# Gröbner Bases Theory 

Fabian Immler and Alexander Maletzky*

September 13, 2023


#### Abstract

This formalization is concerned with the theory of Gröbner bases in (commutative) multivariate polynomial rings over fields, originally developed by Buchberger in his 1965 PhD thesis. Apart from the statement and proof of the main theorem of the theory, the formalization also implements algorithms for actually computing Gröbner bases, thus allowing to effectively decide ideal membership in finitely generated polynomial ideals. Furthermore, all functions can be executed on a concrete representation of multivariate polynomials as association lists.


## Contents

## 1 Introduction <br> 6

1.1 Related Work ..... 6
1.2 Future Work ..... 7
2 General Utilities ..... 7
2.1 Lists ..... 7
2.1.1 max-list ..... 11
2.1.2 insort-wrt ..... 12
2.1.3 diff-list and insert-list ..... 14
2.1.4 remdups-wrt ..... 14
2.1.5 map-idx ..... 16
2.1.6 map-dup ..... 18
2.1.7 Filtering Minimal Elements ..... 18
3 Properties of Binary Relations ..... 25
3.1 wfp-on ..... 26
3.2 Relations ..... 28
3.3 Setup for Connection to Theory Abstract-Rewriting.Abstract-Rewriting ..... 29

[^0]3.4 Simple Lemmas ..... 30
3.5 Advanced Results and the Generalized Newman Lemma ..... 34
4 Polynomial Reduction ..... 42
4.1 Basic Properties of Reduction ..... 42
4.2 Reducibility and Addition \& Multiplication ..... 53
4.3 Confluence of Reducibility ..... 61
4.4 Reducibility and Module Membership ..... 62
4.5 More Properties of red, red-single and is-red ..... 65
4.6 Well-foundedness and Termination ..... 77
4.7 Algorithms ..... 86
4.7.1 Function find-adds ..... 86
4.7.2 Function trd ..... 89
5 Gröbner Bases and Buchberger's Theorem ..... 93
5.1 Critical Pairs and S-Polynomials ..... 94
5.2 Buchberger's Theorem ..... 104
5.3 Buchberger's Criteria for Avoiding Useless Pairs ..... 110
5.4 Weak and Strong Gröbner Bases ..... 113
5.5 Alternative Characterization of Gröbner Bases via Represen- tations of S-Polynomials ..... 120
5.6 Replacing Elements in Gröbner Bases ..... 134
5.7 An Inconstructive Proof of the Existence of Finite Gröbner Bases ..... 139
5.8 Relation red-supset ..... 143
5.9 Context od-term ..... 147
6 A General Algorithm Schema for Computing Gröbner Bases 147
6.1 processed ..... 148
6.2 Algorithm Schema ..... 151
6.2.1 const-lt-component ..... 151
6.2.2 Type synonyms ..... 152
6.2.3 Specification of the selector parameter ..... 152
6.2.4 Specification of the add-basis parameter ..... 153
6.2.5 Specification of the add-pairs parameter ..... 153
6.2.6 Function args-to-set ..... 159
6.2.7 Functions count-const-lt-components, count-rem-comps and full-gb ..... 160
6.2.8 Specification of the completion parameter ..... 163
6.2.9 Function $g b$-schema-dummy ..... 166
6.2.10 Function gb-schema-aux ..... 203
6.2.11 Functions gb-schema-direct and term gb-schema-incr ..... 209
6.3 Suitable Instances of the add-pairs Parameter ..... 215
6.3.1 Specification of the crit parameters ..... 215
6.3.2 Suitable instances of the crit parameters ..... 218
6.3.3 Creating Initial List of New Pairs ..... 225
6.3.4 Applying Criteria to New Pairs ..... 233
6.3.5 Applying Criteria to Old Pairs ..... 238
6.3.6 Creating Final List of Pairs ..... 239
6.4 Suitable Instances of the completion Parameter ..... 245
6.5 Suitable Instances of the add-basis Parameter ..... 249
6.6 Special Case: Scalar Polynomials ..... 250
7 Buchberger's Algorithm ..... 251
7.1 Reduction ..... 252
7.2 Pair Selection ..... 257
7.3 Buchberger's Algorithm ..... 258
7.3.1 Special Case: punit ..... 258
8 Benchmark Problems for Computing Gröbner Bases ..... 259
8.1 Cyclic ..... 259
8.2 Katsura ..... 260
8.3 Eco ..... 260
8.4 Noon ..... 260
9 Code Equations Related to the Computation of Gröbner Bases ..... 261
10 Sample Computations with Buchberger's Algorithm ..... 263
10.1 Scalar Polynomials ..... 263
10.2 Vector Polynomials ..... 266
11 Further Properties of Multivariate Polynomials ..... 270
11.1 Modules and Linear Hulls ..... 270
11.2 Ordered Polynomials ..... 271
11.2.1 Sets of Leading Terms and -Coefficients ..... 271
11.2.2 Monicity ..... 274
12 Auto-reducing Lists of Polynomials ..... 278
12.1 Reduction and Monic Sets ..... 278
12.2 Minimal Bases and Auto-reduced Bases ..... 279
12.3 Computing Minimal Bases ..... 284
12.4 Auto-Reduction ..... 285
12.5 Auto-Reduction and Monicity ..... 294
13 Reduced Gröbner Bases ..... 295
13.1 Definition and Uniqueness of Reduced Gröbner Bases ..... 295
13.2 Computing Reduced Gröbner Bases by Auto-Reduction ..... 299
13.2.1 Minimal Bases ..... 299
13.2.2 Computing Minimal Bases ..... 301
13.2.3 Computing Reduced Bases ..... 301
13.2.4 Computing Reduced Gröbner Bases ..... 302
13.2.5 Properties of the Reduced Gröbner Basis of an Ideal ..... 308
13.2.6 Context od-term ..... 309
14 Sample Computations of Reduced Gröbner Bases ..... 310
15 Macaulay Matrices ..... 312
15.1 More about Vectors ..... 313
15.2 More about Matrices ..... 314
15.2.1 nzrows ..... 314
15.2.2 row-space ..... 314
15.2.3 row-echelon ..... 316
15.3 Converting Between Polynomials and Macaulay Matrices ..... 321
15.4 Properties of Macaulay Matrices ..... 328
15.5 Functions Macaulay-mat and Macaulay-list ..... 335
16 Faugère's F4 Algorithm ..... 339
16.1 Symbolic Preprocessing ..... 339
16.2 lin-red ..... 363
16.3 Reduction ..... 365
16.4 Pair Selection ..... 378
16.5 The F4 Algorithm ..... 379
16.5.1 Special Case: punit ..... 379
17 Sample Computations with the F4 Algorithm ..... 380
17.1 Preparations ..... 380
17.2 Computations ..... 383
18 Syzygies of Multivariate Polynomials ..... 384
18.1 Syzygy Modules Generated by Sets ..... 385
18.2 Polynomial Mappings on List-Indices ..... 392
18.3 POT Orders ..... 397
18.4 Gröbner Bases of Syzygy Modules ..... 399
18.4.1 lift-poly-syz ..... 400
18.4.2 proj-poly-syz ..... 402
18.4.3 cofactor-list-syz ..... 406
18.4.4 init-syzygy-list ..... 406
18.4.5 proj-orig-basis ..... 408
18.4.6 filter-syzygy-basis ..... 409
18.4.7 syzygy-module-list ..... 409
18.4.8 Cofactors ..... 414
18.4.9 Modules ..... 415
18.4.10 Gröbner Bases ..... 416
19 Sample Computations of Syzygies ..... 418
19.1 Preparations ..... 418
19.2 Computations ..... 423
19.3 Univariate Polynomials ..... 426
19.4 Homogeneity ..... 430

## 1 Introduction

The theory of Gröbner bases, invented by Buchberger in [2, 3], is ubiquitous in many areas of computer algebra and beyond, as it allows to effectively solve a multitude of interesting, non-trivial problems of polynomial ideal theory. Since its invention in the mid-sixties, the theory has already seen a whole range of extensions and generalizations, some of which are present in this formalization:

- Following [11], the theory is formulated for vector-polynomials instead of ordinary scalar polynomials, thus allowing to compute Gröbner bases of syzygy modules.
- Besides Buchberger's original algorithm, the formalization also features Faugère's $F_{4}$ algorithm [8] for computing Gröbner bases.
- All algorithms for computing Gröbner bases incorporate criteria to avoid useless pairs; see [4] for details.
- Reduced Gröbner bases have been formalized and can be computed by a formally verified algorithm, too.

For further information about Gröbner bases theory the interested reader may consult the introductory paper [5] or literally any book on commutative/computer algebra, e. g. [1, 11].

### 1.1 Related Work

The theory of Gröbner bases has already been formalized in a couple of other proof assistants, listed below in alphabetical order:

- ACL2 [13],
- $\operatorname{Coq}[16,10]$,
- Mizar [15], and
- Theorema $[6,12]$.

Please note that this formalization must not be confused with the algebra proof method based on Gröbner bases [7], which is a completely independent piece of work: our results could in principle be used to formally prove the correctness and, to some extent, completeness of said proof method.

### 1.2 Future Work

This formalization can be extended in several ways:

- One could formalize signature-based algorithms for computing Gröbner bases, as for instance Faugère's $F_{5}$ algorithm [9]. Such algorithms are typically more efficient than Buchberger's algorithm.
- One could establish the connection to elimination theory, exploiting the well-known elimination property of Gröbner bases w.r.t. certain term-orders (e.g. the purely lexicographic one). This would enable the effective simplification (and even solution, in some sense) of systems of algebraic equations.
- One could generalize the theory further to cover also non-commutative Gröbner bases [14].


## 2 General Utilities

```
theory General
    imports Polynomials.Utils
begin
```

A couple of general-purpose functions and lemmas, mainly related to lists.

### 2.1 Lists

lemma distinct-reorder: distinct $(x s @(y \# y s))=\operatorname{distinct}(y \#(x s @ y s))$ by auto
lemma set-reorder: set $(x s$ @ $(y \# y s))=\operatorname{set}(y \#(x s @ y s))$ by simp

```
lemma distinctI:
    assumes \(\bigwedge i j . i<j \Longrightarrow i<\) length \(x s \Longrightarrow j<\) length \(x s \Longrightarrow x s!i \neq x s!j\)
    shows distinct xs
    using assms
proof (induct \(x s\) )
    case Nil
    show? case by simp
next
    case (Cons \(x\) xs)
    show ?case
    proof (simp, intro conjI, rule)
        assume \(x \in\) set \(x s\)
        then obtain \(j\) where \(j<\) length \(x s\) and \(x=x s!j\) by (metis in-set-conv-nth)
        hence Suc \(j<\) length ( \(x \#\) xs) by simp
        have \((x \# x s)!0 \neq(x \# x s)!(S u c j)\) by (rule Cons(2), simp, simp, fact)
        thus False by ( \(\operatorname{simp}\) add: \(\langle x=x s!j\rangle\) )
```

```
    next
        show distinct xs
        proof (rule Cons(1))
            fix ij
            assume i<j and i< length xs and j< length xs
            hence Suc i<Suc j and Suc i<length (x# xs) and Suc j<length ( }x
xs) by simp-all
            hence (x# #s)!(Suc i)}\not=(x#xs)!(Suc j) by (rule Cons(2))
            thus xs ! i\not= xs ! j by simp
        qed
    qed
qed
lemma filter-nth-pairE:
    assumes i<j and i< length (filter P xs) and j<length (filter P xs)
    obtains }\mp@subsup{i}{}{\prime}\mp@subsup{j}{}{\prime}\mathrm{ where }\mp@subsup{i}{}{\prime}<\mp@subsup{j}{}{\prime}\mathrm{ and }\mp@subsup{i}{}{\prime}<length xs and j'<length xs
        and (filter P xs)!i=xs! ' ' and (filter P xs)! j = xs ! j}\mp@subsup{}{}{\prime
    using assms
proof (induct xs arbitrary: i j thesis)
    case Nil
    from Nil(3) show ?case by simp
next
    case (Cons x xs)
    let ?ys = filter P (x# xs)
    show ?case
    proof (cases P x)
        case True
        hence *: ?ys = x # (filter P xs) by simp
    from }\langlei<j\rangle\mathrm{ obtain j0 where j: j=Suc j0 using lessE by blast
    have len-ys:length ?ys = Suc (length (filter P xs)) and ys-j: ?ys ! j = (filter
P xs)!j0
        by (simp only: * length-Cons, simp only: j * nth-Cons-Suc)
    from Cons(5) have j0 < length (filter P xs) unfolding len-ys j by auto
    show ?thesis
    proof (cases i=0)
        case True
        from <j0<length (filter P xs)> obtain j' where j}\mp@subsup{j}{}{\prime}<length xs and **: (filter
P xs)!j0 = xs ! j'
        by (metis (no-types, lifting) in-set-conv-nth mem-Collect-eq nth-mem set-filter)
        have 0<Suc j' by simp
        thus ?thesis
            by (rule Cons(2), simp, simp add: <j' < length xs`, simp only: True *
nth-Cons-0,
            simp only: ys-j nth-Cons-Suc **)
    next
        case False
        then obtain i0 where i: i=Suc i0 using lessE by blast
        have ys-i: ?ys ! i=(filter P xs) ! i0 by (simp only: i* nth-Cons-Suc)
        from Cons(3) have i0 < j0 by (simp add: ij)
```

```
    from Cons(4) have i0 < length (filter P xs) unfolding len-ys i by auto
    from - <i0 < j0> this <j0 < length (filter P xs)>obtain i' j'
            where }\mp@subsup{i}{}{\prime}<\mp@subsup{j}{}{\prime}\mathrm{ and }\mp@subsup{i}{}{\prime}<length xs and j'<length xs
                and }\mp@subsup{i}{}{\prime}:\mathrm{ filter P xs!i0 =xs! i' and j': filter P xs ! j0 = xs! j'
            by (rule Cons(1))
    from «i'< j'〉 have Suc i'<Suc j' by simp
    thus ?thesis
            by (rule Cons(2), simp add: < i'< length xs`, simp add: <j'< length xs>,
                simp only: ys-i nth-Cons-Suc i', simp only: ys-j nth-Cons-Suc j')
    qed
    next
        case False
    hence *: ?ys = filter P xs by simp
    with Cons(4) Cons(5) have i<length (filter P xs) and j<length (filter P
xs) by simp-all
    with - <i< j> obtain }\mp@subsup{i}{}{\prime}\mp@subsup{j}{}{\prime}\mathrm{ where }\mp@subsup{i}{}{\prime}<\mp@subsup{j}{}{\prime}\mathrm{ and }\mp@subsup{i}{}{\prime}<length xs and j'<length
xs
        and i': filter P xs !i=xs! i' and j': filter P xs ! j=xs! j}\mp@subsup{j}{}{\prime
        by (rule Cons(1))
    from < i'< j'〉 have Suc i'< Suc j' by simp
    thus ?thesis
    by (rule Cons(2), simp add: <i' < length xs`, simp add: <j' < length xs>,
            simp only: * nth-Cons-Suc i', simp only: * nth-Cons-Suc j')
    qed
qed
lemma distinct-filterI:
    assumes }\bigwedgeij.i<j\Longrightarrowi< length xs \Longrightarrow < < length xs \LongrightarrowP(xs!i)\Longrightarrow
(xs!j)\Longrightarrowxs!i\not=xs!j
    shows distinct (filter P xs)
proof (rule distinctI)
    fix i j::nat
    assume i<j and i<length (filter P xs) and j<length (filter P xs)
    then obtain }\mp@subsup{i}{}{\prime}\mp@subsup{j}{}{\prime}\mathrm{ where }\mp@subsup{i}{}{\prime}<\mp@subsup{j}{}{\prime}\mathrm{ and }\mp@subsup{i}{}{\prime}<length xs and j'<length xs
        and i:(filter P xs)!i=xs! i' and j:(filter P xs)! j = xs ! j' by (rule
filter-nth-pairE)
    from }\langle\mp@subsup{i}{}{\prime}<\mp@subsup{j}{}{\prime}\rangle\langle\mp@subsup{i}{}{\prime}<length xs\rangle<\mp@subsup{j}{}{\prime}<length xs\rangle show (filter P xs)!i\not=(filter P
xs)! j unfolding i j
    proof (rule assms)
        from <i< length (filter P xs)> show P (xs! i') unfolding i[symmetric] using
nth-mem by force
    next
            from < j< length (filter P xs)> show P (xs! j') unfolding j[symmetric] using
nth-mem by force
    qed
qed
lemma set-zip-map: set (zip (map f xs) (map g xs)) = (\lambdax. (fx,g x))'(set xs)
proof -
```

```
    have {(map f xs ! i, map g xs ! i)| i. i< length xs } = {(f(xs!i),g(xs!i)) |i.
i< length xs}
    proof (rule Collect-eqI, rule, elim exE conjE, intro exI conjI, simp add: map-nth,
assumption,
            elim exE conjE, intro exI)
            fix }x
    assume }x=(f(xs!i),g(xs!i)) and i< length xs
    thus x=(map fxs!i, map gxs!i)^i<length xs by (simp add:map-nth)
    qed
    also have ... = (\lambdax.(fx,gx))'{xs!i|i.i< length xs} by blast
    finally show set (zip (map fxs) (map g xs)) = (\lambdax. (fx,g x))'(set xs)
    by (simp add: set-zip set-conv-nth[symmetric])
qed
lemma set-zip-map1: set (zip (map f xs) xs) = (\lambdax. (f x, x))'(set xs)
proof -
    have set (zip (map fxs) (map id xs)) = ( \lambdax. (f x, id x))' (set xs) by (rule
set-zip-map)
    thus ?thesis by simp
qed
lemma set-zip-map2: set (zip xs (map f xs)) = (\lambdax. (x,f x))'(set xs)
proof -
    have set (zip (map id xs) (map fxs)) = (\lambdax. (id x, f x))' (set xs) by (rule
set-zip-map)
    thus ?thesis by simp
qed
lemma UN-upt: (\bigcupi\in{0..<length xs}.f(xs!i)) =(\bigcupx\inset xs.fx)
    by (metis image-image map-nth set-map set-upt)
lemma sum-list-zeroI':
    assumes }\bigwedgei.i<length xs \Longrightarrowxs!i=
    shows sum-list xs = 0
proof (rule sum-list-zeroI, rule, simp)
    fix }
    assume }x\in\mathrm{ set xs
    then obtain i where i< length xs and x=xs!i by (metis in-set-conv-nth)
    from this(1) show }x=0\mathrm{ unfolding }\langlex=xs!i\rangle by (rule assms
qed
lemma sum-list-map2-plus:
    assumes length xs = length ys
    shows sum-list (map2 (+) xs ys) = sum-list xs + sum-list (ys::'a::comm-monoid-add
list)
    using assms
proof (induct rule: list-induct2)
    case Nil
    show ?case by simp
```

```
next
    case (Cons x xs y ys)
    show ?case by (simp add: Cons(2) ac-simps)
qed
lemma sum-list-eq-nthI:
    assumes i< length xs and }\bigwedgej.j< length xs \Longrightarrowj\not=i\Longrightarrowxs!j=
    shows sum-list xs = xs!i
    using assms
proof (induct xs arbitrary: i)
    case Nil
    from Nil(1) show ?case by simp
next
    case (Cons x xs)
    have *: xs ! j=0 if j< length xs and Suc j\not=i for j
    proof -
    have xs ! j = (x # xs)! (Suc j) by simp
    also have ... = 0 by (rule Cons(3), simp add: <j < length xs`, fact)
    finally show ?thesis .
    qed
    show ?case
    proof (cases i)
        case 0
        have sum-list xs = 0 by (rule sum-list-zeroI', erule *, simp add: 0)
        with 0 show ?thesis by simp
    next
        case (Suc k)
        with Cons(2) have k< length xs by simp
    hence sum-list xs=xs!k
    proof (rule Cons(1))
            fix j
            assume j < length xs
            assume j\not=k
            hence Suc j\not=i by (simp add: Suc)
            with < j < length xs` show xs ! j = 0 by (rule *)
        qed
    moreover have }x=
    proof -
            have }x=(x#xs)!0 by sim
            also have ... = 0 by (rule Cons(3), simp-all add: Suc)
            finally show ?thesis.
    qed
    ultimately show ?thesis by (simp add: Suc)
    qed
qed
```


