{\prec}{t}{}\mathrm{ punit.lt (rep-list y)}
lt x
                    by (simp add: t1-def t2-def u v term-is-le-rel-minus-minus adds-lcs
adds-lcs-2)
            qed (simp|fact)+
    thus is-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq){y}(bs!i1) by (simp add: x is-sig-red-top-tail-cases
    next
        have lt y \preceq\preceqt 0 \oplus lt y by (simp only: term-simps)
        also have ... \preceq\preceqt v unfolding v using zero-min by (rule splus-mono-left)
        also have ... }\mp@subsup{\prec}{t}{}u\mathrm{ by fact
            finally have *:lt (bs!i2) \prec}\mp@subsup{t}{t}{lt}(bs!i1) by (simp only:<lt x = u〉 y
x[symmetric])
    have i1 < i2
    proof (rule linorder-cases)
            assume i2 < i1
            with sorted have lt (bs!i1) \prec}\mp@subsup{l}{l}{lt}(bs!i2) using <i1 < length bs
                by (rule sorted-wrt-nth-less)
            with * show ?thesis by simp
            next
                assume i2 = i1
                with * show ?thesis by simp
            qed
            hence Suc i1 \leqi2 by simp
            thus y fet (drop (Suc i1) bs) unfolding y using <i2 < length bs> by
(rule nth-in-set-dropI)
    qed
    ultimately show ?thesis ..
    next
    assume u=v
    hence punit.lt (rep-list x)\opluslt y= punit.lt (rep-list y) \opluslt x
    by (simp add: t1-def t2-def u v term-is-le-rel-minus-minus adds-lcs adds-lcs-2)
    moreover from〈is-regular-spair x y>
        have punit.lt (rep-list y) \opluslt x\not= punit.lt (rep-list x) \oplus lt y by (rule
```

```
is-regular-spairD3)
            ultimately show ?thesis by simp
            qed
    qed
    moreover from assms(1, 3)<p\in set (p# ps)> have lt b \preceq. sig-of-pair p by
(rule rb-aux-inv-D7)
    ultimately show ?thesis by simp
next
    fix }
    assume p: p= Inr j
    have Inr j \in set (p# ps) by (simp add: p)
    with assms(1) have lt b}\mp@subsup{\prec}{t}{}\mathrm{ term-of-pair (0, j) using assms(3) by (rule rb-aux-inv-D4)
    thus ?thesis by (simp add: p)
qed
context
    assumes fs-distinct: distinct fs
    assumes fs-nonzero: 0 & set fs
begin
lemma rep-list-monomial': rep-list (monomial 1 (term-of-pair (0, j))) = ((fs!j)
when j < length fs)
    by (simp add: rep-list-monomial fs-distinct term-simps)
lemma new-syz-sigs-is-syz-sig:
    assumes rb-aux-inv (bs, ss, p # ps) and v\in set (new-syz-sigs ss bs p)
    shows is-syz-sig dgrad v
proof (rule sum-prodE)
    fix ab
    assume p=Inl (a,b)
    with assms(2) have v\in set ss by simp
    with assms(1) show ?thesis by (rule rb-aux-inv-D2)
next
    fix }
    assume p:p=Inr j
    let ?f = \lambdab. term-of-pair (punit.lt (rep-list b), j)
    let ?a = monomial (1::'b) (term-of-pair (0,j))
    from assms(1) have inv1: rb-aux-inv1 bs by (rule rb-aux-inv-D1)
    have Inr j\in set (p# ps) by (simp add: p)
    with assms(1) have j< length fs by (rule rb-aux-inv-D4)
    hence a: rep-list ?a = fs ! j by (simp add:rep-list-monomial')
    show ?thesis
    proof (cases is-pot-ord)
    case True
    with assms(2) have v\inset (filter-min-append (addst) ss (filter-min (addst)
(map ?f bs)))
            by (simp add: p)
            hence v}\mathrm{ & set ss U ?f' set bs using filter-min-append-subset filter-min-subset
by fastforce
```


## thus ?thesis

## proof

assume $v \in$ set ss
with assms(1) show ?thesis by (rule rb-aux-inv-D2)
next
assume $v \in$ ?f ' set bs
then obtain $b$ where $b \in$ set $b s$ and $v=$ ?f $b$..
have comp-b: component-of-term (lt b) < component-of-term (lt ?a)
proof (rule ccontr)
have $*:$ pp-of-term (term-of-pair $(0, j)) \preceq p p$-of-term (lt b)
by (simp add: pp-of-term-of-pair zero-min)
assume $\neg$ component-of-term (lt b) < component-of-term (lt ?a)
hence component-of-term (term-of-pair $(0, j)) \leq$ component-of-term (lt b)
by (simp add: lt-monomial)
with $*$ have term-of-pair $(0, j) \preceq_{t} l t b$ by (rule ord-termI)
moreover from $\operatorname{assms}(1)\langle I n r j \in \operatorname{set}(p \# p s)\rangle\langle b \in$ set $b s\rangle$ have $l t b \prec_{t}$
term-of-pair ( $0, j$ )
by (rule rb-aux-inv-D4)
ultimately show False by simp
qed
have $v=$ punit.lt (rep-list b) $\oplus l t$ ?a
by (simp add: $\langle v=$ ?f b〉 a lt-monomial splus-def term-simps)
also have $\ldots=$ ord-term-lin.max $($ punit.lt $($ rep-list $b) \oplus l t ? a)($ punit.lt (rep-list
?a) $\oplus l t b)$

## proof -

have component-of-term (punit.lt (rep-list ?a) $\oplus l t b)=$ component-of-term
(lt b)
by (simp only: term-simps)
also have ... < component-of-term (lt ?a) by (fact comp-b)
also have $\ldots=$ component-of-term $($ punit.lt $($ rep-list $b) \oplus l t ? a)$
by (simp only: term-simps)
finally have component-of-term (punit.lt (rep-list ?a) $\oplus l t b)<$ component-of-term (punit.lt (rep-list b) $\oplus l t$ ?a) .
with True have punit.lt (rep-list ?a) $\oplus l t b \prec_{t}$ punit.lt (rep-list b) $\oplus l t$ ?a
by (rule is-pot-ordD)
thus ?thesis by (auto simp: ord-term-lin.max-def)
qed
finally have $v: v=$ ord-term-lin.max (punit.lt $($ rep-list b) $\oplus l t$ ?a) (punit.lt $($ rep-list ? $a) \oplus l t b)$.
show ?thesis unfolding $v$ using dgrad
proof (rule Koszul-syz-is-syz-sig)
from inv1 have set bs $\subseteq$ dgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
with $\langle b \in$ set $b s\rangle$ show $b \in d g r a d-s i g-s e t ~ d g r a d ~ . . ~$
next
show $? a \in d g r a d-s i g-s e t ~ d g r a d$
by (rule dgrad-sig-set-closed-monomial, simp-all add: term-simps dgrad-max-0 $\langle j<$ length $f s\rangle$ )
next
from inv1 have $0 \notin$ rep-list' set bs by (rule rb-aux-inv1-D2)

```
            with }\langleb\in\mathrm{ set bs> show rep-list b}\not=0\mathrm{ by fastforce
            next
            from <j< length fs> have fs ! j E set fs by (rule nth-mem)
            with fs-nonzero show rep-list ?a }\not=0\mathrm{ by (auto simp: a)
            qed (fact comp-b)
        qed
    next
        case False
    with assms(2) have v\in set ss by (simp add: p)
    with assms(1) show ?thesis by (rule rb-aux-inv-D2)
    qed
qed
lemma new-syz-sigs-minimal:
    assumes }\bigwedge\mp@subsup{u}{}{\prime}\mp@subsup{v}{}{\prime}.\mp@subsup{u}{}{\prime}\in\mathrm{ set ss # v
    assumes u\in set (new-syz-sigs ss bs p) and v\in set (new-syz-sigs ss bs p) and
u addst v
    shows }u=
proof (rule sum-prodE)
    fix ab
    assume p:p=\operatorname{Inl}(a,b)
    from assms(2, 3) have }u\in\mathrm{ set ss and ve set ss by (simp-all add: p)
    thus ?thesis using assms(4) by (rule assms(1))
next
    fix }
    assume p:p=Inr j
    show ?thesis
    proof (cases is-pot-ord)
        case True
        have transp (addst) by (rule transpI, drule adds-term-trans)
            define ss' where ss' = filter-min (addst) (map ( }\lambda\mathrm{ b . term-of-pair (punit.lt
(rep-list b), j)) bs)
    note assms(1)
    moreover have }\mp@subsup{u}{}{\prime}=\mp@subsup{v}{}{\prime}\mathrm{ if }\mp@subsup{u}{}{\prime}\in\operatorname{set}s\mp@subsup{s}{}{\prime}\mathrm{ and }\mp@subsup{v}{}{\prime}\in\operatorname{set ss}\mp@subsup{}{}{\prime}\mathrm{ and }\mp@subsup{u}{}{\prime}\mathrm{ addst v}\mp@subsup{v}{}{\prime}\mathrm{ for }\mp@subsup{u}{}{\prime
v
            using <transp (addst)> that unfolding ss'-def by (rule filter-min-minimal)
    moreover from True assms(2, 3) have u\in set (filter-min-append (addst) ss
ss')
            and v\in set (filter-min-append (addst) ss ss') by (simp-all add: p ss'-def)
            ultimately show ?thesis using assms(4) by (rule filter-min-append-minimal)
    next
        case False
        with assms(2, 3) have }u\in\mathrm{ set ss and veset ss by (simp-all add: p)
        thus ?thesis using assms(4) by (rule assms(1))
    qed
qed
lemma new-syz-sigs-distinct:
    assumes distinct ss
```

```
    shows distinct (new-syz-sigs ss bs p)
proof (rule sum-prodE)
    fix ab
    assume p = Inl (a,b)
    with assms show ?thesis by simp
next
    fix }
    assume p:p=Inr j
    show ?thesis
    proof (cases is-pot-ord)
        case True
            define ss' where ss' = filter-min (addst) (map (\lambdab. term-of-pair (punit.lt
(rep-list b), j)) bs)
    from adds-term-refl have reflp (adds}\mp@subsup{)}{t}{})\mathrm{ by (rule reflpI)
    moreover note assms
    moreover have distinct ss' unfolding ss'-def using <reflp (addst)> by (rule
filter-min-distinct)
    ultimately have distinct (filter-min-append (addst) ss ss') by (rule filter-min-append-distinct)
    thus ?thesis by (simp add: p ss'-def True)
    next
        case False
        with assms show ?thesis by (simp add: p)
    qed
qed
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
proof -
    have rl: is-syz-sig dgrad u
        if is-pred-syz (new-syz-sigs ss bs p) u and dgrad (pp-of-term u) \leqdgrad-max
dgrad for u
    proof -
        from that(1) obtain s where s\in set (new-syz-sigs ss bs p) and adds: s addst
u
            unfolding is-pred-syz-def ..
    from assms(1) this(1) have is-syz-sig dgrad s by (rule new-syz-sigs-is-syz-sig)
    with dgrad show ?thesis using adds that(2) by (rule is-syz-sig-adds)
    qed
    from assms(1) have rb-aux-inv1 bs by (rule rb-aux-inv-D1)
    hence bs-sub: set bs\subseteqdgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
    show ?thesis
    proof (rule sum-prodE)
    fix ab
    assume p:p=\operatorname{Inl}(a,b)
    hence Inl (a,b) \in set (p# ps) by simp
    with assms(1) have }a\in\mathrm{ set bs and b}\mathrm{ set bs by (rule rb-aux-inv-D3)+
        with bs-sub have a-in: a \indgrad-sig-set dgrad and b-in: b\indgrad-sig-set
dgrad by fastforce+
```

define $t 1$ where $t 1=$ punit.lt (rep-list a)
define $t 2$ where $t 2=$ punit.lt (rep-list b)
define $u$ where $u=$ fst (spair-sigs a b)
define $v$ where $v=$ snd (spair-sigs a b)
from dgrad a-in have dgrad t1 $\leq$ dgrad-max dgrad unfolding $t 1$-def by (rule dgrad-sig-setD-rep-list-lt)
moreover from dgrad b-in have dgrad t2 $\leq$ dgrad-max dgrad
unfolding t2-def by (rule dgrad-sig-setD-rep-list-lt)
ultimately have ord-class.max (dgrad t1) (dgrad t2) $\leq$ dgrad-max dgrad by $\operatorname{simp}$
with dickson-grading-lcs[OF dgrad] have dgrad (lcs t1 t2) $\leq$ dgrad-max dgrad by (rule le-trans)
have $u: u=($ lcs t1 t2 - t1 $) \oplus l t ~ a ~ b y ~(s i m p ~ a d d: ~ u-d e f ~ s p a i r-s i g s-d e f ~ t 1-d e f ~$ t2-def Let-def)
have $v: v=(l c s t 1$ t2 $-t 2) \oplus l t b$ by (simp add: $v$-def spair-sigs-def t1-def t2-def Let-def)
have 1: spair-sigs a $b=(u, v)$ by (simp add: $u$-def $v$-def $)$
from assms(2) have (is-pred-syz (new-syz-sigs ss bs p) $u \vee$ is-pred-syz (new-syz-sigs ss bs $p) v) \vee$
(is-rewritable bs a $u \vee i s$-rewritable bs $b v$ ) by (simp add: $p$ 1)
thus ?thesis
proof
assume is-pred-syz (new-syz-sigs ss bs $p$ ) $u \vee$ is-pred-syz (new-syz-sigs ss bs p) $v$
thus ?thesis
proof
assume is-pred-syz (new-syz-sigs ss bs p) u
moreover have dgrad (pp-of-term u) $\leq$ dgrad-max dgrad
proof (simp add: u term-simps dickson-gradingD1[OF dgrad], rule)
from dgrad adds-lcs have dgrad (lcs t1 t2 - t1) $\leq$ dgrad (lcs t1 t2)
by (rule dickson-grading-minus)
also have $\ldots \leq$ dgrad-max dgrad by fact
finally show dgrad (lcs t1 t2 - t1) $\leq$ dgrad-max dgrad.
next

qed
ultimately have is-syz-sig dgrad $u$ by (rule rl)
thus ?thesis by (simp add: p 1)
next
assume is-pred-syz (new-syz-sigs ss bs p) v
moreover have dgrad (pp-of-term $v$ ) $\leq$ dgrad-max dgrad
proof (simp add: v term-simps dickson-gradingD1[OF dgrad], rule)
from dgrad adds-lcs-2 have dgrad (lcs t1 t2 - t2) $\leq$ dgrad (lcs t1 t2)
by (rule dickson-grading-minus)
also have $\ldots \leq$ dgrad-max dgrad by fact
finally show dgrad (lcs t1 t2 - t2) $\leq$ dgrad-max dgrad.
next
from $b$-in show dgrad (lp b) $\leq$ dgrad-max dgrad by (rule dgrad-sig-setD-lp) qed

```
            ultimately have is-syz-sig dgrad v by (rule rl)
            thus ?thesis by (simp add: p 1)
        qed
    next
        assume is-rewritable bs a u\vee is-rewritable bs b v
        thus ?thesis by (simp add: p 1)
    qed
next
    fix }
    assume p=Inr j
    with assms(2) have is-pred-syz (new-syz-sigs ss bs p) (term-of-pair (0, j)) by
simp
    moreover have dgrad (pp-of-term (term-of-pair ( }0,j)))\leqdgrad-max dgrad
        by (simp add: pp-of-term-of-pair dgrad-max-0)
    ultimately have is-syz-sig dgrad (term-of-pair ( O, j)) by (rule rl)
    thus ?thesis by (simp add: <p = Inr j>)
    qed
qed
lemma rb-aux-inv-preserved-0:
    assumes rb-aux-inv (bs, ss, p # ps)
    and \s.s set ss'}\Longrightarrow is-syz-sig dgrad 
    and \ab.a\in set bs \Longrightarrowb\in set bs \Longrightarrow is-regular-spair a b \ Inl (a,b)\not\in
set ps \Longrightarrow
        Inl (b,a) & set ps \Longrightarrow\neg is-RB-in dgrad rword (set bs) (lt (spair a b))\Longrightarrow
        \existsq\inset ps. sig-of-pair q=lt (spair a b) ^\neg sig-crit' bs q
    and }\bigwedgej.j< length fs \Longrightarrowp= Inr j\Longrightarrow Inr j\not\in set ps \Longrightarrow is-RB-in dgrad
rword (set bs) (term-of-pair (0, j)) ^
        rep-list (monomial 1 (term-of-pair (0,j))) \in ideal (rep-list'set bs)
    shows rb-aux-inv (bs, s\mp@subsup{s}{}{\prime},ps)
proof -
    from assms(1) have rb-aux-inv1 bs by (rule rb-aux-inv-D1)
    show ?thesis unfolding rb-aux-inv.simps
    proof (intro conjI ballI allI impI)
        fix }
        assume s\in set ss'
        thus is-syz-sig dgrad s by (rule assms(2))
    next
        fix q1 q2
        assume Inl (q1, q2) \in set ps
        hence Inl (q1, q2) \in set ( }p#\mathrm{ # ps) by simp
        with assms(1) show is-regular-spair q1 q2 and q1 \in set bs and q2 \in set bs
            by (rule rb-aux-inv-D3)+
    next
        fix }
        assume Inr j\in set ps
        hence Inr j fet (p# ps) by simp
            with assms(1) have j< length fs and length (filter (\lambdaq. sig-of-pair q =
term-of-pair (0,j)) (p# ps)) \leq 1
```

by（rule rb－aux－inv－D4）＋
have length（filter $(\lambda q$ ．sig－of－pair $q=$ term－of－pair $(0, j)) p s) \leq$
length（filter $(\lambda q$ ．sig－of－pair $q=$ term－of－pair $(0, j))(p \# p s))$ by simp also have $\ldots \leq 1$ by fact
finally show length（filter $(\lambda q$ ．sig－of－pair $q=$ term－of－pair $(0, j)) p s) \leq 1$ ． show $j<$ length $f s$ by fact
fix $b$
assume $b \in$ set $b s$
with $\operatorname{assms}(1)\langle\operatorname{Inr} j \in \operatorname{set}(p \# p s)\rangle$ show lt $b \prec_{t}$ term－of－pair $(0, j)$ by（rule rb－aux－inv－D4）
next
from assms（1）have sorted－wrt pair－ord（ $p \# p s$ ）by（rule rb－aux－inv－D5）
thus sorted－wrt pair－ord ps by simp
next
fix $q$
assume $q \in$ set $p s$
from assms（1）have sorted－wrt pair－ord（ $p \# p s$ ）by（rule rb－aux－inv－D5）
hence $\bigwedge p^{\prime} . p^{\prime} \in$ set $p s \Longrightarrow$ sig－of－pair $p \preceq_{t}$ sig－of－pair $p^{\prime}$ by（simp add：
pair－ord－def）
with $\langle q \in$ set $p s\rangle$ have 1：sig－of－pair $p \preceq_{t}$ sig－of－pair $q$ by blast
\｛
fix $b 1$ b2
note assms（1）
moreover from $\langle q \in$ set $p s\rangle$ have $q \in \operatorname{set}(p \# p s)$ by simp
moreover assume $b 1 \in$ set bs and $b 2 \in$ set bs and is－regular－spair b1 b2 and 2：sig－of－pair $q \prec_{t}$ lt（spair b1 b2）
ultimately show $\operatorname{Inl}(b 1, b 2) \in$ set $p s \vee \operatorname{Inl}(b 2, b 1) \in$ set $p s$
proof（rule rb－aux－inv－D6－1）
assume Inl $(b 1, b 2) \in \operatorname{set}(p \# p s)$
moreover from 12 have sig－of－pair $p \prec_{t} l t$（spair b1 b2）by simp
ultimately have $\operatorname{Inl}(b 1, b 2) \in$ set $p s$
by（auto simp：sig－of－spair〈is－regular－spair b1 b2〉 simp del：sig－of－pair．simps）
thus ？thesis ．．
next
assume $\operatorname{Inl}(b 2, b 1) \in \operatorname{set}(p \# p s)$
moreover from 12 have sig－of－pair $p \prec_{t} l t$（spair b1 b2）by simp
ultimately have $\operatorname{Inl}(b 2, b 1) \in$ set $p s$
by（auto simp：sig－of－spair 〈is－regular－spair b1 b2〉 sig－of－spair－commute simp del：sig－of－pair．simps）
thus ？thesis ．．
qed
\}
\｛
fix $j$
note assms（1）
moreover from $\langle q \in$ set $p s\rangle$ have $q \in$ set（ $p \# p s$ ）by simp
moreover assume $j<$ length $f s$ and 2：sig－of－pair $q \prec_{t}$ term－of－pair（ $0, j$ ）
ultimately have $\operatorname{Inr} j \in$ set（ $p \# p s$ ）by（rule rb－aux－inv－D6－2）

```
        moreover from 12 have sig-of-pair p < < sig-of-pair (Inr j) by simp
        ultimately show Inr j\in set ps by auto
    }
    next
    fix b q
    assume b\in set bs and q\in set ps
    hence b\in set bs and q\in set ( }p#\mathrm{ #s) by simp-all
    with assms(1) show lt b}\mp@subsup{\preceq}{t}{}\mathrm{ sig-of-pair q by (rule rb-aux-inv-D7)
    next
    fix }
    assume j< length fs and Inr j & set ps
    have is-RB-in dgrad rword (set bs) (term-of-pair ( }0,j))
                rep-list (monomial 1 (term-of-pair (0, j))) \in ideal (rep-list'set bs)
    proof (cases p = Inr j)
        case True
        with «j<length fs` show ?thesis using \Inr j & set ps` by (rule assms(4))
    next
            case False
        with «Inr j\not\in set ps` have Inr j\not\in set ( p# ps) by simp
        with assms(1)<j < length fs` rb-aux-inv-D9 show ?thesis by blast
    qed
    thus is-RB-in dgrad rword (set bs) (term-of-pair (0,j))
        and rep-list (monomial }1\mathrm{ (term-of-pair (0,j))) E ideal (rep-list'set bs) by
simp-all
    qed (fact, rule assms(3))
qed
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)
proof -
    from assms(1) have rb-aux-inv1 bs by (rule rb-aux-inv-D1)
    hence bs-sub: set bs \subseteqdgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
    from assms(1, 2) have sig-crit' bs p by (rule sig-crit'I-sig-crit)
    from assms(1) show ?thesis
    proof (rule rb-aux-inv-preserved-0)
        fix }
        assume s}\in\mathrm{ set (new-syz-sigs ss bs p)
        with assms(1) show is-syz-sig dgrad s by (rule new-syz-sigs-is-syz-sig)
    next
    fix ab
    assume 1:a\in set bs and 2:b\in set bs and 3:is-regular-spair a b and 4:Inl
(a,b)\not\in set ps
            and 5: Inl (b,a)\not\in set ps and 6:\neg is-RB-in dgrad rword (set bs) (lt (spair
a b))
    from assms(1, 2) have sig-crit' bs p by (rule sig-crit'I-sig-crit)
    show \existsq\inset ps. sig-of-pair q=lt (spair a b) ^\neg sig-crit' bs q
    proof (cases p=Inl (a,b)\vee p=Inl (b,a))
            case True
```

```
    hence sig-of-p:lt (spair ab) \(=\) sig-of-pair \(p\)
    proof
    assume \(p: p=\operatorname{Inl}(a, b)\)
    from 3 show ?thesis by (simp only: p sig-of-spair)
    next
    assume \(p: p=\operatorname{Inl}(b, a)\)
    from 3 have is-regular-spair \(b a\) by (rule is-regular-spair-sym)
    thus ?thesis by (simp only: p sig-of-spair spair-comm[of a] lt-uminus)
    qed
    note assms(1)
    moreover have is-RB-upt dgrad rword (set bs) (lt (spair a b)) unfolding
sig-of-p
    using assms(1) by (rule rb-aux-inv-is-RB-upt-Cons)
    moreover have dgrad (lp (spair a b)) \(\leq\) dgrad-max dgrad
    proof (rule dgrad-sig-setD-lp, rule dgrad-sig-set-closed-spair, fact dgrad)
        from \(\langle a \in\) set \(b s\rangle\) bs-sub show \(a \in d g r a d\)-sig-set dgrad ..
    next
        from \(\langle b \in\) set \(b s\rangle b s\)-sub show \(b \in\) dgrad-sig-set dgrad ..
    qed
    moreover obtain \(c\) where crw: is-canon-rewriter rword (set bs) (lt (spair a
b)) \(c\)
    proof (rule ord-term-lin.linorder-cases)
        from 3 have rep-list \(b \neq 0\) by (rule is-regular-spairD2)
        moreover assume punit.lt (rep-list b) \(\oplus l t a \prec_{t}\) punit.lt (rep-list \(\left.a\right) \oplus l t b\)
        ultimately have lt (spair ba) \(=(\) lcs (punit.lt (rep-list b)) (punit.lt (rep-list
a)) - punit.lt (rep-list b)) \(\oplus l t b\)
        by (rule lt-spair)
            hence lt (spair ab) \(=(\) lcs \((\) punit.lt \((\) rep-list b) \()(\) punit.lt \((\) rep-list a) \()-\)
        punit.lt \((\) rep-list \(b)) \oplus l t b\)
            by (simp add: spair-comm[of a])
            hence lt \(b\) addst \(l t\) (spair a b) by (rule adds-termI)
            from 〈rep-list \(b \neq 0\rangle\) have \(b \neq 0\) by (auto simp: rep-list-zero)
            show ?thesis by (rule is-rewrite-ord-finite-canon-rewriterE, fact rword, fact
finite-set, fact+)
    next
            from 3 have rep-list \(a \neq 0\) by (rule is-regular-spairD1)
            moreover assume punit.lt (rep-list a) \(\oplus l t b \prec_{t}\) punit.lt \((\) rep-list b) \(\oplus l t a\)
            ultimately have lt (spair a b) \(=(\) lcs (punit.lt (rep-list a) ) (punit.lt (rep-list
                b)) - punit.lt (rep-list a)) \(\oplus l t\) a
            by (rule lt-spair)
            hence lt a addst lt (spair a b) by (rule adds-termI)
            from 〈rep-list \(a \neq 0\rangle\) have \(a \neq 0\) by (auto simp: rep-list-zero)
            show ?thesis by (rule is-rewrite-ord-finite-canon-rewriterE, fact rword, fact
finite-set, fact+)
    next
            from 3 have punit.lt (rep-list b) \(\oplus l t a \neq\) punit.lt (rep-list a) \(\oplus l t b\)
                by (rule is-regular-spairD3)
            moreover assume punit.lt (rep-list b) \(\oplus l t a=\) punit.lt (rep-list \(a) \oplus l t b\)
            ultimately show ?thesis ..
```

qed
moreover from 6 have $\neg$ is-syz-sig dgrad (lt (spair a b)) by (simp add: is-RB-in-def)
moreover from 6 crw have $i s$-sig-red $\left(\prec_{t}\right)(=)($ set bs) (monom-mult 1 ( $l p$ $($ spair $a b)-l p c) c$ )
by (simp add: is-RB-in-def)
ultimately obtain $x y$ where 7: $x \in$ set bs and 8: $y \in$ set bs and 9: is-regular-spair $x$ y
and 10: lt (spair $x y)=l t($ spair a b) and 11: $\neg$ sig-crit' bs $(\operatorname{Inl}(x, y))$
by (rule lemma-12)
from this(5) 〈sig-crit' bs $p\rangle$ have $\operatorname{Inl}(x, y) \neq p$ and $\operatorname{Inl}(y, x) \neq p$
by (auto simp only: sig-crit'-sym)
show ?thesis
proof (cases Inl $(x, y) \in$ set $p s \vee \operatorname{Inl}(y, x) \in$ set ps)
case True
thus?thesis
proof
assume $\operatorname{Inl}(x, y) \in$ set ps
show ? thesis
proof (intro bexI conjI)
show sig-of-pair $(\operatorname{Inl}(x, y))=l t$ (spair a b) by (simp only: sig-of-spair
9 10)
qed fact+
next
assume $\operatorname{Inl}(y, x) \in$ set ps
show ?thesis
proof (intro bexI conjI)
from 9 have is-regular-spair y $x$ by (rule is-regular-spair-sym)
thus sig-of-pair $(\operatorname{Inl}(y, x))=l t($ spair a $b)$
by (simp only: sig-of-spair spair-comm[of y] lt-uminus 10)
next
from 11 show $\neg$ sig-crit' bs $(\operatorname{Inl}(y, x))$ by (auto simp only: sig-crit'-sym)
qed fact
qed
next
case False
note $\operatorname{assms}(1) 789$
moreover from False $\langle\operatorname{Inl}(x, y) \neq p\rangle\langle\operatorname{Inl}(y, x) \neq p\rangle$ have $\operatorname{Inl}(x, y) \notin$ set ( $p$ \# ps)
and $\operatorname{Inl}(y, x) \notin \operatorname{set}(p \# p s)$ by auto
moreover from 6 have $\neg$ is-RB-in dgrad rword (set bs) (lt (spair $x y$ )) by (simp add: 10)
ultimately obtain $q$ where 12: $q \in \operatorname{set}(p \# p s)$ and 13: sig-of-pair $q=$ lt (spair $x y$ )
and 14: $\neg$ sig-crit' bs $q$ by (rule rb-aux-inv-D8)
from 1214 〈sig-crit' bs $p\rangle$ have $q \in$ set $p s$ by auto
with 1314 show ?thesis unfolding 10 by blast
qed
next
case False
with 45 have $\operatorname{Inl}(a, b) \notin \operatorname{set}(p \# p s)$ and $\operatorname{Inl}(b, a) \notin \operatorname{set}(p \# p s)$ by auto
with $\operatorname{assms}(1) 123$ obtain $q$ where 7：$q \in \operatorname{set}(p \# p s)$ and 8：sig－of－pair $q=l t($ spair $a b)$
and 9：$\neg$ sig－crit＇bs $q$ using 6 by（rule rb－aux－inv－D8）
from 79 〈sig－crit＇$b s p\rangle$ have $q \in$ set $p s$ by auto
with 89 show ？thesis by blast
qed
next
fix $j$
assume $j<$ length $f s$
assume $p: p=\operatorname{Inr} j$
with 〈sig－crit＇bs p〉 have is－syz－sig dgrad（term－of－pair $(0, j)$ ）by simp
hence is－RB－in dgrad rword（set bs）（term－of－pair $(0, j)$ ）by（rule is－RB－inI2）
moreover have rep－list（monomial 1 （term－of－pair $(0, j))$ ）ideal（rep－list ‘ set bs）
proof（rule sig－red－zero－idealI，rule syzygy－crit）
from assms（1）have is－RB－upt dgrad rword（set bs）（sig－of－pair p） by（rule rb－aux－inv－is－RB－upt－Cons）
with dgrad have is－sig－GB－upt dgrad（set bs）（sig－of－pair p）
by（rule is－RB－upt－is－sig－GB－upt）
thus is－sig－GB－upt dgrad（set bs）（term－of－pair $(0, j))$ by（simp add：p）
next
show monomial 1 （term－of－pair $(0, j)) \in$ dgrad－sig－set dgrad
by（rule dgrad－sig－set－closed－monomial，simp－all add：term－simps dgrad－max－0 $\langle j<$ length fs $\rangle$ ）
next
show lt（monomial $(1:: ' b)($ term－of－pair $(0, j)))=$ term－of－pair $(0, j)$ by （simp add：lt－monomial）
qed（fact dgrad，fact）
ultimately show 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 $b s)$ ．．
qed
qed
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 $p, p s$ ）
proof－
let $? p=$ sig－trd bs（poly－of－pair $p$ ）
have 0：$\left(\text { sig－red }\left(\prec_{t}\right)(\preceq)(\text { set bs })\right)^{* *}($ poly－of－pair $p)$ ？p
by（rule sig－trd－red－rtrancl）
hence eq：lt ？$p=l t$（poly－of－pair $p$ ）by（rule sig－red－regular－rtrancl－lt）
from $\operatorname{assms}(1)$ have inv1：rb－aux－inv1 bs by（rule rb－aux－inv－D1）
have $*$ ：is－syz－sig dgrad（lt（poly－of－pair p））
proof（rule is－syz－sigI）
have poly－of－pair $p \neq 0$ by（rule pair－list－nonzero，fact，simp）