### 2.1.1 max-list

```
fun (in ord) max-list \(::\) ' \(a\) list \(\Rightarrow{ }^{\prime} a\) where
```

```
    max-list (x # xs)=( case xs of [] # x | - m max x (max-list xs))
context linorder
begin
lemma max-list-Max:xs \not=[]\Longrightarrow max-list xs = Max (set xs)
    by (induct xs rule: induct-list012, auto)
lemma max-list-ge:
    assumes x\in set xs
    shows x < max-list xs
proof -
    from assms have xs }\not=[] by aut
    from finite-set assms have x\leqMax (set xs) by (rule Max-ge)
    also from }\langlexs\not=[]\rangle\mathrm{ have Max (set xs) = max-list xs by (rule max-list-Max[symmetric])
    finally show ?thesis.
qed
lemma max-list-boundedI:
    assumes }xs\not=[] \mathrm{ and }\bigwedgex.x\in set xs \Longrightarrowx\leq
    shows max-list xs \leqa
proof -
    from assms(1) have set xs }\not={}\mathrm{ by simp
    from assms(1) have max-list xs = Max (set xs) by (rule max-list-Max)
    also from finite-set 〈set xs }\not={}`\operatorname{assms(2) have .. . \leqa by (rule Max.boundedI)
    finally show ?thesis.
qed
end
```


### 2.1.2 insort-wrt

```
primrec insort-wrt :: \(\left({ }^{\prime} c \Rightarrow{ }^{\prime} c \Rightarrow\right.\) bool \() \Rightarrow{ }^{\prime} c \Rightarrow{ }^{\prime} c\) list \(\Rightarrow{ }^{\prime} c\) list where
    insort-wrt - x []=[x]|
    insort-wrt r x (y# ys) =
        (if r x y then (x#y# ys) else y # (insort-wrt r x ys))
lemma insort-wrt-not-Nil [simp]: insort-wrt r x xs \not= []
    by (induct xs, simp-all)
lemma length-insort-wrt [simp]: length (insort-wrt r x xs) = Suc (length xs)
    by (induct xs, simp-all)
lemma set-insort-wrt [simp]: set (insort-wrt r x xs) = insert x (set xs)
    by (induct xs, auto)
lemma sorted-wrt-insort-wrt-imp-sorted-wrt:
    assumes sorted-wrt r (insort-wrt s x xs)
    shows sorted-wrt r xs
```

```
    using assms
proof (induct xs)
    case Nil
    show ?case by simp
next
    case (Cons a xs)
    show ?case
    proof (cases s x a)
        case True
        with Cons.prems have sorted-wrt r (x # a # xs) by simp
        thus ?thesis by simp
    next
        case False
        with Cons(2) have sorted-wrt r (a # (insort-wrt s x xs)) by simp
        hence *: (\forally\inset xs.r a y) and sorted-wrt r (insort-wrt s x xs)
            by (simp-all)
        from this(2) have sorted-wrt r xs by (rule Cons(1))
        with * show ?thesis by (simp)
    qed
qed
lemma sorted-wrt-imp-sorted-wrt-insort-wrt:
    assumes transp r and \bigwedge\a.r a x\veerx a and sorted-wrt r xs
    shows sorted-wrt r (insort-wrt r x xs)
    using assms(3)
proof (induct xs)
    case Nil
    show ?case by simp
next
    case (Cons a xs)
    show ?case
    proof (cases r x a)
        case True
        with Cons(2) assms(1) show ?thesis by (auto dest: transpD)
    next
        case False
        with assms(2) have r a x by blast
        from Cons(2) have *: (\forally\inset xs.r a y) and sorted-wrt r xs
            by (simp-all)
        from this(2) have sorted-wrt r (insort-wrt r x xs) by (rule Cons(1))
        with \langler a x\rangle* show ?thesis by (simp add: False)
    qed
qed
corollary sorted-wrt-insort-wrt:
    assumes transpr and \\a.r a x \veerxa
    shows sorted-wrt r (insort-wrt r x xs) \longleftrightarrow sorted-wrt r xs (is ?l \longleftrightarrow \longleftrightarrowr)
proof
    assume ?l
```

```
    then show ?r by (rule sorted-wrt-insort-wrt-imp-sorted-wrt)
next
    assume ?r
    with assms show ?l by (rule sorted-wrt-imp-sorted-wrt-insort-wrt)
qed
```


### 2.1.3 diff-list and insert-list

definition diff-list $::$ 'a list $\Rightarrow$ 'a list $\Rightarrow$ 'a list (infixl -- 65)
where diff-list xs ys $=$ fold removeAll ys xs
lemma set-diff-list: set $(x s--y s)=$ set $x s-$ set $y s$
by (simp only: diff-list-def, induct ys arbitrary: xs, auto)
lemma diff-list-disjoint: set ys $\cap$ set $(x s--y s)=\{ \}$
unfolding set-diff-list by (rule Diff-disjoint)
lemma subset-append-diff-cancel:
assumes set ys $\subseteq$ set $x s$
shows set $(y s$ @ $(x s--y s))=$ set $x s$
by (simp only: set-append set-diff-list Un-Diff-cancel, rule Un-absorb1, fact)
definition insert-list $::$ ' $a \Rightarrow$ 'a list $\Rightarrow$ 'a list
where insert-list $x$ xs $=($ if $x \in$ set $x s$ then $x s$ else $x \# x s)$
lemma set-insert-list: set (insert-list $x$ xs) $=$ insert $x$ (set $x s$ )
by (auto simp add: insert-list-def)

### 2.1.4 remdups-wrt

primrec remdups-wrt :: ('a $\left.{ }^{\prime} ' b\right) \Rightarrow{ }^{\prime}$ a list $\Rightarrow$ 'a list where
remdups-wrt-base: remdups-wrt - [] = [] |
remdups-wrt-rec: remdups-wrt $f(x \# x s)=\left(\right.$ if $f x \in f^{\prime}$ set xs then remdups-wrt
f xs else $x$ \# remdups-wrt f xs)

```
lemma set-remdups-wrt: \(f\) ' set (remdups-wrt \(f\) xs \()=f\) ' set xs
proof (induct xs)
    case Nil
    show ?case unfolding remdups-wrt-base ..
next
    case (Cons a xs)
    show ?case unfolding remdups-wrt-rec
    proof (simp only: split: if-splits, intro conjI, intro impI)
    assume \(f a \in f\) ' set xs
        have \(f\) 'set \((a \# x s)=\operatorname{insert}(f a)(f\) 'set \(x s)\) by simp
    have \(f\) ' set (remdups-wrt \(f\) xs) \(=f\) ' set xs by fact
    also from \(\left\langle f a \in f^{\prime}\right.\) set \(\left.x s\right\rangle\) have \(\ldots=\operatorname{insert}(f a)(f\) ' set \(x s)\) by (simp add:
insert-absorb)
    also have \(\ldots=f\) ' \(\operatorname{set}(a \# x s)\) by simp
    finally show \(f\) ' set (remdups-wrt \(f x s)=f ‘ \operatorname{set}(a \# x s)\).
```

qed (simp add: Cons.hyps)
qed
lemma subset-remdups-wrt: set (remdups-wrt f xs) $\subseteq$ set xs by (induct xs, auto)
lemma remdups-wrt-distinct-wrt:
assumes $x \in$ set (remdups-wrt $f x s$ ) and $y \in \operatorname{set}$ (remdups-wrt $f x s$ ) and $x \neq y$
shows $f x \neq f y$
using assms(1) assms(2)
proof (induct xs)
case Nil
thus?case unfolding remdups-wrt-base by simp
next
case (Cons a xs)
from Cons(2) Cons(3) show ?case unfolding remdups-wrt-rec
proof (simp only: split: if-splits)
assume $x \in \operatorname{set}$ (remdups-wrt $f x s$ ) and $y \in \operatorname{set}$ (remdups-wrt $f x s$ )
thus $f x \neq f y$ by (rule Cons.hyps)
next
assume $\neg$ True
thus $f x \neq f y$ by simp
next
assume $f a \notin f^{\prime}$ set xs and xin: $x \in \operatorname{set}(a \#$ remdups-wrt $f x s)$ and yin: $y \in$ set ( $a \neq$ remdups-wrt $f$ xs)
from yin have $y: y=a \vee y \in \operatorname{set}$ (remdups-wrt f xs) by simp
from xin have $x=a \vee x \in \operatorname{set}$ (remdups-wrt $f$ xs) by simp
thus $f x \neq f y$
proof
assume $x=a$
from $y$ show ?thesis
proof
assume $y=a$
with $\langle x \neq y\rangle$ show ?thesis unfolding $\langle x=a\rangle$ by simp
next
assume $y \in \operatorname{set}$ (remdups-wrt $f x s$ )
have $y \in$ set xs by (rule, fact, rule subset-remdups-wrt)
hence $f y \in f$ ' set $x s$ by simp
with $\langle f a \notin f$ ' set $x s\rangle$ show ?thesis unfolding $\langle x=a\rangle$ by auto
qed
next
assume $x \in$ set (remdups-wrt $f$ xs)
from $y$ show ?thesis
proof
assume $y=a$
have $x \in$ set xs by (rule, fact, rule subset-remdups-wrt)
hence $f x \in f$ ' set $x s$ by simp
with $\langle f a \notin f$ ' set $x s\rangle$ show ?thesis unfolding $\langle y=a\rangle$ by auto next

```
            assume y f set (remdups-wrt f xs)
            with <x set (remdups-wrt f xs)\rangle show ?thesis by (rule Cons.hyps)
            qed
        qed
    qed
qed
lemma distinct-remdups-wrt: distinct (remdups-wrt f xs)
proof (induct xs)
    case Nil
    show ?case unfolding remdups-wrt-base by simp
next
    case (Cons a xs)
    show ?case unfolding remdups-wrt-rec
    proof (split if-split, intro conjI impI, rule Cons.hyps)
        assume f a\not\inf' set xs
        hence a & set xs by auto
        hence a & set (remdups-wrt f xs) using subset-remdups-wrt[of f xs] by auto
        with Cons.hyps show distinct (a# remdups-wrt f xs) by simp
    qed
qed
lemma map-remdups-wrt: map f (remdups-wrt f xs) = remdups (map f xs)
    by (induct xs,auto)
lemma remdups-wrt-append:
    remdups-wrt f(xs @ ys)=(filter (\lambdaa.f a\not\inf'set ys)(remdups-wrt fxs))@
(remdups-wrt f ys)
    by (induct xs, auto)
```