```
    hence lc (poly-of-pair p)\not=0 by (rule lc-not-0)
    moreover from 0 have lc ?p = lc (poly-of-pair p) by (rule sig-red-regular-rtrancl-lc)
    ultimately have lc ? p}\not=0\mathrm{ by simp
    thus ?p p=0 by (simp add:lc-eq-zero-iff)
    next
    note dgrad(1)
    moreover from inv1 have set bs\subseteqdgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
    moreover have poly-of-pair p \in dgrad-sig-set dgrad by (rule pair-list-dgrad-sig-set,
fact, simp)
    ultimately show ?p }\in\mathrm{ dgrad-sig-set dgrad using 0 by (rule dgrad-sig-set-closed-sig-red-rtrancl)
    qed (fact eq, fact assms(2))
    hence rb: is-RB-in dgrad rword (set bs) (lt (poly-of-pair p)) by (rule is-RB-inI2)
    from assms(1) show ?thesis
    proof (rule rb-aux-inv-preserved-0)
        fix }
        assume s\in set (lt ?p # new-syz-sigs ss bs p)
    hence s=lt (poly-of-pair p)\vees\in set (new-syz-sigs ss bs p) by (simp add: eq)
    thus is-syz-sig dgrad s
    proof
        assume s=lt (poly-of-pair p)
        with * show ?thesis by simp
    next
        assume s\in set (new-syz-sigs ss bs p)
        with assms(1) show ?thesis by (rule new-syz-sigs-is-syz-sig)
    qed
next
    fix ab
    assume 1:a\in set bs and 2: b\in set bs and 3:is-regular-spair a b and 4:Inl
(a,b) & set ps
    and 5: Inl (b,a)\not\in set ps and 6: ᄀ is-RB-in dgrad rword (set bs) (lt (spair a
b))
    have p}\in\operatorname{set}(p#ps) by sim
            with assms(1) have sig-of-p: sig-of-pair p = lt (poly-of-pair p) by (rule
pair-list-sig-of-pair)
    from rb 6 have neq: lt (poly-of-pair p) #lt (spair a b) by auto
    hence }p\not=\operatorname{Inl}(a,b)\mathrm{ and }p\not=\operatorname{Inl}(b,a)\mathrm{ by (auto simp: spair-comm[of a])
    with 4 5 have Inl ( }a,b)\not\in\operatorname{set (p# ps) and Inl ( b,a) & set ( p# ps) by auto
    with assms(1) 12 3 obtain q where 7: q\in set ( }p#ps)\mathrm{ and 8: sig-of-pair
q=lt(spair a b)
            and 9:\neg sig-crit' bs q using 6 by (rule rb-aux-inv-D8)
    from this(1, 2) neq have q\in set ps by (auto simp: sig-of-p)
    thus \existsq\inset ps. sig-of-pair q=lt (spair a b)^\neg sig-crit' bs qusing 8 9 by
blast
    next
        fix }
        assume j< length fs
        assume p:p=Inr j
        from rb have is-RB-in dgrad rword (set bs) (term-of-pair ( }0,j\mathrm{ )) by (simp add:
plt-monomial)
```

moreover have rep-list (monomial 1 (term-of-pair $(0, j))$ ) ideal (rep-list ‘ set bs)
proof (rule sig-red-zero-ideall, rule sig-red-zeroI)
from 0 show $\left(\text { sig-red }\left(\prec_{t}\right)(\preceq)(\text { set bs })\right)^{* *}($ monomial 1 (term-of-pair $\left.(0, j))\right)$
? $p$ by (simp add: $p$ )
qed fact
ultimately show 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) ..
qed
qed
lemma rb-aux-inv-preserved-3:
assumes rb-aux-inv (bs, ss, $p \# p s)$ 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 $l t$ (sig-trd bs (poly-of-pair p)) $\notin l t$ 'set bs
proof -
have $p \in \operatorname{set}(p \# p s)$ by $\operatorname{simp}$
with assms(1) have sig-of-p: sig-of-pair $p=l t($ poly-of-pair $p)$
and $p$-in: poly-of-pair $p \in$ dgrad-sig-set dgrad
by (rule pair-list-sig-of-pair, rule pair-list-dgrad-sig-set)
define $p^{\prime}$ where $p^{\prime}=$ sig-trd bs (poly-of-pair $p$ )
from assms(1) have inv1: rb-aux-inv1 bs by (rule rb-aux-inv-D1)
hence bs-sub: set bs $\subseteq$ dgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
have $p$-red: $\left(\text { sig-red }\left(\prec_{t}\right)(\preceq)(\text { set bs) })\right)^{* *}\left(\right.$ poly-of-pair p) $p^{\prime}$
and $p^{\prime}$-irred: $\neg$ is-sig-red $\left(\prec_{t}\right)(\preceq)\left(\right.$ set bs) $p^{\prime}$
unfolding $p^{\prime}$-def by (rule sig-trd-red-rtrancl, rule sig-trd-irred)
from dgrad bs-sub $p$-in $p$-red have $p^{\prime}$-in: $p^{\prime} \in d g r a d$-sig-set dgrad
by (rule dgrad-sig-set-closed-sig-red-rtrancl)
from $p$-red have $l t-p^{\prime}: l t p^{\prime}=l t($ poly-of-pair $p$ ) by (rule sig-red-regular-rtrancl-lt)
have sig-merge: sig-of-pair $p \preceq_{t}$ sig-of-pair $q$ if $q \in \operatorname{set}$ (add-spairs ps bs $p^{\prime}$ ) for $q$
using that unfolding add-spairs-def set-merge-wrt
proof
assume $q \in$ set (new-spairs bs $p^{\prime}$ )
then obtain $b 0$ where is-regular-spair $p^{\prime} b 0$ and $q=\operatorname{Inl}\left(p^{\prime}, b 0\right)$ by (rule in-new-spairsE)
hence sig-of-q: sig-of-pair $q=l t$ (spair $p^{\prime}$ b0) by (simp only: sig-of-spair)
show ?thesis unfolding sig-of-q sig-of-p lt-p ${ }^{\prime}$ [symmetric $]$ by (rule is-regular-spair-lt-ge-1, fact)
next
assume $q \in$ set $p s$
moreover from assms (1) have sorted-wrt pair-ord ( $p \#$ ps) by (rule rb-aux-inv-D5)
ultimately show?thesis by (simp add: pair-ord-def)
qed
have sig-of-p-less: sig-of-pair $p \prec_{t}$ term-of-pair $(0, j)$ if Inr $j \in$ set ps for $j$
proof (intro ord-term-lin.le-neq-trans)

```
    from assms(1) have sorted-wrt pair-ord ( \(p \# p s\) ) by (rule rb-aux-inv-D5)
    with \(\langle\operatorname{Inr} j \in\) set \(p s\rangle\) show sig-of-pair \(p \preceq_{t}\) term-of-pair \((0, j)\)
    by (auto simp: pair-ord-def)
next
    from \(\operatorname{assms}(1)\) that show sig-of-pair \(p \neq\) term-of-pair \((0, j)\) by (rule Inr-in-tailD)
qed
have lt-p-gr: lt \(b \prec_{t} l t\) (poly-of-pair \(p\) ) if \(b \in\) set \(b s\) for \(b\) unfolding sig-of-p[symmetric]
    using \(\operatorname{assms}(1,2)\) that by (rule not-sig-crit)
have inv1: rb-aux-inv1 ( \(p^{\prime} \#\) bs) unfolding rb-aux-inv1-def
proof (intro conjI impI allI)
    from \(b s\)-sub \(p^{\prime}\)-in show set \(\left(p^{\prime} \# b s\right) \subseteq\) dgrad-sig-set dgrad by simp
next
    from inv1 have \(0 \notin\) rep-list‘ set bs by (rule rb-aux-inv1-D2)
    with \(\operatorname{assms}(3)\) show \(0 \notin\) rep-list' \(\operatorname{set}\left(p^{\prime} \#\right.\) bs) by (simp add: \(p^{\prime}\)-def)
next
    from inv1 have sorted-wrt ( \(\lambda x y\). lt \(y \prec_{t}\) lt \(x\) ) bs by (rule rb-aux-inv1-D3)
    with \(l t-p\)-gr show sorted-wrt ( \(\lambda x y\). lt \(\left.y \prec_{t} l t x\right)\left(p^{\prime} \#\right.\) bs) by (simp add: lt-p')
next
    fix \(i\)
    assume \(i<\) length ( \(p^{\prime} \# b s\) )
    have \(\left(\neg\right.\) is-sig-red \(\left.\left(\prec_{t}\right)(\preceq)\left(\operatorname{set}\left(\operatorname{drop}(S u c i)\left(p^{\prime} \# b s\right)\right)\right)\left(\left(p^{\prime} \# b s\right)!i\right)\right) \wedge\)
                            \(\left(\left(\exists j<\right.\right.\) length \(f\) s. lt \(\left(\left(p^{\prime} \# b s\right)!i\right)=l t\) (monomial \((1:: ' b)\) (term-of-pair ( 0 ,
j))) \(\wedge\)
                            punit.lt (rep-list \(\left(\left(p^{\prime} \#\right.\right.\) bs)! i)) \(\preceq\) punit.lt (rep-list (monomial 1
(term-of-pair \((0, j))))) \vee\)
    \(\left(\exists p \in \operatorname{set}\left(p^{\prime} \# b s\right) . \exists q \in \operatorname{set}\left(p^{\prime} \# b s\right)\right.\). is-regular-spair \(p q \wedge\) rep-list (spair
\(p q) \neq 0 \wedge\)
                \(l t\left(\left(p^{\prime} \# b s\right)!i\right)=l t(\) spair \(p q) \wedge\)
                    punit.lt \(\left(\right.\) rep-list \(\left.\left(\left(p^{\prime} \# b s\right)!i\right)\right) \preceq\) punit.lt \((\) rep-list \((\) spair p q) \(\left.))\right) \wedge\)
            is-RB-upt dgrad rword (set (drop (Suc i) \(\left.\left.\left(p^{\prime} \# b s\right)\right)\right)\left(l t\left(\left(p^{\prime} \# b s\right)!i\right)\right)\)
        (is ?thesis1 \(\wedge\) ?thesis2 \(\wedge\) ?thesis3)
    proof (cases \(i\) )
        case 0
        show ?thesis
        proof (simp add: \(\langle i=0\rangle p^{\prime}\)-irred del: bex-simps, rule conjI)
            show \(\left(\exists j<\right.\) length fs. lt \(p^{\prime}=l t(\) monomial \((1:: ' b)(\) term-of-pair \((0, j))) \wedge\)
                    punit.lt (rep-list \(p^{\prime}\) ) \(\preceq\) punit.lt (rep-list (monomial 1 (term-of-pair
\((0, j))))) \vee\)
            \(\left(\exists p \in\right.\) insert \(p^{\prime}(\) set \(b s) . \exists q \in\) insert \(p^{\prime}(\) set \(b s)\). is-regular-spair \(p q \wedge\)
rep-list \((\) spair \(p q) \neq 0 \wedge\)
                        lt \(p^{\prime}=l t(\) spair \(p q) \wedge\) punit.lt (rep-list \(\left.p^{\prime}\right) \preceq\) punit.lt (rep-list
(spair \(p q)\) ))
    proof (rule sum-prodE)
            fix \(a b\)
            assume \(p: p=\operatorname{Inl}(a, b)\)
            have \(\operatorname{Inl}(a, b) \in \operatorname{set}(p \# p s)\) by ( \(\operatorname{simp} a d d: p)\)
            with assms(1) have \(a \in\) set bs and \(b \in\) set bs and is-regular-spair a \(b\)
                by (rule rb-aux-inv-D3)+
            from \(p\)-red have \(p^{\prime}\)-red: \(\left(\right.\) sig-red \(\left(\prec_{t}\right)(\preceq)(\) set bs) \(){ }^{* *}\left(\right.\) spair a b) \(p^{\prime}\) by
```

(simp add: p)
hence $(\text { punit.red }(\text { rep-list ' set bs) }))^{* *}($ rep-list (spair a b)) (rep-list p')
by (rule sig-red-red-rtrancl)
moreover from assms(3) have rep-list $p^{\prime} \neq 0$ by (simp add: $p^{\prime}$-def)
ultimately have rep-list (spair a $b$ ) $\neq 0$ by (auto dest: punit.rtrancl-0)
moreover from $p^{\prime}$-red have $l t p^{\prime}=l t$ (spair a b)
and punit.lt (rep-list $\left.p^{\prime}\right) \preceq$ punit.lt (rep-list (spair a b))
by (rule sig-red-regular-rtrancl-lt, rule sig-red-rtrancl-lt-rep-list)
ultimately show ?thesis using $\langle a \in$ set $b s\rangle\langle b \in$ set bs〉〈is-regular-spair $a b$ by blast
next
fix $j$
assume $p=\operatorname{Inr} j$
hence $\operatorname{Inr} j \in \operatorname{set}(p \# p s)$ by $\operatorname{simp}$
with $\operatorname{assms}$ (1) have $j<$ length fs by (rule rb-aux-inv-D4)
from $p$-red have $\left(\text { sig-red }\left(\prec_{t}\right)(\preceq)(\text { set } b s)\right)^{* *}($ monomial 1 (term-of-pair $(0, j))) p^{\prime}$
by (simp add: $\langle p=\operatorname{Inr} j\rangle)$
hence lt $p^{\prime}=l t$ (monomial $(1:: ' b)$ (term-of-pair $\left.(0, j)\right)$ )
and punit.lt (rep-list $p^{\prime}$ ) $\preceq$ punit.lt (rep-list (monomial 1 (term-of-pair $(0, j))))$
by (rule sig-red-regular-rtrancl-lt, rule sig-red-rtrancl-lt-rep-list)
with $\langle j<$ length $f s\rangle$ show ?thesis by blast
qed
next
from $\operatorname{assms}(1)$ show is-RB-upt dgrad rword (set bs) (lt p') unfolding lt-p' sig-of-p[symmetric]
by (rule rb-aux-inv-is-RB-upt-Cons)
qed
next
case (Suc $i^{\prime}$ )
with $\left\langle i<\right.$ length $\left.\left(p^{\prime} \# b s\right)\right\rangle$ have $i^{\prime}: i^{\prime}<$ length $b s$ by simp
show ?thesis
proof (simp add: «i=Suc $\left.i^{\prime}\right\rangle$ del: bex-simps, intro conjI)
from inv1 $i^{\prime}$ show $\neg$ is-sig-red $\left(\prec_{t}\right)(\preceq)\left(\right.$ set $\left(\right.$ drop $\left.\left.\left(S u c i^{\prime}\right) b s\right)\right)\left(b s!i^{\prime}\right)$
by (rule rb-aux-inv1-D4)
next
from ${ }^{\text {inv1 }} i^{\prime}$
show $\left(\exists j<\right.$ length $f s . l t\left(b s!i^{\prime}\right)=l t($ monomial $(1:: ' b)($ term-of-pair $(0, j)))$
$\wedge$
punit.lt (rep-list $\left.\left(b s!i^{\prime}\right)\right) \preceq$ punit.lt (rep-list (monomial 1 (term-of-pair $(0, j))))) \vee$
$\left(\exists p \in\right.$ insert $p^{\prime}($ set $b s) . \exists q \in$ insert $p^{\prime}($ set bs). is-regular-spair $p q \wedge$ rep-list $($ spair $p q) \neq 0 \wedge$

$$
\text { lt }\left(b s!i^{\prime}\right)=l t(\text { spair } p q) \wedge \text { punit.lt }\left(\text { rep-list }\left(b s!i^{\prime}\right)\right) \preceq \text { punit.lt }
$$

(rep-list (spair p q)))
by (auto elim!: rb-aux-inv1-E) next
from inv1 $i^{\prime}$ show is-RB-upt dgrad rword (set (drop (Suc $\left.i^{\prime}\right)$ bs)) (lt (bs !
$\left.i^{\prime}\right)$ )
by (rule rb-aux-inv1-D5)
qed
qed
thus?thesis1 and ?thesis2 and ?thesis3 by simp-all
qed
have rb: is-RB-in dgrad rword (set ( $\left.p^{\prime} \# b s\right)$ ) (sig-of-pair $p$ ) proof (rule is-RB-inI1)
have $p^{\prime} \in \operatorname{set}\left(p^{\prime} \# b s\right)$ by simp
with inv1 have is-canon-rewriter rword (set ( $\left.p^{\prime} \# b s\right)$ ) (lt $\left.p^{\prime}\right) p^{\prime}$ by (rule is-canon-rewriterI-eq-sig)
thus is-canon-rewriter rword (set ( $p^{\prime} \#$ bs)) (sig-of-pair p) $p^{\prime}$ by (simp add:
lt-p ${ }^{\prime}$ sig-of-p)
next
from $p^{\prime}$-irred have $\neg$ is-sig-red $\left(\prec_{t}\right)(=)\left(\right.$ set bs) $p^{\prime}$
by (simp add: is-sig-red-top-tail-cases)
with sig-irred-regular-self have $\neg$ is-sig-red $\left(\prec_{t}\right)(=)\left(\left\{p^{\prime}\right\} \cup\right.$ set bs) $p^{\prime}$ by (simp add: is-sig-red-Un del: Un-insert-left)
thus $\neg$ is-sig-red $\left(\prec_{t}\right)(=)\left(\right.$ set $\left.\left(p^{\prime} \# b s\right)\right)$ (monom-mult 1 (pp-of-term (sig-of-pair $\left.\left.p)-l p p^{\prime}\right) p^{\prime}\right)$ by (simp add: lt-p' sig-of-p)
qed
show rb-aux-inv ( $p^{\prime} \#$ bs, new-syz-sigs ss bs $p$, add-spairs ps bs $p^{\prime}$ )
unfolding rb-aux-inv.simps
proof (intro conjI ballI allI impI)
show rb-aux-inv1 ( $p^{\prime} \#$ bs) by (fact inv1)
next
fix $s$
assume $s \in$ set (new-syz-sigs ss bs $p$ )
with assms(1) show is-syz-sig dgrad s by (rule new-syz-sigs-is-syz-sig)
next
fix $q 1 q 2$
assume $\operatorname{Inl}(q 1, q 2) \in \operatorname{set}\left(\right.$ add-spairs $p s$ bs $\left.p^{\prime}\right)$
hence $\operatorname{Inl}\left(q 1, q^{2}\right) \in \operatorname{set}\left(\right.$ new-spairs bs $\left.p^{\prime}\right) \vee \operatorname{Inl}(q 1, q 2) \in \operatorname{set}(p \# p s)$
by (auto simp: add-spairs-def set-merge-wrt)
hence is-regular-spair $q 1 q 2 \wedge q 1 \in \operatorname{set}\left(p^{\prime} \# b s\right) \wedge q 2 \in \operatorname{set}\left(p^{\prime} \# b s\right)$
proof
assume $\operatorname{Inl}(q 1, q 2) \in$ set (new-spairs bs $\left.p^{\prime}\right)$
hence $q 1=p^{\prime}$ and $q 2 \in$ set bs and is-regular-spair $p^{\prime} q^{2}$ by (rule in-new-spairsD) + thus ?thesis by simp
next
assume $\operatorname{Inl}(q 1, q 2) \in \operatorname{set}(p \# p s)$
with $\operatorname{assms}(1)$ have is-regular-spair q1 $q 2$ and $q 1 \in$ set bs and $q 2 \in$ set bs
by (rule rb-aux-inv-D3)+
thus ?thesis by simp
qed
thus is-regular-spair q1 q2 and $q 1 \in \operatorname{set}\left(p^{\prime} \# b s\right)$ and $q 2 \in \operatorname{set}\left(p^{\prime} \# b s\right)$ by simp-all
next
fix $j$
assume Inr $j \in$ set (add-spairs ps bs $p^{\prime}$ )
hence $\operatorname{Inr} j \in$ set ps by (simp add: add-spairs-def set-merge-wrt Inr-not-in-new-spairs)
hence $\operatorname{Inr} j \in \operatorname{set}(p \# p s)$ by $\operatorname{simp}$
with $\operatorname{assms}(1)$ show $j<$ length $f s$ by (rule rb-aux-inv-D4)
fix $b$
assume $b \in \operatorname{set}\left(p^{\prime} \# b s\right)$
hence $b=p^{\prime} \vee b \in$ set bs by simp
thus lt $b \prec_{t}$ term-of-pair $(0, j)$
proof
assume $b=p^{\prime}$
hence $l t b=$ sig-of-pair $p$ by (simp only: lt-p' sig-of-p)
also from $\langle\operatorname{Inr} j \in$ set $p s\rangle$ have $\ldots \prec_{t}$ term-of-pair $(0, j)$ by (rule sig-of-p-less)
finally show ?thesis.
next
assume $b \in$ set $b s$
with $\operatorname{assms}(1)\langle\operatorname{Inr} j \in \operatorname{set}(p \# p s)\rangle$ show ?thesis by (rule rb-aux-inv-D4)
qed
next
fix $j$
assume $\operatorname{Inr} j \in$ set (add-spairs ps bs $p^{\prime}$ )
hence Inr $j \in$ set ps by (simp add: add-spairs-def set-merge-wrt Inr-not-in-new-spairs)
hence $\operatorname{Inr} j \in \operatorname{set}(p \# p s)$ by $\operatorname{simp}$
let ? $P=\lambda q$. sig-of-pair $q=$ term-of-pair $(0, j)$
have filter ?P (add-spairs ps bs $p^{\prime}$ ) $=$ filter ?P ps unfolding add-spairs-def
proof (rule filter-merge-wrt-2)
fix $q$
assume $q \in$ set (new-spairs bs $p^{\prime}$ )
then obtain $b$ where $b \in$ set $b s$ and is-regular-spair $p^{\prime} b$ and $q=\operatorname{Inl}\left(p^{\prime}, b\right)$
by (rule in-new-spairsE)
moreover assume sig-of-pair $q=$ term-of-pair $(0, j)$
ultimately have $l t\left(\right.$ spair $\left.p^{\prime} b\right)=$ term-of-pair $(0, j)$
by (simp add: sig-of-spair del: sig-of-pair.simps)
hence eq: component-of-term (lt (spair $\left.p^{\prime} b\right)$ ) $=j$ by (simp add: compo-
nent-of-term-of-pair)
have component-of-term (lt $p^{\prime}$ ) $<j$
proof (rule ccontr)
assume $\neg$ component-of-term (lt $p^{\prime}$ ) $<j$
hence component-of-term (term-of-pair $(0, j)) \leq$ component-of-term (lt p')
by (simp add: component-of-term-of-pair)
moreover have pp-of-term (term-of-pair $(0, j)) \preceq p p$-of-term (lt p')
by (simp add: pp-of-term-of-pair zero-min)
ultimately have term-of-pair $(0, j) \preceq_{t}$ lt $p^{\prime}$ using ord-termI by blast
moreover have $l t p^{\prime} \prec_{t}$ term-of-pair $(0, j)$ unfolding $l t-p^{\prime}$ sig-of- $p[$ symmetric]
using $\langle I n r j \in$ set $p s\rangle$ by (rule sig-of-p-less)
ultimately show False by simp
qed
moreover have component-of-term (lt b) $<j$

```
    proof (rule ccontr)
        assume ᄀ component-of-term (lt b)<j
        hence component-of-term (term-of-pair ( }0,j))\leq\mathrm{ component-of-term (lt b)
            by (simp add: component-of-term-of-pair)
    moreover have pp-of-term (term-of-pair (0, j))\preceq pp-of-term (lt b)
            by (simp add: pp-of-term-of-pair zero-min)
    ultimately have term-of-pair (0,j) \preceq́t lt b using ord-termI by blast
    moreover from assms(1) \Inr j set ( }p#\mathrm{ # ps)〉<b set bs>
    have lt b}\mp@subsup{\prec}{t}{}\mathrm{ term-of-pair ( 0,j) by (rule rb-aux-inv-D4)
    ultimately show False by simp
        qed
        ultimately have component-of-term (lt (spair p' b)) < j
        using is-regular-spair-component-lt-cases[OF〈is-regular-spair p' b>] by auto
        thus False by (simp add: eq)
    qed
    hence length (filter ?P (add-spairs ps bs p
        by simp
    also from assms(1)<Inr j\in set (p# ps)\rangle have ... \leq1 by (rule rb-aux-inv-D4)
    finally show length (filter ?P (add-spairs ps bs p
next
    from assms(1) have sorted-wrt pair-ord ( }p#\mathrm{ # ps) by (rule rb-aux-inv-D5)
    hence sorted-wrt pair-ord ps by simp
    thus sorted-wrt pair-ord (add-spairs ps bs p') by (rule sorted-add-spairs)
next
    fix qb1 b2
    assume 1:q\in set (add-spairs ps bs p') and 2: is-regular-spair b1 b2
        and 3: sig-of-pair q < <t lt (spair b1 b2)
    assume b1 \in set ( }\mp@subsup{p}{}{\prime}#|bs)\mathrm{ and b2 }\in\operatorname{set}(\mp@subsup{p}{}{\prime}#bs
    hence b1 = p'\vee b1 \in set bs and b2 = p'\vee b2 \in set bs by simp-all
    thus Inl (b1, b2) \in set (add-spairs ps bs p
ps bs p')
    proof (elim disjE)
        assume b1 = p' and b2 = p'
        with 2 show ?thesis by (simp add: is-regular-spair-def)
    next
        assume b1 = p' and b2 \in set bs
        from this(2) 2 have Inl (b1, b2) \in set (new-spairs bs p') unfolding <b1 =
p'>
            by (rule in-new-spairsI)
        with 2 show ?thesis by (simp add: sig-of-spair add-spairs-def set-merge-wrt
image-Un del: sig-of-pair.simps)
    next
        assume b2 = p' and b1 \in set bs
        note this(2)
        moreover from 2 have is-regular-spair b2 b1 by (rule is-regular-spair-sym)
        ultimately have Inl (b2, b1) \in set (new-spairs bs p') unfolding < b2 = p'>
            by (rule in-new-spairsI)
        with 2 show ?thesis
            by (simp add: sig-of-spair-commute sig-of-spair add-spairs-def set-merge-wrt
```

```
image-Un del: sig-of-pair.simps)
    next
        note assms(1)<p\in set (p#ps)\rangle
        moreover assume b1 \in set bs and b2 \in set bs
        moreover note 2
        moreover have 4: sig-of-pair p}\mp@subsup{\prec}{t}{lt (spair b1 b2)
            by (rule ord-term-lin.le-less-trans, rule sig-merge, fact 1, fact 3)
        ultimately show ?thesis
        proof (rule rb-aux-inv-D6-1)
            assume Inl (b1, b2) \in set (p# ps)
            with }4\mathrm{ have Inl (b1,b2) G set ps
            by (auto simp: sig-of-spair 〈is-regular-spair b1 b2` simp del: sig-of-pair.simps)
            thus ?thesis by (simp add: add-spairs-def set-merge-wrt)
        next
            assume Inl (b2,b1) \in set (p# ps)
            with }4\mathrm{ have Inl (b2, b1) G set ps
            by (auto simp: sig-of-spair sig-of-spair-commute〈is-regular-spair b1 b2`
simp del: sig-of-pair.simps)
            thus ?thesis by (simp add: add-spairs-def set-merge-wrt)
        qed
    qed
next
    fix qj
    assume j< length fs
    assume q\in set (add-spairs ps bs p
    hence sig-of-pair p \preceq\preceq_ sig-of-pair q by (rule sig-merge)
    also assume sig-of-pair q}\mp@subsup{\prec}{t}{}\mathrm{ term-of-pair (0,j)
    finally have 1: sig-of-pair p}\mp@subsup{\prec}{t}{}\mathrm{ term-of-pair (0, j).
    with assms(1)<p\in\operatorname{set ( }p# # ps)\rangle<j<length fs> have Inr j fet (p# # ps)
        by (rule rb-aux-inv-D6-2)
    with 1 show Inr j \in set (add-spairs ps bs p') by (auto simp: add-spairs-def
set-merge-wrt)
    next
        fix b q
        assume b fet ( }\mp@subsup{p}{}{\prime}#|s)\mathrm{ and q-in: q eset (add-spairs ps bs p
        from this(1) have b= p
        hence lt b \preceq_t lt p'
        proof
            note assms(1)
            moreover assume b\in set bs
            moreover have p}\operatorname{set}(p#ps) by sim
            ultimately have lt b \preceq. sig-of-pair p by (rule rb-aux-inv-D7)
            thus ?thesis by (simp only:lt-p' sig-of-p)
    qed simp
    also have ... = sig-of-pair p by (simp only: sig-of-p lt-p')
    also from q-in have ... \preceq}\mp@subsup{\}{t}{\mathrm{ sig-of-pair q by (rule sig-merge)}
    finally show lt b}\mp@subsup{\preceq}{t}{}\mathrm{ sig-of-pair q.
next
    fix ab
```