### 2.1.5 map-idx

```
primrec map-idx :: \(\left({ }^{\prime} a \Rightarrow\right.\) nat \(\left.\Rightarrow{ }^{\prime} b\right) \Rightarrow{ }^{\prime} a\) list \(\Rightarrow\) nat \(\Rightarrow\) 'b list where
    map-idx f [] n = []|
    map-idx f(x# xs) n = (fx n) # (map-idx fxs (Suc n))
lemma map-idx-eq-map2: map-idx f xs n = map2 f xs [n..<n + length xs]
proof (induct xs arbitrary: n)
    case Nil
    show ?case by simp
next
    case (Cons x xs)
    have eq: [n..<n + length (x # xs)] = n # [Suc n..<Suc ( }n+l\mathrm{ length xs)]
    by (metis add-Suc-right length-Cons less-add-Suc1 upt-conv-Cons)
    show ?case unfolding eq by (simp add: Cons del: upt-Suc)
qed
lemma length-map-idx [simp]: length (map-idx f xs n) = length xs
    by (simp add: map-idx-eq-map2)
```

lemma map-idx-append: map-idx $f(x s$ @ ys) $n=(\operatorname{map}-i d x f x s n) @(\operatorname{map}-i d x f$ ys ( $n+$ length $x s$ ))
by (simp add: map-idx-eq-map2 ab-semigroup-add-class.add-ac(1) zip-append1)
lemma map-idx-nth:
assumes $i<$ length xs
shows $(\operatorname{map}-i d x f x s n)!i=f(x s!i)(n+i)$
using assms by (simp add: map-idx-eq-map2)
lemma map-map-idx: map $f(\operatorname{map-idx} g$ xs $n)=\operatorname{map-idx}(\lambda x i . f(g x i))$ xs $n$ by (auto simp add: map-idx-eq-map2)
lemma map-idx-map: map-idx $f($ map $g x s) n=\operatorname{map-idx}(f \circ g)$ xs $n$ by (simp add: map-idx-eq-map2 map-zip-map)
lemma map-idx-no-idx: map-idx $(\lambda x-. f x)$ xs $n=\operatorname{map} f x s$ by (induct xs arbitrary: $n$, simp-all)
lemma map-idx-no-elem: map-idx $(\lambda$-. $f)$ xs $n=\operatorname{map} f[n . .<n+$ length $x s]$
proof (induct xs arbitrary: n)
case Nil
show? case by simp
next
case (Cons $x$ xs)
have eq: $[n . .<n+$ length $(x \# x s)]=n \#[$ Suc $n . .<$ Suc $(n+$ length $x s)]$
by (metis add-Suc-right length-Cons less-add-Suc1 upt-conv-Cons)
show ? case unfolding eq by (simp add: Cons del: upt-Suc)
qed
lemma map-idx-eq-map: map-idx fxs $n=\operatorname{map}(\lambda i . f(x s!i)(i+n))[0 . .<l e n g t h$ $x s]$
proof (induct xs arbitrary: n)
case Nil
show ? case by simp
next
case (Cons $x$ xs)
have eq: $[0 . .<$ length $(x \# x s)]=0 \#[$ Suc $0 . .<$ Suc (length $x s)]$
by (metis length-Cons upt-conv-Cons zero-less-Suc)
have map $(\lambda i . f((x \# x s)!i)(i+n))[$ Suc 0.. $<$ Suc $($ length $x s)]=$ $\operatorname{map}((\lambda i . f((x \# x s)!i)(i+n)) \circ S u c)[0 . .<$ length $x s]$
by (metis map-Suc-upt map-map)
also have $\ldots=\operatorname{map}(\lambda i . f(x s!i)(S u c(i+n)))[0 . .<$ length $x s]$
by (rule map-cong, fact refl, simp)
finally show ?case unfolding eq by (simp add: Cons del: upt-Suc)
qed
lemma set-map-idx: set (map-idxfxs $n)=(\lambda i . f(x s!i)(i+n)) '\{0 . .<$ length $x s\}$

```
by (simp add: map-idx-eq-map)
```


### 2.1.6 map-dup

primrec map-dup $::\left({ }^{\prime} a \Rightarrow{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} b\right) \Rightarrow$ 'a list $\Rightarrow{ }^{\prime} b$ list where map-dup - - [] = []
map-dup $f(x \# x s)=($ if $x \in$ set $x s$ then $g x$ else $f x) \#($ map-dup $f g x s)$
lemma length-map-dup[simp]: length (map-dup $\mathrm{f} g \mathrm{xs}$ ) $=$ length $x s$
by (induct xs, simp-all)
lemma map-dup-distinct:
assumes distinct $x s$
shows map-dup f g xs $=$ map $f$ xs
using assms by (induct xs, simp-all)
lemma filter-map-dup-const:
filter $(\lambda x . x \neq c)(\operatorname{map}-d u p f(\lambda-. c) x s)=$ filter $(\lambda x . x \neq c)($ map $f($ remdups xs))
by (induct $x s$, simp-all)
lemma filter-zip-map-dup-const:
filter $(\lambda(a, b) . a \neq c)(z i p($ map-dup $f(\lambda-. c) x s) x s)=$
filter $(\lambda(a, b) . a \neq c)(z i p($ map $f(r e m d u p s ~ x s))($ remdups $x s))$
by (induct xs, simp-all)

### 2.1.7 Filtering Minimal Elements

```
context
    fixes rel :: 'a # 'a b bool
begin
```

primrec filter-min-aux :: 'a list $\Rightarrow$ 'a list $\Rightarrow$ 'a list where
filter-min-aux [] ys =ys|
filter-min-aux ( $x \#$ xs) ys $=$
(if $(\exists y \in($ set $x s \cup$ set $y s)$. rel $y x)$ then (filter-min-aux xs ys)
else (filter-min-aux xs (x \# ys)))
definition filter-min :: 'a list $\Rightarrow$ 'a list
where filter-min xs $=$ filter-min-aux xs []
definition filter-min-append $::$ ' ${ }^{\prime}$ list $\Rightarrow$ 'a list $\Rightarrow$ 'a list
where filter-min-append xs ys $=$
(let $P=(\lambda z s . \lambda x$. $\neg(\exists z \in$ set $z s$. rel $z x))$; ys $1=$ filter ( $P$ xs) ys in
(filter (Pys1) xs) @ ys1)
lemma filter-min-aux-supset: set ys $\subseteq$ set (filter-min-aux xs ys)
proof (induct xs arbitrary: ys)
case Nil
show ?case by simp

```
next
    case (Cons x xs)
    have set ys \subseteqset (x # ys) by auto
    also have set (x # ys)\subseteq set (filter-min-aux xs (x # ys)) by (rule Cons.hyps)
    finally have set ys \subseteqset (filter-min-aux xs (x # ys)).
    moreover have set ys \subseteqset (filter-min-aux xs ys) by (rule Cons.hyps)
    ultimately show ?case by simp
qed
lemma filter-min-aux-subset: set (filter-min-aux xs ys)\subseteq set xs \cup set ys
proof (induct xs arbitrary: ys)
    case Nil
    show ?case by simp
next
    case (Cons x xs)
    note Cons.hyps
    also have set xs U set ys \subseteqset (x# xs)\cup set ys by fastforce
    finally have c1: set (filter-min-aux xs ys)\subseteq set (x # xs) \cup set ys.
    note Cons.hyps
    also have set xs \cup set (x# ys) = set (x# xs) \cup set ys by simp
    finally have set (filter-min-aux xs (x # ys))\subseteq set (x # xs)\cup set ys .
    with c1 show ?case by simp
qed
lemma filter-min-aux-relE:
    assumes transp rel and x\in set xs and x & set (filter-min-aux xs ys)
    obtains y where }y\in\mathrm{ set (filter-min-aux xs ys) and rel y x
    using assms(2, 3)
proof (induct xs arbitrary:x ys thesis)
    case Nil
    from Nil(2) show ?case by simp
next
    case (Cons x0 xs)
    from Cons(3) have x=x0\vee x\in set xs by simp
    thus ?case
    proof
    assume x = x0
    from Cons(4) have *: \existsy\inset xs \cup set ys. rel y x0
    proof (simp add:<x = x0` split: if-splits)
            assume x0 # set (filter-min-aux xs (x0 # ys))
            moreover from filter-min-aux-supset have x0 \in set (filter-min-aux xs (x0
# ys))
            by (rule subsetD) simp
            ultimately show False ..
    qed
    hence eq: filter-min-aux (x0 # xs) ys = filter-min-aux xs ys by simp
    from * obtain x1 where x1 \in set xs \cup set ys and rel x1 x unfolding <x =
x0> ..
```

```
    from this(1) show ?thesis
    proof
        assume x1 \in set xs
        show ?thesis
        proof (cases x1 \in set (filter-min-aux xs ys))
            case True
            hence x1 \in set (filter-min-aux (x0 # xs) ys) by (simp only: eq)
            thus ?thesis using \langlerel x1 x〉 by (rule Cons(2))
        next
            case False
            with}\langlex1\in\mathrm{ set xs` obtain y where y fet (filter-min-aux xs ys) and rel
y x1
                using Cons.hyps by blast
            from this(1) have y\in set (filter-min-aux (x0 # xs) ys) by (simp only:eq)
            moreover from assms(1)\langlerel y x1〉\langlerel x1 x\rangle have rel y x by (rule transpD)
                ultimately show ?thesis by (rule Cons(2))
            qed
    next
            assume x1 \in set ys
            hence x1 \in set (filter-min-aux (x0 # xs) ys) using filter-min-aux-supset ..
            thus ?thesis using <rel x1 x〉 by (rule Cons(2))
    qed
    next
        assume x f set xs
        show ?thesis
        proof (cases \existsy\inset xs U set ys. rel y x0)
            case True
            hence eq: filter-min-aux (x0 # xs) ys = filter-min-aux xs ys by simp
            with Cons(4) have x & set (filter-min-aux xs ys) by simp
            with}\langlex\in\mathrm{ set xs` obtain y where }y\in\mathrm{ set (filter-min-aux xs ys) and rel y x
                using Cons.hyps by blast
            from this(1) have y set (filter-min-aux (x0 # xs) ys) by (simp only: eq)
            thus ?thesis using <rel y x〉 by (rule Cons(2))
    next
        case False
        hence eq: filter-min-aux (x0 # xs) ys = filter-min-aux xs (x0 # ys) by simp
        with Cons(4) have x & set (filter-min-aux xs (x0 # ys)) by simp
        with }\langlex\in\operatorname{set}xs\rangle\mathrm{ obtain }y\mathrm{ where }y\in\operatorname{set}(filter-min-aux xs (x0 # ys)) and
rel y x
            using Cons.hyps by blast
            from this(1) have y set (filter-min-aux (x0 # xs) ys) by (simp only: eq)
            thus ?thesis using <rel y x\rangle by (rule Cons(2))
        qed
    qed
qed
lemma filter－min－aux－minimal：
assumes transp rel and \(x \in\) set（filter－min－aux xs ys）and \(y \in\) set（filter－min－aux xs ys）
```

```
    and rel x y
    assumes \bigwedgeab.a set xs \cup set ys \Longrightarrowb set ys \Longrightarrowrel a b\Longrightarrowa=b
    shows }x=
    using assms(2-5)
proof (induct xs arbitrary: x y ys)
    case Nil
    from Nil(1) have x\in set [] \cup set ys by simp
    moreover from Nil(2) have y\in set ys by simp
    ultimately show ?case using Nil(3) by (rule Nil(4))
next
    case (Cons x0 xs)
    show ?case
    proof (cases \existsy\inset xs U set ys. rel y x0)
    case True
    hence eq: filter-min-aux (x0 # xs) ys = filter-min-aux xs ys by simp
    with Cons(2, 3) have x\in set (filter-min-aux xs ys) and y set (filter-min-aux
xs ys)
            by simp-all
            thus ?thesis using Cons(4)
            proof (rule Cons.hyps)
            fix ab
            assume a set xs \cup set ys
            hence a\in set (x0 # xs) \cup set ys by simp
            moreover assume b\in set ys and rel a b
            ultimately show }a=b\mathrm{ by (rule Cons(5))
    qed
    next
    case False
    hence eq: filter-min-aux (x0 # xs) ys = filter-min-aux xs (x0 # ys) by simp
            with Cons(2, 3) have }x\in\operatorname{set}(filter-min-aux xs (x0 # ys)) and y se
(filter-min-aux xs (x0 # ys))
            by simp-all
            thus ?thesis using Cons(4)
                    proof (rule Cons.hyps)
            fix ab
            assume a: a\in set xs \cup set (x0 # ys) and b\inset (x0 # ys) and rel a b
            from this(2) have b=x0\veeb\in set ys by simp
            thus }a=
            proof
                assume b=x0
                from a have a=x0\vee a\in set xs \cup set ys by simp
                thus ?thesis
                proof
                assume a=x0
                with }\langleb=x0\rangle\mathrm{ show ?thesis by simp
                    next
                assume a set xs \cup set ys
                hence \existsy\inset xs \cup set ys. rel y x0 using <rel a b> unfolding <b = x0〉 ..
                with False show ?thesis ..
```

```
            qed
            next
                from a have }a\in\operatorname{set}(x0# #s)\cup set ys by sim
                moreover assume b\in set ys
                ultimately show ?thesis using <rel a b〉 by (rule Cons(5))
            qed
    qed
    qed
qed
lemma filter-min-aux-distinct:
    assumes reflp rel and distinct ys
    shows distinct (filter-min-aux xs ys)
    using assms(2)
proof (induct xs arbitrary: ys)
    case Nil
    thus ?case by simp
next
    case (Cons x xs)
    show ?case
    proof (simp split: if-split, intro conjI impI)
        from Cons(2) show distinct (filter-min-aux xs ys) by (rule Cons.hyps)
    next
        assume a: \forally\inset xs \cup set ys. \neg rel y x
        show distinct (filter-min-aux xs (x # ys))
        proof (rule Cons.hyps)
            have }x\not\in\mathrm{ set ys
            proof
                assume x\in set ys
                    hence }x\in\mathrm{ set xs U set ys by simp
                    with a have \neg rel x x ..
                    moreover from assms(1) have rel x x by (rule reflpD)
                    ultimately show False ..
            qed
            with Cons(2) show distinct (x # ys) by simp
        qed
    qed
qed
lemma filter-min-subset: set (filter-min xs)\subseteq set xs
    using filter-min-aux-subset[of xs []] by (simp add: filter-min-def)
lemma filter-min-cases:
    assumes transp rel and x\in set xs
    assumes }x\in\mathrm{ set (filter-min xs) }\Longrightarrow\mathrm{ thesis
    assumes }\y.y\in\operatorname{set}(\mathrm{ filter-min xs) }\Longrightarrowx\not\in\operatorname{set (filter-min xs) \Longrightarrow rel y x \Longrightarrow
thesis
    shows thesis
proof (cases x set (filter-min xs))
```

```
    case True
    thus ?thesis by (rule assms(3))
next
    case False
    with assms(1, 2) obtain y where y fet (filter-min xs) and rel y x
        unfolding filter-min-def by (rule filter-min-aux-relE)
    from this(1) False this(2) show ?thesis by (rule assms(4))
qed
corollary filter-min-relE:
    assumes transp rel and reflp rel and x\in set xs
    obtains y where }y\in\mathrm{ set (filter-min xs) and rel y x
    using assms(1, 3)
proof (rule filter-min-cases)
    assume x f set (filter-min xs)
    moreover from assms(2) have rel x x by (rule reflpD)
    ultimately show ?thesis ..
qed
lemma filter-min-minimal:
    assumes transp rel and x\in set (filter-min xs) and y\in set (filter-min xs) and
rel x y
    shows }x=
    using assms unfolding filter-min-def by (rule filter-min-aux-minimal) simp
lemma filter-min-distinct:
    assumes reflp rel
    shows distinct (filter-min xs)
    unfolding filter-min-def by (rule filter-min-aux-distinct, fact, simp)
lemma filter-min-append-subset: set (filter-min-append xs ys)\subseteq set xs \cup set ys
    by (auto simp: filter-min-append-def)
lemma filter-min-append-cases:
    assumes transp rel and x\in set xs U set ys
    assumes }x\in\mathrm{ set (filter-min-append xs ys) }\Longrightarrow\mathrm{ thesis
    assumes }\y.y\in\operatorname{set}(\mathrm{ filter-min-append xs ys) }\Longrightarrowx\not\in set (filter-min-append xs
ys)\Longrightarrow rel y x \ thesis
    shows thesis
proof (cases x set (filter-min-append xs ys))
    case True
    thus ?thesis by (rule assms(3))
next
    case False
    define P where P}=(\lambdazs.\lambdaa.\neg(\existsz\in\mathrm{ set zs. rel z a)}
    from assms(2) obtain y where y\in set (filter-min-append xs ys) and rel y x
    proof
        assume x\in set xs
        with False obtain y where y}\in\operatorname{set}(\mathrm{ (ilter-min-append xs ys) and rel y x
```

```
    by (auto simp: filter-min-append-def P-def)
    thus ?thesis ..
    next
    assume x f set ys
    with False obtain }y\mathrm{ where }y\in\mathrm{ set xs and rel y x
        by (auto simp: filter-min-append-def P-def)
    show ?thesis
    proof (cases y fet (filter-min-append xs ys))
        case True
        thus ?thesis using <rel y x〉 ..
    next
        case False
        with }\langley\in\mathrm{ set xs> obtain }\mp@subsup{y}{}{\prime}\mathrm{ where }\mp@subsup{y}{}{\prime}:\mp@subsup{y}{}{\prime}\in\mathrm{ set (filter-min-append xs ys) and
rel y'y
            by (auto simp: filter-min-append-def P-def)
        from assms(1) this(2)<rel y x> have rel y' x by (rule transpD)
        with y' show ?thesis ..
    qed
    qed
    from this(1) False this(2) show ?thesis by (rule assms(4))
qed
corollary filter-min-append-relE:
    assumes transp rel and reflp rel and x\in set xs \cup set ys
    obtains }y\mathrm{ where }y\in\mathrm{ set (filter-min-append xs ys) and rel y x
    using assms(1, 3)
proof (rule filter-min-append-cases)
    assume x\in set (filter-min-append xs ys)
    moreover from assms(2) have rel x x by (rule reflpD)
    ultimately show ?thesis ..
qed
lemma filter-min-append-minimal:
    assumes }\\mp@subsup{x}{}{\prime}\mp@subsup{y}{}{\prime}.\mp@subsup{x}{}{\prime}\in\mathrm{ set xs # y y
    and }\bigwedge\mp@subsup{x}{}{\prime}\mp@subsup{y}{}{\prime}.\mp@subsup{x}{}{\prime}\in\mathrm{ set ys ఋ y'}\in\mathrm{ set ys # rel }\mp@subsup{x}{}{\prime}\mp@subsup{y}{}{\prime}\Longrightarrow\mp@subsup{x}{}{\prime}=\mp@subsup{y}{}{\prime
        and }x\in\mathrm{ set (filter-min-append xs ys) and y set (filter-min-append xs ys)
and rel x y
    shows }x=
proof -
    define P where P}=(\lambdazs.\lambdaa.\neg(\existsz\in\mathrm{ set zs. rel z a ))
    define ys1 where ys1 = filter (P xs) ys
    from assms(3) have }x\in\mathrm{ set xs }\cup\mathrm{ set ys1
    by (auto simp: filter-min-append-def P-def ys1-def)
    moreover from assms(4) have y fet (filter (P ys1) xs) \cup set ys1
    by (simp add: filter-min-append-def P-def ys1-def)
    ultimately show ?thesis
    proof (elim UnE)
    assume x f set xs
    assume y fet (filter (P ys1) xs)
```

```
    hence y\in set xs by simp
    with }\langlex\in\mathrm{ set xs` show ?thesis using assms(5) by (rule assms(1))
    next
    assume y f set ys1
    hence }\bigwedgez.z\in\mathrm{ set xs }\Longrightarrow\neg\mathrm{ rel z y by (simp add: ys1-def P-def)
    moreover assume x\in set xs
    ultimately have }\neg\mathrm{ rel x y by blast
    thus ?thesis using <rel x y>..
    next
    assume y f set (filter (P ys1) xs)
    hence }\z.z\in\mathrm{ set ys1 }\Longrightarrow\neg\mathrm{ rel zy by (simp add:P-def)
    moreover assume x set ys1
    ultimately have }\neg\mathrm{ rel x y by blast
    thus ?thesis using <rel x y> ..
next
    assume }x\in\mathrm{ set ys1 and }y\in\mathrm{ set ys1
    hence }x\in\mathrm{ set ys and }y\in\mathrm{ set ys by (simp-all add: ys1-def)
    thus ?thesis using assms(5) by (rule assms(2))
    qed
qed
lemma filter-min-append-distinct:
    assumes reflp rel and distinct xs and distinct ys
    shows distinct (filter-min-append xs ys)
proof -
    define P where P}=(\lambdazs.\lambdaa.\neg(\existsz\in\mathrm{ set zs. rel z a )}
    define ys1 where ys1 = filter ( }Pxs\mathrm{ ) ys
    from assms(2) have distinct (filter (P ys1) xs) by simp
    moreover from assms(3) have distinct ys1 by (simp add: ys1-def)
    moreover have set (filter (P ys1) xs) \cap set ys1 = {}
    proof (simp add: set-eq-iff, intro allI impI notI)
        fix }
        assume P ys1 x
        hence }\z.z\in\mathrm{ set ys1 }\Longrightarrow\neg\operatorname{rel zx by (simp add: P-def)
        moreover assume x set ys1
        ultimately have }\neg\mathrm{ rel x x by blast
        moreover from assms(1) have rel x x by (rule reflpD)
        ultimately show False ..
    qed
    ultimately show ?thesis by (simp add: filter-min-append-def ys1-def P-def)
qed
end
end
```


## 3 Properties of Binary Relations

theory Confluence
imports Abstract-Rewriting.Abstract-Rewriting Open-Induction.Restricted-Predicates begin

This theory formalizes some general properties of binary relations, in particular a very weak sufficient condition for a relation to be Church-Rosser.

## 3.1 wfp-on

lemma wfp-on-imp-wfP:
assumes wfp-on r $A$
shows $w f P(\lambda x y . r x y \wedge x \in A \wedge y \in A)($ is $w f P$ ? $r)$
proof (simp add: wfP-def wf-def, intro allI impI)
fix $P x$
assume $\forall x$. $(\forall y . r y x \wedge y \in A \wedge x \in A \longrightarrow P y) \longrightarrow P x$
hence $*: \bigwedge x$. $(\bigwedge y . x \in A \Longrightarrow y \in A \Longrightarrow r y x \Longrightarrow P y) \Longrightarrow P x$ by blast
from assms have $* *: \bigwedge a . a \in A \Longrightarrow(\bigwedge x . x \in A \Longrightarrow(\bigwedge y . y \in A \Longrightarrow r y x \Longrightarrow$ $P y) \Longrightarrow P x) \Longrightarrow P a$
by (rule wfp-on-induct) blast+
show $P x$
proof (cases $x \in A$ )
case True
from this * show ?thesis by (rule **)
next
case False
show ?thesis
proof (rule *)
fix $y$
assume $x \in A$
with False show P y ..
qed
qed
qed
lemma wfp-onI-min:
assumes $\bigwedge x Q . x \in Q \Longrightarrow Q \subseteq A \Longrightarrow \exists z \in Q . \forall y \in A . r y z \longrightarrow y \notin Q$
shows wfp-on $r A$
proof (intro inductive-on-imp-wfp-on minimal-imp-inductive-on allI impI)
fix $Q x$
assume $x \in Q \wedge Q \subseteq A$
hence $x \in Q$ and $Q \subseteq A$ by simp-all
hence $\exists z \in Q . \forall y \in A$. r y $z \longrightarrow y \notin Q$ by (rule assms)
then obtain $z$ where $z \in Q$ and $1: \bigwedge y . y \in A \Longrightarrow r y z \Longrightarrow y \notin Q$ by blast
show $\exists z \in Q . \forall y . r y z \longrightarrow y \notin Q$
proof (intro bexI allI impI)
fix $y$
assume $r y z$
show $y \notin Q$
proof (cases $y \in A$ )
case True

```
        thus ?thesis using <r y z\rangle by (rule 1)
        next
            case False
            with }\langleQ\subseteqA\rangle\mathrm{ show ?thesis by blast
    qed
    qed fact
qed
lemma wfp-onE-min:
    assumes wfp-on r A and x\inQ and Q\subseteqA
    obtains z where z\inQ and \\y.ryz\Longrightarrowy\not\inQ
    using wfp-on-imp-minimal[OF assms(1)] assms(2, 3) by blast
lemma wfp-onI-chain: \neg(\existsf.\foralli.fi\inA\wedger(f(Suc i)) (fi))\Longrightarrowwfp-on r A
    by (simp add: wfp-on-def)
lemma finite-minimalE:
    assumes finite }A\mathrm{ and }A\not={}\mathrm{ and irreflp rel and transp rel
    obtains a where }a\inA\mathrm{ and }\b\mathrm{ . rel b a > b#A
    using assms(1, 2)
proof (induct arbitrary: thesis)
    case empty
    from empty(2) show ?case by simp
next
    case (insert a A)
    show ?case
    proof (cases A={})
        case True
        show ?thesis
        proof (rule insert(4))
        fix b
        assume rel b a
        with assms(3) show b\not\in insert a A by (auto simp: True irreflp-def)
    qed simp
    next
    case False
    with insert(3) obtain z}\mathrm{ where z 
    show ?thesis
    proof (cases rel a z)
        case True
        show ?thesis
        proof (rule insert(4))
            fix b
            assume rel b a
            with assms(4) have rel b z using <rel a z> by (rule transpD)
            hence }b\not\inA\mathrm{ by (rule *)
            moreover from «rel b a` assms(3) have b}\not=a\mathrm{ by (auto simp: irreflp-def)
            ultimately show b\not\in insert a A by simp
            qed simp
```

```
    next
            case False
            show ?thesis
            proof (rule insert(4))
                fix b
                    assume rel bz
            hence b}\not\inA\mathrm{ by (rule *)
            moreover from 〈rel b z` False have b}\not=a\mathrm{ by blast
            ultimately show b\not\in insert a A by simp
            next
            from}\langlez\inA\rangle\mathrm{ show z insert a A by simp
            qed
        qed
        qed
qed
lemma wfp-on-finite:
        assumes irreflp rel and transp rel and finite A
        shows wfp-on rel A
proof (rule wfp-onI-min)
    fix }x
    assume x G Q and Q\subseteqA
    from this(2) assms(3) have finite Q by (rule finite-subset)
    moreover from <x \inQ> have Q}={}\mathrm{ by blast
    ultimately obtain z where z\inQ and \y. rel y z\Longrightarrowy\not\inQ using assms(1,
2)
    by (rule finite-minimalE) blast
    thus }\existsz\inQ.\forally\inA.rel yz\longrightarrowy\not\inQ by blas
qed
```


### 3.2 Relations

locale relation $=$ fixes $r::^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infixl $\rightarrow 50$ )
begin
abbreviation $r t c::^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infixl $\rightarrow{ }^{*} 50$ ) where rtc a $b \equiv r^{* *} a b$
abbreviation $s c::^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infixl $\leftrightarrow 50$ )
where sc $a b \equiv a \rightarrow b \vee b \rightarrow a$
definition is-final::' $a \Rightarrow$ bool where
$i s$-final $a \equiv \neg(\exists b . r a b)$
definition srtc::' $a \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infixl $\leftrightarrow^{*} 50$ ) where
srtc $a b \equiv s c^{* *} a b$
definition $c s::^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infixl $\downarrow^{*} 50$ ) where
$c s a b \equiv\left(\exists s .\left(a \rightarrow^{*} s\right) \wedge\left(b \rightarrow^{*} s\right)\right)$

```
definition is-confluent-on :: 'a set }=>\mathrm{ bool
    where is-confluent-on }A\longleftrightarrow(\foralla\inA.\forallb1b2. (a ->* b1 ^a ->** b2) \longrightarrowb1 \downarrow**
b2)
definition is-confluent :: bool
    where is-confluent \equivis-confluent-on UNIV
definition is-loc-confluent :: bool
    where is-loc-confluent \equiv(\foralla b1 b2. (a->b1^a->b2) \longrightarrowb1 \downarrow* b2)
definition is-ChurchRosser :: bool
    where is-ChurchRosser }\equiv(\forallab.a\leftrightarrow\mp@subsup{\leftrightarrow}{}{*}b\longrightarrowa\mp@subsup{\downarrow}{}{*}b
definition dw-closed :: 'a set }=>\mathrm{ bool
    where dw-closed }A\longleftrightarrow(\foralla\inA.\forallb.a->b\longrightarrowb\inA
lemma dw-closedI [intro]:
    assumes \ab. a\inA\Longrightarrowa->b\Longrightarrowb\inA
    shows dw-closed A
    unfolding dw-closed-def using assms by auto
lemma dw-closedD:
    assumes dw-closed A and a\inA and a->b
    shows }b\in
    using assms unfolding dw-closed-def by auto
lemma dw-closed-rtrancl:
    assumes dw-closed A and a\inA and a ->*
    shows }b\in
    using assms(3)
proof (induct b)
    case base
    from assms(2) show ?case.
next
    case (step y z)
    from assms(1) step(3) step(2) show ?case by (rule dw-closedD)
qed
lemma dw-closed-empty:dw-closed {}
    by (rule, simp)
lemma dw-closed-UNIV:dw-closed UNIV
    by (rule, intro UNIV-I)
```