```
    assume 1: a \in set ( }\mp@subsup{p}{}{\prime}#bs)\mathrm{ and 2: b f set ( }\mp@subsup{p}{}{\prime}#|s)\mathrm{ and 3: is-regular-spair
ab
    assume 6: ᄀis-RB-in dgrad rword (set ( }\mp@subsup{p}{}{\prime}##bs))(lt (spair a b))
    with rb have neq:lt (spair a b) \not=lt (poly-of-pair p) by (auto simp: sig-of-p)
    assume Inl (a,b)\not\in set (add-spairs ps bs p')
    hence 40: Inl (a,b) & set (new-spairs bs p') and Inl (a,b) & set ps
        by (simp-all add: add-spairs-def set-merge-wrt)
    from this(2) neq have 4: Inl (a,b)\not\in set (p# ps) by auto
    assume Inl (b,a)\not\in set (add-spairs ps bs p')
    hence 50: Inl (b,a) & set (new-spairs bs p') and Inl (b,a) & set ps
        by (simp-all add: add-spairs-def set-merge-wrt)
    from this(2) neq have 5: Inl (b,a)\not\in set ( }p##\mathrm{ ps) by (auto simp: spair-comm[of
a])
    have }a\not=\mp@subsup{p}{}{\prime
    proof
        assume a= p'
        with 3 have b\not= p' by (auto simp: is-regular-spair-def)
        with 2 have b\in set bs by simp
        hence Inl (a,b)\in set (new-spairs bs p') using 3 unfolding <a = p
(rule in-new-spairsI)
        with 40 show False ..
    qed
    with 1 have a\in set bs by simp
    have b}\not=\mp@subsup{p}{}{\prime
    proof
        assume b= p'
        with 3 have a\not= p' by (auto simp: is-regular-spair-def)
        with 1 have a\in set bs by simp
        moreover from 3 have is-regular-spair b a by (rule is-regular-spair-sym)
        ultimately have Inl (b,a)\in set (new-spairs bs p') unfolding <b = p >}>\mathrm{ by
(rule in-new-spairsI)
        with }50\mathrm{ show False ..
    qed
    with 2 have b f set bs by simp
    have lt-sp:lt (spair a b) \prec}\mp@subsup{\}{l}{lt p
    proof (rule ord-term-lin.linorder-cases)
        assume lt (spair a b) =lt p'
        with neq show ?thesis by (simp add: lt-p')
    next
        assume lt p}\mp@subsup{p}{}{\prime}\mp@subsup{\prec}{t}{lt (spair a b)
        hence sig-of-pair p < lt lt (spair a b) by (simp only:lt-p' sig-of-p)
        with assms(1)<p\in set (p# ps)\rangle\langlea\in set bs\rangle\langleb\in set bs\rangle 3 show ?thesis
        proof (rule rb-aux-inv-D6-1)
            assume Inl (a,b) \in set (p#ps)
            with 4 show ?thesis ..
        next
            assume Inl (b,a)\in set (p# ps)
            with 5 show ?thesis ..
        qed
```

```
    qed
    have }\negis-RB-in dgrad rword (set bs) (lt (spair a b)
    proof
    assume is-RB-in dgrad rword (set bs) (lt (spair a b))
    hence is-RB-in dgrad rword (set ( }\mp@subsup{p}{}{\prime}#|s)) (lt (spair a b)) unfolding set-simp
using lt-sp
            by (rule is-RB-in-insertI)
        with 6 show False ..
    qed
    with assms(1)\langlea\in set bs\rangle\langleb\in set bs\rangle 34 5
    obtain q}\mathrm{ where q}\operatorname{set (p#ps) and 8: sig-of-pair q=lt (spair a b) and 9:
\neg sig-crit' bs q
    by (rule rb-aux-inv-D8)
    from this(1, 2) lt-sp have q\in set ps by (auto simp:lt-p' sig-of-p)
    show \existsq\inset (add-spairs ps bs p'). sig-of-pair q=lt (spair a b) ^\neg sig-crit'
( }\mp@subsup{p}{}{\prime}#bs)
    proof (intro bexI conjI)
        show \neg sig-crit' ( }\mp@subsup{p}{}{\prime}##bs)
        proof
            assume sig-crit' (p' # bs)q
            moreover from lt-sp have sig-of-pair q}\mp@subsup{\prec}{t}{}lt\mp@subsup{p}{}{\prime}\mathrm{ by (simp only: 8)
            ultimately have sig-crit' bs q by (rule sig-crit'-ConsD)
            with 9 show False ..
        qed
    next
            from }\langleq\in\mathrm{ set ps` show q & set (add-spairs ps bs p') by (simp add:
add-spairs-def set-merge-wrt)
    qed fact
next
    fix }
    assume j< length fs
    assume Inr j # set (add-spairs ps bs p')
    hence Inr j & set ps by (simp add:add-spairs-def set-merge-wrt)
    show is-RB-in dgrad rword (set ( }\mp@subsup{p}{}{\prime}#|s)) (term-of-pair (0, j)
    proof (cases term-of-pair (0, j) = sig-of-pair p)
        case True
        with rb show ?thesis by simp
    next
        case False
        with}<Inr j\not\in set ps` have Inr j\not\in set (p# ps) by aut
            with assms(1)<j < length fs` have rb': is-RB-in dgrad rword (set bs)
(term-of-pair (0,j))
            by (rule rb-aux-inv-D9)
    have term-of-pair (0,j) \prec}\mp@subsup{t}{l}{lt p'
    proof (rule ord-term-lin.linorder-cases)
            assume term-of-pair (0,j) =lt p'
            with False show ?thesis by (simp add:lt-p' sig-of-p)
            next
```

```
            assume lt p}\mp@subsup{p}{}{\prime}\mp@subsup{\prec}{t}{}\mathrm{ term-of-pair (0,j)
            hence sig-of-pair p}\mp@subsup{\prec}{t}{}\mathrm{ term-of-pair ( 0, j) by (simp only:lt-p' sig-of-p)
            with assms(1) <p\in set (p#ps)\rangle\langlej<length fs> have Inr j f set ( p# # ps)
            by (rule rb-aux-inv-D6-2)
            with \Inr j\not\in set (p# ps)> show ?thesis ..
            qed
            with rb' show ?thesis unfolding set-simps by (rule is-RB-in-insertI)
    qed
    show rep-list (monomial 1 (term-of-pair (0, j))) \in ideal (rep-list` set (p' #
bs))
    proof (cases p = Inr j)
        case True
    show ?thesis
    proof (rule sig-red-zero-idealI, rule sig-red-zeroI)
        from p-red have (sig-red ( }\mp@subsup{\prec}{t}{})(\preceq)(\mathrm{ set bs))** (monomial 1 (term-of-pair
(0,j))) p
            by (simp add: True)
            moreover have set bs \subseteqset ( }\mp@subsup{p}{}{\prime}#|s)\mathrm{ by fastforce
    ultimately have (sig-red ( }\mp@subsup{\prec}{t}{})(\preceq)(\mathrm{ set ( }\mp@subsup{p}{}{\prime}#|s))\mp@subsup{)}{}{**}(\mathrm{ monomial 1 (term-of-pair
(0,j))) p
                by (rule sig-red-rtrancl-mono)
            hence (sig-red ( }\mp@subsup{\preceq}{t}{})(\preceq)(\operatorname{set}(\mp@subsup{p}{}{\prime}#|s)))** (monomial 1 (term-of-pair (0
j))) p
            by (rule sig-red-rtrancl-sing-regI)
            also have sig-red (\mp@subsup{\preceq}{t}{})(\preceq)(set (\mp@subsup{p}{}{\prime}#bs)) p'0 unfolding sig-red-def
            proof (intro exI bexI)
            from assms(3) have rep-list p'}=0\mathrm{ by (simp add: p}\mp@subsup{p}{}{\prime}\mathrm{ -def)
            show sig-red-single (\mp@subsup{\preceq}{t}{})(\preceq) p
            proof (rule sig-red-singleI)
            show rep-list p'\not=0 by fact
            next
                from 〈rep-list p'\not= 0〉 have punit.lt (rep-list p') \in keys (rep-list p')
                by (rule punit.lt-in-keys)
            thus 0 + punit.lt (rep-list p}\mp@subsup{p}{}{\prime})\in\mathrm{ keys (rep-list p') by simp
            next
                    from \langlerep-list p' f 0〉 have punit.lc (rep-list p') \not=0 by (rule
punit.lc-not-0)
                    thus 0 = p' - monom-mult (lookup (rep-list p})(0+\mathrm{ punit.lt (rep-list
p
                    by (simp add: punit.lc-def[symmetric])
            qed (simp-all add: term-simps)
        qed simp
        finally show (sig-red (\preceq}\mp@subsup{\}{t}{})(\preceq)(\operatorname{set}(\mp@subsup{p}{}{\prime}#bs))\mp@subsup{)}{}{**}(\mathrm{ monomial 1 (term-of-pair
(0,j))) 0 .
        qed (fact rep-list-zero)
    next
        case False
        with «Inr j\not\in set ps` have Inr j & set (p# ps) by simp
```

```
    with assms(1)<j < length fs>
    have rep-list (monomial 1 (term-of-pair (0,j)))\in ideal (rep-list'set bs)
        by (rule rb-aux-inv-D9)
        also have ...\subseteq ideal (rep-list' set (p'# bs)) by (rule ideal.span-mono,
fastforce)
    finally show ?thesis .
    qed
    qed
    show lt p' }\not=lt'set bs unfolding lt-p
    proof
        assume lt (poly-of-pair p)\inlt set bs
        then obtain b where b\in set bs and lt (poly-of-pair p)=lt b ..
    note this(2)
    also from }\langleb\in\mathrm{ set bs> have lt b}\mp@subsup{\prec}{t}{lt (poly-of-pair p) by (rule lt-p-gr)
    finally show False ..
    qed
qed
lemma rb-aux-inv-init: rb-aux-inv ([],Koszul-syz-sigs fs, map Inr [0..<length fs])
proof (simp add: rb-aux-inv.simps rb-aux-inv1-def o-def, intro conjI ballI allI
impI)
    fix v
    assume v\in set (Koszul-syz-sigs fs)
    with dgrad fs-distinct fs-nonzero show is-syz-sig dgrad v by (rule Koszul-syz-sigs-is-syz-sig)
next
    fix p q :: 't }\mp@subsup{=>}{0}{\prime}\mp@subsup{}{}{\prime}
    show Inl (p,q)\not\in Inr ` {0..<length fs } by blast
next
    fix }
    assume Inr j\inInr ' {0..<length fs}
    thus j< length fs by fastforce
next
    fix }
    have eq: (term-of-pair (0,i)=term-of-pair (0,j))\longleftrightarrow(j=i) for i
        by (auto dest: term-of-pair-injective)
    show length (filter (\lambdai.term-of-pair (0,i)= term-of-pair (0,j)) [0..<length fs])
s Suc 0
    by (simp add: eq)
next
    show sorted-wrt pair-ord (map Inr [0..<length fs])
    proof (simp add: sorted-wrt-map pair-ord-def sorted-wrt-upt-iff, intro allI impI)
        fix i j :: nat
        assume i<j
        hence }i\leqj\mathrm{ by simp
        show term-of-pair (0,i) \preceq term-of-pair ( 0, j) by (rule ord-termI, simp-all
add: term-simps <i \leq j`)
    qed
qed
```

```
corollary rb-aux-inv-init-fst:
    rb-aux-inv (fst (([], Koszul-syz-sigs \(f s, \operatorname{map} \operatorname{Inr}[0 . .<\) length \(f s]), z))\)
    using rb-aux-inv-init by simp
function (domintros) rb-aux :: ((('t \(\left.\Rightarrow_{0}{ }^{\prime} b\right)\) list \(\times{ }^{\prime} t\) list \(\times\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}\right.\right.\)
'b) \()+n a t)\) list \() \times n a t) \Rightarrow\)
        \(\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right.\) list \(\times{ }^{\prime} t\) list \(\times\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)\)
\(+n a t)\) list) \(\times n a t)\)
    where
        rb-aux \(((b s, s s,[]), z)=((b s, s s,[]), z) \mid\)
    rb-aux \(((b s, s s, p \# p s), z)=\)
        (let \(s s^{\prime}=\) new-syz-sigs ss bs \(p\) in
                if sig-crit bs ss' \(p\) then
                \(r b-a u x\left(\left(b s, s s^{\prime}, p s\right), z\right)\)
                else
                    let \(p^{\prime}=\) sig-trd bs (poly-of-pair \(\left.p\right)\) in
                    if rep-list \(p^{\prime}=0\) then
                        rb-aux ((bs,lt \(\left.p^{\prime} \# s s^{\prime}, p s\right)\), Suc z)
                        else
                        rb-aux (( \(p^{\prime} \#\) bs, ss' \({ }^{\prime}\) add-spairs ps bs \(\left.\left.\left.p^{\prime}\right), z\right)\right)\)
    by pat-completeness auto
```

definition $r b::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\times$ nat
where $r b=($ let $((b s,-,-), z)=r b-a u x(([]$, Koszul-syz-sigs $f$ s, map Inr $[0 . .<$ length
$f s])$, 0) in ( $b s, z)$ )
$r b$ 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 $\operatorname{sig}$ - $g b$, defined below. The second return value of $r b$ 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-aux-term1 $\equiv\{(x, y) . \exists z \cdot x=z \# y\}$
qualified definition $r b$-aux-term2 $\equiv\{(x, y) .(f s t x, f s t y) \in r b$-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 $r b$-aux-term $\equiv$ rb-aux-term2 $\cap\{(x, y) . r b-a u x-i n v x \wedge r b$-aux-inv $y\}$
lemma wfp-on-rb-aux-term1: wfp-on $(\lambda x y .(x, y) \in$ rb-aux-term1) (Collect rb-aux-inv1)
proof (rule wfp-onI-chain, rule, elim exE)
fix $s e q^{\prime}$
assume $\forall i$. seq' $i \in$ Collect rb-aux-inv1 $\wedge\left(s e q^{\prime}(S u c i)\right.$, seq $\left.^{\prime} i\right) \in$ rb-aux-term1
hence inv: rb-aux-inv1 (seq' $j$ ) and cons: $\exists b$. seq' $(S u c j)=b \#$ seq' $j$ for $j$ by ( simp-all add: rb-aux-term1-def)
from this(2) have 1: thesis0 if $\bigwedge j . i<$ length $(s e q ' ~ j) \Longrightarrow$ thesis0 for $i$ thesis0
using that by (rule list-seq-indexE-length)
define seq where seq $=\left(\lambda i\right.$. let $j=\left(S O M E\right.$ k. $i<$ length $\left.\left(s e q{ }^{\prime} k\right)\right)$ in rev $\left(s e q^{\prime}\right.$ j) ! i)
have 2: seq $i=\operatorname{rev}\left(s e q^{\prime} j\right)!i$ if $i<$ length $\left(s e q{ }^{\prime} j\right)$ for $i j$
proof -
define $k$ where $k=\left(\operatorname{SOME} k . i<\right.$ length $\left.\left(s e q^{\prime} k\right)\right)$
from that have $i<l e n g t h\left(s e q q^{\prime} k\right)$ unfolding $k$-def by (rule someI)
with cons that have rev $\left(s e q^{\prime} k\right)!i=\operatorname{rev}\left(s e q^{\prime} j\right)!i$ by (rule list-seq-nth')
thus ?thesis by (simp add: seq-def $k$-def[symmetric])
qed
have 3: seq $i \in \operatorname{set}\left(s e q^{\prime} j\right)$ if $i<$ length $\left(s e q^{\prime} j\right)$ for $i j$
proof -
from that have $i<$ length (rev $\left(s e q^{\prime} j\right)$ ) by simp
moreover from that have seq $i=\operatorname{rev}\left(s e q q^{\prime} j\right)!i$ by (rule 2)
ultimately have seq $i \in \operatorname{set}\left(r e v\left(s e q^{\prime} j\right)\right)$ by (metis nth-mem)
thus ?thesis by simp
qed
have 4: seq' $\{0 . .<i\}=$ set $($ take $i($ rev $(s e q ' j)))$ if $i<l e n g t h(s e q ' j)$ for $i j$
proof -
from refl have seq' $\{0 . .<i\}=(!)\left(\right.$ rev $\left.\left(s e q^{\prime} j\right)\right)$ ' $\{0 . .<i\}$
proof (rule image-cong)
fix $i^{\prime}$
assume $i^{\prime} \in\{0 . .<i\}$
hence $i^{\prime}<i$ by simp
hence $i^{\prime}<$ length $\left(s e q^{\prime} j\right)$ using that by simp
thus seq $i^{\prime}=$ rev $\left(s e q^{\prime} j\right)!i^{\prime}$ by (rule 2)
qed
also have $\ldots=\operatorname{set}\left(\right.$ take $i\left(\right.$ rev $\left(\right.$ seq $\left.\left.^{\prime} j\right)\right)$ ) by (rule nth-image, simp add: that less-imp-le-nat)
finally show ?thesis .
qed
from dgrad show False
proof (rule rb-termination)
have seq $i \in d g r a d$-sig-set dgrad for $i$
proof -
obtain $j$ where $i<$ length (seq' $j$ ) by (rule 1)
hence seq $i \in \operatorname{set}\left(s e q^{\prime} j\right)$ by (rule 3)
moreover from inv have set (seq' $j$ ) $\subseteq$ dgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
ultimately show ?thesis ..
qed
thus range seq $\subseteq$ dgrad-sig-set dgrad by blast
next
have rep-list (seq $i) \neq 0$ for $i$
proof -
obtain $j$ where $i<$ length (seq' $j$ ) by (rule 1)
hence seq $i \in \operatorname{set}\left(s e q^{\prime} j\right)$ by (rule 3)
moreover from inv have $0 \notin$ rep-list 'set (seq' j) by (rule rb-aux-inv1-D2)

```
            ultimately show ?thesis by auto
    qed
    thus 0 &rep-list'range seq by fastforce
next
    fix i1 i2 :: nat
    assume i1<i2
    also obtain j where i2: i2 < length (seq' j) by (rule 1)
    finally have i1: i1 < length (seq' j).
    from i1 have s1: seq i1 = rev (seq' j)!i1 by (rule 2)
    from i2 have s2: seq i2 = rev (seq' j)!i2 by (rule 2)
    from inv have sorted-wrt ( }\lambdaxy\mathrm{ . lt }y\mp@subsup{\prec}{t}{}\mathrm{ lt }x\mathrm{ ) (seq' j) by (rule rb-aux-inv1-D3)
    hence sorted-wrt ( }\lambdaxy\mathrm{ . lt }x\mp@subsup{\prec}{t}{}\mathrm{ lt y) (rev (seq' j)) by (simp add: sorted-wrt-rev)
    moreover note <i1 < i2>
    moreover from iQ have iQ < length (rev (seq' j)) by simp
    ultimately have lt (rev (seq' j) ! i1) < <t lt (rev (seq' j) ! iQ) by (rule
sorted-wrt-nth-less)
    thus lt (seq i1) \prec}\mp@subsup{t}{l}{lt (seq i2) by (simp only: s1 s2)
next
    fix }
    obtain j where i: i< length (seq' j) by (rule 1)
    hence eq1: seq i = rev (seq' j)!i and eq2: seq' {0..<i} = set (take i (rev
(seq' j)))
        by (rule 2, rule 4)
    let ?i = length (seq' j) - Suc i
    from i have ?i < length (seq' j) by simp
    with inv have }\neg\mathrm{ is-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq)(\mathrm{ set (drop (Suc ?i) (seq' j))) ((seq' j)!?i)
        by (rule rb-aux-inv1-D4)
    thus \negis-sig-red ( }\mp@subsup{\prec}{t}{})(\preceq)(seq`{0..<i})(seq i
        using i by (simp add: eq1 eq2 rev-nth take-rev Suc-diff-Suc)
    from inv <?i < length (seq' j)>
    show (\existsj<length fs.lt (seq i) = lt (monomial (1::'b) (term-of-pair (0, j))) ^
        punit.lt (rep-list (seq i)) \preceq punit.lt (rep-list (monomial 1 (term-of-pair
(0,j))))) \vee
            (\existsjk. is-regular-spair (seq j) (seq k) ^ rep-list (spair (seq j) (seq k))\not=0^
                        lt (seq i)}=lt(\mathrm{ spair (seq j) (seq k)) ^
                        punit.lt (rep-list (seq i)) \preceq punit.lt (rep-list (spair (seq j)(seq k))))
(is ?l \vee ?r)
    proof (rule rb-aux-inv1-E)
        fix j0
        assume j0 < length fs
        and lt (seq' j! (length (seq' j) - Suc i)) =lt (monomial (1::'b) (term-of-pair
(0,j0)))
            and punit.lt (rep-list (seq' j!(length (seq' j) - Suc i))) \preceq
                punit.lt (rep-list (monomial 1 (term-of-pair (0,j0))))
        hence ?l using i by (auto simp: eq1 eq2 rev-nth take-rev Suc-diff-Suc)
        thus ?thesis..
    next
        fix pq
```

```
    assume p \in set (seq'
    then obtain pi where pi<length (seq' j) and p = (seq' j) ! pi by (metis
in-set-conv-nth)
    hence p:p = seq (length (seq' j) - Suc pi)
        by (metis 2 <p\in set (seq' j)\rangle diff-Suc-less length-pos-if-in-set length-rev
rev-nth rev-rev-ident)
    assume q\in set (seq'}
    then obtain qi where qi<length (seq'}j)\mathrm{ and }q=(seq' j)!qi by (meti
in-set-conv-nth)
    hence q: q = seq (length (seq' j) - Suc qi)
```



```
rev-nth rev-rev-ident)
    assume is-regular-spair p q and rep-list (spair p q) \not=0
            and lt (seq' j! (length (seq' j) - Suc i)) =lt (spair p q)
            and punit.lt (rep-list (seq' j!(length (seq' j) - Suc i))) \preceq punit.lt (rep-list
(spair p q))
            hence ?r using i by (auto simp: eq1 eq2 p q rev-nth take-rev Suc-diff-Suc)
            thus ?thesis..
    qed
    from inv <?i < length (seq' j)>
    have is-RB-upt dgrad rword (set (drop (Suc ?i) (seq' j))) (lt ((seq' j)!?i))
        by (rule rb-aux-inv1-D5)
    with dgrad have is-sig-GB-upt dgrad (set (drop (Suc ?i) (seq' j))) (lt ((seq' j)
!?i))
            by (rule is-RB-upt-is-sig-GB-upt)
    thus is-sig-GB-upt dgrad (seq'{0..<i}) (lt (seq i))
            using i by (simp add: eq1 eq2 rev-nth take-rev Suc-diff-Suc)
    qed
qed
lemma wfp-on-rb-aux-term2: wfp-on (\lambdax y. (x,y)\inrb-aux-term2) (Collect rb-aux-inv)
proof (rule wfp-onI-min)
    fix x Q
    assume x\inQ and Q-sub: Q\subseteqCollect rb-aux-inv
    from this(1) have fst }x\infst' Q by (rule imageI
    have fst' }Q\subseteq\mathrm{ Collect rb-aux-inv1
    proof
    fix y
    assume y ffst' }
    then obtain z where z\inQ and y:y=fst z by fastforce
    obtain bs ss ps where z:z=(bs,ss,ps) by (rule rb-aux-inv.cases)
    from }\langlez\inQ\rangleQ\mathrm{ -sub have rb-aux-inv z by blast
    thus y Collect rb-aux-inv1 by (simp add: y z rb-aux-inv.simps)
    qed
```



```
    and z'-min: \bigwedgey.(y,z')\inrb-aux-term1 \Longrightarrowy\not\infst'Q by (rule wfp-onE-min)
blast
    from this(1) obtain z0 where z0 \inQ and \mp@subsup{z}{}{\prime}:\mp@subsup{z}{}{\prime}=fst z0 by fastforce
```

define $Q 0$ where $Q 0=\{z . z \in Q \wedge$ fst $z=f s t z 0\}$
from $\langle z 0 \in Q\rangle$ have $z 0 \in Q 0$ by (simp add: Q0-def)
hence length (snd (snd z0)) $\in$ length 'snd 'snd' $Q 0$ by (intro imageI)
with wf-less obtain $n$ where $n 1: n \in$ length'snd'snd ' $Q 0$
and $n 2: \bigwedge n$ '. $n$ ' $<n \Longrightarrow n^{\prime} \notin$ length 'snd'snd ' $Q 0$ by (rule wfE-min, blast)
from $n 1$ obtain $z$ where $z \in Q 0$ and $n 3: n=$ length (snd (snd $z$ )) by fastforce
have $z$-min: $y \notin Q 0$ if length (snd (snd $y)$ ) < length (snd (snd $z$ )) for $y$
proof
assume $y \in Q 0$
hence length (snd (snd y)) $\in$ length 'snd' snd ' Q0 by (intro imageI)
with n2 have $\neg$ length $($ snd $($ snd $y))<$ length (snd (snd z)) unfolding $n 3$ [symmetric] by blast
thus False using that ..
qed
show $\exists z \in Q . \forall y \in$ Collect rb-aux-inv. $(y, z) \in r b$-aux-term2 $\longrightarrow y \notin Q$
proof (intro bexI ballI impI)
fix $y$
assume $y \in$ Collect rb-aux-inv
assume $(y, z) \in$ rb-aux-term2
hence $(f s t y, f s t z) \in r b$-aux-term $1 \vee($ fst $y=$ fst $z \wedge$ length $($ snd $($ snd $y))<$ length (snd (snd z)))
by (simp add: rb-aux-term2-def)
thus $y \notin Q$
proof
assume (fst $y$, fst $z) \in$ rb-aux-term1
moreover from $\langle z \in Q 0\rangle$ have $f s t z=f s t z 0$ by (simp add: Q0-def)
ultimately have $\left(f s t y, z^{\prime}\right) \in$ rb-aux-term1 by (simp add: rb-aux-term1-def
$\left.z^{\prime}\right)$
hence fst $y \notin f s t$ ' $Q$ by (rule $z^{\prime}$-min)
thus ?thesis by blast
next
assume fst $y=$ fst $z \wedge$ length $($ snd $($ snd $y))<$ length (snd (snd $z)$ )
hence fst $y=$ fst $z$ and length (snd (snd $y)$ ) $<$ length (snd (snd $z)$ ) by simp-all
from this(2) have $y \notin Q 0$ by (rule z-min)
moreover from $\langle z \in Q 0\rangle$ have $f s t y=f$ st $z 0$ by (simp add: Q0-def $\langle f s t y$ $=f s t z\rangle)$
ultimately show ?thesis by (simp add: Q0-def)
qed
next
from $\langle z \in Q 0\rangle$ show $z \in Q$ by (simp add: Q0-def)
qed
qed
corollary wf-rb-aux-term: wf rb-aux-term
proof (rule wfI-min)
fix $x::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\times{ }^{\prime} t$ list $\times\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+\right.$ nat $)$ list and $Q$
assume $x \in Q$
show $\exists z \in Q . \forall y .(y, z) \in r b$-aux-term $\longrightarrow y \notin Q$

```
    proof (cases rb-aux-inv x)
    case True
    let ? \(Q=Q \cap\) Collect rb-aux-inv
    note wfp-on-rb-aux-term2
    moreover from \(\langle x \in Q\rangle\) True have \(x \in\) ? \(Q\) by simp
    moreover have ? \(Q \subseteq\) Collect rb-aux-inv by simp
    ultimately obtain \(z\) where \(z \in ? Q\) and \(z\)-min: \(\bigwedge y .(y, z) \in\) rb-aux-term2
\(\Longrightarrow y \notin ? Q\)
        by (rule wfp-onE-min) blast
    show ?thesis
    proof (intro bexI allI impI)
        fix \(y\)
        assume \((y, z) \in\) rb-aux-term
    hence \((y, z) \in\) rb-aux-term2 and rb-aux-inv \(y\) by (simp-all add: rb-aux-term-def)
        from this(1) have \(y \notin ? Q\) by (rule \(z\)-min)
        with 〈rb-aux-inv \(y\rangle\) show \(y \notin Q\) by simp
    next
        from \(\langle z \in ? Q\rangle\) show \(z \in Q\) by simp
    qed
    next
    case False
    show ?thesis
    proof (intro bexI allI impI)
        fix \(y\)
        assume \((y, x) \in r b\)-aux-term
        hence rb-aux-inv \(x\) by (simp add: rb-aux-term-def)
        with False show \(y \notin Q\)..
    qed fact
    qed
qed
lemma rb-aux-domI:
    assumes rb-aux-inv (fst args)
    shows rb-aux-dom args
proof -
    let ?rel \(=\) rb-aux-term \(<*\) lex \(*>(\{ \}::(n a t \times n a t)\) set \()\)
    from wf-rb-aux-term wf-empty have wf ?rel ..
    thus ?thesis using assms
    proof (induct args)
    case (less args)
        obtain bs ss ps0 \(z\) where args: args \(=((b s, s s, p s 0), z)\) using prod.exhaust
by metis
    show ?case
    proof (cases ps0)
        case Nil
        show ?thesis unfolding args Nil by (rule rb-aux.domintros)
    next
        case (Cons p ps)
        from less(1) have 1: \(\bigwedge y .(y,((b s, s s, p \# p s), z)) \in ? r e l \Longrightarrow r b-a u x-i n v(f s t\)
```