### 3.3 Setup for Connection to Theory Abstract-Rewriting.Abstract-Rewriting

 abbreviation (input) relset::(' $a *$ ' $a$ ) set where$$
\text { relset } \equiv\{(x, y) . x \rightarrow y\}
$$

```
lemma rtc-rtranclI:
    assumes a }\mp@subsup{->}{}{*}
    shows (a,b) \in relset*
using assms by (simp only: Enum.rtranclp-rtrancl-eq)
lemma final-NF:(is-final a)}=(a\inNF relset) 
unfolding is-final-def NF-def by simp
lemma sc-symcl: (a\leftrightarrowb) =((a,b)\in relset }\leftrightarrow
by simp
lemma srtc-conversion: (a ↔* b) =((a,b)\in relset }\mp@subsup{}{}{\leftrightarrow*}
proof -
    have {(a,b). (a,b) \in{(x,y).x->y}\leftrightarrow} ={(a,b).a->b}\leftrightarrow by auto
    thus ?thesis unfolding srtc-def conversion-def sc-symcl Enum.rtranclp-rtrancl-eq
by simp
qed
lemma cs-join: ( }a\mp@subsup{\downarrow}{}{*}b)=((a,b)\in\mp@subsup{relset }{}{\downarrow}
    unfolding cs-def join-def by (auto simp add: Enum.rtranclp-rtrancl-eq rtrancl-converse)
lemma confluent-CR: is-confluent =CR relset
    by (auto simp add: is-confluent-def is-confluent-on-def CR-defs Enum.rtranclp-rtrancl-eq
cs-join)
lemma ChurchRosser-conversion: is-ChurchRosser =(relset }\mp@subsup{}{}{\leftrightarrow*}\subseteq\mp@subsup{rrelset}{}{\downarrow}
    by (auto simp add: is-ChurchRosser-def cs-join srtc-conversion)
lemma loc-confluent-WCR:
    shows is-loc-confluent =WCR relset
unfolding is-loc-confluent-def WCR-defs by (auto simp add: cs-join)
lemma wf-converse:
    shows (wfP r^--1) = (wf (relset }\mp@subsup{}{}{-1})
unfolding wfP-def converse-def by simp
lemma wf-SN:
    shows (wfPr^--1) = (SN relset )
unfolding wf-converse wf-iff-no-infinite-down-chain SN-on-def by auto
```


### 3.4 Simple Lemmas

```
lemma rtrancl-is-final:
    assumes a ->** b and is-final a
    shows }a=
proof -
    from rtranclpD[OF<a ->* b〉] show ?thesis
    proof
        assume }a\not=b\wedge(->\mp@subsup{)}{}{++}a
```

```
        hence ( }->\mp@subsup{)}{}{++}\mathrm{ a b by simp
        from <is-final a〉 final-NF have a N NF relset by simp
        from NF-no-trancl-step[OF this] have (a,b)\not\in{(x,y).x->y}+..
        thus ?thesis using <( }->\mp@subsup{)}{}{++}\mathrm{ a b> unfolding tranclp-unfold ..
        qed
qed
lemma cs-refl:
    shows }x\mp@subsup{\downarrow}{}{*}
unfolding cs-def
proof
    show }x\mp@subsup{->}{}{*}x\wedgex\mp@subsup{->}{}{*}x\mathrm{ by simp
qed
lemma cs-sym:
    assumes }x\mp@subsup{\downarrow}{}{*}
    shows }y\mp@subsup{\downarrow}{}{*}
using assms unfolding cs-def
proof
    fix z
    assume a: }x\mp@subsup{->}{}{*}z\wedgey\mp@subsup{->}{}{*}
    show \exists s.y ->* s^x 険s
    proof
        from }a\mathrm{ show }y\mp@subsup{->}{}{*}z\wedgex\mp@subsup{->}{}{*}z\mathrm{ by simp
    qed
qed
lemma rtc-implies-cs:
    assumes }x\mp@subsup{->}{}{*}
    shows }x\mp@subsup{\downarrow}{}{*}
proof -
    from joinI-left[OF rtc-rtranclI[OF assms]] cs-join show ?thesis by simp
qed
lemma rtc-implies-srtc:
    assumes }a->\mp@subsup{->}{}{*}
    shows }a\mp@subsup{\leftrightarrow}{}{*}
proof -
    from conversionI'[OF rtc-rtranclI[OF assms]] srtc-conversion show ?thesis by
simp
qed
lemma srtc-symmetric:
    assumes }a\mp@subsup{\leftrightarrow}{}{*}
    shows b}\mp@subsup{\leftrightarrow}{}{*}
proof -
    from symD[OF conversion-sym[of relset], of a b] assms srtc-conversion show
?thesis by simp
qed
```

```
lemma srtc-transitive
    assumes }a\mp@subsup{\leftrightarrow}{}{*}b\mathrm{ and b ↔*}
    shows }a\mp@subsup{\leftrightarrow}{}{*}
proof -
    from rtranclp-trans[of (\leftrightarrow) a b c] assms show }a\mp@subsup{\leftrightarrow}{}{*}c\mathrm{ unfolding srtc-def .
qed
lemma cs-implies-srtc:
    assumes }a\mp@subsup{\downarrow}{}{*}
    shows }a\mp@subsup{\leftrightarrow}{}{*}
proof -
    from assms cs-join have (a,b)\in relset }\downarrow\mathrm{ by simp
    hence (a,b) \in relset }->*\mathrm{ using join-imp-conversion by auto
    thus ?thesis using srtc-conversion by simp
qed
lemma confluence-equiv-ChurchRosser: is-confluent = is-ChurchRosser
    by (simp only:ChurchRosser-conversion confluent-CR, fact CR-iff-conversion-imp-join)
corollary confluence-implies-ChurchRosser:
    assumes is-confluent
    shows is-ChurchRosser
    using assms by (simp only: confluence-equiv-ChurchRosser)
lemma ChurchRosser-unique-final:
    assumes is-ChurchRosser and a ->* b1 and a ->* b2 and is-final b1 and
is-final b2
    shows b1 = b2
proof -
    from 〈is-ChurchRosser` confluence-equiv-ChurchRosser confluent-CR have CR
relset by simp
    from CR-imp-UNF[OF this] assms show ?thesis unfolding UNF-defs normal-
izability-def
    by (auto simp add: Enum.rtranclp-rtrancl-eq final-NF)
qed
lemma wf-on-imp-nf-ex:
    assumes wfp-on (( }->\mp@subsup{)}{}{-1-1})A\mathrm{ and dw-closed A and a 
    obtains b}\mathrm{ where }a->\mp@subsup{->}{}{*}b\mathrm{ and is-final b
proof -
    let ?A = {b\inA. a 玍 b}
    note assms(1)
    moreover from assms(3) have a\in?A by simp
    moreover have ?A }\subseteqA\mathrm{ by auto
    ultimately show ?thesis
    proof (rule wfp-onE-min)
        fix z
    assume z & ?A and }\bigwedgey.(->\mp@subsup{)}{}{-1-1}yz\Longrightarrowy\not\in?
```

```
    from this(2) have *: \bigwedgey.z->y\Longrightarrowy\not\in?A by simp
    from }\langlez\in?A\rangle have z\inA and a->** z by simp-all
    show thesis
    proof (rule, fact)
        show is-final z unfolding is-final-def
        proof
            assume }\existsy.z->
            then obtain }y\mathrm{ where z}->y\mathrm{ ..
            hence y}\not\in?A\mathrm{ by (rule *)
            moreover from assms(2) {z\inA\rangle\langlez->y\rangle have y\inA by (rule dw-closedD)
            ultimately have }\neg(a\mp@subsup{->}{}{*}y)\mathrm{ by simp
            with rtranclp-trans[OF <a 旃 z\rangle, of y]}\langlez->y\rangle\mathrm{ show False by auto
        qed
    qed
    qed
qed
lemma unique-nf-imp-confluence-on:
    assumes major: \a b1 b2. }a\inA\Longrightarrow(a\mp@subsup{->}{}{*}b1)\Longrightarrow(a\mp@subsup{->}{}{*}b2)\Longrightarrowis-final b1
ls-final b2 \Longrightarrowb1 = b2
    and wf:wfp-on ((->)
    shows is-confluent-on A
    unfolding is-confluent-on-def
proof (intro ballI allI impI)
    fix a b1 b2
    assume a ->* b1 ^a ->* b2
    hence }a\mp@subsup{->}{}{*}b1\mathrm{ and }a\mp@subsup{->}{}{*}\mathrm{ b2 by simp-all
    assume a\inA
    from dw this }{a\mp@subsup{->}{}{*}b1\rangle\mathrm{ have b1 }\inA\mathrm{ by (rule dw-closed-rtrancl)
    from wf dw this obtain c1 where b1 梠c1 and is-final c1 by (rule wf-on-imp-nf-ex)
    from dw <a \inA\rangle\langlea ->* b2\rangle have b2 \inA by (rule dw-closed-rtrancl)
    from wf dw this obtain c2 where b2 ->* c2 and is-final c2 by (rule wf-on-imp-nf-ex)
    have c1 = c2
            by (rule major, fact, rule rtranclp-trans[OF <a ->** b1>], fact, rule rtran-
clp-trans[OF <a ->* b2`], fact+)
    show b1 \downarrow* b2 unfolding cs-def
    proof (intro exI, intro conjI)
        show b1 }\mp@subsup{->}{}{*}\mathrm{ c1 by fact
    next
        show b2 }\mp@subsup{->}{}{*}c1\mathrm{ unfolding <c1 = c2` by fact
    qed
qed
corollary wf-imp-nf-ex:
    assumes wfP ((->\mp@subsup{)}{}{-1-1})
    obtains b where a ->**}b\mathrm{ and is-final b
proof -
    from assms have wfp-on (r--1) UNIV by simp
    moreover note dw-closed-UNIV
```

```
    moreover have a\inUNIV ..
    ultimately obtain b}\mathrm{ where }a->**\mathrm{ and is-final b by (rule wf-on-imp-nf-ex)
    thus ?thesis ..
qed
corollary unique-nf-imp-confluence:
    assumes \a b1 b2. (a ->* b1)\Longrightarrow(a 椟 b2) \Longrightarrow is-final b1 \Longrightarrow is-final b2
"b1 = b2
    and wfP(( }->\mp@subsup{)}{}{-1-1}
    shows is-confluent
    unfolding is-confluent-def
    by (rule unique-nf-imp-confluence-on, erule assms(1), assumption+, simp add:
assms(2), fact dw-closed-UNIV)
end
```


### 3.5 Advanced Results and the Generalized Newman Lemma

definition relbelow-on $::$ 'a set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow{ }^{\prime} a \Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow b o o l\right) \Rightarrow$ ( ${ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow b o o l$ )
where relbelow-on $A$ ord $z$ rel $a b \equiv(a \in A \wedge b \in A \wedge$ rel $a b \wedge$ ord $a z \wedge$ ord $b$ z)
definition cbelow-on-1 :: 'a set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow{ }^{\prime} a \Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow$ ( ${ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow b o o l$ )
where cbelow-on-1 A ord z rel $\equiv(\text { relbelow-on } A \text { ord } z \text { rel })^{++}$
definition cbelow-on :: 'a set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow{ }^{\prime} a \Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow$ ( ${ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow b o o l$ )
where cbelow-on $A$ ord $z$ rel $a b \equiv(a=b \wedge b \in A \wedge$ ord $b z) \vee$ cbelow-on-1 $A$ ord $z$ rel $a b$

Note that cbelow-on cannot be defined as the reflexive-transitive closure of relbelow-on, since it is in general not reflexive!
definition is-loc-connective-on :: 'a set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow b o o l\right) \Rightarrow\left({ }^{\prime} a \Rightarrow^{\prime} a \Rightarrow\right.$ bool $)$ $\Rightarrow$ bool
where is-loc-connective-on $A$ ord $r \longleftrightarrow(\forall a \in A . \forall b 1$ b2. rab1 $\wedge r a b 2 \longrightarrow$ cbelow-on A ord a (relation.sc r) b1 b2)

Note that wfp-on is not the same as $S N$-on, since in the definition of $S N$-on only the first element of the chain must be in the set.

```
lemma cbelow-on-first-below:
    assumes cbelow-on A ord z rel a b
    shows ord a z
    using assms unfolding cbelow-on-def
proof
    assume cbelow-on-1 A ord z rel a b
    thus ord a z unfolding cbelow-on-1-def by (induct rule: tranclp-induct, simp
add: relbelow-on-def)
```

```
qed simp
lemma cbelow-on-second-below:
    assumes cbelow-on A ord z rel a b
    shows ord b z
    using assms unfolding cbelow-on-def
proof
    assume cbelow-on-1 A ord z rel a b
    thus ord b z unfolding cbelow-on-1-def
        by (induct rule: tranclp-induct, simp-all add: relbelow-on-def)
qed simp
lemma cbelow-on-first-in:
    assumes cbelow-on A ord z rel a b
    shows }a\in
    using assms unfolding cbelow-on-def
proof
    assume cbelow-on-1 A ord z rel a b
    thus ?thesis unfolding cbelow-on-1-def by (induct rule: tranclp-induct, simp
add: relbelow-on-def)
qed simp
lemma cbelow-on-second-in:
    assumes cbelow-on A ord z rel a b
    shows b\inA
    using assms unfolding cbelow-on-def
proof
    assume cbelow-on-1 A ord z rel a b
    thus ?thesis unfolding cbelow-on-1-def
        by (induct rule: tranclp-induct, simp-all add: relbelow-on-def)
qed simp
lemma cbelow-on-intro [intro]:
    assumes main: cbelow-on A ord z rel a b and c\inA and rel b c and ord c z
    shows cbelow-on A ord z rel a c
proof -
    from main have b\inA by (rule cbelow-on-second-in)
    from main show ?thesis unfolding cbelow-on-def
    proof (intro disjI2)
    assume cases: ( }a=b\wedgeb\inA\wedge\mathrm{ ord b z) V cbelow-on-1 A ord z rel a b
```



```
                have bc: relbelow-on A ord z rel b c by (simp add:relbelow-on-def)
    from cases show cbelow-on-1 A ord z rel a c
    proof
                assume }a=b\wedgeb\inA\wedge\mathrm{ ord b z
                from this bc have relbelow-on A ord z rel a c by simp
                thus?thesis by (simp add: cbelow-on-1-def)
            next
                    assume cbelow-on-1 A ord z rel a b
```

from this bc show ?thesis unfolding cbelow-on-1-def by (rule tranclp.intros(2)) qed
qed
qed
lemma cbelow-on-induct [consumes 1, case-names base step]:
assumes a: cbelow-on $A$ ord $z$ rel $a b$
and base: $a \in A \Longrightarrow$ ord $a z \Longrightarrow P a$
and ind: $\bigwedge b c$. $[\mid$ cbelow-on $A$ ord $z$ rel $a b ;$ rel $b c ; c \in A ;$ ord $c z ; P b \mid]==>$ Pc
shows $P b$
using a unfolding cbelow-on-def
proof
assume $a=b \wedge b \in A \wedge$ ord $b z$
from this base show $P b$ by simp
next
assume cbelow-on-1 $A$ ord $z$ rel $a b$
thus $P b$ unfolding cbelow-on-1-def
proof (induct $x \equiv a b$ )
fix $b$
assume relbelow-on $A$ ord $z$ rel a $b$
hence rel $a b$ and $a \in A$ and $b \in A$ and ord $a z$ and ord $b z$
by (simp-all add: relbelow-on-def)
hence cbelow-on $A$ ord $z$ rel a a by (simp add: cbelow-on-def)
from this $\langle$ rel $a b\rangle\langle b \in A\rangle\langle o r d b z\rangle$ base $[O F\langle a \in A\rangle\langle\operatorname{ord} a z\rangle]$ show $P b$ by (rule ind)
next
fix $b c$
assume $I H:(\text { relbelow-on } A \text { ord } z \text { rel })^{++} a b$ and $P b$ and relbelow-on $A$ ord $z$ rel b c
hence rel $b c$ and $b \in A$ and $c \in A$ and ord $b z$ and ord $c z$
by (simp-all add: relbelow-on-def)
from $I H$ have cbelow-on $A$ ord $z$ rel a by (simp add: cbelow-on-def cbe-low-on-1-def)
from this $\langle$ rel $b c\rangle\langle c \in A\rangle\langle o r d c z\rangle\langle P b\rangle$ show $P c$ by (rule ind)
qed
qed
lemma cbelow-on-symmetric:
assumes main: cbelow-on $A$ ord $z$ rel $a b$ and symp rel
shows cbelow-on $A$ ord $z$ rel $b$ a
using main unfolding cbelow-on-def
proof
assume $a 1: a=b \wedge b \in A \wedge$ ord $b z$
show $b=a \wedge a \in A \wedge$ ord $a z \vee$ cbelow-on-1 $A$ ord $z$ rel $b a$
proof
from $a 1$ show $b=a \wedge a \in A \wedge$ ord $a z$ by simp
qed
next

```
    assume a2: cbelow-on-1 A ord z rel a b
    show b=a\wedgea\inA\wedge ord a z\vee cbelow-on-1 A ord z rel b a
    proof (rule disjIL)
    from <symp rel` have symp (relbelow-on A ord z rel) unfolding symp-def
    proof (intro allI impI)
            fix }x
            assume rel-sym: }\forallxy.rel xy\longrightarrow rel y x
            assume relbelow-on A ord z rel x y
            hence rel x y and x\inA and y\inA and ord xz and ord yz
            by (simp-all add: relbelow-on-def)
            show relbelow-on A ord z rel y x unfolding relbelow-on-def
            proof (intro conjI)
                    from rel-sym〈rel x y` show rel y }x\mathrm{ by simp
            qed fact+
    qed
    from sym-trancl[to-pred, OF this] a2 show cbelow-on-1 A ord z rel b a
        by (simp add: symp-def cbelow-on-1-def)
    qed
qed
lemma cbelow-on-transitive:
    assumes cbelow-on A ord z rel a b and cbelow-on A ord z rel b c
    shows cbelow-on A ord z rel a c
proof (induct rule: cbelow-on-induct[OF <cbelow-on A ord z rel b c>])
    from 〈cbelow-on A ord z rel a b〉 show cbelow-on A ord z rel a b .
next
    fix c0 c
    assume cbelow-on A ord z rel b c0 and rel c0 c and c\inA and ord c z and
cbelow-on A ord z rel a c0
    show cbelow-on A ord z rel a c by (rule, fact+)
qed
lemma cbelow-on-mono:
    assumes cbelow-on A ord z rel a b and A\subseteqB
    shows cbelow-on B ord z rel a b
    using assms(1)
proof (induct rule: cbelow-on-induct)
    case base
    show ?case by (simp add: cbelow-on-def, intro disjI1 conjI, rule, fact+)
next
    case (step b c)
    from step(3) assms(2) have c\inB ..
    from step(5) this step(2) step (4) show ?case ..
qed
locale relation-order = relation +
    fixes ord::' }a>>'' a=> boo
    fixes A::'a set
    assumes trans: ord x y cord y z\Longrightarrow ord x z
```

```
    assumes wf: wfp-on ord \(A\)
    assumes refines: \((\rightarrow) \leq\) ord \(^{-1-1}\)
begin
lemma relation-refines:
    assumes \(a \rightarrow b\)
    shows ord \(b a\)
    using refines assms by auto
lemma relation-wf: wfp-on \((\rightarrow)^{-1-1} A\)
    using subset-refl-wf
proof (rule wfp-on-mono)
    fix \(x y\)
    assume \((\rightarrow)^{-1-1} x y\)
    hence \(y \rightarrow x\) by simp
    with refines have (ord \()^{-1-1}\) y \(x\)..
    thus ord \(x y\) by simp
qed
lemma rtc-implies-cbelow-on:
    assumes dw-closed \(A\) and main: \(a \rightarrow^{*} b\) and \(a \in A\) and ord \(a c\)
    shows cbelow-on \(A\) ord \(c(\leftrightarrow) a b\)
    using main
proof (induct rule: rtranclp-induct)
    from \(\operatorname{assms}(3) \operatorname{assms}(4)\) show cbelow-on A ord \(c(\leftrightarrow)\) a a by (simp add: cbe-
low-on-def)
next
    fix \(b 0 b\)
    assume \(a \rightarrow^{*} b 0\) and \(b 0 \rightarrow b\) and \(I H:\) cbelow-on \(A\) ord \(c(\leftrightarrow) a b 0\)
    from \(\operatorname{assms}(1) \operatorname{assms}(3)\left\langle a \rightarrow^{*} b 0\right\rangle\) have \(b 0 \in A\) by (rule dw-closed-rtrancl)
    from \(\operatorname{assms}(1)\) this \(\langle b 0 \rightarrow b\rangle\) have \(b \in A\) by (rule dw-closed \(D\) )
    show cbelow-on \(A\) ord \(c(\leftrightarrow) a b\)
    proof
        from \(\langle b 0 \rightarrow b\rangle\) show \(b 0 \leftrightarrow b\) by simp
    next
    from relation-refines \([O F\langle b 0 \rightarrow b\rangle]\) cbelow-on-second-below \([O F I H]\) show ord
b c by (rule trans)
    qed fact+
qed
lemma cs-implies-cbelow-on:
    assumes \(d w\)-closed \(A\) and \(a \downarrow^{*} b\) and \(a \in A\) and \(b \in A\) and ord \(a c\) and ord \(b\)
c
    shows cbelow-on A ord \(c(\leftrightarrow) a b\)
proof -
    from \(\left\langle a \downarrow^{*} b\right\rangle\) obtain \(s\) where \(a \rightarrow^{*} s\) and \(b \rightarrow^{*} s\) unfolding \(c s\)-def by auto
    have sym: symp \((\leftrightarrow)\) unfolding symp-def
    proof (intro allI, intro impI)
    fix \(x y\)
```

```
        assume }x\leftrightarrow
        thus }y\leftrightarrowx\mathrm{ by auto
    qed
    from assms(1)\langlea ->* s`assms(3) assms(5) have cbelow-on A ord c (\leftrightarrow) a s
    by (rule rtc-implies-cbelow-on)
    also have cbelow-on A ord c(\leftrightarrow)sb
    proof (rule cbelow-on-symmetric)
    from assms(1)\langleb ->* s\rangle assms(4) assms(6) show cbelow-on A ord c ( }\leftrightarrow)b
        by (rule rtc-implies-cbelow-on)
    qed fact
    finally(cbelow-on-transitive) show ?thesis .
qed
The generalized Newman lemma, taken from [17]:
lemma loc-connectivity-implies-confluence:
assumes is-loc-connective-on \(A\) ord \((\rightarrow)\) and dw-closed \(A\)
shows is-confluent-on \(A\)
using assms(1) unfolding is-loc-connective-on-def is-confluent-on-def
proof (intro ballI allI impI)
fix \(z x y::^{\prime} a\)
assume \(\forall a \in A . \forall b 1\) b2. \(a \rightarrow b 1 \wedge a \rightarrow b 2 \longrightarrow\) cbelow-on \(A\) ord \(a(\leftrightarrow) b 1\) b2
hence \(A: \bigwedge a b 1\) b2. \(a \in A \Longrightarrow a \rightarrow b 1 \Longrightarrow a \rightarrow b 2 \Longrightarrow\) cbelow-on \(A\) ord \(a(\leftrightarrow)\)
b1 b2 by \(\operatorname{simp}\)
assume \(z \in A\) and \(z \rightarrow^{*} x \wedge z \rightarrow^{*} y\)
with \(w f\) show \(x \downarrow^{*} y\)
proof (induct \(z\) arbitrary: \(x\) y rule: wfp-on-induct)
fix \(z x y::^{\prime} a\)
assume \(I H: \bigwedge z 0 x 0 y 0 . z 0 \in A \Longrightarrow\) ord \(z 0 z \Longrightarrow z 0 \rightarrow^{*} x 0 \wedge z 0 \rightarrow^{*} y 0 \Longrightarrow\)
\(x 0 \downarrow^{*} y 0\)
and \(z \rightarrow^{*} x \wedge z \rightarrow^{*} y\)
hence \(z \rightarrow^{*} x\) and \(z \rightarrow^{*} y\) by auto
assume \(z \in A\)
from converse-rtranclpE[OF \(\left\langle z \rightarrow^{*} x\right\rangle\) obtain \(x 1\) where \(x=z \vee(z \rightarrow x 1 \wedge\)
\(x 1 \rightarrow^{*} x\) ) by auto
thus \(x \downarrow^{*} y\)
proof
assume \(x=z\)
show ?thesis unfolding cs-def proof
from \(\langle x=z\rangle\left\langle z \rightarrow^{*} y\right\rangle\) show \(x \rightarrow^{*} y \wedge y \rightarrow^{*} y\) by simp
qed
next
assume \(z \rightarrow x 1 \wedge x 1 \rightarrow^{*} x\)
hence \(z \rightarrow x 1\) and \(x 1 \rightarrow^{*} x\) by auto
from \(\operatorname{assms}(2)\langle z \in A\rangle\) this(1) have \(x 1 \in A\) by (rule dw-closedD)
from converse-rtranclp \(E\left[O F\left\langle z \rightarrow^{*} y\right\rangle\right]\) obtain \(y 1\) where \(y=z \vee(z \rightarrow y 1\)
\(\left.\wedge y 1 \rightarrow^{*} y\right)\) by auto
thus ?thesis
proof
```

```
    assume \(y=z\)
    show ?thesis unfolding \(c s\)-def
    proof
        from \(\langle y=z\rangle\left\langle z \rightarrow^{*} x\right\rangle\) show \(x \rightarrow^{*} x \wedge y \rightarrow^{*} x\) by simp
    qed
    next
        assume \(z \rightarrow y 1 \wedge y 1 \rightarrow^{*} y\)
    hence \(z \rightarrow y 1\) and \(y 1 \rightarrow^{*} y\) by auto
    from \(\operatorname{assms}(2)\langle z \in A\rangle\) this(1) have \(y 1 \in A\) by (rule dw-closedD)
    have \(x 1 \downarrow^{*} y 1\)
    proof (induct rule: cbelow-on-induct[OF \(A[O F\langle z \in A\rangle\langle z \rightarrow x 1\rangle\langle z \rightarrow\)
y1 >]])
            from \(c s\)-refl[of \(x 1]\) show \(x 1 \downarrow^{*} x 1\).
    next
        fix \(b c\)
        assume cbelow-on \(A\) ord \(z(\leftrightarrow) x 1 b\) and \(b \leftrightarrow c\) and \(c \in A\) and ord \(c z\)
and \(x 1 \downarrow^{*} b\)
    from this(1) have \(b \in A\) by (rule cbelow-on-second-in)
    from \(\left\langle x 1 \downarrow^{*} b\right\rangle\) obtain \(w 1\) where \(x 1 \rightarrow^{*} w 1\) and \(b \rightarrow^{*} w 1\) unfolding
cs-def by auto
    from \(\langle b \leftrightarrow c\rangle\) show \(x 1 \downarrow^{*} c\)
    proof
            assume \(b \rightarrow c\)
            hence \(b \rightarrow^{*} c\) by simp
                from 〈cbelow-on \(A\) ord \(z(\leftrightarrow) x 1 b\rangle\) have ord \(b z\) by (rule cbe-
low-on-second-below)
            from \(I H\left[O F\langle b \in A\rangle\right.\) this] \(\left\langle b \rightarrow^{*} c\right\rangle\left\langle b \rightarrow^{*} w 1\right\rangle\) have \(c \downarrow^{*} w 1\) by simp
            then obtain \(w 2\) where \(c \rightarrow^{*} w 2\) and \(w 1 \rightarrow^{*} w 2\) unfolding \(c s\)-def by
auto
            show ?thesis unfolding cs-def
            proof
                    from rtranclp-trans \(\left[O F\left\langle x 1 \rightarrow^{*} w 1\right\rangle\left\langle w 1 \rightarrow^{*} w 2\right\rangle\right]\left\langle c \rightarrow^{*} w 2\right\rangle\)
                    show \(x 1 \rightarrow^{*} w 2 \wedge c \rightarrow^{*} w 2\) by \(\operatorname{simp}\)
            qed
        next
            assume \(c \rightarrow b\)
            hence \(c \rightarrow^{*} b\) by simp
            show ?thesis unfolding cs-def
            proof
                from rtranclp-trans \(\left[O F\left\langle c \rightarrow^{*} b\right\rangle\left\langle b \rightarrow^{*} w 1\right\rangle\right]\left\langle x 1 \rightarrow^{*} w 1\right\rangle\)
                    show \(x 1 \rightarrow^{*} w 1 \wedge c \rightarrow^{*} w 1\) by simp
            qed
        qed
    qed
    then obtain \(w 1\) where \(x 1 \rightarrow^{*} w 1\) and \(y 1 \rightarrow^{*} w 1\) unfolding \(c s\)-def by
auto
    from \(I H[O F\langle x 1 \in A\rangle\) relation-refines \([O F\langle z \rightarrow x 1\rangle]]\left\langle x 1 \rightarrow^{*} x\right\rangle\left\langle x 1 \rightarrow^{*}\right.\)
w1 >
    have \(x \downarrow^{*} w 1\) by simp
```

```
            then obtain v where x ->** v and w1 ->** unfolding cs-def by auto
            from IH[OF <y1 \in A relation-refines[OF <z->y1\rangle]]
```



```
            have }v\mp@subsup{\downarrow}{}{*}y\mathrm{ by simp
            then obtain w where v ->** w and y ->** unfolding cs-def by auto
            show ?thesis unfolding cs-def
            proof
```



```
** w by simp
            qed
            qed
    qed
    qed
qed
end
theorem loc-connectivity-equiv-ChurchRosser:
    assumes relation-order r ord UNIV
    shows relation.is-ChurchRosser r = is-loc-connective-on UNIV ord r
proof
    assume relation.is-ChurchRosser r
    show is-loc-connective-on UNIV ord r unfolding is-loc-connective-on-def
    proof (intro ballI allI impI)
            fix a b1 b2
    assume rab1^rab2
    hence r a b1 and r a b2 by simp-all
    hence r r* a b1 and re* a b2 by simp-all
    from relation.rtc-implies-srtc[OF << ** a b1>] have relation.srtc r b1 a by (rule
relation.srtc-symmetric)
    from relation.srtc-transitive[OF this relation.rtc-implies-srtc[OF << [r* a b2>]]
have relation.srtc r b1 b2 .
    with 〈relation.is-ChurchRosser r> have relation.cs r b1 b2 by (simp add:
relation.is-ChurchRosser-def)
    from relation-order.cs-implies-cbelow-on[OF assms relation.dw-closed-UNIV
this]
            relation-order.relation-refines[OF assms, of a] <r a b1\rangle\langler a b2\rangle
            show cbelow-on UNIV ord a (relation.sc r) b1 b2 by simp
    qed
next
    assume is-loc-connective-on UNIV ord r
    from assms this relation.dw-closed-UNIV have relation.is-confluent-on r UNIV
            by (rule relation-order.loc-connectivity-implies-confluence)
    hence relation.is-confluent r by (simp only: relation.is-confluent-def)
    thus relation.is-ChurchRosser r by (simp add: relation.confluence-equiv-ChurchRosser)
qed
end
```