```
y)\Longrightarrow rb-aux-dom y
    by (simp only: args Cons)
    from less(2) have 2: rb-aux-inv (bs, ss, p# ps) by (simp only: args Cons
fst-conv)
    show ?thesis unfolding args Cons
    proof (rule rb-aux.domintros)
            assume sig-crit bs (new-syz-sigs ss bs p) p
                with 2 have a: rb-aux-inv (bs, (new-syz-sigs ss bs p), ps) by (rule
rb-aux-inv-preserved-1)
    with 2 have ((bs, (new-syz-sigs ss bs p), ps), bs, ss, p # ps) \in rb-aux-term
                by (simp add: rb-aux-term-def rb-aux-term2-def)
        hence (((bs, (new-syz-sigs ss bs p), ps), z), (bs, ss, p # ps),z)\in ?rel by
simp
    moreover from a have rb-aux-inv (fst ((bs, (new-syz-sigs ss bs p), ps), z))
        by (simp only: fst-conv)
        ultimately show rb-aux-dom((bs, (new-syz-sigs ss bs p), ps),z) by (rule
1)
    next
        assume rep-list (sig-trd bs (poly-of-pair p)) = 0
        with 2 have a: rb-aux-inv (bs,lt (sig-trd bs (poly-of-pair p)) # new-syz-sigs
ss bs p,ps)
            by (rule rb-aux-inv-preserved-2)
    with 2 have ((bs, lt (sig-trd bs (poly-of-pair p)) # new-syz-sigs ss bs p, ps),
bs,ss, p# ps)\in
                                    rb-aux-term
                            by (simp add: rb-aux-term-def rb-aux-term2-def)
                            hence (((bs, lt (sig-trd bs (poly-of-pair p)) # new-syz-sigs ss bs p, ps), Suc
z),(bs, ss, p # ps), z)\in
                    ?rel by simp
        moreover from a have rb-aux-inv (fst ((bs,lt (sig-trd bs (poly-of-pair p))
# new-syz-sigs ss bs p, ps), Suc z))
            by (simp only: fst-conv)
                ultimately show rb-aux-dom ((bs,lt (sig-trd bs (poly-of-pair p)) #
new-syz-sigs ss bs p, ps), Suc z)
            by (rule 1)
    next
        let ?args = (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)))
            assume ᄀ sig-crit bs (new-syz-sigs ss bs p) p and rep-list (sig-trd bs
(poly-of-pair p))}\not=
            with 2 have a: rb-aux-inv ?args by (rule rb-aux-inv-preserved-3)
            with 2 have (?args, bs, ss, p # ps) \inrb-aux-term
                    by (simp add: rb-aux-term-def rb-aux-term2-def rb-aux-term1-def)
            hence ((?args,z), (bs, ss, p# ps),z)\in ?rel by simp
            moreover from a have rb-aux-inv (fst (?args,z)) by (simp only: fst-conv)
            ultimately show rb-aux-dom (?args,z) by (rule 1)
        qed
    qed
qed
```


## qed

## Invariant

```
lemma rb-aux-inv-invariant:
    assumes rb-aux-inv (fst args)
    shows rb-aux-inv (fst (rb-aux args))
proof -
    from assms have rb-aux-dom args by (rule rb-aux-domI)
    thus ?thesis using assms
    proof (induct args rule: rb-aux.pinduct)
        case (1 bs ss z)
        thus ?case by (simp only: rb-aux.psimps(1))
    next
        case (2 bs ss p ps z)
        from 2(5) have *: rb-aux-inv (bs, ss, p # ps) by (simp only: fst-conv)
    show ?case
    proof (simp add: rb-aux.psimps(2)[OF 2(1)] Let-def, intro conjI impI)
        assume a: sig-crit bs (new-syz-sigs ss bs p) p
        with * have rb-aux-inv (bs, new-syz-sigs ss bs p,ps)
            by (rule rb-aux-inv-preserved-1)
            hence rb-aux-inv (fst ((bs, new-syz-sigs ss bs p, ps), z)) by (simp only:
fst-conv)
        with refl a show rb-aux-inv (fst (rb-aux ((bs,new-syz-sigs ss bs p,ps),z)))
by (rule 2(2))
            thus rb-aux-inv (fst (rb-aux ((bs, new-syz-sigs ss bs p, ps),z))).
    next
            assume a: ᄀ sig-crit bs (new-syz-sigs ss bs p)p
            assume b: rep-list (sig-trd bs (poly-of-pair p)) = 0
            with * have rb-aux-inv (bs, lt (sig-trd bs (poly-of-pair p)) # new-syz-sigs ss
bs p,ps)
            by (rule rb-aux-inv-preserved-2)
            hence rb-aux-inv (fst ((bs,lt (sig-trd bs (poly-of-pair p)) # new-syz-sigs ss bs
p,ps),Suc z))
            by (simp only: fst-conv)
        with refl a refl b
    show rb-aux-inv (fst (rb-aux ((bs,lt (sig-trd bs (poly-of-pair p)) # new-syz-sigs
ss bs p, ps), Suc z)))
            by (rule 2(3))
    next
        let ?args = (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)))
            assume a: ᄀ sig-crit bs (new-syz-sigs ss bs p) p and b: rep-list (sig-trd bs
(poly-of-pair p)) =0}
            with * have rb-aux-inv ?args by (rule rb-aux-inv-preserved-3)
            hence rb-aux-inv (fst (?args,z)) by (simp only: fst-conv)
            with refl a refl b
            show rb-aux-inv (fst (rb-aux (?args,z)))
            by (rule 2(4))
    qed
```


## qed

qed
lemma rb-aux-inv-last-Nil:
assumes rb-aux-dom args
shows snd $($ snd $(f s t(r b-a u x$ args $)))=[]$
using assms
proof (induct args rule: rb-aux.pinduct)
case ( $1 \mathrm{bs} \mathrm{ss} z$ )
thus ?case by (simp add: rb-aux.psimps(1))
next
case (2 bs ss p ps z)
show ?case
proof (simp add: rb-aux.psimps(2)[OF 2(1)] Let-def, intro conjI impI)
assume sig-crit bs (new-syz-sigs ss bs p) p
with refl show snd (snd $($ fst $($ rb-aux $((b s$, new-syz-sigs ss bs $p, p s), z))))=[]$
and snd (snd (fst (rb-aux ((bs, new-syz-sigs ss bs p, ps), z)))) = []
by (rule 2(2))+
next
assume $a$ : $\neg$ sig-crit bs (new-syz-sigs ss bs $p$ ) $p$ and b: rep-list (sig-trd bs
$($ poly-of-pair $p))=0$
from refl a refl $b$
show snd (snd (fst (rb-aux ((bs, lt (sig-trd bs (poly-of-pair p)) \# new-syz-sigs ss bs $p, p s)$, Suc $z))$ )) $=[]$
by (rule 2(3))
next
assume $a: \neg$ sig-crit $b s$ (new-syz-sigs ss bs $p$ ) $p$ and $b$ : rep-list (sig-trd bs
(poly-of-pair p)) $\neq 0$
from refl a refl $b$
show snd (snd (fst (rb-aux ((sig-trd bs (poly-of-pair p) \# bs, new-syz-sigs ss bs $p$,

$$
\text { add-spairs ps bs }(\text { sig-trd bs }(\text { poly-of-pair } p))), z))))=[]
$$

by (rule 2(4))
qed
qed
corollary rb-aux-shape:
assumes rb-aux-dom args
obtains bs ss $z$ where $r b$-aux args $=((b s, s s,[]), z)$
proof -
obtain bs ss ps $z$ where rb-aux args $=((b s, s s, p s), z)$ using prod.exhaust by metis
moreover from assms have snd (snd (fst (rb-aux args)) ) = [] by (rule rb-aux-inv-last-Nil)
ultimately have rb-aux args $=((b s, s s,[]), z)$ by simp
thus ?thesis ..
qed
lemma rb-aux-is-RB-upt:
is-RB-upt dgrad rword (set (fst (fst (rb-aux (([], Koszul-syz-sigs fs, map Inr

```
[0..<length fs]), z))))) u
proof -
    let ?args = (([], Koszul-syz-sigs fs, map Inr [0..<length fs]), z)
    from rb-aux-inv-init-fst have rb-aux-dom ?args by (rule rb-aux-domI)
    then obtain bs ss z' where eq: rb-aux ?args = ((bs,ss, []), z') by (rule
rb-aux-shape)
    moreover from rb-aux-inv-init-fst have rb-aux-inv (fst (rb-aux ?args))
    by (rule rb-aux-inv-invariant)
    ultimately have rb-aux-inv (bs, ss, []) by simp
    have is-RB-upt dgrad rword (set bs) u by (rule rb-aux-inv-is-RB-upt, fact, simp)
    thus ?thesis by (simp add: eq)
qed
corollary rb-is-RB-upt: is-RB-upt dgrad rword (set (fst rb)) u
    using rb-aux-is-RB-upt[of 0 u] by (auto simp add: rb-def split: prod.split)
corollary rb-aux-is-sig-GB-upt:
    is-sig-GB-upt dgrad (set (fst (fst (rb-aux (([],Koszul-syz-sigs fs, map Inr [0..<length
fs]),z))))) u
    using dgrad rb-aux-is-RB-upt by (rule is-RB-upt-is-sig-GB-upt)
corollary rb-aux-is-sig-GB-in:
    is-sig-GB-in dgrad (set (fst (fst (rb-aux (([], Koszul-syz-sigs fs, map Inr [0..<length
fs]),z))))) u
proof -
    let ?u = term-of-pair (pp-of-term u,Suc (component-of-term u))
    have }u\mp@subsup{\prec}{t}{}\mathrm{ ? u
    proof (rule ord-term-lin.le-neq-trans)
            show }u\mp@subsup{\preceq}{t}{}\mathrm{ ? u by (rule ord-termI, simp-all add: term-simps)
    next
        show }u\not=?
        proof
            assume u= ?u
            hence component-of-term u= component-of-term ?u by simp
            thus False by (simp add: term-simps)
        qed
    qed
    with rb-aux-is-sig-GB-upt show ?thesis by (rule is-sig-GB-uptD2)
qed
corollary rb-aux-is-Groebner-basis:
    assumes hom-grading dgrad
    shows punit.is-Groebner-basis (set (map rep-list (fst (fst (rb-aux (([], Koszul-syz-sigs
fs, map Inr [0..<length fs]), z))))))
proof -
    let ?args =(([], Koszul-syz-sigs fs, map Inr [0..<length fs ]),z)
    from rb-aux-inv-init-fst have rb-aux-dom ?args by (rule rb-aux-domI)
        then obtain bs ss z' where eq: rb-aux ?args = ((bs,ss, []), z') by (rule
rb-aux-shape)
```

```
    moreover from rb-aux-inv-init-fst have rb-aux-inv (fst (rb-aux ?args))
    by (rule rb-aux-inv-invariant)
    ultimately have rb-aux-inv (bs, ss, []) by simp
    hence rb-aux-inv1 bs by (rule rb-aux-inv-D1)
    hence set bs \subseteqdgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
    hence set (fst (fst (rb-aux ?args))) \subseteq dgrad-max-set dgrad by (simp add: eq
dgrad-sig-set'-def)
    with dgrad assms have punit.is-Groebner-basis (rep-list'set (fst (fst (rb-aux
?args))))
    using rb-aux-is-sig-GB-in by (rule is-sig-GB-is-Groebner-basis)
    thus ?thesis by simp
qed
lemma ideal-rb-aux:
    ideal (set (map rep-list (fst (fst (rb-aux (([], Koszul-syz-sigs fs, map Inr [0..<length
fs]),z)))))) =
    ideal (set fs) (is ideal ?l = ideal ?r)
proof
    show ideal ?l \subseteqideal ?r by (rule ideal.span-subset-spanI, auto simp: rep-list-in-ideal)
next
    show ideal ?r }\subseteq\mathrm{ ideal ?l
    proof (rule ideal.span-subset-spanI, rule subsetI)
        fix f
        assume f}\in\mathrm{ set fs
        then obtain j where j< length fs and f:f=fs! j by (metis in-set-conv-nth)
        let ?args = (([],Koszul-syz-sigs fs, map Inr [0..<length fs]),z)
        from rb-aux-inv-init-fst have rb-aux-dom?args by (rule rb-aux-domI)
            then obtain bs ss z' where eq: rb-aux ?args = ((bs,ss, []), z') by (rule
rb-aux-shape)
            moreover from rb-aux-inv-init-fst have rb-aux-inv (fst (rb-aux ?args))
                by (rule rb-aux-inv-invariant)
        ultimately have rb-aux-inv (bs, ss, []) by simp
        moreover note <j < length fs>
        moreover have Inr j\not\in set [] by simp
        ultimately have rep-list (monomial 1 (term-of-pair (0,j))) \in ideal ?l
            unfolding eq set-map fst-conv by (rule rb-aux-inv-D9)
        thus f\inideal ?l by (simp add: rep-list-monomial'}\langlej<length fs`f
    qed
qed
corollary ideal-rb: ideal (rep-list'set (fst rb)) = ideal (set fs)
proof -
    have ideal (rep-list ' set (fst rb)) =
            ideal (set (map rep-list (fst (fst (rb-aux (([], Koszul-syz-sigs fs, map Inr
[0..<length fs]),0))))))
    by (auto simp: rb-def split: prod.splits)
    also have ... = ideal (set fs) by (fact ideal-rb-aux)
    finally show ?thesis .
qed
```

```
lemma
    shows dgrad-max-set-closed-rb-aux:
        set (map rep-list (fst (fst (rb-aux (([], Koszul-syz-sigs fs, map Inr [0..<length
fs]), z)))|)\subseteq
        punit-dgrad-max-set dgrad (is ?thesis1)
    and rb-aux-nonzero:
    0 & set (map rep-list (fst (fst (rb-aux (([], Koszul-syz-sigs fs, map Inr [0..<length
fs]), z)))))
        (is ?thesis2)
proof -
    let ?args = (([], Koszul-syz-sigs fs, map Inr [0..<length fs]),z)
    from rb-aux-inv-init-fst have rb-aux-dom ?args by (rule rb-aux-domI)
    then obtain bs ss z' where eq: rb-aux ?args = ((bs, ss, []), z') by (rule
rb-aux-shape)
    moreover from rb-aux-inv-init-fst have rb-aux-inv (fst (rb-aux ?args))
    by (rule rb-aux-inv-invariant)
    ultimately have rb-aux-inv (bs, ss, []) by simp
    hence rb-aux-inv1 bs by (rule rb-aux-inv-D1)
    hence set bs\subseteqdgrad-sig-set dgrad and *: 0 & rep-list' set bs
    by (rule rb-aux-inv1-D1, rule rb-aux-inv1-D2)
    from this(1) have set bs\subseteqdgrad-max-set dgrad by (simp add: dgrad-sig-set'-def)
    with dgrad show ?thesis1 by (simp add: eq dgrad-max-3)
    from * show ?thesis2 by (simp add: eq)
qed
```


### 4.2.11 Minimality of the Computed Basis

lemma rb-aux-top-irred':
assumes rword-strict $=r w$-rat-strict and rb-aux-inv $(b s, s s, p \# p s)$
and $\neg$ sig-crit bs (new-syz-sigs ss bs p) p
shows $\neg i$ s-sig-red $\left(\preceq_{t}\right)(=)($ set bs) $($ sig-trd bs $($ poly-of-pair $p))$
proof -
have rword $=r w-r a t$ by (intro ext, simp only: rword-def rw-rat-alt, simp add: $\operatorname{assms}(1)$ )
have lt-p: sig-of-pair $p=l t$ (poly-of-pair $p$ ) by (rule pair-list-sig-of-pair, fact, simp)
define $p^{\prime}$ where $p^{\prime}=$ sig-trd bs (poly-of-pair $p$ )
have red-p: $\left(\text { sig-red }\left(\prec_{t}\right)(\preceq)(\text { set bs) })\right)^{* *}\left(\right.$ poly-of-pair p) $p^{\prime}$
unfolding $p^{\prime}$-def by (rule sig-trd-red-rtrancl)
hence $l t$ - $p^{\prime}: l t p^{\prime}=$ sig-of-pair $p$
and $l t-p^{\prime \prime}:$ punit.lt (rep-list $\left.p^{\prime}\right) \preceq$ punit.lt (rep-list (poly-of-pair p))
unfolding $l t-p$ by (rule sig-red-regular-rtrancl-lt, rule sig-red-rtrancl-lt-rep-list)
have $\neg i s$-sig-red $(=)(=)\left(\right.$ set bs) $p^{\prime}$
proof
assume is-sig-red $(=)(=)\left(\right.$ set bs) $p^{\prime}$
then obtain $b$ where $b \in$ set $b s$ and rep-list $b \neq 0$ and rep-list $p^{\prime} \neq 0$
and 1: punit.lt (rep-list b) adds punit.lt (rep-list p')
and 2：punit．lt $\left(\right.$ rep－list $\left.p^{\prime}\right) \oplus l t b=$ punit．lt $($ rep－list $b) \oplus l t p^{\prime}$
by（rule is－sig－red－top－addsE）
note this（3）
moreover from red－p have（punit．red（rep－list＇set bs））＊＊（rep－list（poly－of－pair
p））（rep－list $p^{\prime}$ ）
by（rule sig－red－red－rtrancl）
ultimately have rep－list（poly－of－pair $p$ ）$\neq 0$ by（auto simp：punit．rtrancl－ 0 ）
define $x$ where $x=$ punit．lt（rep－list $p^{\prime}$ ）－punit．lt（rep－list b）
from 12 have $x 1: x \oplus l t b=l t p^{\prime}$ by（simp add：term－is－le－rel－minus $x$－def）
from this［symmetric］have $l t b$ addst sig－of－pair $p$ unfolding $l t$－$p^{\prime}$ by（rule adds－termI）
from 1 have $x 2: x+$ punit．lt（rep－list b）$=$ punit．lt（rep－list $p^{\prime}$ ）by（simp add： $x$－def adds－minus）
from 〈rep－list $b \neq 0\rangle$ have $b \neq 0$ by（auto simp：rep－list－zero）
show False
proof（rule sum－prodE）
fix $a 0$ b0
assume $p: p=\operatorname{Inl}(a 0, b 0)$
hence $\operatorname{Inl}(a 0, b 0) \in \operatorname{set}(p \# p s)$ by $\operatorname{simp}$
with $\operatorname{assms}(2)$ have reg：is－regular－spair $a 0$ b0 and $a 0 \in$ set bs and $b 0 \in$ set bs
by（rule rb－aux－inv－D3）＋
from assms（2）have inv1：rb－aux－inv1 bs by（rule rb－aux－inv－D1）
hence $0 \notin$ rep－list＇set bs by（rule rb－aux－inv1－D2）
with $\langle a 0 \in$ set $b s\rangle\langle b 0 \in$ set $b s\rangle$ have rep－list $a 0 \neq 0$ and rep－list $b 0 \neq 0$ by fastforce＋
hence $a 0 \neq 0$ and $b 0 \neq 0$ by（auto simp：rep－list－zero）
let ？ $\mathrm{t} 1=$ punit．lt $($ rep－list a0 $)$
let ？t2 $=$ punit．lt $($ rep－list b0 $)$

from 〈rep－list（poly－of－pair $p) \neq 0\rangle$ have punit．spoly（rep－list a0）（rep－list b0）$\neq 0$
by（simp add：$p$ rep－list－spair）
with $\langle$ rep－list a0 $\neq 0\rangle\langle$ rep－list b0 $\neq 0\rangle$
have punit．lt（punit．spoly（rep－list a0）（rep－list b0））々？l
by（rule punit．lt－spoly－less－lcs［simplified］）
obtain $b^{\prime}$ where 3：is－canon－rewriter rword（set bs）（sig－of－pair p）$b^{\prime}$
and 4：punit．lt（rep－list（poly－of－pair p））$\prec$
（pp－of－term（sig－of－pair p）－lp $\left.b^{\prime}\right)+$ punit．lt（rep－list $\left.b^{\prime}\right)$
proof $\left(\right.$ cases $\left.(? l-? t 1) \oplus l t a 0 \preceq_{t}(? l-? t 2) \oplus l t b 0\right)$
case True
have sig－of－pair $p=l t$（spair a0 b0）unfolding $l t-p$ by（simp add：$p$ ）
also from reg have $\ldots=(? l-? t 2) \oplus l t b 0$
by（simp add：True is－regular－spair－lt ord－term－lin．max－def）
finally have eq1：sig－of－pair $p=(? l-? t 2) \oplus l t b 0$.
hence lt b0 adds $s_{t}$ sig-of-pair $p$ by (rule adds-termI)
moreover from $\operatorname{assms}(3)$ have $\neg i s$-rewritable bs b0 $((? l-$ ?t2) $\oplus l t b 0)$
by (simp add: p spair-sigs-def Let-def)
ultimately have is-canon-rewriter rword (set bs) (sig-of-pair p) b0
unfolding eq1 [symmetric] using inv1 $\langle b 0 \in$ set $b s\rangle\langle b 0 \neq 0\rangle$ is-rewritableI-is-canon-rewriter by blast
thus ?thesis
proof
have punit.lt (rep-list (poly-of-pair p)) = punit.lt (punit.spoly (rep-list a0)
(rep-list b0))
by (simp add: p rep-list-spair)
also have ... $\prec$ ?l by fact
also have $\ldots=(? l-? t 2)+? t 2$ by (simp only: adds-minus adds-lcs-2)
also have $\ldots=(p p$-of-term (sig-of-pair $p)-l p b 0)+$ ? $t 2$
by (simp only: eq1 pp-of-term-splus add-diff-cancel-right')
finally show punit.lt (rep-list (poly-of-pair p)) $\prec$ pp-of-term (sig-of-pair
$p)-l p b 0+? t 2$.
qed
next
case False
have sig-of-pair $p=l t$ (spair a0 b0) unfolding $l t-p$ by (simp add: $p$ )
also from reg have $\ldots=(? l-? t 1) \oplus l t a 0$
by (simp add: False is-regular-spair-lt ord-term-lin.max-def)
finally have eq1: sig-of-pair $p=(? l-? t 1) \oplus l t$ a 0 .
hence lt a0 addst sig-of-pair $p$ by (rule adds-termI)
moreover from $\operatorname{assms}(3)$ have $\neg$ is-rewritable bs a0 $((? l-? t 1) \oplus l t a 0)$
by (simp add: p spair-sigs-def Let-def)
ultimately have is-canon-rewriter rword (set bs) (sig-of-pair p) a0
unfolding eq1 [symmetric] using inv1 $\langle a 0 \in$ set $b s\rangle\langle a 0 \neq 0\rangle$ is-rewritableI-is-canon-rewriter
by blast
thus ?thesis
proof
have punit.lt (rep-list (poly-of-pair p)) $=$ punit.lt (punit.spoly (rep-list a0)
(rep-list b0))
by (simp add: p rep-list-spair)
also have ... $\prec$ ?l by fact
also have $\ldots=(? l-? t 1)+? t 1$ by (simp only: adds-minus adds-lcs)
also have $\ldots=(p p$-of-term $($ sig-of-pair $p)-l p a 0)+$ ? t1
by (simp only: eq1 pp-of-term-splus add-diff-cancel-right')
finally show punit.lt (rep-list (poly-of-pair p)) ఒpp-of-term (sig-of-pair p) $-l p a 0+? t 1$.
qed
qed
define $y$ where $y=p p$-of-term (sig-of-pair $p)-l p b^{\prime}$
from $l t$ - $p^{\prime \prime} 4$ have $y 2$ : punit.lt (rep-list $p^{\prime}$ ) $\prec y+$ punit.lt (rep-list $b^{\prime}$ )
unfolding $y$-def by (rule ordered-powerprod-lin.le-less-trans)
from 3 have $l t b^{\prime}$ addst sig-of-pair $p$ by (rule is-canon-rewriterD3)
hence $l p b^{\prime}$ adds $l p p^{\prime}$ and component-of-term ( $l t b^{\prime}$ ) $=$ component-of-term ( $l t$

```
p')
            by (simp-all add: adds-term-def lt-p')
            hence y1:y \opluslt b}\mp@subsup{}{}{\prime}=lt \mp@subsup{p}{}{\prime}\mathbf{by (simp add: y-def splus-def lt-p' adds-minus
term-simps)
            from 3 <b cet bs\rangle\langleb}\not=0\\langlelt b addst sig-of-pair p>
            have rword (spp-of b) (spp-of b}\mp@subsup{b}{}{\prime})\mathrm{ by (rule is-canon-rewriterD)
            hence punit.lt (rep-list b}\mp@subsup{b}{}{\prime})\opluslt b\mp@subsup{\preceq}{t}{}\mathrm{ punit.lt (rep-list b) }\opluslt b
            by (auto simp: <rword = rw-rat> rw-rat-def Let-def spp-of-def)
                            hence }(x+y)\oplus(punit.lt (rep-list b) \oplus lt b) \preceq\preceq (x+y)\oplus (punit.lt (rep-list
b) }\opluslt b'
            by (rule splus-mono)
    hence (y + punit.lt (rep-list b
(y\opluslt b}
            by (simp add: ac-simps)
            hence (y + punit.lt (rep-list b}\mp@subsup{b}{}{\prime}))\opluslt \mp@subsup{p}{}{\prime}\mp@subsup{\preceq}{t}{}\mathrm{ punit.lt (rep-list p
            by (simp only: x1 x2 y1)
    hence }y+\mathrm{ punit.lt (rep-list b}\mp@subsup{b}{}{\prime})\preceq\mathrm{ punit.lt (rep-list p}\mp@subsup{p}{}{\prime})\mathbf{by}(\mathrm{ rule ord-term-canc-left)
            with y2 show ?thesis by simp
    next
            fix }
            assume p:p=Inr j
            hence lt p' = term-of-pair ( 0, j) by (simp add:lt-p')
            with x1 term-of-pair-pair[of lt b] have lt b = term-of-pair ( }0,j
            by (auto simp: splus-def dest!: term-of-pair-injective plus-eq-zero-2)
            moreover have lt b}\mp@subsup{\prec}{t}{}\mathrm{ term-of-pair (0,j) by (rule rb-aux-inv-D4, fact, simp
add: p, fact)
            ultimately show ?thesis by simp
    qed
qed
moreover have }\negis\mathrm{ -sig-red ( }\mp@subsup{\prec}{t}{})(=)(\mathrm{ set bs) p
proof
    assume is-sig-red (}\mp@subsup{\prec}{t}{})(=)(\mathrm{ set bs) p'
    hence is-sig-red ( }\mp@subsup{\prec}{t}{\prime}\mathrm{ )(々) (set bs) p' by (simp add: is-sig-red-top-tail-cases)
    with sig-trd-irred show False unfolding p'-def ..
    qed
    ultimately show ?thesis by (simp add: p'-def is-sig-red-sing-reg-cases)
qed
lemma rb-aux-top-irred:
    assumes rword-strict =rw-rat-strict and rb-aux-inv (fst args) and b set (fst
(fst (rb-aux args)))
    and \b0.b0 \in set (fst (fst args)) \Longrightarrow\negis-sig-red (\preceq}\mp@subsup{\}{t}{})(=)(\operatorname{set}(fst (fst args)
- {b0}) b0
    shows \neg is-sig-red (\mp@subsup{\preceq}{t}{})(=)(set (fst (fst (rb-aux args))) - {b})b
proof -
    from assms(2) have rb-aux-dom args by (rule rb-aux-domI)
    thus ?thesis using assms(2, 3, 4)
    proof (induct args rule: rb-aux.pinduct)
    case (1 bs ss z)
```

$$
\text { let ?nil }=[]::\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+\text { nat }\right) \text { list }
$$

from $1(3)$ have $b \in \operatorname{set}(f s t(f s t((b s, s s, ? n i l), z))$ by (simp add: rb-aux.psimps(1)[OF 1(1)])
hence $\neg$ is-sig-red $\left(\preceq_{t}\right)(=)($ set $(f s t(f s t((b s, s s, ? n i l), z)))-\{b\}) b$ by (rule 1 (4))
thus ?case by (simp add: rb-aux.psimps(1)[OF 1 (1)])
next
case (2 bs ss p ps z)
from 2(5) have *: rb-aux-inv (bs, ss, p\# ps) by (simp only: fst-conv)
define $p^{\prime}$ where $p^{\prime}=$ sig-trd bs (poly-of-pair $p$ )
from 2(6) show ?case
proof (simp add: rb-aux.psimps(2)[OF 2(1)] Let-def $p^{\prime}$-def[symmetric] split: $i f$-splits)
note refl
moreover assume sig-crit bs (new-syz-sigs ss bs p) $p$
moreover from * this have rb-aux-inv (fst ((bs, new-syz-sigs ss bs p, ps),
z))
unfolding $f$ ft-conv by (rule rb-aux-inv-preserved-1)
moreover assume $b \in \operatorname{set}(f s t(f s t$ (rb-aux ((bs, new-syz-sigs ss bs p, ps), z))))
ultimately show $\neg$ is-sig-red $\left(\preceq_{t}\right)(=)($ set $(f s t(f s t(r b-a u x)((b s$, new-syz-sigs ss bs $p, p s), z)))$ ) $-\{b\}) b$
proof (rule 2(2))
fix $b 0$
assume $b 0 \in \operatorname{set}(f s t(f s t((b s, n e w-s y z-s i g s ~ s s ~ b s ~ p, p s), z)))$
hence $b 0 \in \operatorname{set}(f s t(f s t((b s, s s, p \# p s), z)))$ by simp
hence $\neg$ is-sig-red $\left(\preceq_{t}\right)(=)(\operatorname{set}(f s t(f s t((b s, s s, p \# p s), z)))-\{b 0\}) b 0$
by (rule 2(7))
thus $\neg i s$-sig-red $\left(\preceq_{t}\right)(=)($ set $(f s t(f s t((b s$, new-syz-sigs ss bs $p, p s), z)))$
$-\{b 0\}) b 0$
by $\operatorname{simp}$
qed
next
note refl
moreover assume $\neg$ sig-crit bs (new-syz-sigs ss bs p) $p$
moreover note refl
moreover assume rep-list $p^{\prime}=0$
moreover from * this have rb-aux-inv (fst ((bs, lt $p^{\prime} \#$ new-syz-sigs ss bs p, ps), Suc z))
unfolding $p^{\prime}$-def fst-conv by (rule rb-aux-inv-preserved-2)
moreover assume $b \in \operatorname{set}\left(f s t\left(f s t\right.\right.$ (rb-aux ((bs,lt $p^{\prime} \#$ new-syz-sigs ss bs $p$, ps), Suc z))))
ultimately show $\neg i s$-sig-red $\left(\preceq_{t}\right)(=)($ set $(f s t(f s t)(r b-a u x)((b s$,
lt $p^{\prime} \#$ new-syz-sigs ss bs $\left.p, p s\right)$, Suc
z)) )) $-\{b\}) b$
proof (rule 2(3)[simplified $p^{\prime}$-def $[$ symmetric $\left.\left.]\right]\right)$
fix $b 0$
assume $b 0 \in \operatorname{set}\left(f s t\left(f s t\left(\left(b s, l t p^{\prime} \#\right.\right.\right.\right.$ new-syz-sigs ss bs $\left.p, p s\right)$, Suc $\left.\left.\left.z\right)\right)\right)$
hence $b 0 \in \operatorname{set}(f s t(f s t((b s, s s, p \# p s), z)))$ by simp