## 4 Polynomial Reduction

```
theory Reduction
imports Polynomials.MPoly-Type-Class-Ordered Confluence
begin
```

This theory formalizes the concept of reduction of polynomials by polynomials.

```
context ordered-term
begin
```



```
    where red-single p qft \longleftrightarrow(f\not=0\wedge lookup p (t\opluslt f)\not=0^
                        q=p- monom-mult ((lookup p (t\oplusltf))/lc f)tf)
```

definition red $::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.$ field $)$ set $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow$ bool
where red $F$ p $q \longleftrightarrow(\exists f \in F . \exists t$. red-single $p q f t)$
definition is-red :: ( $t \Rightarrow_{0}{ }^{\prime} b::$ field $)$ set $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow$ bool
where is-red $F a \longleftrightarrow \neg$ relation.is-final (red $F$ ) $a$

### 4.1 Basic Properties of Reduction

```
lemma red-setI:
    assumes f\inF and a: red-single p qft
    shows red Fpq
    unfolding red-def
proof
    from}\langlef\inF\rangle\mathrm{ show }f\inF
next
    from a show \existst. red-single pqft..
qed
lemma red-setE:
    assumes red Fpq
    obtains f}\mathrm{ and t where f}\inF\mathrm{ and red-single pqft
proof -
    from assms obtain f}\mathrm{ where f}\inF\mathrm{ and t: }\exists\mathrm{ t. red-single p qft unfolding
red-def by auto
    from t obtain t where red-single p qft..
    from }\langlef\inF\rangle\mathrm{ this show ?thesis ..
qed
lemma red-empty: ᄀ red {} pq
    by (rule, elim red-setE, simp)
lemma red-singleton-zero: }\neg\mathrm{ red {0} pq
    by (rule, elim red-setE, simp add: red-single-def)
```

```
lemma red-union: red (F\cupG) pq=(red Fpq\vee red G pq)
proof
    assume red (F\cupG) pq
    from red-setE[OF this] obtain ft where f\inF\cupG and r: red-single p qft.
    from}\langlef\inF\cupG\rangle\mathrm{ have }f\inF\veef\inG\mathrm{ by simp
    thus red Fpq\vee red Gpq
    proof
        assume f}\in
        show ?thesis by (intro disjI1, rule red-setI[OF<f}\inF>r]
    next
        assume f}\in
        show ?thesis by (intro disjI2, rule red-setI[OF<f}\inG`r]
    qed
next
    assume red Fpq\vee red Gpq
    thus red (F\cupG) pq
    proof
        assume red F pq
        from red-setE[OF this] obtain ft where f}\inF\mathrm{ and red-single p qft.
        show ?thesis by (intro red-setI[off --t], rule UnI1, rule <f \inF\rangle, fact)
    next
        assume red G p q
        from red-setE[OF this] obtain ft where f}\inG\mathrm{ and red-single p qft.
        show ?thesis by (intro red-setI[off --t], rule UnI2, rule <f \inG`, fact)
    qed
qed
lemma red-unionI1:
    assumes red Fpq
    shows red (F\cupG)pq
    unfolding red-union by (rule disjI1, fact)
lemma red-unionI2:
    assumes red Gpq
    shows red (F\cupG)pq
    unfolding red-union by (rule disjI2, fact)
lemma red-subset:
    assumes red G pq and G\subseteqF
    shows red Fpq
proof -
    from «G\subseteqF\rangle\mathrm{ obtain }H\mathrm{ where }F=G\cupH\mathrm{ by auto}
    show ?thesis unfolding }\langleF=G\cupH\rangle\mathrm{ by (rule red-unionI1, fact)
qed
lemma red-union-singleton-zero: red (F\cup{0})=red F
    by (intro ext, simp only: red-union red-singleton-zero, simp)
lemma red-minus-singleton-zero: red (F-{0}) = red F
```

```
    by (metis Un-Diff-cancel2 red-union-singleton-zero)
    lemma red-rtrancl-subset:
    assumes major: (red G)** p q and G\subseteqF
    shows (red F)** p q
    using major
proof (induct rule: rtranclp-induct)
    show (red F)** p p ..
next
    fix r q
    assume red Gr q and (red F)** pr
    show (red F)** p q
    proof
        show (red F)** p r by fact
    next
        from red-subset[OF<red Gr q\rangle\langleG\subseteqF>] show red Fr q.
    qed
qed
lemma red-singleton: red {f} pq\longleftrightarrow(\existst. red-single p q f t)
    unfolding red-def
proof
    assume }\existsf\in{f}.\existst. red-single p qf
    from this obtain f0 where f0\in{f} and a: \existst. red-single p q f0t ..
    from }\langlef0\in{f}\rangle have f0=f by sim
    from this a show }\existst\mathrm{ . red-single pqft by simp
next
    assume a: \existst. red-single pqft
    show }\existsf\in{f}.\existst\mathrm{ . red-single p qft
    proof (rule, simp)
        from a show \existst. red-single pqft.
    qed
qed
lemma red-single-lookup:
    assumes red-single pqft
    shows lookup q(t\oplusltf)=0
    using assms unfolding red-single-def
proof
    assume f}\not=0\mathrm{ and lookup p (t ¢lt f)}\not=0\wedgeq=p - monom-mult (lookup 
(t\oplusltf) /lcf)tf
    hence lookup p (t\opluslt f)\not=0 and q-def: q = p - monom-mult (lookup p (t\oplus
lt f) / lc f) tf
    by auto
    from lookup-minus[of p monom-mult (lookup p (t\opluslt f) /lcf) tft\opluslt f]
            lookup-monom-mult-plus[of lookup p (t\opluslt f) / lc ft flt f]
            lc-not-0[OF< 
            show ?thesis unfolding q-def lc-def by simp
qed
```

```
lemma red-single-higher:
    assumes red-single pqft
    shows higher q}(t\opluslt f)=higher p (t\opluslt f
    using assms unfolding higher-eq-iff red-single-def
proof (intro allI, intro impI)
    fix u
    assume a: t\opluslt f \prec}\mp@subsup{}{t}{}
        and f\not=0^lookup p (t\opluslt f)}\not=0\wedgeq=p\mathrm{ - monom-mult (lookup p (t }
lt f) / lc f) tf
    hence f}\not=
        and lookup p (t\opluslt f)}\not=
        and q-def:q=p-monom-mult (lookup p (t\oplusltf)/lc f) tf
        by simp-all
    from <lookup p (t\opluslt f)\not=0> lc-not-0[OF<f \not= 0〉] have c-not-0:lookup p (t
\opluslt f) / lc f\not=0
    by (simp add: field-simps)
    from q-def lookup-minus[of p monom-mult (lookup p (t\opluslt f) / lc f)tf]
            have q-lookup: \bigwedges. lookup q s lookup p s-lookup (monom-mult (lookup p
(t\opluslt f) / lcf)tf) s
    by simp
    from a lt-monom-mult[OF c-not-0 <f \not= 0\rangle, of t]
            have \negu \preceq. lt (monom-mult (lookup p (t\opluslt f) / lc f) tf) by simp
    with lt-max[of monom-mult (lookup p (t\opluslt f) /lc f) tfu]
    have lookup (monom-mult (lookup p (t\opluslt f) / lc f) tf) u=0 by auto
    thus lookup q u = lookup p u using q-lookup[of u] by simp
qed
lemma red-single-ord:
    assumes red-single pqft
    shows q}\mp@subsup{\prec}{p}{}
    unfolding ord-strict-higher
proof (intro exI, intro conjI)
    from red-single-lookup[OF assms] show lookup q (t\opluslt f)=0.
next
    from assms show lookup p (t\opluslt f)}\not=0\mathrm{ unfolding red-single-def by simp
next
    from red-single-higher[OF assms] show higher q (t\opluslt f) = higher p (t\opluslt f)
```



```
qed
lemma red-single-nonzero1:
    assumes red-single p qft
    shows p}\not=
proof
    assume p=0
    from this red-single-ord[OF assms] ord-p-zero-min[of q] show False by simp
qed
```

```
lemma red-single-nonzero2:
    assumes red-single p qft
    shows }f\not=
proof
    assume f=0
    from assms monom-mult-zero-right have f =0 by (simp add: red-single-def)
    from this }\langlef=0\rangle\mathrm{ show False by simp
qed
lemma red-single-self:
    assumes p\not=0
    shows red-single p 0 p 0
proof -
    from lc-not-O[OF assms] have lc: lc p\not=0.
    show ?thesis unfolding red-single-def
    proof (intro conjI)
        show p\not=0 by fact
    next
            from lc show lookup p (0\opluslt p) \not= 0 unfolding lc-def by (simp add:
term-simps)
    next
                            from lc have (lookup p (0\opluslt p)) / lc p=1 unfolding lc-def by (simp add:
term-simps)
    from this monom-mult-one-left[of p] show 0 = p - monom-mult (lookup p (0
\oplus lt p) / lc p) 0 p
            by simp
        qed
qed
lemma red-single-trans:
    assumes red-single p p0ft and lt gaddst lt f and g\not=0
    obtains p1 where red-single p p1 g(t+(lp f - lpg))
proof -
    let ?s = t + (lpf-lpg)
    let ?p = p - monom-mult (lookup p (?s \oplus lt g) / lc g) ?s g
    have red-single p ?p g ?s unfolding red-single-def
    proof (intro conjI)
        from assms(2) have eq: ?s \opluslt g=t\opluslt f using adds-term-alt splus-assoc
            by (auto simp: term-simps)
    from〈red-single p pOft〉 have lookup p (t\oplusltf)\not=0 unfolding red-single-def
by simp
    thus lookup p (?s \oplus lt g)\not=0 by (simp add: eq)
    qed (fact, fact refl)
    thus ?thesis ..
qed
lemma red-nonzero:
    assumes red Fpq
    shows p}\not=
```

```
proof -
    from red-setE[OF assms] obtain ft where red-single p qft.
    show ?thesis by (rule red-single-nonzero1, fact)
qed
lemma red-self:
    assumes p\not=0
    shows red {p} po
unfolding red-singleton
proof
    from red-single-self[OF assms] show red-single p 0 p 0 . 
qed
lemma red-ord:
    assumes red Fpq
    shows }q\mp@subsup{\prec}{p}{}
proof -
    from red-setE[OF assms] obtain f and t where red-single p qft.
    from red-single-ord[OF this] show q}\mp@subsup{\prec}{p}{}p
qed
lemma red-indI1:
    assumes f\inF and f\not=0 and p\not=0 and adds: lt f addst lt p
    shows red Fp(p-monom-mult (lc p / lc f) (lp p-lpf)f)
proof (intro red-setI[OF<f \inF\rangle])
    let ?s = lp p-lpf
    have c: lookup p (?s \opluslt f) = lc p unfolding lc-def
        by (metis add-diff-cancel-right' adds adds-termE pp-of-term-splus)
    show red-single p(p-monom-mult (lc p / lc f) ?s f) f?s unfolding red-single-def
    proof (intro conjI, fact)
        from clc-not-0[OF<p\not=0〉] show lookup p (?s \oplus lt f) \not=0 by simp
    next
            from c show p - monom-mult (lc p / lc f) ?s f = p - monom-mult (lookup
p(?s}\oplusltf)/lcf) ?s 
            by simp
    qed
qed
lemma red-indI2:
    assumes p\not=0 and r: red F (tail p) q
    shows red Fp(q+ monomial (lc p) (lt p))
proof -
    from red-setE[OF r] obtain ft where f\inF and rs: red-single (tail p) qft by
auto
    from rs have f}\not=0\mathrm{ and ct:lookup (tail p) (t @lt f)}\not=
        and q:q= tail p - monom-mult (lookup (tail p) (t\opluslt f)/lcf) tf
        unfolding red-single-def by simp-all
    from ct lookup-tail[of pt\opluslt f] have t\opluslt f \prec}\mp@subsup{}{t}{lt p by (auto split: if-splits)
    hence c:lookup (tail p) (t\opluslt f) = lookup p (t\oplusltf) using lookup-tail[of p]
```

```
by simp
    show ?thesis
    proof (intro red-setI[OF}\langlef\inF\rangle]
        show red-single p(q+Poly-Mapping.single (lt p) (lc p)) ft unfolding
red-single-def
    proof (intro conjI, fact)
            from ct c show lookup p (t\opluslt f)}\not=0\mathrm{ by simp
    next
            from q have q+ monomial (lc p) (lt p) =
                (monomial (lc p) (lt p) + tail p) - monom-mult (lookup (tail p) (t
\opluslt f) / lc f) tf
                by simp
            also have ... = p - monom-mult (lookup (tail p) (t\opluslt f)/lcf) tf
                using leading-monomial-tail[of p] by auto
                finally show q + monomial (lc p) (lt p) = p - monom-mult (lookup p (t }
lt f) / lc f) tf
                by (simp only: c)
        qed
    qed
qed
lemma red-indE:
    assumes red Fpq
    shows (\existsf\inF.f\not=0\wedgelt faddst lt p\wedge
                (q=p-monom-mult (lc p/lcf) (lp p-lpf)f))\vee
                red F (tail p) (q-monomial (lc p) (lt p))
proof -
    from red-nonzero[OF assms] have p}=0\mathrm{ .
    from red-setE[OF assms] obtain ft where f}\inF\mathrm{ and rs: red-single p qft by
auto
    from rs have f}\not=
        and cn0: lookup p (t\opluslt f)}\not=
        and q: q = p - monom-mult ((lookup p (t\oplusltf)) / lc f) tf
        unfolding red-single-def by simp-all
    show ?thesis
    proof (cases lt p=t\opluslt f)
        case True
        hence lt f addst lt p by (simp add: term-simps)
    from True have eq1: lp p-lpf=t by (simp add: term-simps)
    from True have eq2: lc p = lookup p (t\opluslt f) unfolding lc-def by simp
    show ?thesis
    proof (intro disjI1, rule bexI[of - f], intro conjI, fact+)
            from q eq1 eq2 show q}=p\mathrm{ - monom-mult (lc p / lc f) (lp p - lp f)f
                    by simp
        qed (fact)
    next
        case False
        from this lookup-tail-2[of pt\opluslt f]
            have ct:lookup (tail p) (t\opluslt f)=lookup p (t\opluslt f) by simp
```

```
    show ?thesis
    proof (intro disjI2, intro red-setI[of f], fact)
    show red-single (tail p) (q- monomial (lc p) (lt p)) ft unfolding red-single-def
    proof (intro conjI, fact)
        from cn0 ct show lookup (tail p) (t\opluslt f)}\not=0\mathrm{ by simp
    next
        from leading-monomial-tail[of p]
            have p-monomial (lc p) (lt p)=(monomial (lc p) (lt p) + tail p) -
monomial (lc p) (lt p)
            by simp
        also have ... = tail p by simp
        finally have eq: p - monomial (lc p) (lt p) = tail p .
        from q have q-monomial (lc p) (lt p)=
                            (p - monomial (lc p) (lt p)) - monom-mult ((lookup p (t\opluslt f))
/ lc f) tf by simp
        also from eq have ... = tail p - monom-mult ((lookup p (t\opluslt f)) / lc
f) tf by simp
            finally show q- monomial (lc p) (lt p) = tail p - monom-mult (lookup
(tail p) (t\oplusltf) /lcf) tf
            using ct by simp
        qed
        qed
    qed
qed
lemma is-redI:
    assumes red Fab
    shows is-red Fa
    unfolding is-red-def relation.is-final-def by (simp, intro exI[of - b], fact)
lemma is-redE:
    assumes is-red Fa
    obtains b where red Fab
    using assms unfolding is-red-def relation.is-final-def
proof simp
    assume r:\bigwedgeb. red F a b\Longrightarrow thesis and b: \existsx. red F a x
    from b obtain b where red Fab ..
    show thesis by (rule r[of b], fact)
qed
lemma is-red-alt:
    shows is-red Fa\longleftrightarrow(\existsb. red Fab)
proof
    assume is-red F a
    from is-redE[OF this] obtain b where red F a b .
    show \existsb. red F a b by (intro exI[of - b], fact)
next
    assume \existsb. red F a b
    from this obtain b where red Fab ..
```

```
    show is-red F a by (rule is-redI, fact)
qed
lemma is-red-singletonI:
    assumes is-red Fq
    obtains p where p}\inF\mathrm{ and is-red {p}q
proof -
    from assms obtain q0 where red F q q0 unfolding is-red-alt ..
    from this red-def[of F qq0] obtain p where p\inF and t: \existst. red-single q q0
pt by auto
    have is-red {p} q unfolding is-red-alt
    proof
            from red-singleton[of p q q0] t show red {p} qq0 by simp
    qed
    from }\langlep\inF\rangle\mathrm{ this show ?thesis ..
qed
lemma is-red-singletonD:
    assumes is-red {p} q and p\inF
    shows is-red Fq
proof -
    from assms(1) obtain q0 where red {p} qq0 unfolding is-red-alt ..
    from red-singleton[of p q q0] this have \existst. red-single q q0 p t ..
    from this obtain t where red-single qq0 pt ..
    show ?thesis unfolding is-red-alt
    by (intro exI[of - q0], intro red-setI[OF assms(2), of q q0 t], fact)
qed
lemma is-red-singleton-trans:
    assumes is-red {f} p and lt g addst lt f and g\not=0
    shows is-red {g} p
proof -
    from <is-red {f} p> obtain q where red {f} p q unfolding is-red-alt ..
    from this red-singleton[of f p q] obtain t where red-single p qft by auto
    from red-single-trans[OF this assms(2, 3)] obtain q0 where
        red-single p q0 g (t+(lpf-lpg)).
    show ?thesis
    proof (rule is-redI[of {g} p q0])
        show red {g} p q0 unfolding red-def
            by (intro bexI[of-g], intro exI[of-t + (lpf - lp g)], fact, simp)
    qed
qed
lemma is-red-singleton-not-0:
    assumes is-red {f} p
    shows }f\not=
using assms unfolding is-red-alt
proof
    fix q
```

```
    assume red {f} pq
    from this red-singleton[of f pq] obtain t where red-single p qft by auto
    thus ?thesis unfolding red-single-def ..
qed
lemma irred-0:
    shows \neg is-red F 0
proof (rule, rule is-redE)
    fix b
    assume red F 0 b
    from ord-p-zero-min[of b] red-ord[OF this] show False by simp
qed
lemma is-red-indI1:
    assumes f\inF and f\not=0 and p\not=0 and lt f addst lt p
    shows is-red Fp
by (intro is-redI, rule red-indI1[OF assms])
lemma is-red-indI2:
    assumes p\not=0 and is-red F (tail p)
    shows is-red Fp
proof -
    from is-redE[OF<is-red F (tail p)>] obtain q where red F (tail p) q.
    show ?thesis by (intro is-redI, rule red-indI2[OF}\langlep\not=0\rangle], fact
qed
lemma is-red-indE:
    assumes is-red F p
    shows (\existsf\inF.f\not=0\wedgelt faddst lt p)\vee is-red F (tail p)
proof -
    from is-redE[OF assms] obtain q where red F pq.
    from red-indE[OF this] show ?thesis
    proof
            assume }\existsf\inF.f\not=0\wedgeltfaddsttlt p\wedgeq=p-monom-mult (lc p / lc f)(lp
p-lpf)f
            from this obtain f}\mathrm{ where }f\inF\mathrm{ and f}\not=0\mathrm{ and lt f addst lt p by auto
            show ?thesis by (intro disjI1, rule bexI[of-f], intro conjI, fact+)
    next
            assume red F (tail p) (q- monomial (lc p) (lt p))
            show ?thesis by (intro disjI2, intro is-redI, fact)
    qed
qed
lemma rtrancl-0:
    assumes (red F)** 0x
    shows }x=
proof -
    from irred-0[of F] have relation.is-final (red F) 0 unfolding is-red-def by simp
    from relation.rtrancl-is-final[OF<(red F)** 0 x〉 this] show ?thesis by simp
```