```
    hence }\neg\mathrm{ is-sig-red ({}\mp@subsup{\}{t}{})(=)(\operatorname{set}(fst(fst ((bs, ss, p# ps),z))) - {b0})b
by (rule 2(7))
                            thus \neg is-sig-red (\preceq}\mp@subsup{)}{t}{)}(=)(\mathrm{ set (fst (fst ((bs, lt p' # new-syz-sigs ss bs p,
ps), Suc z))) - {b0}) b0
                by simp
        qed
    next
        note refl
    moreover assume \neg sig-crit bs (new-syz-sigs ss bs p) p
    moreover note refl
    moreover assume rep-list p' }=
    moreover from * 〈\neg sig-crit bs (new-syz-sigs ss bs p) p\rangle this
    have inv: rb-aux-inv (fst (( }\mp@subsup{p}{}{\prime}#|\mp@code{b}\mathrm{ , new-syz-sigs ss bs p, add-spairs ps bs p),
z))
            unfolding p'-def fst-conv by (rule rb-aux-inv-preserved-3)
            moreover assume b set (fst (fst (rb-aux (( p' # bs, new-syz-sigs ss bs p,
add-spairs ps bs p'),z))))
    ultimately show }\neg\mathrm{ is-sig-red ( }\mp@subsup{\preceq}{t}{})(=)(set (fst (fst (rb-aux (( p' # bs
                                    new-syz-sigs ss bs p, add-spairs ps bs p}\mp@subsup{}{}{\prime}),z)))
- {b})b
    proof (rule 2(4)[simplified p'-def[symmetric]])
            fix b0
            assume b0 \in set (fst (fst (( }\mp@subsup{p}{}{\prime}## bs,new-syz-sigs ss bs p, add-spairs ps b
p', z)))
            hence b0 = p'\vee b0 set bs by simp
            hence }\neg\mathrm{ is-sig-red (甶) (=) (({p'}-{b0}) U(set bs - {b0})) b0
            proof
            assume b0 = p'
```



```
            proof
                    assume is-sig-red (\mp@subsup{\preceq}{t}{})(=)(set bs - {b0}) p'
                    moreover have set bs - {b0}\subseteq set bs by fastforce
            ultimately have is-sig-red (\mp@subsup{\preceq}{t}{})(=) (set bs) p' by (rule is-sig-red-mono)
                    moreover have }\neg\mathrm{ is-sig-red (}\mp@subsup{\preceq}{t}{})(=)(set bs) p' unfolding p'-de
                    using assms(1) * \langleᄀ sig-crit bs (new-syz-sigs ss bs p) p\rangle by (rule
rb-aux-top-irred')
            ultimately show False by simp
            qed
            thus ?thesis by (simp add: <b0 = p'`)
    next
            assume b0 \in set bs
            hence b0 \in set (fst (fst ((bs, ss, p # ps),z))) by simp
            hence }\neg\mathrm{ is-sig-red ({}\mp@subsup{\}{t}{})(=)(set (fst (fst ((bs, ss, p# # ps),z))) - {b0}
b0 by (rule 2(7))
                            hence }\neg\mathrm{ is-sig-red ( }\mp@subsup{\Omega}{t}{})(=)(set bs - {b0}) b0 by sim
            moreover have }\neg\mathrm{ is-sig-red ( }\mp@subsup{\preceq}{t}{})(=)({p} - {b0})b
            proof
                assume is-sig-red (\mp@subsup{\preceq}{t}{})(=)({\mp@subsup{p}{}{\prime}}-{b0})b0
```


ultimately have is-sig-red $\left(\preceq_{t}\right)(=)\left\{p^{\prime}\right\} b 0$ by (rule is-sig-red-mono) hence lt $p^{\prime} \preceq_{t} l t$ b0 by (rule is-sig-redD-lt)
from inv have rb-aux-inv ( $p^{\prime} \#$ bs, new-syz-sigs ss bs $p$, add-spairs ps bs $p^{\prime}$ )
by (simp only: fst-conv)
hence rb-aux-inv1 ( $p^{\prime} \# b s$ ) by (rule rb-aux-inv-D1)
hence sorted-wrt ( $\lambda x y$. lt $y \prec_{t}$ lt $\left.x\right)\left(p^{\prime} \#\right.$ bs) by (rule rb-aux-inv1-D3)
with $\langle b 0 \in$ set $b s\rangle$ have $l t b 0 \prec_{t}$ lt $p^{\prime}$ by simp
with $\left\langle l t p^{\prime} \preceq_{t} l t b 0\right\rangle$ show False by simp
qed
ultimately show ?thesis by (simp add: is-sig-red-Un)
qed
thus $\neg$ is-sig-red $\left(\preceq_{t}\right)(=)\left(\right.$ set $\left(f_{s t}\left(f s t\left(\left(p^{\prime} \#\right.\right.\right.\right.$ bs, new-syz-sigs ss bs $p$, add-spairs ps bs $\left.\left.\left.\left.\left.p^{\prime}\right), z\right)\right)\right)-\{b 0\}\right) b 0$
by (simp add: Un-Diff[symmetric])
qed
qed
qed
qed
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 $\left[0 . .<\right.$ length $\left.\left.\left.\left.f_{s}\right]\right), z\right)\right)$ ))
(is is-min-sig-GB - $($ set $(f s t(f s t(r b-a u x ~ ? a r g s)))))$
unfolding is-min-sig-GB-def
proof (intro conjI allI ballI impI)
from rb-aux-inv-init-fst have inv: rb-aux-inv (fst (rb-aux ?args))
and rb-aux-dom?args
by (rule rb-aux-inv-invariant, rule rb-aux-domI)
from this(2) obtain bs ss $z^{\prime}$ where eq: rb-aux ?args $=\left((b s, s s,[]), z^{\prime}\right)$
by (rule rb-aux-shape)
from inv have rb-aux-inv (bs, ss, []) by (simp only: eq fst-conv)
hence rb-aux-inv1 bs by (rule rb-aux-inv-D1)
hence set bs $\subseteq$ dgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
thus set $(f s t(f s t(r b-a u x$ ? args $))) \subseteq$ dgrad-sig-set dgrad by (simp add: eq)
next
fix $u$
show is-sig-GB-in dgrad (set (fst (fst (rb-aux ?args)))) u by (fact rb-aux-is-sig-GB-in)
next
fix $g$
assume $g \in \operatorname{set}(f s t(f s t(r b-a u x$ ? args $)))$
with $\operatorname{assms}(1)$ rb-aux-inv-init-fst
show $\neg$ is-sig-red $\left(\preceq_{t}\right)(=)($ set $(f s t(f s t(r b-a u x$ ? args $)))-\{g\}) g$
by (rule rb-aux-top-irred) simp
qed
corollary rb-is-min-sig-GB:
assumes rword-strict $=$ rw-rat-strict
shows is-min-sig-GB dgrad (set (fst rb))
using rb-aux-is-min-sig-GB[OF assms, of 0] by (auto simp: rb-def split: prod.split)

### 4.2.12 No Zero-Reductions

```
fun rb-aux-inv2 :: (('t \(\left.\Rightarrow_{0}{ }^{\prime} b\right)\) list \(\times{ }^{\prime} t\) list \(\times\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+n a t\right)\)
list) \(\Rightarrow\) bool
    where \(r b-a u x-i n v 2(b s, s s, p s)=\)
            (rb-aux-inv (bs, ss, ps) \(\wedge\)
                            ( \(\forall j<\) length fs. Inr \(j \notin\) set \(p s \longrightarrow\)
                            (fs \(!j \in\) ideal (rep-list'set (filter ( \(\lambda b\). component-of-term (lt b) \(<\) Suc
```

j) $b s)) \wedge$
( $\forall b \in$ set bs. component-of-term (lt b) $<j \longrightarrow$
$(\exists s \in$ set ss. s addst term-of-pair (punit.lt (rep-list b), $j)))))$ )
lemma rb-aux-inv2-D1: rb-aux-inv2 args $\Longrightarrow r b-a u x-i n v$ args
by (metis prod.exhaust rb-aux-inv2.simps)
lemma rb-aux-inv2-D2:
rb-aux-inv2 $(b s, s s, p s) \Longrightarrow j<$ length $f s \Longrightarrow$ Inr $j \notin$ set $p s \Longrightarrow$
fs $!j \in$ ideal (rep-list'set (filter ( $\lambda$ b. component-of-term (lt b) $<$ Suc j) bs))
by $\operatorname{simp}$
lemma rb-aux-inv2-E:
assumes rb-aux-inv2 (bs, ss, ps) and $j<$ length $f s$ and Inr $j \notin$ set ps and $b \in$
set bs
and component-of-term (lt b) $<j$
obtains $s$ where $s \in$ set ss and $s$ adds $s_{t}$ term-of-pair (punit.lt (rep-list b), $j$ )
using assms by auto
context
assumes pot: is-pot-ord
begin
lemma sig-red-zero-filter:
assumes sig-red-zero $\left(\preceq_{t}\right)($ set bs) $r$ and component-of-term (lt r) $<j$
shows sig-red-zero $\left(\preceq_{t}\right)($ set $($ filter $(\lambda b$. component-of-term $(l t ~ b)<j) b s)) r$
proof -
have $\left(\preceq_{t}\right)=\left(\preceq_{t}\right) \vee\left(\preceq_{t}\right)=\left(\prec_{t}\right)$ by simp
with $\operatorname{assms}(1)$ have sig-red-zero $\left(\preceq_{t}\right)\left\{b \in\right.$ set bs. lt $b \preceq_{t}$ lt $\left.r\right\} r$ by (rule sig-red-zero-subset)
moreover have $\left\{b \in\right.$ set $b s$. lt $\left.b \preceq_{t} l t r\right\} \subseteq$ set (filter ( $\lambda b$. component-of-term (lt
b) $<j$ ) $b s$ )
proof
fix $b$
assume $b \in\left\{b \in\right.$ set $b s$. lt $\left.b \preceq_{t} l t r\right\}$
hence $b \in$ set $b s$ and $l t b \preceq_{t} l t r$ by simp-all
from pot this(2) have component-of-term (lt b) $\leq$ component-of-term (lt r) by
(rule is-pot-ordD2)

```
    also have ... < j by (fact assms(2))
    finally have component-of-term (lt b) <j .
    with }\langleb\in\mathrm{ set bs> show b}\in\operatorname{set (filter (\lambdab. component-of-term (lt b)<j) bs)
by simp
    qed
    ultimately show ?thesis by (rule sig-red-zero-mono)
qed
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}\in\mathrm{ set bs and component-of-term (lt b) <j
    shows \existss\inset (new-syz-sigs ss bs p).s addst term-of-pair (punit.lt (rep-list b),
j)
proof (rule sum-prodE)
    fix }x
    assume p:p=\operatorname{Inl (x,y)}
    with assms(3) have Inr j & set ( }p#\mathrm{ # ps) by simp
    with assms(1, 2) obtain s where s\in set ss and *: s addst term-of-pair (punit.lt
(rep-list b), j)
    using assms(4,5) by (rule rb-aux-inv2-E)
    from this(1) have s\in set (new-syz-sigs ss bs p) by (simp add: p)
    with * show ?thesis ..
next
    fix }
    assume p:p=Inr i
    have trans: transp (addst) by (rule transpI, drule adds-term-trans)
    from adds-term-refl have refl: reflp (addst) by (rule reflpI)
    let ?v = term-of-pair (punit.lt (rep-list b), j)
    let ?f = \lambdab. term-of-pair (punit.lt (rep-list b),i)
    define ss' where ss'= filter-min (addst) (map ?f bs)
    have eq: new-syz-sigs ss bs p = filter-min-append (addst) ss ss' by (simp add: p
ss'-def pot)
    show ?thesis
    proof (cases i=j)
        case True
        from }\langleb\in\mathrm{ set bs` have ?v v ?f' set bs unfolding <i=j> by (rule imageI)
    hence ?v \in set ss U set (map ?f bs) by simp
    thus ?thesis
    proof
            assume ?v f set ss
            hence ?v \in set ss U set ss' by simp
            with trans refl obtain s}\mathrm{ where st set (new-syz-sigs ss bs p) and s addst ?v
                unfolding eq by (rule filter-min-append-relE)
            thus ?thesis ..
    next
                assume ?v \in set (map ?f bs)
                with trans refl obtain s}\mathrm{ where s}\in\mathrm{ set ss' and s addst ? v
                unfolding ss''def by (rule filter-min-relE)
            from this(1) have s\in set ss U set ss' by simp
```

with trans refl obtain $s^{\prime}$ where $s^{\prime}: s^{\prime} \in$ set (new-syz-sigs ss bs $p$ ) and $s^{\prime}$ $a d d s_{t} s$
unfolding eq by (rule filter-min-append-relE)
from this(2) $\langle s$ addst $? v\rangle$ have $s^{\prime}$ adds $s_{t} ? v$ by (rule adds-term-trans)
with $s^{\prime}$ show ?thesis ..
qed
next
case False
with $\operatorname{assms}(3)$ have $\operatorname{Inr} j \notin \operatorname{set}(p \# p s)$ by (simp add: $p$ )
with $\operatorname{assms}(1,2)$ obtain $s$ where $s \in$ set ss and $s$ adds $s_{t}$ ? $v$ using $\operatorname{assms}(4,5)$ by (rule rb-aux-inv2-E)
from this(1) have $s \in$ set $s s \cup$ set (map ?f bs) by simp
thus?thesis
proof
assume $s \in$ set ss
hence $s \in$ set ss $\cup$ set ss' by simp
with trans refl obtain $s^{\prime}$ where $s^{\prime}: s^{\prime} \in$ set (new-syz-sigs ss bs $p$ ) and $s^{\prime}$ $a d d s_{t} s$
unfolding eq by (rule filter-min-append-relE)
from this(2) «s addst ? $v>$ have $s^{\prime}$ addst ${ }^{2}$ ?v by (rule adds-term-trans)
with $s^{\prime}$ show ?thesis ..
next
assume $s \in$ set (map ?f bs)
with trans refl obtain $s^{\prime}$ where $s^{\prime} \in$ set $s s^{\prime}$ and $s^{\prime}$ adds $s_{t} s$
unfolding $s s^{\prime}$-def by (rule filter-min-relE)
from this(1) have $s^{\prime} \in$ set ss $\cup$ set $s s^{\prime}$ by simp
with trans refl obtain $s^{\prime \prime}$ where $s^{\prime \prime}: s^{\prime \prime} \in$ set (new-syz-sigs ss bs $p$ ) and $s^{\prime \prime}$ $a d d s_{t} s^{\prime}$
unfolding eq by (rule filter-min-append-relE)
from this(2) $\left\langle s^{\prime}\right.$ adds $\left.s_{t} s\right\rangle$ have $s^{\prime \prime}$ adds $s_{t} s$ by (rule adds-term-trans)
hence $s^{\prime \prime}$ addst ?v using «s addst $\left.? v\right\rangle$ by (rule adds-term-trans)
with $s^{\prime \prime}$ show ?thesis ..
qed
qed
qed
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 $p, p s$ )
unfolding rb-aux-inv2.simps
proof (intro allI conjI impI ballI)
from $\operatorname{assms}(1)$ have inv: rb-aux-inv (bs, ss, p\# ps) by (rule rb-aux-inv2-D1)
thus rb-aux-inv (bs, new-syz-sigs ss bs $p, p s$ )
using assms(2) by (rule rb-aux-inv-preserved-1)
fix $j$
assume $j<$ length fs and Inr $j \notin$ set ps
show $f s!j \in$ ideal (rep-list'set (filter ( $\lambda b$. component-of-term (lt b) $<$ Suc $j$ ) bs))

```
proof (cases p = Inr j)
    case True
    with assms(2) have is-pred-syz (new-syz-sigs ss bs p) (term-of-pair ( O, j)) by
simp
    let ?X = set (filter ( }\lambda\mathrm{ b. component-of-term (lt b) < Suc j) bs)
    have rep-list (monomial 1 (term-of-pair ( 0,j))) \in ideal (rep-list ' ?X)
    proof (rule sig-red-zero-idealI)
            have sig-red-zero (}\mp@subsup{\prec}{t}{})(\mathrm{ set bs) (monomial 1 (term-of-pair (0, j)))
            proof (rule syzygy-crit)
                from inv have is-RB-upt dgrad rword (set bs) (sig-of-pair p)
                    by (rule rb-aux-inv-is-RB-upt-Cons)
            with dgrad have is-sig-GB-upt dgrad (set bs) (sig-of-pair p)
                    by (rule is-RB-upt-is-sig-GB-upt)
            thus is-sig-GB-upt dgrad (set bs) (term-of-pair ( }0,j)\mathrm{ ) by (simp add: <p =
Inr j>)
            next
                show monomial }1\mathrm{ (term-of-pair ( O, j)) & dgrad-sig-set dgrad
            by (rule dgrad-sig-set-closed-monomial, simp-all add: term-simps dgrad-max-0
<j< length fs>)
            next
                show lt (monomial (1::'b) (term-of-pair (0, j))) = term-of-pair (0, j) by
(simp add:lt-monomial)
            next
            from inv assms(2) have sig-crit' bs p by (rule sig-crit'I-sig-crit)
            thus is-syz-sig dgrad (term-of-pair ( }0,j)\mathrm{ ) by (simp add: <p = Inr j`)
            qed (fact dgrad)
            hence sig-red-zero ( }\mp@subsup{\preceq}{t}{})(\mathrm{ set bs) (monomial 1 (term-of-pair (0,j)))
            by (rule sig-red-zero-sing-regI)
            moreover have component-of-term (lt (monomial (1::'b) (term-of-pair (0,
j)))) < Suc j
                by (simp add: lt-monomial component-of-term-of-pair)
            ultimately show sig-red-zero ( }\mp@subsup{\preceq}{t}{})\mathrm{ ?X (monomial 1 (term-of-pair (0, j)))
                by (rule sig-red-zero-filter)
    qed
    thus ?thesis by (simp add: rep-list-monomial' «j < length fs>)
    next
        case False
        with <Inr j & set ps〉 have Inr j & set ( p# ps) by simp
        with assms(1)<j< length fs` show ?thesis by (rule rb-aux-inv2-D2)
    qed
next
    fix jb
    assume j<length fs and Inr j\not\in set ps and b\in set bs and component-of-term
(lt b)<j
    with assms(1) show \exists s\inset (new-syz-sigs ss bs p). s addst term-of-pair (punit.lt
(rep-list b), j)
    by (rule rb-aux-inv2-preserved-0)
qed
```

lemma rb-aux-inv2-preserved-3:
assumes rb-aux-inv2 (bs, ss, $p \# p s$ ) 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-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 -
from assms(1) have inv: rb-aux-inv ( $b s, s s, p \# p s$ ) by (rule rb-aux-inv2-D1) define $p^{\prime}$ where $p^{\prime}=$ sig-trd bs (poly-of-pair $p$ )
from sig-trd-red-rtrancl[of bs poly-of-pair p] have lt $p^{\prime}=l t$ (poly-of-pair $p$ ) unfolding $p^{\prime}$-def by (rule sig-red-regular-rtrancl-lt)
also have $\ldots=$ sig-of-pair $p$ by (rule sym, rule pair-list-sig-of-pair, fact inv, $\operatorname{simp})$
finally have $l t-p^{\prime}$ : lt $p^{\prime}=$ sig-of-pair $p$.
show ?thesis unfolding rb-aux-inv2.simps $p^{\prime}$-def[symmetric]
proof (intro allI conjI impI ballI)
show rb-aux-inv ( $p^{\prime} \#$ bs, new-syz-sigs ss bs $p$, add-spairs ps bs $p^{\prime}$ )
unfolding $p^{\prime}$-def using inv assms(2, 3) by (rule rb-aux-inv-preserved-3)
next
fix $j$
assume $j<$ length $f s$ and $*:$ Inr $j \notin$ set (add-spairs ps bs $p^{\prime}$ )
show $f s!j \in$ ideal (rep-list' set (filter ( $\lambda b$. component-of-term (lt b) $<S u c j$ )
( $\left.p^{\prime} \# b s\right)$ ))
proof (cases $p=\operatorname{Inr} j$ )
case True
let ? $X=\operatorname{set}\left(\right.$ filter $(\lambda b$. component-of-term $\left.(l t b)<S u c j)\left(p^{\prime} \# b s\right)\right)$
have rep-list (monomial 1 (term-of-pair $(0, j))$ ) $\in$ ideal (rep-list ‘?X) proof (rule sig-red-zero-idealI)
have sig-red-zero $\left(\preceq_{t}\right)\left(\operatorname{set}\left(p^{\prime} \# b s\right)\right)($ monomial 1 (term-of-pair $\left.(0, j))\right)$
proof (rule sig-red-zeroI)
have $\left(\text { sig-red }\left(\prec_{t}\right)(\preceq)(\text { set bs })\right)^{* *}($ monomial $1($ term-of-pair $(0, j))) p^{\prime}$
using sig-trd-red-rtrancl[of bs poly-of-pair p] by (simp add: True $p^{\prime}$-def)
moreover have set $b s \subseteq$ set ( $p^{\prime} \# b s$ ) by fastforce
ultimately have $\left(\operatorname{sig}-r e d ~\left(\prec_{t}\right)(\preceq)\left(\text { set }\left(p^{\prime} \# b s\right)\right)\right)^{* *}($ monomial 1
(term-of-pair $(0, j))) p^{\prime}$
by (rule sig-red-rtrancl-mono)
hence $\left(\operatorname{sig} \text {-red }\left(\preceq_{t}\right)(\preceq)\left(\operatorname{set}\left(p^{\prime} \# b s\right)\right)\right)^{* *}($ monomial 1 (term-of-pair ( 0 , j))) $p^{\prime}$
by (rule sig-red-rtrancl-sing-regI)
also have sig-red $\left(\preceq_{t}\right)(\preceq)\left(\right.$ set $\left.\left(p^{\prime} \# b s\right)\right) p^{\prime} 0$ unfolding sig-red-def
proof (intro exI bexI)
from $\operatorname{assms}(3)$ have rep-list $p^{\prime} \neq 0$ by (simp add: $p^{\prime}$-def)
show sig-red-single $\left(\preceq_{t}\right)(\preceq) p^{\prime} 0 p^{\prime} 0$
proof (rule sig-red-singleI)
show rep-list $p^{\prime} \neq 0$ by fact
next
from $\left\langle r e p-l i s t p^{\prime} \neq 0\right\rangle$ have punit.lt $\left(\right.$ rep-list $\left.p^{\prime}\right) \in$ keys $\left(\right.$ rep-list $\left.p^{\prime}\right)$
by (rule punit.lt-in-keys)
thus $0+$ punit.lt (rep-list $\left.p^{\prime}\right) \in$ keys (rep-list $p^{\prime}$ ) by simp
next
from $\left\langle\right.$ rep－list $\left.p^{\prime} \neq 0\right\rangle$ have punit．lc（rep－list $\left.p^{\prime}\right) \neq 0$ by（rule punit．lc－not－0）
thus $0=p^{\prime}-$ monom－mult（lookup（rep－list $\left.p^{\prime}\right)(0+$ punit．lt（rep－list $\left.\left.p^{\prime}\right)\right) /$ punit．lc（rep－list $\left.\left.p^{\prime}\right)\right) 0 p^{\prime}$ by（simp add：punit．lc－def［symmetric］）
qed（simp－all add：term－simps）
qed $\operatorname{simp}$
finally show $\left(\operatorname{sig}-r e d ~\left(\preceq_{t}\right)(\preceq)\left(\operatorname{set}\left(p^{\prime} \# b s\right)\right)\right)^{* *}($ monomial 1 （term－of－pair $(0, j))) 0$ ．
qed（fact rep－list－zero）
moreover have component－of－term（lt（monomial（1：：＇b）（term－of－pair（0， j））））$<$ Suc $j$
by（simp add：lt－monomial component－of－term－of－pair）
ultimately show sig－red－zero $\left(\preceq_{t}\right)$ ？$X($ monomial 1 （term－of－pair $(0, j))$ ）
by（rule sig－red－zero－filter）
qed
thus ？thesis by（simp add：rep－list－monomial＇$\langle j<$ length $f s\rangle)$
next
case False
from＊have Inr $j \notin$ set ps by（simp add：add－spairs－def set－merge－wrt）
hence Inr $j \notin \operatorname{set}(p \# p s)$ using False by simp
with assms（1）〈j＜length fs〉
have $f_{s}!j \in$ ideal（rep－list＇set（filter（ $\lambda b$ ．component－of－term（lt b）$<$ Suc j）$b s$ ）
by（rule rb－aux－inv2－D2）
also have $\ldots \subseteq$ ideal（rep－list＇set（filter（ $\lambda$ b．component－of－term（lt b）$<$ Suc j）$\left.\left(p^{\prime} \# b s\right)\right)$ ）
by（intro ideal．span－mono image－mono，fastforce）
finally show ？thesis ．
qed
next
fix $j$ and $b::^{\prime} t \Rightarrow_{0}{ }^{\prime} b$
assume $j<$ length $f s$ and $*$ ：component－of－term（lt b）$<j$
assume Inr $j \notin$ set（add－spairs ps bs $p^{\prime}$ ）
hence $\operatorname{Inr} j \notin$ set ps by（simp add：add－spairs－def set－merge－wrt）
assume $b \in \operatorname{set}\left(p^{\prime} \# b s\right)$
hence $b=p^{\prime} \vee b \in$ set bs by simp
thus $\exists$ s $\in$ set（new－syz－sigs ss bs p）．s addst term－of－pair（punit．lt（rep－list b），
j）
proof
assume $b=p^{\prime}$
with $*$ have component－of－term（sig－of－pair p）＜component－of－term（term－of－pair $(0, j))$
by（simp only：lt－p＇component－of－term－of－pair）
with pot have $* *$ ：sig－of－pair $p \prec_{t}$ term－of－pair $(0, j)$ by（rule is－pot－ordD）
have $p \in \operatorname{set}(p \# p s)$ by $\operatorname{simp}$
with inv have Inr $j \in \operatorname{set}(p \# p s)$ using $\langle j<l e n g t h ~ f s 〉 * *$ by（rule rb－aux－inv－D6－2）
with $\langle I n r j \notin$ set $p s\rangle$ have $p=\operatorname{Inr} j$ by simp

```
            with ** show ?thesis by simp
        next
            assume b f set bs
            with assms(1)<j<length fs\rangle\langleInr j & set ps〉 show ?thesis
                using * by (rule rb-aux-inv2-preserved-0)
    qed
    qed
qed
lemma rb-aux-inv2-ideal-subset:
    assumes rb-aux-inv2 (bs, ss, ps) and }\bigwedgep0.p0\in set ps \Longrightarrowj\leqcomponent-of-term
(sig-of-pair p0)
    shows ideal (set (take jfs))\subseteqideal (rep-list'set (filter (\lambdab. component-of-term
(lt b)<j) bs))
    (is ideal ?B\subseteqideal ?A)
proof (intro ideal.span-subset-spanI subsetI)
    fix f
    assume }f\in\mathrm{ ? B
    then obtain i where i< length (take jfs) and f=(take jfs)!i
        by (metis in-set-conv-nth)
    hence i< length fs and i<j and f:f=fs !i by auto
    from this(2) have Suc i\leqj by simp
    have f}\in\mathrm{ ideal (rep-list'set (filter ( }\lambdab\mathrm{ . component-of-term (lt b)<Suc i) bs))
        unfolding f using assms(1)<i< length fs`
    proof (rule rb-aux-inv2-D2)
        show Inr i\not\in set ps
        proof
            assume Inr i\in set ps
            hence j\leq component-of-term(sig-of-pair (Inr i)) by (rule assms(2))
            hence j\leqi by (simp add: component-of-term-of-pair)
            with }\langlei<j\rangle\mathrm{ show False by simp
        qed
    qed
    also have ...\subseteq ideal ?A
        by (intro ideal.span-mono image-mono, auto dest: order-less-le-trans[OF - <Suc
i\leqj>])
    finally show f}\in\mathrm{ ideal ?A .
qed
lemma rb-aux-inv-is-Groebner-basis:
    assumes hom-grading dgrad and rb-aux-inv (bs, ss, ps)
        and }\p0.p0\in set ps \Longrightarrowj\leq component-of-term (sig-of-pair p0
    shows punit.is-Groebner-basis (rep-list' set (filter (\lambdab. component-of-term (lt b)
< j)bs))
            (is punit.is-Groebner-basis (rep-list'set ?bs))
    using dgrad assms(1)
proof (rule is-sig-GB-upt-is-Groebner-basis)
    show set ?bs \subseteqdgrad-sig-set' j dgrad
    proof
```

fix $b$
assume $b \in$ set ?bs
hence $b \in$ set $b s$ and component-of-term $(l t b)<j$ by simp-all
show $b \in d g r a d-s i g-s e t^{\prime} j$ dgrad unfolding dgrad-sig-set'-def
proof
from $\operatorname{assms}(2)$ have $r b-a u x-i n v 1$ bs by (rule rb-aux-inv-D1)
hence set bs $\subseteq d g r a d$-sig-set dgrad by (rule rb-aux-inv1-D1)
with $\langle b \in$ set $b s\rangle$ have $b \in$ dgrad-sig-set dgrad ..

next
show $b \in$ sig-inv-set ${ }^{\prime} j$
proof (rule sig-inv-setI')
fix $v$
assume $v \in$ keys $b$
hence $v \preceq_{t}$ lt $b$ by (rule lt-max-keys)
with pot have component-of-term $v \leq$ component-of-term (lt b) by (rule
is-pot-ordD2)
also have $\ldots<j$ by fact
finally show component-of-term $v<j$.
qed
qed
qed
next
fix $u$
assume $u$ : component-of-term $u<j$
from dgrad have is-sig-GB-upt dgrad (set bs) (term-of-pair $(0, j))$
proof (rule is-RB-upt-is-sig-GB-upt)
from $\operatorname{assms}(2)$ show $i s$ - $R B$-upt dgrad rword (set bs) (term-of-pair $(0, j))$ proof (rule rb-aux-inv-is-RB-upt)
fix $p$
assume $p \in$ set $p s$
hence $j \leq$ component-of-term (sig-of-pair $p$ ) by (rule assms(3))
with pot show term-of-pair $(0, j) \preceq_{t}$ sig-of-pair $p$
by (auto simp: is-pot-ord term-simps zero-min)
qed
qed
moreover from pot have $u \prec_{t}$ term-of-pair $(0, j)$
by (rule is-pot-ordD) (simp only: u component-of-term-of-pair)
ultimately have 1: is-sig-GB-in dgrad (set bs) u by (rule is-sig-GB-uptD2)
show is-sig-GB-in dgrad (set ?bs) u
proof (rule is-sig-GB-inI)
fix $r::{ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b$
assume $l t r=u$
assume $r \in d g r a d$-sig-set dgrad
with 1 have sig-red-zero $\left(\preceq_{t}\right)($ set bs) rusing $\langle l t r=u\rangle$ by (rule is-sig-GB-inD) moreover from $u$ have component-of-term (lt $r$ ) < $j$ by (simp only: <lt $r=$ $u\rangle$ )
ultimately show sig-red-zero $\left(\preceq_{t}\right)($ set ?bs) r by (rule sig-red-zero-filter)
qed

## qed

lemma rb-aux-inv2-no-zero-red:
assumes hom-grading dgrad and is-regular-sequence $f s$ and rb-aux-inv2 (bs, ss, $p \# p s)$
and $\neg$ sig-crit bs (new-syz-sigs ss bs p) $p$
shows rep-list (sig-trd bs (poly-of-pair p)) $\neq 0$
proof
from $\operatorname{assms}(3)$ have inv: rb-aux-inv (bs, ss, $p \# p s$ ) by (rule rb-aux-inv2-D1)
moreover have $p \in \operatorname{set}(p \# p s)$ by simp
ultimately have sig-p: sig-of-pair $p=l t$ (poly-of-pair $p$ ) and poly-of-pair $p \neq 0$
and $p$-in: poly-of-pair $p \in d g r a d-s i g-s e t ~ d g r a d ~$
by (rule pair-list-sig-of-pair, rule pair-list-nonzero, rule pair-list-dgrad-sig-set)
from this(2) have $l c$ (poly-of-pair $p) \neq 0$ by (rule lc-not-0)
from inv have rb-aux-inv1 bs by (rule rb-aux-inv-D1)
hence bs-sub: set bs $\subseteq$ dgrad-sig-set dgrad by (rule rb-aux-inv1-D1)
define $p^{\prime}$ where $p^{\prime}=$ sig-trd bs (poly-of-pair $p$ )
define $j$ where $j=$ component-of-term (lt $p^{\prime}$ )
define $q$ where $q=$ lookup (vectorize-poly $p^{\prime}$ ) $j$
let ?bs $=$ filter $(\lambda b$. component-of-term $(l t b)<j) b s$
let $? f s=$ take $(S u c j) f s$
have $p^{\prime} \in d g r a d$-sig-set dgrad unfolding $p^{\prime}$-def using dgrad bs-sub $p$-in sig-trd-red-rtrancl
by (rule dgrad-sig-set-closed-sig-red-rtrancl)
hence $p^{\prime} \in$ sig-inv-set by (simp add: dgrad-sig-set'-def)
have $l t$ - $p^{\prime}: l t p^{\prime}=l t($ poly-of-pair $p)$ and $l c p^{\prime}=l c($ poly-of-pair $p)$
unfolding $p^{\prime}$-def using sig-trd-red-rtrancl
by (rule sig-red-regular-rtrancl-lt, rule sig-red-regular-rtrancl-lc)
from this(2) <lc (poly-of-pair $p) \neq 0 〉$ have $p^{\prime} \neq 0$ by (simp add: lc-eq-zero-iff[symmetric])
hence $l t p^{\prime} \in$ keys $p^{\prime}$ by (rule lt-in-keys)
hence $j \in$ keys (vectorize-poly $p^{\prime}$ ) by (simp add: keys-vectorize-poly $j$-def)
hence $q \neq 0$ by (simp add: $q$-def in-keys-iff)
from $\left\langle p^{\prime} \in\right.$ sig-inv-set $\rangle\left\langle l t p^{\prime} \in\right.$ keys $\left.p^{\prime}\right\rangle$ have $j<$ length $f s$
unfolding $j$-def by (rule sig-inv-set $D^{\prime}$ )
with le-refl have $f s!j \in$ set (drop $j f s$ ) by (rule nth-in-set-dropI)
with $f s$-distinct le-refl have $0: f s!j \notin$ set (take $j f s$ )
by (auto dest: set-take-disj-set-drop-if-distinct)
have 1: $j \leq$ component-of-term (sig-of-pair p0) if $p 0 \in \operatorname{set}(p \# p s)$ for $p 0$
proof -
from that have $p 0=p \vee p 0 \in$ set $p s$ by simp
thus ?thesis
proof
assume $p 0=p$
thus ?thesis by (simp add: $j$-def lt-p' sig-p)
next

```
    assume p0 \in set ps
    from inv have sorted-wrt pair-ord (p# ps) by (rule rb-aux-inv-D5)
    hence Ball (set ps) (pair-ord p) by simp
    hence pair-ord p p0 using <p0 \in set ps`..
    hence lt p' }\mp@subsup{\preceq}{t}{}\mathrm{ sig-of-pair p0 by (simp add: pair-ord-def lt-p' sig-p)
    thus ?thesis using pot by (auto simp add: is-pot-ord j-def term-simps)
    qed
qed
with assms(1) inv have gb: punit.is-Groebner-basis (rep-list'set ?bs)
    by (rule rb-aux-inv-is-Groebner-basis)
have p}\mp@subsup{p}{}{\prime}\in\mathrm{ sig-inv-set' (Suc j)
proof (rule sig-inv-setI')
    fix v
    assume v\in keys p'
    hence v}\mp@subsup{\preceq}{t}{lt p' by (rule lt-max-keys)
    with pot have component-of-term v\leqjunfolding j-def by (rule is-pot-ordD2)
    thus component-of-term v<Suc j by simp
qed
hence 2: keys (vectorize-poly p}\mp@subsup{}{}{\prime})\subseteq{0..<Suc j} by (rule sig-inv-setD
moreover assume rep-list p'=0
ultimately have 0=(\sumk\inkeys (pm-of-idx-pm ?fs (vectorize-poly p}\mp@subsup{p}{}{\prime})\mathrm{ ).
                    lookup (pm-of-idx-pm?fs (vectorize-poly p')) k*k)
    by (simp add: rep-list-def ideal.rep-def pm-of-idx-pm-take)
    also have ... = (\sumk\inset ?fs.lookup (pm-of-idx-pm?fs (vectorize-poly p'))k*
k)
    using finite-set keys-pm-of-idx-pm-subset by (rule sum.mono-neutral-left) (simp
add: in-keys-iff)
    also from 2 have ... = (\sumk\inset ?fs. lookup (pm-of-idx-pm fs (vectorize-poly
p}\mp@subsup{}{}{\prime}))k*k
    by (simp only: pm-of-idx-pm-take)
    also have ... = lookup (pm-of-idx-pm fs (vectorize-poly p')) (fs!j)*fs!j +
                            (\sumk\inset (take jfs). lookup (pm-of-idx-pm fs (vectorize-poly p'))k
* k)
    using <j< length fs` by (simp add: take-Suc-conv-app-nth q-def sum.insert[OF
finite-set 0])
    also have ... = q*fs !j + (\sumk\inset (take j fs). lookup (pm-of-idx-pm fs
(vectorize-poly p')) k*k)
    using fs-distinct <j < length fs> by (simp only:lookup-pm-of-idx-pm-distinct
q-def)
    finally have - (q*fs!j)=
                                    (\sumk\inset (take j fs). lookup (pm-of-idx-pm fs (vectorize-poly
p')) k*k
    by (simp add: add-eq-0-iff)
    hence - (q*fs! j) \inideal (set (take jfs)) by (simp add: ideal.sum-in-spanI)
    hence - (- (q*fs!j)) \in ideal (set (take jfs)) by (rule ideal.span-neg)
    hence q*fs ! j\in ideal (set (take jfs)) by simp
    with assms(2)<j<length fs` have q\inideal (set (take j fs)) by (rule is-regular-sequenceD)
```

also from $\operatorname{assms}(3) 1$ have $\ldots \subseteq$ ideal (rep-list'set ?bs)
by (rule rb-aux-inv2-ideal-subset)
finally have $q \in$ ideal (rep-list' set?bs) .
with $g b$ obtain $g$ where $g \in$ rep-list'set ?bs and $g \neq 0$ and punit.lt $g$ adds punit.lt $q$
using $\langle q \neq 0\rangle$ by (rule punit.GB-adds-lt[simplified])
from this(1) obtain $b$ where $b \in$ set bs and component-of-term (lt $b)<j$ and $g: g=$ rep-list $b$
by auto
from $\operatorname{assms}(3)\langle j<l e n g t h ~ f s\rangle-t h i s(1,2)$
have $\exists$ s $\in$ set (new-syz-sigs ss bs p). s addst term-of-pair (punit.lt (rep-list b), j)
proof (rule rb-aux-inv2-preserved-0)
show Inr $j \notin$ set ps
proof
assume $\operatorname{Inr} j \in$ set $p s$
with inv have sig-of-pair $p \neq$ term-of-pair $(0, j)$ by (rule Inr-in-tailD)
hence lt $p^{\prime} \neq$ term-of-pair $(0, j)$ by (simp add: lt- $p^{\prime}$ sig-p)
from inv have sorted-wrt pair-ord ( $p \# p s$ ) by (rule rb-aux-inv-D5)
hence Ball (set ps) (pair-ord p) by simp
hence pair-ord $p$ (Inr $j$ ) using $\langle\operatorname{Inr} j \in \operatorname{set} p s\rangle .$.
hence lt $p^{\prime} \preceq_{t}$ term-of-pair $(0, j)$ by (simp add: pair-ord-def lt-p' sig-p)
hence $l p p^{\prime} \preceq 0$ using pot by (simp add: is-pot-ord j-def term-simps)
hence $l p p^{\prime}=0$ using zero-min by (rule ordered-powerprod-lin.order-antisym)
hence lt $p^{\prime}=$ term-of-pair $(0, j)$ by (metis $j$-def term-of-pair-pair)
with $\left\langle l t p^{\prime} \neq\right.$ term-of-pair $\left.(0, j)\right\rangle$ show False ..
qed
qed
then obtain $s$ where $s$-in: $s \in$ set (new-syz-sigs ss bs $p$ ) and $s$ adds $s_{t}$ term-of-pair (punit.lt $g, j$ )
unfolding $g$..
from this(2) «punit.lt $g$ adds punit.lt $q\rangle$ have $s$ adds $s_{t}$ term-of-pair (punit.lt $q, j$ ) by (metis adds-minus-splus adds-term-splus component-of-term-of-pair pp-of-term-of-pair)
also have $\ldots=l t p^{\prime}$ by (simp only: $q$-def $j$-def lt-lookup-vectorize term-simps)
finally have $s$ addst sig-of-pair $p$ by (simp only: lt-p' sig-p)
with $s$-in have pred: is-pred-syz (new-syz-sigs ss bs $p$ ) (sig-of-pair $p$ )
by (auto simp: is-pred-syz-def)
have sig-crit bs (new-syz-sigs ss bs p) $p$
proof (rule sum-prodE)
fix $x y$
assume $p=\operatorname{Inl}(x, y)$
thus ?thesis using pred by (auto simp: ord-term-lin.max-def split: if-splits)
next
fix $i$
assume $p=$ Inr $i$
thus ?thesis using pred by simp
qed
with $\operatorname{assms}(4)$ show False ..
qed

```
corollary rb-aux-no-zero-red':
    assumes hom-grading dgrad and is-regular-sequence \(f\) s and rb-aux-inv2 (fst args)
    shows snd (rb-aux args) \(=\) snd args
proof -
    from assms(3) have rb-aux-inv (fst args) by (rule rb-aux-inv2-D1)
    hence rb-aux-dom args by (rule rb-aux-domI)
    thus ?thesis using assms(3)
    proof (induct args rule: rb-aux.pinduct)
        case ( 1 bs ss \(z\) )
        show ?case by (simp only: rb-aux.psimps(1)[OF 1(1)])
    next
        case (2 bs ss p ps z)
        from 2(5) have *: rb-aux-inv2 ( \(b s\), ss, \(p \# p s\) ) by (simp only: fst-conv)
        show ?case
        proof (simp add: rb-aux.psimps(2)[OF 2(1)] Let-def, intro conjI impI)
            note refl
            moreover assume sig-crit bs (new-syz-sigs ss bs p) p
            moreover from * this have rb-aux-inv2 (fst ((bs, new-syz-sigs ss bs p, ps),
z))
            unfolding fst-conv by (rule rb-aux-inv2-preserved-1)
            ultimately have snd (rb-aux ((bs, new-syz-sigs ss bs p, ps), z)) =
                                    snd ((bs, new-syz-sigs ss bs \(p, p s), z)\) by (rule 2(2))
            thus snd (rb-aux ((bs, new-syz-sigs ss bs \(p, p s), z))=z\) by (simp only:
snd-conv)
            thus snd (rb-aux \(((b s\), new-syz-sigs ss bs \(p, p s), z))=z\).
        next
            assume \(\neg\) sig-crit bs (new-syz-sigs ss bs p) \(p\)
            with \(\operatorname{assms}(1,2) *\) have rep-list (sig-trd bs \((\) poly-of-pair \(p)) \neq 0\)
                by (rule rb-aux-inv2-no-zero-red)
            moreover assume rep-list (sig-trd bs (poly-of-pair p)) \(=0\)
            ultimately show snd (rb-aux ((bs, lt (sig-trd bs (poly-of-pair p)) \#
                new-syz-sigs ss bs \(p, p s)\), Suc \(z))=z .\).
    next
            define \(p^{\prime}\) where \(p^{\prime}=\) sig-trd bs (poly-of-pair \(p\) )
            note refl
            moreover assume \(a: \neg\) sig-crit bs (new-syz-sigs ss bs p) \(p\)
            moreover note \(p^{\prime}\)-def
            moreover assume \(b\) : rep-list \(p^{\prime} \neq 0\)
            moreover have rb-aux-inv2 (fst ( \(\left(p^{\prime} \#\right.\) bs, new-syz-sigs ss bs \(p\), add-spairs
ps bs \(\left.\left.p^{\prime}\right), z\right)\) )
            using \(* a b\) unfolding fst-conv \(p^{\prime}\)-def by (rule rb-aux-inv2-preserved-3)
            ultimately have snd (rb-aux ( \(p^{\prime} \#\) bs, new-syz-sigs ss bs \(p\), add-spairs \(p s\)
bs \(\left.\left.p^{\prime}\right), z\right)\) ) \(=\)
                    snd (( \(p^{\prime} \#\) bs, new-syz-sigs ss bs \(p\), add-spairs ps bs \(\left.\left.p^{\prime}\right), z\right)\)
                by (rule 2(4))
            thus snd (rb-aux \(\left(\left(p^{\prime} \# b s\right.\right.\), new-syz-sigs ss bs \(p\), add-spairs ps bs \(\left.\left.\left.p^{\prime}\right), z\right)\right)=z\)
                by (simp only: snd-conv)
    qed
qed
```


## qed

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 $f s]), z))=z$
proof -
let ?args $=\left(\left([]::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right.\right.$ list, Koszul-syz-sigs $f$ s,
$($ map Inr $[0 . .<$ length $f s])::\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)+$ nat $)$ list $\left.), z\right)$
from rb-aux-inv-init have rb-aux-inv2 (fst ?args) by simp
with assms have snd (rb-aux ?args) = snd ?args by (rule rb-aux-no-zero-red')
thus ?thesis by (simp only: snd-conv)
qed
corollary rb-no-zero-red:
assumes hom-grading dgrad and is-regular-sequence fs
shows $s n d r b=0$
using rb-aux-no-zero-red[OF assms, of 0] by (auto simp: rb-def split: prod.split)
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 $(A x)=A-s p p(s p p-o f x)$.
fun spp-inv-pair :: $\left.\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \times\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right)\right)+n a t\right) \Rightarrow$ bool where $\operatorname{spp-inv-pair}(\operatorname{Inl}(p, q))=(\operatorname{spp-inv} p \wedge \operatorname{spp-inv} q) \mid$ spp-inv-pair $(\operatorname{Inr} j)=$ True
fun app-pair :: $\left({ }^{\prime} x \Rightarrow{ }^{\prime} y\right) \Rightarrow\left(\left({ }^{\prime} x \times{ }^{\prime} x\right)+n a t\right) \Rightarrow\left(\left({ }^{\prime} y \times{ }^{\prime} y\right)+n a t\right)$ where app-pair $f(\operatorname{Inl}(p, q))=\operatorname{Inl}(f p, f q) \mid$
$\operatorname{app}-$ pair $f(\operatorname{Inr} j)=\operatorname{Inr} j$
fun app-args :: $\left({ }^{\prime} x \Rightarrow{ }^{\prime} y\right) \Rightarrow\left(\left({ }^{\prime} x\right.\right.$ list $\times{ }^{\prime} z \times\left(\left(\left({ }^{\prime} x \times{ }^{\prime} x\right)+n a t\right)\right.$ list $\left.)\right) \times$ nat $) \Rightarrow$

$$
\left(\left({ }^{\prime} y \text { list } \times{ }^{\prime} z \times\left(\left(\left({ }^{\prime} y \times{ }^{\prime} y\right)+n a t\right) \text { list }\right)\right) \times n a t\right) \text { where }
$$

app-args $f((a s, b s, c s), n)=(($ map $f a s, b s$, map $($ app-pair $f) c s), n)$
lemma app-pair-spp-of-vec-of:
assumes spp-inv-pair $p$
shows app-pair spp-of (app-pair vec-of $p$ ) $=p$
proof (rule sum-prodE)
fix $a b$
assume $p: p=\operatorname{Inl}(a, b)$
from assms have spp-inv $a$ and spp-inv b by (simp-all add: $p$ )
thus ?thesis by (simp add: p spp-of-vec-of)
qed $\operatorname{simp}$

```
lemma map-app-pair-spp-of-vec-of:
    assumes list-all spp-inv-pair ps
    shows map (app-pair spp-of ○ app-pair vec-of) ps=ps
proof (rule map-idI)
    fix p
    assume p\in set ps
    with assms have spp-inv-pair p by (simp add: list-all-def)
    hence app-pair spp-of (app-pair vec-of p) = p by (rule app-pair-spp-of-vec-of)
    thus (app-pair spp-of ○ app-pair vec-of) p=p by simp
qed
lemma snd-app-args: snd (app-args f args) = snd args
    by (metis prod.exhaust app-args.simps snd-conv)
lemma new-syz-sigs-alt-spp:
    new-syz-sigs ss bs p = new-syz-sigs-spp ss (map spp-of bs) (app-pair spp-of p)
proof (rule sum-prodE)
    fix ab
    assume p=Inl (a,b)
    thus ?thesis by simp
next
    fix }
    assume p = Inr j
    thus ?thesis by (simp add: comp-def spp-of-def)
qed
lemma is-rewritable-alt-spp:
    assumes 0 & set bs
    shows is-rewritable bs pu= is-rewritable-spp (map spp-of bs) (spp-of p)u
proof -
    from assms have b\in set bs \Longrightarrowb\not=0 for b by blast
    thus ?thesis by (auto simp: is-rewritable-def is-rewritable-spp-def fst-spp-of)
qed
lemma spair-sigs-alt-spp: spair-sigs p q = spair-sigs-spp (spp-of p) (spp-of q)
    by (simp add: spair-sigs-def spair-sigs-spp-def Let-def fst-spp-of snd-spp-of)
lemma sig-crit-alt-spp:
    assumes 0 & set bs
    shows sig-crit bs ss p = sig-crit-spp (map spp-of bs) ss (app-pair spp-of p)
proof (rule sum-prodE)
    fix ab
    assume p:p=Inl (a,b)
    from assms show ?thesis by (simp add: p spair-sigs-alt-spp is-rewritable-alt-spp)
qed simp
lemma spair-alt-spp:
    assumes is-regular-spair p q
    shows spp-of (spair p q) = spair-spp (spp-of p) (spp-of q)
```

```
proof -
    let ?t1 = punit.lt (rep-list p)
    let ?t2 = punit.lt (rep-list q)
    let ?l = lcs ?t1 ?t2
    from assms have p: rep-list p\not=0 and q: rep-list q}\not=
        by (rule is-regular-spairD1, rule is-regular-spairD2)
    hence }p\not=0\mathrm{ and }q\not=0\mathrm{ and 1: punit.lc (rep-list p)}\not=0\mathrm{ and 2: punit.lc (rep-list
q) }\not=
    by (auto simp: rep-list-zero punit.lc-eq-zero-iff)
    from assms have lt (monom-mult (1 / punit.lc (rep-list p)) (?l - ?t1) p)\not=
                                    lt (monom-mult (1 / punit.lc (rep-list q)) (?l - ?t2) q) (is ?u =
?v)
    by (rule is-regular-spairD3)
    hence lt (monom-mult (1 / punit.lc (rep-list p)) (?l - ?t1) p - monom-mult (1
/ punit.lc (rep-list q)) (?l - ?t2) q)=
    ord-term-lin.max ?u ?v by (rule lt-minus-distinct-eq-max)
    moreover from \langlep\not=0\rangle1 have ?u = (?l - ?t1) \oplus fst (spp-of p) by (simp
add: lt-monom-mult fst-spp-of)
    moreover from }\langleq\not=0\rangle2\mathrm{ have ?v = (?l - ?t2) }\oplus\mathrm{ fst (spp-of q) by (simp add:
lt-monom-mult fst-spp-of)
    ultimately show ?thesis
    by (simp add: spair-spp-def spair-def Let-def spp-of-def rep-list-minus rep-list-monom-mult)
qed
lemma sig-trd-spp-body-alt-Some:
    assumes find-sig-reducer (map spp-of bs) v (punit.lt p) 0 = Some i
    shows sig-trd-spp-body (map spp-of bs) v (p,r)=
                            (punit.lower (p - local.punit.monom-mult (punit.lc p / punit.lc (rep-list
(bs!i)))
                    (punit.lt p - punit.lt (rep-list (bs!i))) (rep-list (bs!i))) (punit.lt
p),r)
            (is ?thesis1)
        and sig-trd-spp-body (map spp-of bs) v (p,r) =
            (p - local.punit.monom-mult (punit.lc p / punit.lc (rep-list (bs!i)))
                    (punit.lt p - punit.lt (rep-list (bs!i)))(rep-list (bs!i)),r)
            (is ?thesis2)
proof -
    have ?thesis1 ^ ?thesis2
    proof (cases p=0)
        case True
        show ?thesis by (simp add: assms, simp add: True)
    next
        case False
        from assms have i< length bs by (rule find-sig-reducer-SomeD)
        hence eq1: snd (map spp-of bs!i)= rep-list (bs!i) by (simp add: snd-spp-of)
            from assms have rep-list (bs!i)}\not=0\mathrm{ and punit.lt (rep-list (bs !i)) adds
punit.lt p
            by (rule find-sig-reducer-SomeD)+
            hence nz: rep-list (bs!i)\not=0 and adds: punit.lt (rep-list (bs!i)) adds punit.lt
```

p
by (simp-all add: snd-spp-of)
from $n z$ have punit.lc (rep-list $(b s!i)) \neq 0$ by (rule punit.lc-not- 0 )
moreover from False have punit.lc $p \neq 0$ by (rule punit.lc-not-0)
ultimately have eq2: punit.lt (punit.monom-mult (punit.lc p / punit.lc (rep-list ( $b s!i)$ )

$$
(\text { punit.lt } p-\text { punit.lt }(\text { rep-list }(b s!i)))(\text { rep-list }(b s!i)))=
$$

punit.lt $p$
(is punit.lt ? $p=-$ ) using $n z$ adds by (simp add: lp-monom-mult adds-minus)
have ?thesis1 by (simp add: assms Let-def eq1 punit.lower-minus punit.tail-monom-mult[symmetric], simp add: punit.tail-def eq2)
moreover have ?thesis2
proof (simp add:〈?thesis1〉 punit.lower-id-iff disj-commute $[$ of $p=$ ? $p]$ del:
sig-trd-spp-body.simps)
show punit.lt $(p-$ ? $p) \prec$ punit.lt $p \vee p=$ ?p
proof (rule disjCI)
assume $p \neq ? p$
hence $p-? p \neq 0$ by simp
moreover note eq2
moreover from <punit.lc (rep-list $(b s!i)) \neq 0\rangle$ have punit.lc ? $p=$ punit.lc
$p$ by $\operatorname{simp}$
ultimately show punit.lt $(p-? p) \prec$ punit.lt $p$ by (rule punit.lt-minus-lessI)
qed
qed
ultimately show ?thesis ..
qed
thus ?thesis1 and ?thesis2 by blast+
qed
lemma sig-trd-aux-alt-spp:
assumes fst args $\in$ keys (rep-list (snd args))
shows rep-list (sig-trd-aux bs args) $=$
sig-trd-spp-aux (map spp-of bs) (lt (snd args))
(rep-list (snd args) - punit.higher (rep-list (snd args)) (fst args), punit.higher (rep-list (snd args)) (fst args))
proof -
from assms have sig-trd-aux-dom bs args by (rule sig-trd-aux-domI)
thus ?thesis using assms
proof (induct args rule: sig-trd-aux.pinduct)
case ( $1 t p$ )
define $p^{\prime}$ where $p^{\prime}=($ case find-sig-reducer (map spp-of bs) (lt p) t 0 of

$$
\text { None } \Rightarrow p
$$

| Some $i \Rightarrow p-$ monom-mult (lookup (rep-list p) $t /$ punit.lc (rep-list (bs
! i) ) )

$$
(t-\text { punit.lt }(\text { rep-list }(b s!i)))(b s!i))
$$

define $p^{\prime \prime}$ where $p^{\prime \prime}=$ punit.lower (rep-list $p^{\prime}$ ) $t$
from 1 (3) have $t$-in: $t \in$ keys (rep-list p) by simp
hence $t \in$ keys (rep-list $p-$ punit.higher (rep-list $p) t$ ) (is $-\in$ keys ?p)

```
        by (simp add: punit.keys-minus-higher)
    hence ? }p\not=0\mathrm{ by auto
    hence eq1: sig-trd-spp-aux bs0 v0 (?p,r0) = sig-trd-spp-aux bs0 v0 (sig-trd-spp-body
bs0 v0 (?p,r0))
            for bs0 v0 r0 by (simp add: sig-trd-spp-aux-simps del: sig-trd-spp-body.simps)
    from t-in have lt-p: punit.lt ?p = t and lc-p: punit.lc ?p = lookup (rep-list p)
t
            and tail-p: punit.tail ?p = punit.lower (rep-list p) t
    by (rule punit.lt-minus-higher, rule punit.lc-minus-higher, rule punit.tail-minus-higher)
    have lt p' = lt p ^ punit.higher (rep-list p')t= punit.higher (rep-list p) t\wedge
            (\forall i. find-sig-reducer (map spp-of bs) (lt p)t 0=Some i \longrightarrowlookup (rep-list
p') }t=0\mathrm{ )
            (is ?A}\wedge?B\wedge?C
    proof (cases find-sig-reducer (map spp-of bs) (lt p) t 0)
            case None
            thus ?thesis by (simp add: p'-def)
    next
            case (Some i)
            hence }\mp@subsup{p}{}{\prime}:\mp@subsup{p}{}{\prime}=p-monom-mult (lookup (rep-list p) t / punit.lc (rep-list (bs
! i)))
\[
(t-\text { punit.lt }(\text { rep-list }(b s!i)))(b s!i) \text { by }\left(\text { simp add: } p^{\prime}-d e f\right)
\]
```

from Some have punit.lt (rep-list $(b s!i))$ adds $t$ by (rule find-sig-reducer-SomeD)
hence eq: $t$ - punit.lt (rep-list $(b s!i))+$ punit.lt $($ rep-list $(b s!i))=t$ by (rule adds-minus)
from $t$-in Some have $*$ : sig-red-single $\left(\prec_{t}\right)(\preceq) p p^{\prime}(b s!i)(t-p u n i t . l t$ (rep-list $(b s!i))$ )
unfolding $p^{\prime}$ by (rule find-sig-reducer-SomeD-red-single)
hence $* *$ : punit.red-single (rep-list p) (rep-list p') (rep-list (bs ! i)) (tpunit.lt (rep-list (bs!i)))
by (rule sig-red-single-red-single)
from $*$ have ?A by (rule sig-red-single-regular-lt)
moreover from punit.red-single-higher[OF **] have ?B by (simp add: eq)
moreover have ? $C$
proof (intro allI impI)
from punit.red-single-lookup[OF **] show lookup (rep-list p') $t=0$ by (simp add: eq)
qed
ultimately show ?thesis by (intro conjI)
qed
hence $l t-p^{\prime}:$ lt $p^{\prime}=l t p$ and higher- $p^{\prime}:$ punit.higher (rep-list $\left.p^{\prime}\right) t=$ punit.higher (rep-list p) $t$
and lookup-p': ^i. find-sig-reducer (map spp-of bs) (lt p) t $0=$ Some $i \Longrightarrow$ lookup (rep-list $\left.p^{\prime}\right) t=0$
by blast+
show ?case
proof (simp add: sig-trd-aux.psimps[OF 1(1)] Let-def $p^{\prime}$-def[symmetric] $p^{\prime \prime}$-def[symmetric], intro conjI impI)
assume $p^{\prime \prime}=0$
hence $p^{\prime}$-decomp: punit.higher (rep-list p) $t+$ monomial (lookup (rep-list p')
t) $t=r e p-l i s t p^{\prime}$
using punit.higher-lower-decomp $\left[\right.$ of rep-list $\left.p^{\prime} t\right]$ by (simp add: $p^{\prime \prime}$-def higher-p')
show rep-list $p^{\prime}=$ sig-trd-spp-aux (map spp-of bs) (lt $\left.p\right)(? p$, punit.higher $($ rep-list $p) t$ )
proof (cases find-sig-reducer (map spp-of bs) (lt p) t 0)
case None
hence $p^{\prime}: p^{\prime}=p$ by (simp add: $p^{\prime}$-def)
from $\left\langle p^{\prime \prime}=0\right\rangle$ have eq2: punit.tail $? p=0$ by (simp add: tail-p $p^{\prime \prime}$-def $p^{\prime}$ )
from $p^{\prime}$-decomp show ?thesis by (simp add: $p^{\prime}$ eq1 lt-p lc-p None eq2 sig-trd-spp-aux-simps)

## next

case (Some i)
hence $p^{\prime}: p^{\prime}=p$ - monom-mult (lookup (rep-list p) $t /$ punit.lc (rep-list (bs!i)))
$(t-$ punit.lt $($ rep-list $(b s!i)))(b s!i)$ by (simp add: $p^{\prime}$-def)
from $\left\langle p^{\prime \prime}=0\right\rangle$ have eq2: punit.lower (rep-list $p-$ punit.higher (rep-list $p$ )
$t-$
punit.monom-mult (lookup (rep-list p) t/punit.lc (rep-list (bs!i))) ( $t$ - punit.lt (rep-list $(b s!i)))($ rep-list $(b s!i)))$ $t=0$
by (simp add: $p^{\prime \prime}$-def $p^{\prime}$ rep-list-minus rep-list-monom-mult punit.lower-minus punit.lower-higher-zeroI)
from Some have lookup (rep-list $p^{\prime}$ ) $t=0$ by (rule lookup-p ${ }^{\prime}$ )
with $p^{\prime}$-decomp have eq3: rep-list $p^{\prime}=$ punit.higher (rep-list p) $t$ by simp
show ?thesis by (simp add: sig-trd-spp-body-alt-Some(1) eq1 eq2 lt-p lc-p Some del: sig-trd-spp-body.simps,
simp add: sig-trd-spp-aux-simps eq3)
qed
next
assume $p^{\prime \prime} \neq 0$
hence punit.lt $p^{\prime \prime} \prec t$ unfolding $p^{\prime \prime}$-def by (rule punit.lt-lower-less)
have higher-p'-2: punit.higher (rep-list $\left.p^{\prime}\right)\left(\right.$ punit.lt $\left.p^{\prime \prime}\right)=$
punit.higher $($ rep-list $p) t+$ monomial (lookup $\left(\right.$ rep-list $\left.\left.p^{\prime}\right) t\right) t$
proof (simp add: higher-p' ${ }^{\prime}$ symmetric], rule poly-mapping-eqI)
fix $s$
show lookup (punit.higher (rep-list p') (punit.lt $\left.\left.p^{\prime \prime}\right)\right) s=$ lookup (punit.higher (rep-list $\left.p^{\prime}\right) t+$ monomial (lookup (rep-list $\left.p^{\prime}\right) t$ )
t) $s$
proof (rule ordered-powerprod-lin.linorder-cases)
assume $t \prec s$
moreover from <punit.lt $\left.p^{\prime \prime} \prec t\right\rangle$ this have punit.lt $p^{\prime \prime} \prec s$
by (rule ordered-powerprod-lin.less-trans)
ultimately show ?thesis by (simp add: lookup-add punit.lookup-higher-when lookup-single)
next
assume $t=s$
with $\left\langle\right.$ punit.lt $\left.p^{\prime \prime} \prec t\right\rangle$ show ?thesis by (simp add: lookup-add punit.lookup-higher-when) next

```
        assume }s\prec
        show ?thesis
        proof (cases punit.lt p'l}\precs
        case True
        hence lookup (punit.higher (rep-list p') (punit.lt p'')) s= lookup (rep-list
p) s
            by (simp add: punit.lookup-higher-when)
                        also from }\langles\prect\rangle\mathrm{ have ... = lookup p' s by (simp add: p''-def
punit.lookup-lower-when)
            also from True have ... = 0 using punit.lt-le-iff by auto
            finally show ?thesis using }\langles\prect
                    by (simp add: lookup-add lookup-single punit.lookup-higher-when)
            next
                case False
            with «s \prect` show ?thesis by (simp add: lookup-add punit.lookup-higher-when
lookup-single)
            qed
            qed
        qed
        have rep-list (sig-trd-aux bs (punit.lt p'\prime, p'))=
            sig-trd-spp-aux (map spp-of bs) (lt (snd (punit.lt p'\prime, p')))
                (rep-list (snd (punit.lt p', p
                punit.higher (rep-list (snd (punit.lt p'\prime, p'))) (fst (punit.lt p'\prime},\mp@subsup{p}{}{\prime}))
                    punit.higher (rep-list (snd (punit.lt p', p
            using p}\mp@subsup{p}{}{\prime}-\operatorname{def}\mp@subsup{p}{}{\prime\prime}-\operatorname{def}\langle\mp@subsup{p}{}{\prime\prime}\not=0
    proof (rule 1(2))
            from }\langle\mp@subsup{p}{}{\prime\prime}\not=0\rangle\mathrm{ have punit.lt p" }\mp@subsup{p}{}{\prime\prime}\mathrm{ keys p'l by (rule punit.lt-in-keys)
            also have ... \subseteq keys (rep-list p') by (auto simp: p"'def punit.keys-lower)
            finally show fst (punit.lt p'\prime, p') \in keys (rep-list (snd (punit.lt p'\prime, p}))\mathrm{ ) by
simp
            qed
            also have ... = sig-trd-spp-aux (map spp-of bs) (lt p)
                    (rep-list p' - punit.higher (rep-list p') (punit.lt p'),
                    punit.higher (rep-list p') (punit.lt p'\prime))
            by (simp only: lt-p' fst-conv snd-conv)
                            also have ... = sig-trd-spp-aux (map spp-of bs) (lt p) (?p, punit.higher (rep-list
p) t)
    proof (cases find-sig-reducer (map spp-of bs) (lt p) t 0)
            case None
            hence }\mp@subsup{p}{}{\prime}:\mp@subsup{p}{}{\prime}=p\mathrm{ by (simp add: p}\mp@subsup{p}{}{\prime}-def
            have rep-list p - (punit.higher (rep-list p) t + monomial (lookup (rep-list
                p) t) t)=
                punit.lower (rep-list p) t
            using punit.higher-lower-decomp[of rep-list p t] by (simp add: diff-eq-eq
ac-simps
            with higher-p'-2 show ?thesis by (simp add: eq1 lt-p lc-p tail-p p' None)
            next
            case (Some i)
            hence p': rep-list p - punit.monom-mult (lookup (rep-list p) t / punit.lc
```

```
(rep-list (bs!i)))
                    (t - punit.lt (rep-list (bs!i))) (rep-list (bs!i)) = rep-list p}\mp@subsup{}{}{\prime
    by (simp add: p'-def rep-list-minus rep-list-monom-mult)
    from Some have lookup (rep-list p') t=0 by (rule lookup-p')
    with higher-p'-2 show ?thesis
    by (simp add: sig-trd-spp-body-alt-Some(2) eq1 lt-p lc-p tail-p Some
        diff-right-commute[of rep-list p punit.higher (rep-list p)t] p' del:
sig-trd-spp-body.simps)
    qed
    finally show rep-list (sig-trd-aux bs (punit.lt p'\prime, p')) =
                sig-trd-spp-aux (map spp-of bs) (lt p) (?p, punit.higher (rep-list
p) t).
    qed
    qed
qed
lemma sig-trd-alt-spp: spp-of (sig-trd bs p) = sig-trd-spp (map spp-of bs) (spp-of
p)
    unfolding sig-trd-def
proof (split if-split, intro conjI impI)
    assume rep-list p=0
    thus spp-of p = sig-trd-spp (map spp-of bs) (spp-of p) by (simp add: spp-of-def
sig-trd-spp-aux-simps)
next
    let ?args = (punit.lt (rep-list p), p)
    assume rep-list p\not=0
    hence a: fst ?args \in keys (rep-list (snd ?args)) by (simp add: punit.lt-in-keys)
    hence (sig-red (< ) (\preceq) (set bs))** (snd ?args) (sig-trd-aux bs ?args)
        by (rule sig-trd-aux-red-rtrancl)
    hence eq1:lt (sig-trd-aux bs ?args) =lt (snd ?args) by (rule sig-red-regular-rtrancl-lt)
    have eq2: punit.higher (rep-list p) (punit.lt (rep-list p)) = 0
        by (auto simp: punit.higher-eq-zero-iff punit.lt-max simp flip: not-in-keys-iff-lookup-eq-zero
                dest: punit.lt-max-keys)
    show spp-of (sig-trd-aux bs (punit.lt (rep-list p), p)) = sig-trd-spp (map spp-of
bs) (spp-of p)
    by (simp add: spp-of-def eq1 eq2 sig-trd-aux-alt-spp[OF a])
qed
lemma is-regular-spair-alt-spp: is-regular-spair p q \longleftrightarrow is-regular-spair-spp (spp-of
p) (spp-of q)
    by (auto simp: is-regular-spair-spp-def fst-spp-of snd-spp-of intro: is-regular-spairI
        dest: is-regular-spairD1 is-regular-spairD2 is-regular-spairD3)
lemma sig-of-spair-alt-spp: sig-of-pair p = sig-of-pair-spp (app-pair spp-of p)
proof (rule sum-prodE)
    fix ab
    assume p:p=\operatorname{Inl}(a,b)
    show ?thesis by (simp add: p spair-sigs-def spair-sigs-spp-def spp-of-def)
qed simp
```

lemma pair-ord-alt-spp: pair-ord $x y \longleftrightarrow$ pair-ord-spp (app-pair spp-of $x$ ) (app-pair spp-of $y$ )
by (simp add: pair-ord-spp-def pair-ord-def sig-of-spair-alt-spp)
lemma new-spairs-alt-spp:
map (app-pair spp-of) (new-spairs bs $p$ ) $=$ new-spairs-spp (map spp-of bs) (spp-of
p)
proof (induct bs)
case Nil
show? case by simp
next
case (Cons b bs)
have map (app-pair spp-of) (insort-wrt pair-ord $(\operatorname{Inl}(p, b))($ new-spairs bs $p))$
$=$
insort-wrt pair-ord-spp (app-pair spp-of $(\operatorname{Inl}(p, b)))($ map (app-pair spp-of)
(new-spairs bs p))
by (rule map-insort-wrt, rule pair-ord-alt-spp[symmetric])
thus ?case by (simp add: is-regular-spair-alt-spp Cons)
qed
lemma add-spairs-alt-spp:
assumes $\bigwedge x . x \in$ set $b s \Longrightarrow$ Inl (spp-of $p$, spp-of $x) \notin$ app-pair spp-of'set ps
shows map (app-pair spp-of) (add-spairs ps bs $p$ ) =
add-spairs-spp (map (app-pair spp-of) ps) (map spp-of bs) (spp-of p)
proof -
have map (app-pair spp-of) (merge-wrt pair-ord (new-spairs bs p) ps) = merge-wrt pair-ord-spp (map (app-pair spp-of) (new-spairs bs p)) (map
(app-pair spp-of) ps)
proof (rule map-merge-wrt, rule ccontr)
assume app-pair spp-of 'set (new-spairs bs $p$ ) $\cap$ app-pair spp-of' set ps $\neq\{ \}$
then obtain $q^{\prime}$ where $q^{\prime} \in$ app-pair spp-of ' set (new-spairs bs $p$ )
and $q^{\prime}$-in: $q^{\prime} \in$ app-pair spp-of ' set ps by blast
from this(1) obtain $q$ where $q \in$ set (new-spairs bs $p$ ) and $q^{\prime}: q^{\prime}=a p p-p a i r$ spp-of $q$..
from this(1) obtain $x$ where $x$-in: $x \in$ set bs and $q: q=\operatorname{Inl}(p, x)$
by (rule in-new-spairsE)
have $q^{\prime}: q^{\prime}=\operatorname{Inl}(s p p-o f p, s p p-o f x)$ by (simp add: $\left.q q^{\prime}\right)$
have $q^{\prime} \notin$ app-pair spp-of'set ps unfolding $q^{\prime}$ using $x$-in by (rule assms)
thus False using $q^{\prime}$-in ..
qed (simp only: pair-ord-alt-spp)
thus ?thesis by (simp add: add-spairs-def add-spairs-spp-def new-spairs-alt-spp) qed
lemma rb-aux-invD-app-args:
assumes rb-aux-inv (fst (app-args vec-of ((bs,ss, ps), z)))
shows list-all spp-inv bs and list-all spp-inv-pair ps
proof -
from assms(1) have inv: rb-aux-inv (map vec-of bs, ss, map (app-pair vec-of)

```
ps) by simp
    hence rb-aux-inv1 (map vec-of bs) by (rule rb-aux-inv-D1)
    hence 0 # rep-list'set (map vec-of bs) by (rule rb-aux-inv1-D2)
    hence 0 &vec-of'set bs using rep-list-zero by fastforce
    hence 1:b\in set bs \Longrightarrowspp-inv b for b by (auto simp: spp-inv-alt)
    thus list-all spp-inv bs by (simp add: list-all-def)
    have 2: }x\in\mathrm{ set bs if vec-of }x\in\mathrm{ set (map vec-of bs) for x
    proof -
    from that have vec-of x\invec-of'set bs by simp
    then obtain y where y\inset bs and eq: vec-of }x=vec-of y ..
    from this(1) have spp-inv y by (rule 1)
    moreover have vec-of y=vec-of x by (simp only: eq)
    ultimately have }y=x\mathrm{ by (rule vec-of-inj)
    with }<y\in\mathrm{ set bs` show ?thesis by simp
qed
show list-all spp-inv-pair ps unfolding list-all-def
proof (rule ballI)
    fix p
    assume p}\in\mathrm{ set ps
    show spp-inv-pair p
    proof (rule sum-prodE)
        fix ab
        assume p:p=\operatorname{Inl}(a,b)
        from }\langlep\in\mathrm{ set ps> have Inl (a,b) E set ps by (simp only: p)
        hence app-pair vec-of (Inl (a,b))\in app-pair vec-of'set ps by (rule imageI)
        hence Inl (vec-of a, vec-of b) \in set (map (app-pair vec-of) ps) by simp
        with inv have vec-of a\inset (map vec-of bs) and vec-of b \in set (map vec-of
bs)
            by (rule rb-aux-inv-D3)+
            have spp-inv a by (rule 1, rule 2, fact)
            moreover have spp-inv b by (rule 1, rule 2, fact)
            ultimately show ?thesis by (simp add: p)
        qed simp
    qed
qed
lemma app-args-spp-of-vec-of:
    assumes rb-aux-inv (fst (app-args vec-of args))
    shows app-args spp-of (app-args vec-of args) = args
proof -
    obtain bs ss ps z where args: args = ((bs, ss, ps),z) using prod.exhaust by
metis
    from assms have list-all spp-inv bs and *: list-all spp-inv-pair ps unfolding
args
    by (rule rb-aux-invD-app-args)+
    from this(1) have map (spp-of o vec-of) bs =bs by (rule map-spp-of-vec-of)
    moreover from * have map (app-pair spp-of ○ app-pair vec-of) ps = ps
```

```
    by (rule map-app-pair-spp-of-vec-of)
    ultimately show ?thesis by (simp add: args)
qed
lemma poly-of-pair-alt-spp:
    assumes distinct fs and rb-aux-inv (bs, ss, p# ps)
    shows spp-of (poly-of-pair p) = spp-of-pair (app-pair spp-of p)
proof -
    show ?thesis
    proof (rule sum-prodE)
        fix ab
        assume p:p=\operatorname{Inl}(a,b)
    hence Inl (a,b) \in set (p#ps) by simp
    with assms(2) have is-regular-spair a b by (rule rb-aux-inv-D3)
    thus ?thesis by (simp add: p spair-alt-spp)
    next
    fix }
    assume p: p= Inr j
    hence Inr j 榇 (p#ps) by simp
    with assms(2) have j< length fs by (rule rb-aux-inv-D4)
        thus ?thesis by (simp add: p spp-of-def lt-monomial rep-list-monomial[OF
assms(1)] term-simps)
    qed
qed
lemma rb-aux-alt-spp:
    assumes rb-aux-inv (fst args)
    shows app-args spp-of (rb-aux args) = rb-spp-aux (app-args spp-of args)
proof -
    from assms have rb-aux-dom args by (rule rb-aux-domI)
    thus ?thesis using assms
    proof (induct args rule: rb-aux.pinduct)
        case (1 bs ss z)
        show ?case by (simp add: rb-aux.psimps(1)[OF 1(1)] rb-spp-aux-Nil)
    next
        case (2 bs ss p ps z)
        let ?q = sig-trd bs (poly-of-pair p)
    from 2(5) have *: rb-aux-inv (bs, ss, p # ps) by (simp only: fst-conv)
    hence rb-aux-inv1 bs by (rule rb-aux-inv-D1)
    hence 0 # rep-list'set bs by (rule rb-aux-inv1-D2)
    hence 0 & set bs by (force simp: rep-list-zero)
    hence eq1: sig-crit-spp (map spp-of bs) ss' (app-pair spp-of p)\longleftrightarrow sig-crit bs
ss'p for ss'
            by (simp add: sig-crit-alt-spp)
    from fs-distinct * have eq2: sig-trd-spp (map spp-of bs) (spp-of-pair (app-pair
spp-of p))}=spp-of ?
            by (simp only: sig-trd-alt-spp poly-of-pair-alt-spp)
```


## show ?case

proof (simp add: rb-aux.psimps(2)[OF 2(1)] Let-def, intro conjI impI) note refl
moreover assume $a$ : sig-crit bs (new-syz-sigs ss bs p) $p$
moreover from * this have rb-aux-inv (fst ((bs, new-syz-sigs ss bs p, ps),
z))
unfolding fst-conv by (rule rb-aux-inv-preserved-1)
ultimately have app-args spp-of (rb-aux ((bs, new-syz-sigs ss bs p, ps), z)) $=$
rb-spp-aux (app-args spp-of ((bs, new-syz-sigs ss bs $p, p s), z))$
by (rule 2(2))
also have $\ldots=$ rb-spp-aux ( $($ map spp-of bs, ss, app-pair spp-of $p$ \# map (app-pair spp-of) ps), z)
by (simp add: rb-spp-aux-Cons eq1 a new-syz-sigs-alt-spp[symmetric])
finally show app-args spp-of (rb-aux ((bs, new-syz-sigs ss bs $p, p s), z))=$ rb-spp-aux ((map spp-of bs, ss, app-pair spp-of p \# map (app-pair spp-of) $p s), z)$.
thus app-args spp-of $(r b-a u x((b s$, new-syz-sigs ss bs $p, p s), z))=$ rb-spp-aux ((map spp-of bs, ss, app-pair spp-of p \# map (app-pair spp-of) $p s), z)$.
next
assume $a: \neg$ sig-crit bs (new-syz-sigs ss bs $p$ ) $p$ and $b:$ rep-list $? q=0$
from $* b$ have rb-aux-inv $(f s t((b s, l t ? q \# n e w-s y z-s i g s ~ s s ~ b s ~ p, p s), ~ S u c ~ z))$ unfolding fst-conv by (rule rb-aux-inv-preserved-2)
with refl a refl b have app-args spp-of (rb-aux ( $(b s$, lt ? $q$ \# new-syz-sigs ss bs $p, p s)$, Suc $z))=$
rb-spp-aux (app-args spp-of ((bs,lt ?q \# new-syz-sigs ss
bs $p, p s)$, Suc z))
by (rule 2(3))
also have $\ldots=$ rb-spp-aux ( $($ map spp-of bs, ss, app-pair spp-of $p \#$ map (app-pair spp-of) ps), z)
by (simp add: rb-spp-aux-Cons eq1 a Let-def eq2 snd-spp-of b fst-spp-of new-syz-sigs-alt-spp[symmetric])
finally show app-args spp-of (rb-aux ( $b s, l t ? q$ \# new-syz-sigs ss bs $p, p s$ ), Suc z)) $=$
rb-spp-aux ((map spp-of bs, ss, app-pair spp-of $p \#$ map (app-pair spp-of) $p s), z)$.
next
assume $a: \neg$ sig-crit bs (new-syz-sigs ss bs p) $p$ and $b:$ rep-list $? q \neq 0$
have Inl (spp-of ? $q$, spp-of $x$ ) $\notin$ app-pair spp-of'set ps for $x$
proof
assume Inl (spp-of ? $q$, spp-of $x) \in$ app-pair spp-of'set ps
then obtain $y$ where $y \in$ set ps and eq0: Inl $($ spp-of ? $q$, spp-of $x)=$ app-pair spp-of $y$..
obtain $a b$ where $y: y=\operatorname{Inl}(a, b)$ and $s p p-o f ? q=\operatorname{spp}$-of $a$
proof (rule sum-prodE)
fix $a b$
assume $y=\operatorname{Inl}(a, b)$

```
            moreover from eq0 have spp-of ?q = spp-of a by (simp add: <y = Inl
(a,b)>)
            ultimately show ?thesis ..
            next
                fix }
                    assume y = Inr j
                    with eq0 show ?thesis by simp
            qed
            from this(2) have lt ?q = lt a by (simp add: spp-of-def)
            from }\langley\in\mathrm{ set ps` have }y\in\operatorname{set}(p#ps) by sim
            with * have a fet bs unfolding y by (rule rb-aux-inv-D3(1))
            hence lt ?q\inlt 'set bs unfolding <lt ?q=lt a> by (rule imageI)
                            moreover from *ab}\mathrm{ have lt ?q q &lt'set bs by (rule rb-aux-inv-preserved-3)
            ultimately show False by simp
    qed
    hence eq3: add-spairs-spp (map (app-pair spp-of) ps) (map spp-of bs) (spp-of
?q) =
                    map (app-pair spp-of) (add-spairs ps bs ?q) by (simp add:
add-spairs-alt-spp)
```

from * $a$ b have rb-aux-inv (fst ((?q \# bs, new-syz-sigs ss bs p, add-spairs ps bs ?q), z))
unfolding fst-conv by (rule rb-aux-inv-preserved-3)
with refl a refl $b$
have app-args spp-of (rb-aux ((?q \# bs, new-syz-sigs ss bs $p$, add-spairs ps bs ? $q), z)$ ) $=$
rb-spp-aux (app-args spp-of ((?q \# bs, new-syz-sigs ss bs p, add-spairs ps bs ?q), z))
by (rule 2(4))
also have $\ldots=$ rb-spp-aux ( $($ map spp-of bs, ss, app-pair spp-of $p \#$ map (app-pair spp-of) ps), z)
by (simp add: rb-spp-aux-Cons eq1 a Let-def eq2 fst-spp-of snd-spp-of beq3 new-syz-sigs-alt-spp [symmetric])
finally show app-args spp-of (rb-aux ((?q \# bs, new-syz-sigs ss bs p, add-spairs $p s(b s q), z))=$
rb-spp-aux ((map spp-of bs, ss, app-pair spp-of $p \#$ map (app-pair spp-of) $p s), z)$.
qed
qed
qed
corollary rb-spp-aux-alt:
rb-aux-inv (fst (app-args vec-of args)) $\Longrightarrow$
rb-spp-aux args $=$ app-args spp-of (rb-aux (app-args vec-of args))
by (simp only: rb-aux-alt-spp app-args-spp-of-vec-of)
corollary rb-spp-aux:
hom-grading dgrad $\Longrightarrow$ punit.is-Groebner-basis (set (map snd (fst (fst (rb-spp-aux (([], Koszul-syz-sigs
fs, map Inr $[0 . .<$ length $f s]), z)))))$ )
(is - $\Longrightarrow$ ?thesis1)
ideal (set (map snd (fst (fst (rb-spp-aux (([], Koszul-syz-sigs fs, map Inr [0..<length $\left.\left.\left.\left.\left.\left.\left.f_{s}\right]\right), z\right)\right)\right)\right)\right)$ ) $=\operatorname{ideal}\left(\right.$ set $\left.f_{s}\right)$
(is ?thesis2)
set (map snd (fst (fst (rb-spp-aux (([], Koszul-syz-sigs fs, map Inr $[0 . .<$ length $\left.\left.\left.\left.\left.\left.\left.f_{s}\right]\right), z\right)\right)\right)\right)\right) \subseteq$ punit-dgrad-max-set dgrad
(is?thesis3)
$0 \notin$ set (map snd (fst (fst (rb-spp-aux (([], Koszul-syz-sigs fs, map Inr [0..<length $f s]$ ), z) )) )
(is ?thesis4)
hom-grading dgrad $\Longrightarrow$ is-pot-ord $\Longrightarrow$ is-regular-sequence $f s \Longrightarrow$
snd (rb-spp-aux (([], Koszul-syz-sigs fs, map Inr $[0 . .<$ length $f s]), z))=z$
(is $-\Longrightarrow-\Longrightarrow-\Longrightarrow$ ?thesis5)
rword-strict $=r w$-rat-strict $\Longrightarrow p \in \operatorname{set}(f s t(f s t$ (rb-spp-aux (([], Koszul-syz-sigs fs, map Inr $[0 . .<$ length $f s]), z)))) \Longrightarrow$
$q \in \operatorname{set}(f s t(f s t(r b-s p p-a u x)(([]$, Koszul-syz-sigs fs, map Inr $[0 . .<$ length $f s])$, $z))) \Longrightarrow p \neq q \Longrightarrow$
punit.lt (snd $p$ ) adds punit.lt (snd $q) \Longrightarrow$ punit.lt (snd $p) \oplus$ fst $q \prec_{t}$ punit.lt $($ snd $q) \oplus$ fst $p$
proof -
let ?args $=(([]$, Koszul-syz-sigs fs, map Inr $[0 . .<$ length fs $]), z)$
have eq0: app-pair vec-of $\circ$ Inr $=\operatorname{Inr}$ by (intro ext, simp)
have eq1: fst (fst (app-args spp-of a)) = map spp-of $(f s t(f s t a))$ for $a::(-\times(' t$ list) $\times-) \times-$
proof -
obtain $b s$ ss $p s z$ where $a=((b s, s s, p s), z)$ using prod.exhaust by metis
thus ?thesis by simp
qed
have eq2: snd $\circ$ spp-of $=$ rep-list by (intro ext, simp add: snd-spp-of)
have rb-aux-inv (fst (app-args vec-of ?args)) by (simp add: eq0 rb-aux-inv-init)
hence eq3: rb-spp-aux ? args $=$ app-args spp-of $(r b-a u x ~(a p p-a r g s ~ v e c-o f ~ ? a r g s)) ~$
by (rule rb-spp-aux-alt)

## \{

assume hom-grading dgrad
with rb-aux-is-Groebner-basis show ?thesis1 by (simp add: eq0 eq1 eq2 eq3 del: set-map)
\}
from ideal-rb-aux show ?thesis2 by (simp add: eq0 eq1 eq2 eq3 del: set-map)
from dgrad-max-set-closed-rb-aux show ?thesis3 by (simp add: eq0 eq1 eq2 eq3 del: set-map)
from rb-aux-nonzero show ?thesis4 by (simp add: eq0 eq1 eq2 eq3 del: set-map)
\{
assume is-pot-ord and hom-grading dgrad and is-regular-sequence fs

```
    hence snd (rb-aux ?args) =z by (rule rb-aux-no-zero-red)
    thus?thesis5 by (simp add: snd-app-args eq0 eq3)
}
{
    from rb-aux-nonzero have 0 & rep-list ' set (fst (fst (rb-aux ?args)))
            (is 0& rep-list'?G) by simp
    assume rword-strict =rw-rat-strict
    hence is-min-sig-GB dgrad ?G by (rule rb-aux-is-min-sig-GB)
    hence rl: \g.g\in? G \Longrightarrow\negis-sig-red ( }\mp@subsup{\Omega}{t}{\prime})(=)(?G-{g})g\mathrm{ by (simp add:
is-min-sig-GB-def)
    assume p\in set (fst (fst (rb-spp-aux ?args)))
    also have ... = spp-of '?G by (simp add: eq0 eq1 eq3)
    finally obtain }\mp@subsup{p}{}{\prime}\mathrm{ where }\mp@subsup{p}{}{\prime}\in?G\mathrm{ and }p:p=spp-of \mp@subsup{p}{}{\prime}.
    assume q\in set (fst (fst (rb-spp-aux ?args)))
    also have ... = spp-of '?G by (simp add: eq0 eq1 eq3)
    finally obtain q' where q}\mp@subsup{q}{}{\prime}\in?G\mathrm{ and }q:q=spp-of \mp@subsup{q}{}{\prime}.
    from this(1) have 1: ᄀis-sig-red (\mp@subsup{\preceq}{t}{})(=)(?G-{q'}) q' by (rule rl)
    assume p\not=q and punit.lt (snd p) adds punit.lt (snd q)
    hence }\mp@subsup{p}{}{\prime}\not=\mp@subsup{q}{}{\prime}\mathrm{ and adds: punit.lt (rep-list p}\mp@subsup{p}{}{\prime})\mathrm{ adds punit.lt (rep-list q}\mp@subsup{q}{}{\prime}
        by (auto simp: p q snd-spp-of)
    show punit.lt (snd p)\oplus fst q < t punit.lt (snd q) \oplus fst p
    proof (rule ccontr)
        assume }\neg\mathrm{ punit.lt (snd p) }\mp@subsup{f}{\mathrm{ st }q\mp@subsup{\prec}{t}{}\mathrm{ punit.lt (snd q) }\oplus\mathrm{ fst p}}{
        hence le: punit.lt (rep-list q')}\opluslt p\mp@subsup{p}{}{\prime}\mp@subsup{\preceq}{t}{}\mathrm{ punit.lt (rep-list p}\mp@subsup{p}{}{\prime})\opluslt \mp@subsup{q}{}{\prime
            by (simp add: p q spp-of-def)
    from }\langle\mp@subsup{p}{}{\prime}\not=\mp@subsup{q}{}{\prime}\rangle\langle\mp@subsup{p}{}{\prime}\in?G`\mathrm{ have }\mp@subsup{p}{}{\prime}\in??G-{\mp@subsup{q}{}{\prime}}\mathrm{ by simp
    moreover from \langlep'\in?G\rangle\langle0 & rep-list '?G` have rep-list p' 
    moreover from }\langle\mp@subsup{q}{}{\prime}\in?G\rangle\langle0\not\in\mathrm{ rep-list '?G` have rep-list q' }\mp@subsup{q}{}{\prime}=0\mathrm{ by fastforce
    moreover note adds
    moreover have ord-term-lin.is-le-rel (\mp@subsup{\preceq}{t}{})\mathrm{ by simp}
            ultimately have is-sig-red ( }\mp@subsup{\Omega}{t}{})(=)(?G-{q'}) q' using le by (rule
is-sig-red-top-addsI)
            with 1 show False ..
    qed
    }
qed
end
end
end
end
end
definition gb-sig-z ::
```

```
    \(\left(\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\right.\) bool \() \Rightarrow\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a\right.\right.\)
\(\Rightarrow_{0}{ }^{\prime} b::\) field \()\) ) list \(\times\) nat)
    where gb-sig-z rword-strict fs0 \(=\)
                        (let fs \(=\) rev (remdups (rev (removeAll 0 fs0)));
                        res \(=\) rb-spp-aux fs rword-strict (([], Koszul-syz-sigs fs, map Inr
\([0 . .<\) length \(f s]), 0)\) in
    (fst (fst res), snd res))
```

The second return value of $g b-s i g-z$ is the total number of zero-reductions.

```
definition \(g b\)-sig : : \(\left(\left(^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\left({ }^{\prime} t \times\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\right) \Rightarrow\right.\) bool \() \Rightarrow\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right)\)
```

list $\Rightarrow\left(^{\prime} a \Rightarrow_{0}{ }^{\prime} b::\right.$ field $)$ list
where $g b$-sig rword-strict $f s 0=\operatorname{map}$ snd $(f s t(g b-s i g-z$ rword-strict $f s 0))$

## theorem

assumes $\bigwedge f$ s. is-strict-rewrite-ord fs rword-strict
shows gb-sig-isGB: punit.is-Groebner-basis (set (gb-sig rword-strict fs)) (is ?thesis1)
and gb-sig-ideal: ideal (set (gb-sig rword-strict fs)) ideal (set fs) (is ?thesis2)
and dgrad-p-set-closed-gb-sig:
dickson-grading $d \Longrightarrow$ set $f s \subseteq$ punit.dgrad-p-set $d m \Longrightarrow$ set (gb-sig
rword-strict fs) $\subseteq$ punit.dgrad-p-set d $m$
(is $-\Longrightarrow-\Longrightarrow$ ?thesis3)
and gb-sig-nonzero: $0 \notin \operatorname{set}$ (gb-sig rword-strict fs) (is ?thesis4)
and gb-sig-no-zero-red: is-pot-ord $\Longrightarrow$ is-regular-sequence $f s \Longrightarrow$ snd (gb-sig-z
rword-strict $f_{s}$ ) $=0$
proof -
from ex-hgrad obtain $d 0:^{\prime} a \Rightarrow$ nat where dickson-grading d0 $\wedge$ hom-grading
d0 ..
hence $d g$ : dickson-grading $d 0$ and $h g$ : hom-grading $d 0$ by simp-all
define $f_{s 1} 1$ where $f_{s 1}=$ rev $($ remdups $($ rev $($ removeAll $0 f s)))$
note assms $d g$
moreover have distinct fs1 and $0 \notin$ set fs1 by (simp-all add: fs1-def)
ultimately have ideal (set (gb-sig rword-strict fs)) =ideal (set fs1) and ?thesis4
unfolding $g b$-sig-def gb-sig-z-def fst-conv fs1-def Let-def by (rule rb-spp-aux)+
thus ?thesis2 and ?thesis 4 by (simp-all add: fs1-def ideal.span-Diff-zero)
from assms dg 〈distinct $\left.f_{s 1} 1\right\rangle\langle 0 \notin$ set fs1〉hg show ?thesis1
unfolding gb-sig-def gb-sig-z-def fst-conv fs1-def Let-def by (rule rb-spp-aux)
\{
assume dg: dickson-grading $d$ and $*:$ set $f s \subseteq$ punit.dgrad-p-set d m
show ?thesis3
proof (cases set $f s \subseteq\{0\}$ )
case True
hence removeAll $0 f_{s}=[]$
by (metis (no-types, lifting) Diff-iff ex-in-conv set-empty2 set-removeAll
subset-singleton-iff)
thus ?thesis by (simp add: gb-sig-def gb-sig-z-def Let-def rb-spp-aux-Nil)
next
case False

```
    have set fs1\subseteq set fs by (fastforce simp: fs1-def)
    hence Keys (set fs1)\subseteqKeys (set fs) by (rule Keys-mono)
    hence d'Keys (set fs1)\subseteqd'Keys (set fs) by (rule image-mono)
    hence insert (d 0) (d'Keys (set fs1))\subseteq insert (d 0) (d'Keys (set fs)) by
(rule insert-mono)
    moreover have insert (d 0) (d'Keys (set fs1)) f={} by simp
    moreover have finite (insert (d 0) (d`'Keys (set fs)))
        by (simp add: finite-Keys)
    ultimately have le: Max (insert (d 0) (d'Keys (set fs1)))\leq
                        Max (insert (d 0) (d'Keys (set fs))) by (rule Max-mono)
        from assms dg have set (gb-sig rword-strict fs) \subseteq punit-dgrad-max-set
(TYPE('b)) fs1 d
            using <distinct fs1><0 # set fs1>
            unfolding gb-sig-def gb-sig-z-def fst-conv fs1-def Let-def by (rule rb-spp-aux)
                also have punit-dgrad-max-set (TYPE('b)) fs1 d \subseteq punit-dgrad-max-set
(TYPE('b)) fs d
            by (rule punit.dgrad-p-set-subset, simp add: dgrad-max-def le)
            also from dg* False have ...\subseteq punit.dgrad-p-set d m
                by (rule punit-dgrad-max-set-subset-dgrad-p-set)
            finally show ?thesis .
    qed
    }
    {
    assume is-regular-sequence fs
    note assms dg <distinct fs1〉〈0 # set fs1〉 hg
    moreover assume is-pot-ord
    moreover from <is-regular-sequence fs` have is-regular-sequence fs1 unfolding
fs1-def
            by (intro is-regular-sequence-remdups is-regular-sequence-removeAll-zero)
    ultimately show snd (gb-sig-z rword-strict fs) =0
    unfolding gb-sig-def gb-sig-z-def snd-conv fs1-def Let-def by (rule rb-spp-aux)
    }
qed
theorem gb-sig-z-is-min-sig-GB:
    assumes p \in set (fst (gb-sig-z rw-rat-strict fs)) and q \in set (fst (gb-sig-z
rw-rat-strict fs))
    and p\not=q and punit.lt (snd p) adds punit.lt (snd q)
    shows punit.lt (snd p)\oplus fst q}\mp@subsup{\prec}{t}{}\mathrm{ punit.lt (snd q) }\oplus\mathrm{ fst p
proof -
    define fs1 where fs1 = rev (remdups (rev (removeAll 0 fs)))
    from ex-hgrad obtain d0::'a m nat where dickson-grading d0 ^ hom-grading
d0 ..
    hence dickson-grading d0 ..
    note rw-rat-strict-is-strict-rewrite-ord this
    moreover have distinct fs1 and 0}#\mathrm{ set fs1 by (simp-all add: fs1-def)
    moreover note refl assms
    ultimately show ?thesis unfolding gb-sig-z-def fst-conv fs1-def Let-def by (rule
```

rb-spp-aux)
qed
Summarizing, these are the four main results proved in this theory:

- ( $\bigwedge f s . i s$-strict-rewrite-ord fs ?rword-strict $) \Longrightarrow$ punit.is-Groebner-basis (set (gb-sig ?rword-strict ?fs)),
- $(\bigwedge f s$. is-strict-rewrite-ord fs ? rword-strict) $\Longrightarrow$ ideal (set (gb-sig ?rword-strict $? f s))=$ ideal (set ?fs),
- $\llbracket \bigwedge f s . i s$-strict-rewrite-ord fs ?rword-strict; is-pot-ord; is-regular-sequence ? $f s \rrbracket \Longrightarrow$ snd (gb-sig-z ?rword-strict ? $f s)=0$, and
- $\llbracket ? p \in \operatorname{set}(f s t(g b-s i g-z r w-r a t-s t r i c t ~ ? f s)) ; ? q \in \operatorname{set}(f s t(g b-s i g-z r w-r a t-s t r i c t$ ?fs) ) ; ? $p \neq$ ? $q$; punit.lt (snd ?p) adds punit.lt (snd ?q) 】 $\Longrightarrow$ punit.lt $\left(\right.$ 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<br>imports Signature-Groebner Groebner-Bases.Benchmarks Groebner-Bases.Code-Target-Rat begin

### 5.1 Setup

lift-definition except-pp :: ('a, 'b) pp $\Rightarrow^{\prime} a$ set $\Rightarrow\left({ }^{\prime} a\right.$, 'b::zero) pp is except.
lemma hom-grading-varnum-pp: hom-grading (varnum-pp::('a::countable, 'b::add-wellorder)
$p p \Rightarrow n a t)$
proof -
define $f$ where $f=(\lambda n t$. (except-pp $t(-\{x$. elem-index $\left.x<n\}))::\left({ }^{\prime} a,{ }^{\prime} b\right) p p\right)$
show ?thesis unfolding hom-grading-def hom-grading-fun-def
proof (intro exI allI conjI impI)
fix $n s t$
show $f n(s+t)=f n s+f n t$ unfolding $f$-def by transfer (rule except-plus)
next
fix $n t$
show varnum-pp $(f n t) \leq n$ unfolding $f$-def by transfer (simp add: varnum-le-iff keys-except)
next
fix $n t$
show varnum-pp $t \leq n \Longrightarrow f n t=t$ unfolding $f$-def

```
        by transfer (auto simp: except-id-iff varnum-le-iff)
    qed
qed
```

instance $p p::$ (countable, add-wellorder) quasi-pm-powerprod
by (standard, intro exI conjI, fact dickson-grading-varnum-pp, fact hom-grading-varnum-pp)

### 5.1.1 Projections of Term Orders to Orders on Power-Products

definition proj-comp :: (('a::nat, 'b::nat) pp $\times$ nat) nat-term-order $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b\right)$ pp $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b\right) p p \Rightarrow$ order
where proj-comp cmp $=(\lambda x y$. nat-term-compare $c m p(x, 0)(y, 0))$
definition proj-ord :: (('a::nat, 'b::nat) pp $\times$ nat) nat-term-order $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b\right)$ 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 ' $a \times n a t$, 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 $x=($ rep-nat-pp $x, 0)$.
lemma comparator-proj-comp: comparator ( proj-comp cmp)
proof -
interpret cmp : comparator nat-term-compare cmp by (rule comparator-nat-term-compare)
show ?thesis unfolding proj-comp-def
proof
fix $x y::\left({ }^{\prime} a,^{\prime} b\right) p p$
show invert-order (nat-term-compare cmp $(x, 0)(y, 0))=$ nat-term-compare
$c m p(y, 0)(x, 0)$
by (simp only: cmp.sym)
next
fix $x y::\left({ }^{\prime} a, ' b\right) p p$
assume nat-term-compare cmp $(x, 0)(y, 0)=E q$
hence $(x, 0)=(y, 0:: n a t)$ by (rule cmp.weak-eq)
thus $x=y$ by simp
next
fix $x y z::\left({ }^{\prime} a,{ }^{\prime} b\right) p p$
assume nat-term-compare cmp $(x, 0)(y, 0)=L t$ and nat-term-compare cmp
$(y, 0)(z, 0)=L t$
thus nat-term-compare cmp $(x, 0)(z, 0)=L t$ by (rule cmp.comp-trans) qed
qed
lemma nat-term-comp-proj-comp: nat-term-comp (proj-comp cmp)
proof -
have 1: fst (rep-nat-term $(u, i))=$ rep-nat-pp $u$ for $u::\left({ }^{\prime} a,{ }^{\prime} b\right) p p$ and $i:: n a t$
by (simp add: rep-nat-term-prod-def)
have 2: snd (rep-nat-term $(u, i))=i$ for $u::(' a, ' b) p p$ and $i:: n a t$
by (simp add: rep-nat-term-prod-def rep-nat-nat-def)

## show ?thesis

proof (rule nat-term-compI)
fix $u v::\left({ }^{\prime} a,{ }^{\prime} b\right) p p$
assume a: fst (rep-nat-term u) $=0$
note nat-term-comp-nat-term-compare
moreover have snd (rep-nat-term $(u, 0:: n a t))=\operatorname{snd}($ rep-nat-term $(v, 0::$ nat $))$
by (simp only: 2)
moreover from $a$ have $f s t$ (rep-nat-term $(u, 0::$ nat $))=0$ by (simp add: 1 rep-nat-term-pp-def)
ultimately have nat-term-compare $\operatorname{cmp}(u, 0)(v, 0) \neq G t$ by (rule nat-term-compD1)
thus proj-comp cmp $u v \neq G t$ by (simp add: proj-comp-def)
next
fix $u v::\left({ }^{\prime} a,{ }^{\prime} b\right) p p$
assume snd (rep-nat-term u) < snd (rep-nat-term v)
thus proj-comp cmp uv=Lt by (simp add: rep-nat-term-pp-def)
next
fix $t u v::\left({ }^{\prime} a,{ }^{\prime} b\right) p p$
assume proj-comp cmp $u v=L t$
hence nat-term-compare cmp $(u, 0)(v, 0)=L t$ by (simp add: proj-comp-def)
with nat-term-comp-nat-term-compare have nat-term-compare cmp (splus ( $t$,
$0)(u, 0))(\operatorname{splus}(t, 0)(v, 0))=L t$ by (rule nat-term-compD3)
thus proj-comp cmp (splus tu) (splus $t v)=L t$ by (simp add: proj-comp-def splus-prod-def pprod.splus-def splus-pp-term)
next
fix $u$ v a $b::\left({ }^{\prime} a,^{\prime} b\right) p p$
assume $u$ : fst (rep-nat-term $u)=f s t($ rep-nat-term $a)$ and $v$ : fst (rep-nat-term
$v)=f_{s t}($ rep-nat-term b)
and a: proj-comp cmp a $b=L t$
note nat-term-comp-nat-term-compare
moreover from $u$ have $f$ st (rep-nat-term $(u, 0:: n a t))=$ fst (rep-nat-term ( $a$, 0::nat))
by (simp add: 1 rep-nat-term-pp-def)
moreover from $v$ have fst (rep-nat-term $(v, 0:: n a t))=$ fst (rep-nat-term $(b$, $0:: n a t)$ )
by (simp add: 1 rep-nat-term-pp-def)
moreover have snd (rep-nat-term ( $u, 0::$ nat $)$ ) $=\operatorname{snd}($ rep-nat-term ( $v, 0::$ nat) $)$
and snd (rep-nat-term ( $a, 0::$ nat) $)=$ snd (rep-nat-term (b, 0::nat)) by
(simp-all only: 2)
moreover from $a$ have nat-term-compare cmp $(a, 0)(b, 0)=L t$ by (simp add: proj-comp-def)
ultimately have nat-term-compare $\operatorname{cmp}(u, 0)(v, 0)=L t$ by (rule nat-term-compD4)
thus proj-comp cmp u $v=L t$ by (simp add: proj-comp-def)
qed
qed
corollary nat-term-compare-proj-ord: nat-term-compare $($ proj-ord cmp $)=$ proj-comp cmp
unfolding proj-ord-def using comparator-proj-comp nat-term-comp-proj-comp

```
    by (rule nat-term-compare-Abs-nat-term-order-id)
lemma proj-ord-LEX [code]: proj-ord LEX = LEX
proof -
    have nat-term-compare (proj-ord LEX) = nat-term-compare LEX
    by (auto simp: nat-term-compare-proj-ord nat-term-compare-LEX proj-comp-def
lex-comp
            lex-comp-aux-def rep-nat-term-prod-def rep-nat-term-pp-def intro!: ext split:
order.split)
    thus ?thesis by (simp only: nat-term-compare-inject)
qed
lemma proj-ord-DRLEX [code]: proj-ord DRLEX = DRLEX
proof -
    have nat-term-compare (proj-ord DRLEX) = nat-term-compare DRLEX
    by (auto simp: nat-term-compare-proj-ord nat-term-compare-DRLEX proj-comp-def
deg-comp pot-comp
lex-comp lex-comp-aux-def rep-nat-term-prod-def rep-nat-term-pp-def intro!:
ext split: order.split)
    thus ?thesis by (simp only: nat-term-compare-inject)
qed
lemma proj-ord-DEG [code]: proj-ord (DEG to) = DEG (proj-ord to)
proof -
    have nat-term-compare (proj-ord (DEG to)) = nat-term-compare (DEG (proj-ord
to))
    by (simp add: nat-term-compare-proj-ord nat-term-compare-DEG proj-comp-def
deg-comp
            rep-nat-term-prod-def rep-nat-term-pp-def)
    thus ?thesis by (simp only: nat-term-compare-inject)
qed
lemma proj-ord-POT [code]: proj-ord (POT to) = proj-ord to
proof -
    have nat-term-compare (proj-ord (POT to)) = nat-term-compare (proj-ord to)
    by (simp add: nat-term-compare-proj-ord nat-term-compare-POT proj-comp-def
pot-comp
        rep-nat-term-prod-def rep-nat-term-pp-def)
    thus ?thesis by (simp only: nat-term-compare-inject)
qed
```


### 5.1.2 Locale Interpretation

```
locale qpm-nat-inf-term \(=g d\)-nat-term \(\lambda x . x \lambda x . x\) to for \(t o::\left(\left({ }^{\prime} a:: n a t\right.\right.\), ' \(\left.\left.b:: n a t\right) p p \times n a t\right)\) 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
proof intro-locales
    show ordered-powerprod-axioms (le-of-nat-term-order (proj-ord to))
    by (unfold-locales, fact le-of-nat-term-order-zero-min, auto dest:le-of-nat-term-order-plus-monotone
simp: ac-simps)
next
    show ordered-term-axioms ( }\lambdax.x)(\lambdax.x) (le-of-nat-term-order (proj-ord to))
(le-of-nat-term-order to)
    proof
        fix vwt
        assume le-of-nat-term-order to v w
        thus le-of-nat-term-order to (local.splus t v) (local.splus t w)
            by (simp add: le-of-nat-term-order nat-term-compare-splus splus-eq-splus)
    next
        fix v}
        assume le-of-nat-term-order (proj-ord to) (pp-of-term v) (pp-of-term w)
            and component-of-term v}\leq\mathrm{ component-of-term w
    hence nat-term-compare to (fst v,0) (fst w,0) \not=Gt and snd v\leq snd w
    by (simp-all add: le-of-nat-term-order nat-term-compare-proj-ord proj-comp-def)
            from comparator-nat-term-compare nat-term-comp-nat-term-compare - . - 
this(1)
            have nat-term-compare to v w\not=Gt
            by (rule nat-term-compD4') (simp-all add: rep-nat-term-prod-def ord-iff[symmetric]
<nd v\leqsnd w`)
    thus le-of-nat-term-order to v w by (simp add:le-of-nat-term-order)
    qed
qed
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 $^{\prime}$ : qpm-nat-inf-term to
rewrites pprod.pp-of-term $=f s t$
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 $=l t-$ punit $($ proj-ord to $)$
and pprod'.aux.punit.lc $=l c-$ 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' ${ }^{\prime}$ aux.Koszul-syz-sigs
and find-sig-reducer-pprod $=$ pprod' $^{\prime}$.aux.find-sig-reducer
and sig-trd-spp-body-pprod $=$ pprod' ${ }^{\prime}$ aux.sig-trd-spp-body
and sig-trd-spp-aux-pprod $=$ pprod'.aux.sig-trd-spp-aux
and sig-trd-spp-pprod $=$ pprod $^{\prime} . a u x . s i g-t r d-s p p$
and spair-sigs-spp-pprod $=$ pprod' $^{\prime}$.aux.spair-sigs-spp
and $i s$-pred-syz-pprod $=$ pprod $^{\prime} . a u x . i s-p r e d-s y z$
and $i s$-rewritable-spp-pprod $=$ pprod $^{\prime} . a u x . i s$-rewritable-spp
and sig-crit-spp-pprod $=$ pprod $^{\prime} \cdot$ aux.sig-crit-spp
and spair-spp-pprod $=$ pprod $^{\prime} \cdot a u x . s p a i r-s p p$
and spp-of-pair-pprod $=$ pprod $^{\prime} . a u x . s p p-o f-p a i r$
and pair-ord-spp-pprod $=$ pprod $^{\prime}$. aux.pair-ord-spp
and sig-of-pair-spp-pprod $=$ pprod' ${ }^{\prime}$.aux.sig-of-pair-spp
and new-spairs-spp-pprod $=$ pprod' ${ }^{\prime}$ aux.new-spairs-spp
and $i s$-regular-spair-spp-pprod $=$ pprod ${ }^{\prime}$.aux.is-regular-spair-spp
and add-spairs-spp-pprod $=$ pprod $^{\prime} \cdot a u x . a d d-$ spairs-spp
and $i s$-pot-ord-pprod $=$ pprod $^{\prime} . i s$-pot-ord
and new-syz-sigs-spp-pprod $=$ pprod $^{\prime} \cdot a u x . n e w-s y z-s i g s-s p p ~$
and $r b$-spp-body-pprod $=$ pprod $^{\prime} . a u x . r b-s p p-b o d y$
and $r b$-spp-aux-pprod $=$ pprod'.aux.rb-spp-aux
and gb-sig-z-pprod ${ }^{\prime}=$ pprod $^{\prime} \cdot a u x . g b-$ sig-z
and $g b$-sig-pprod ${ }^{\prime}=$ pprod $^{\prime} . a u x . g b-s i g$
and rw-rat-strict-pprod $=$ pprod' ${ }^{\prime}$ aux.rw-rat-strict
and $r w-a d d$-strict-pprod $=$ pprod $^{\prime} . a u x . r w-a d d$-strict
subgoal by (rule qpm-nat-inf-term.intro, fact gd-nat-term-id)
subgoal by (fact pprod-pp-of-term)
subgoal by (fact pprod-component-of-term)
subgoal by (simp only: splus-pprod-def)
subgoal by (simp only: adds-term-pprod-def)
subgoal by (simp only: monom-mult-punit-def)
subgoal by (simp only: lt-punit-def)
subgoal by (simp only: lc-punit-def)
subgoal by (simp only: tail-punit-def)
done

### 5.1.3 More Lemmas and Definitions

lemma compute-adds-term-pprod [code]:
adds-term-pprod $u v=($ snd $u=$ snd $v \wedge$ adds-pp-add-linorder $($ fst $u)($ fst $v))$
by (simp add: adds-term-pprod-def pprod.adds-term-def adds-pp-add-linorder-def)
lemma compute-splus-pprod [code]: splus-pprod $t(s, i)=(t+s, i)$
by (simp add: splus-pprod-def pprod.splus-def)
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 $\Rightarrow$ (tail-punit (proj-ord to) p, plus-monomial-less r (lc-punit (proj-ord to) p) (lt-punit (proj-ord to) p))
$\mid$ Some $i \Rightarrow$ let $b=$ snd $(b s!i)$ in (tail-punit (proj-ord to) $p$ - monom-mult-punit (lc-punit (proj-ord to) p/ lc-punit (proj-ord to) b)
 to) $b$ ), $r$ )
by (simp add: plus-monomial-less-def split: option.split)
lemma compute-sig-trd-spp-pprod [code]:
sig-trd-spp-pprod to bs $(v, p) \equiv(v$, sig-trd-spp-aux-pprod to bs $v$ ( $p$, change-ord (proj-ord to) 0))
by (simp add: change-ord-def)
lemmas $[$ code $]=$ conversep-iff
lemma compute-is-pot-ord [code]:
is-pot-ord-pprod (LEX::(('a::nat, 'b::nat) pp $\times$ nat) nat-term-order $)=$ False
(is is-pot-ord-pprod ?lex $=-$ )
is-pot-ord-pprod (DRLEX::(('a::nat, 'b::nat) pp $\times$ nat) nat-term-order $)=$ False
(is is-pot-ord-pprod?drlex $=-$ )
is-pot-ord-pprod (DEG (to:: (('a::nat, 'b::nat) pp $\times$ nat) nat-term-order $)$ ) $=$ False
is-pot-ord-pprod (POT (to:: (('a::nat, 'b::nat) pp $\times$ nat) nat-term-order $)$ ) $=$ True

## proof -

have eq1: snd ((Term-Order.of-exps a b $\left.i)::\left({ }^{\prime} a,{ }^{\prime} b\right) p p \times n a t\right)=i$ for $a b$ and $i:$ :nat
proof -
have snd ((Term-Order.of-exps a bi)::('a, 'b) pp $\times$ nat $)=$ snd (rep-nat-term ((Term-Order.of-exps a b $\left.\left.i)::\left({ }^{\prime} a,{ }^{\prime} b\right) p p \times n a t\right)\right)$
by (simp add: rep-nat-term-prod-def rep-nat-nat-def)
also have $\ldots=i$
proof (rule snd-of-exps)
show snd (rep-nat-term (undefined, $i$ ) ) $i$ by (simp add: rep-nat-term-prod-def rep-nat-nat-def)
qed
finally show? ?thesis.
qed
let ? $u=($ Term-Order.of-exps 100$)::\left({ }^{\prime} a, ' b\right) p p \times n a t$
let $? v=($ Term-Order.of-exps 001$)::\left({ }^{\prime} a, ~ ' b\right) p p \times n a t$
have $\neg$ is-pot-ord-pprod ?lex
proof
assume is-pot-ord-pprod ?lex
moreover have le-of-nat-term-order ?lex ?v ?u
by (simp add: le-of-nat-term-order nat-term-compare-LEX lex-comp lex-comp-aux-def comp-of-ord-def lex-pp-of-exps eq-of-exps)
ultimately have snd ? $v \leq s n d ? u$ by (rule pprod'. $i s$-pot-ordD2)

```
    thus False by (simp add: eq1)
qed
thus is-pot-ord-pprod ?lex = False by simp
have \neg is-pot-ord-pprod ?drlex
proof
    assume is-pot-ord-pprod ?drlex
    moreover have le-of-nat-term-order ?drlex ?v ?u
        by (simp add: le-of-nat-term-order nat-term-compare-DRLEX deg-comp com-
parator-of-def)
    ultimately have snd ?v \leq snd ?u by (rule pprod'.is-pot-ordD2)
    thus False by (simp add: eq1)
qed
thus is-pot-ord-pprod ?drlex = False by simp
have ᄀ is-pot-ord-pprod (DEG to)
proof
    assume is-pot-ord-pprod (DEG to)
    moreover have le-of-nat-term-order (DEG to) ?v ?u
        by (simp add: le-of-nat-term-order nat-term-compare-DEG deg-comp compara-
tor-of-def)
    ultimately have snd ?v \leq snd ?u by (rule pprod'.is-pot-ordD2)
    thus False by (simp add: eq1)
    qed
    thus is-pot-ord-pprod (DEG to) = False by simp
    have is-pot-ord-pprod (POT to)
    by (rule pprod'.is-pot-ordI, simp add: lt-of-nat-term-order nat-term-compare-POT
pot-comp rep-nat-term-prod-def,
        simp add: comparator-of-def)
    thus is-pot-ord-pprod (POT to) = True by simp
qed
corollary is-pot-ord-POT: is-pot-ord-pprod (POT to)
    by (simp only: compute-is-pot-ord)
definition gb-sig-z-pprod to rword-strict fs \equiv
                            (let res = gb-sig-z-pprod' to (rword-strict to) (map (change-ord
(proj-ord to)) fs) in
(length (fst res), snd res))
definition gb-sig-pprod to rword-strict fs \equivgb-sig-pprod' to (rword-strict to) (map
(change-ord (proj-ord to)) fs)
lemma snd-gb-sig-z-pprod'-eq-gb-sig-z-pprod:
    snd (gb-sig-z-pprod' to (rword-strict to) fs) = snd (gb-sig-z-pprod to rword-strict
fs)
    by (simp add: gb-sig-z-pprod-def change-ord-def Let-def)
```

lemma gb-sig-pprod'-eq-gb-sig-pprod:
$g b$-sig-pprod' to (rword-strict to) $f_{s}=g b$-sig-pprod to rword-strict $f s$ by (simp add: gb-sig-pprod-def change-ord-def)
thm pprod'.aux.gb-sig-isGB[OF pprod'.aux.rw-rat-strict-is-strict-rewrite-ord, simplified gb-sig-pprod'-eq-gb-sig-pprod]
thm pprod'.aux.gb-sig-no-zero-red[OF pprod'.aux.rw-rat-strict-is-strict-rewrite-ord is-pot-ord-POT, simplified snd-gb-sig-z-pprod'-eq-gb-sig-z-pprod]

### 5.2 Computations

experiment begin interpretation trivariate $0_{0}$ rat .

## lemma

gb-sig-pprod DRLEX rw-rat-strict-pprod $\left[X^{2} * Z^{\wedge} 3+3 * X^{2} * Y, X * Y * Z\right.$ $\left.+2 * Y^{2}\right]=$
$\left[C_{0}(3 / 4) * X^{\wedge} 3 * Y^{2}-2 * Y^{\wedge} 4,-4 * Y^{\wedge} 3 * Z-3 * X^{2} * Y^{2}, X\right.$ $\left.* Y * Z+2 * Y^{2}, X^{2} * Z^{\wedge} 3+3 * X^{2} * Y\right]$
by eval
end
Recall that the first return value of $g b$-sig-z-pprod is the size of the computed Gröbner basis, and the second return value is the total number of useless zero-reductions:

## lemma

gb-sig-z-pprod (POT DRLEX) rw-rat-strict-pprod ((cyclic DRLEX 6$)::\left(-\Rightarrow_{0}\right.$ rat) list $)=(155,8)$
by eval

## lemma

gb-sig-z-pprod (POT DRLEX) rw-rat-strict-pprod ((katsura DRLEX 5)::(- $\Rightarrow_{0}$ rat) list $)=(29,0)$
by eval

## lemma

gb-sig-z-pprod (POT DRLEX) rw-rat-strict-pprod ((eco DRLEX 8)::(- $\Rightarrow_{0}$ rat) list $)=(76,0)$
by eval

## lemma

gb-sig-z-pprod (POT DRLEX) rw-rat-strict-pprod ((noon DRLEX 5)::(- $\Rightarrow_{0}$ rat) list $)=(83,0)$
by eval
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 $\left(F_{5}\right)$. 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.


[^0]:    *Supported by the Austrian Science Fund (FWF): P 29498-N31