## qed

## lemma red-rtrancl-ord:

    assumes \((\text { red } F)^{* *} p q\)
    shows \(q \preceq_{p} p\)
    using assms
    proof induct
case base
show ?case ..
next
case (step y z)
from step(2) have $z \prec_{p} y$ by (rule red-ord)
hence $z \preceq_{p} y$ by simp
also note step (3)
finally show ?case .
qed
lemma components-red-subset:
assumes red $F p q$
shows component-of-term' keys $q \subseteq$ component-of-term' keys $p \cup$ compo-
nent-of-term' Keys F
proof -
from assms obtain $f t$ where $f \in F$ and red-single $p q f t$ by (rule red-setE)
from this(2) have $q: q=p$ - monom-mult $(($ lookup $p(t \oplus l t f)) / l c f) t f$
by (simp add: red-single-def)
have component-of-term' keys $q \subseteq$
component-of-term' (keys $p \cup$ keys (monom-mult ((lookup $p(t \oplus l t f)) / l c$
f) $t f)$ )
by (rule image-mono, simp add: q keys-minus)
also have $\ldots \subseteq$ component-of-term' keys $p \cup$ component-of-term'Keys $F$
proof (simp add: image-Un, rule)
fix $k$
assume $k \in$ component-of-term'keys (monom-mult (lookup $p(t \oplus l t f) / l c$
f) $t f$ )
then obtain $v$ where $v \in$ keys (monom-mult (lookup $p(t \oplus l t f) / l c f) t f)$
and $k=$ component-of-term $v$..
from this(1) keys-monom-mult-subset have $v \in(\oplus) t$ ' keys $f$..
then obtain $u$ where $u \in$ keys $f$ and $v=t \oplus u$..
have $k=$ component-of-term $u$ by (simp add: $\langle k=$ component-of-term $v\rangle\langle v=$
$t \oplus u\rangle$ term-simps)
with $\langle u \in$ keys $f\rangle$ have $k \in$ component-of-term'keys $f$ by fastforce
also have $\ldots \subseteq$ component-of-term 'Keys $F$ by (rule image-mono, rule keys-subset-Keys,
fact)
finally show $k \in$ component-of-term' keys $p \cup$ component-of-term' Keys $F$
by $\operatorname{simp}$
qed
finally show ?thesis.
qed

```
corollary components-red-rtrancl-subset:
    assumes (red F)** p q
    shows component-of-term' keys q\subseteqcomponent-of-term'keys p U compo-
nent-of-term' Keys F
    using assms
proof (induct)
    case base
    show ?case by simp
next
    case (step q r)
    from step(2) have component-of-term ' keys r\subseteqcomponent-of-term'keys q U
component-of-term' Keys F
    by (rule components-red-subset)
    also from step(3) have ...\subseteq component-of-term'keys p\cup component-of-term
    ' Keys F by blast
    finally show ?case.
qed
```


### 4.2 Reducibility and Addition \& Multiplication

lemma red-single-monom-mult:
assumes red-single $p q f t$ and $c \neq 0$
shows red-single (monom-mult $c$ s $p$ ) (monom-mult $c s q) f(s+t)$
proof -
from $\operatorname{assms}(1)$ have $f \neq 0$
and lookup $p(t \oplus l t f) \neq 0$
and $q$-def: $q=p$ - monom-mult $((l o o k u p ~ p(t \oplus l t f)) / l c f) t f$
unfolding red-single-def by auto
have assoc: $(s+t) \oplus l t f=s \oplus(t \oplus l t f)$ by (simp add: ac-simps)
have $g 2$ : lookup (monom-mult csp) $((s+t) \oplus l t f) \neq 0$
proof
assume lookup (monom-mult csep) $((s+t) \oplus l t f)=0$
hence $c *$ lookup $p(t \oplus l t f)=0$ using assoc by (simp add: lookup-monom-mult-plus)
thus False using $\langle c \neq 0\rangle\langle$ lookup $p(t \oplus l t f) \neq 0\rangle$ by simp
qed
have g3: monom-mult c s $q=$
(monom-mult cse ) - monom-mult ((lookup (monom-mult csp) $((s+t) \oplus l t$
f)) / lcf $)(s+t) f$
proof -
from $q$-def monom-mult-dist-right-minus $\left[\begin{array}{lll}c & s & p\end{array}\right]$
have monom-mult cs $q=$
monom-mult csp-monom-mult cs(monom-mult (lookup $p(t \oplus l t f)$
/ lcf) tf) by $\operatorname{simp}$
also from monom-mult-assoc[of c s lookup p $(t \oplus l t f) / l c f t f]$ assoc
have monom-mult cs (monom-mult (lookup $p(t \oplus l t f) / l c f) t f)=$
monom-mult $((l o o k u p$ (monom-mult csp) $((s+t) \oplus l t f)) / l c f)(s+$
t) $f$
by (simp add: lookup-monom-mult-plus)
finally show?thesis .
qed
from $\langle f \neq 0\rangle g 2 g 3$ show ?thesis unfolding red-single-def by auto qed
lemma red-single-plus-1:
assumes red-single p qft and $t \oplus l t f \notin$ keys $(p+r)$
shows red-single $(q+r)(p+r) f t$
proof -
from assms have $f \neq 0$ and lookup $p(t \oplus l t f) \neq 0$
and $q: q=p$ - monom-mult $((l o o k u p p(t \oplus l t f)) / l c f) t f$
by (simp-all add: red-single-def)
from $\operatorname{assms}(1)$ have $c q-0$ : lookup $q(t \oplus l t f)=0$ by (rule red-single-lookup)
from $\operatorname{assms}(2)$ have lookup $(p+r)(t \oplus l t f)=0$
by (simp add: in-keys-iff)
with neg-eq-iff-add-eq-O[of lookup p $(t \oplus l t f)$ lookup $r(t \oplus l t f)$ ]
have cr: lookup $r(t \oplus l t f)=-($ lookup $p(t \oplus l t f))$ by (simp add: lookup-add)
hence cr-not- 0 : lookup $r(t \oplus l t f) \neq 0$ using 〈lookup $p(t \oplus l t f) \neq 0$ by simp
from $\langle f \neq 0\rangle$ show ?thesis unfolding red-single-def
proof (intro conjI)
from cr-not-0 show lookup $(q+r)(t \oplus l t f) \neq 0$ by (simp add: lookup-add $c q-0)$
next
from lc-not- $0[O F\langle f \neq 0\rangle]$
have monom-mult ((lookup $(q+r)(t \oplus l t f)) / l c f) t f=$ monom-mult $((l o o k u p r(t \oplus l t f)) / l c f) t f$
by (simp add: field-simps lookup-add cq-0)
thus $p+r=q+r$ - monom-mult (lookup $(q+r)(t \oplus l t f) / l c f) t f$
by (simp add: cr q monom-mult-uminus-left)
qed
qed
lemma red-single-plus-2:
assumes red-single $p q f t$ and $t \oplus l t f \notin$ keys $(q+r)$
shows red-single $(p+r)(q+r) f t$
proof -
from assms have $f \neq 0$ and cp: lookup $p(t \oplus l t f) \neq 0$
and $q: q=p$ - monom-mult $((l o o k u p ~ p(t \oplus l t f)) / l c f) t f$
by (simp-all add: red-single-def)
from $\operatorname{assms}(1)$ have $c q-0$ : lookup $q(t \oplus l t f)=0$ by (rule red-single-lookup)
with $\operatorname{assms}(2)$ have $c r-0$ : lookup $r(t \oplus l t f)=0$
by (simp add: lookup-add in-keys-iff)
from $\langle f \neq 0\rangle$ show ?thesis unfolding red-single-def
proof (intro conjI)
from $c p$ show lookup $(p+r)(t \oplus l t f) \neq 0$ by (simp add: lookup-add cr-0)
next
show $q+r=p+r-$ monom-mult (lookup $(p+r)(t \oplus l t f) / l c f) t f$
by (simp add: cr-0 q lookup-add)
qed
qed

```
lemma red-single-plus-3:
    assumes red-single p qft and t}\oplusltf\inkeys (p+r) and t\opluslt f\inkeys(
+r)
    shows \existss. red-single (p+r)sft\wedge red-single (q+r)sft
proof -
    let ?}t=t\opluslt
    from assms have f}\not=0\mathrm{ and lookup p ?t }\not=
        and q: q=p - monom-mult ((lookup p ?t) / lc f) tf
        by (simp-all add: red-single-def)
    from assms(2) have cpr: lookup (p+r) ?t f=0 by (simp add: in-keys-iff)
    from assms(3) have cqr: lookup ( q+r) ?t \not=0 by (simp add: in-keys-iff)
    from assms(1) have cq-0: lookup q ?t = 0 by (rule red-single-lookup)
    let ?s = (p+r) - monom-mult ((lookup (p+r) ?t) /lcf)tf
    from}\langlef\not=0\rangle cpr have red-single (p+r) ?s ft by (simp add: red-single-def
    moreover from }\langlef\not=0\rangle\mathrm{ have red-single (q+r) ?s ft unfolding red-single-def
    proof (intro conjI)
        from cqr show lookup (q+r) ?t }\not=0\mathrm{ .
    next
        from lc-not-0[OF< 
            monom-mult-dist-left[of (lookup p ?t) / lc f (lookup r ?t) / lc f t f]
            have monom-mult ((lookup (p+r) ?t) / lc f) tf=
                    (monom-mult ((lookup p ?t) / lc f) t f) +
                            (monom-mult ((lookup r?t) / lc f) tf)
                by (simp add: field-simps lookup-add)
        moreover from lc-not-O[OF< < 
            monom-mult-dist-left[of (lookup q ?t) / lc f (lookup r ?t) / lc f t f]
            have monom-mult ((lookup (q+r) ?t) / lc f) tf=
                    monom-mult ((lookup r ?t) / lc f) tf
                    by (simp add: field-simps lookup-add cq-0)
        ultimately show p +r - monom-mult (lookup (p+r) ?t / lc f) tf =
                        q+r - monom-mult (lookup (q+r) ?t / lc f) tf by (simp add:
q)
    qed
    ultimately show ?thesis by auto
qed
lemma red-single-plus:
    assumes red-single pqft
    shows red-single (p+r) (q+r)ft\vee
            red-single }(q+r)(p+r)ft
            (\existss.red-single }(p+r)sft\wedgered-single (q+r)sft)(is ?A \vee ?B\vee ?C)
proof (cases t }\oplusltf\in\mathrm{ keys ( }p+r\mathrm{ ))
    case True
    show ?thesis
    proof (cases t }\oplusltf\in\mathrm{ keys (q+r))
        case True
        with assms <t \opluslt f\in keys ( }p+r\mathrm{ )〉 have ?C by (rule red-single-plus-3)
        thus ?thesis by simp
```

```
    next
        case False
        with assms have ?A by (rule red-single-plus-2)
        thus ?thesis ..
    qed
next
    case False
    with assms have ?B by (rule red-single-plus-1)
    thus ?thesis by simp
qed
lemma red-single-diff:
    assumes red-single ( }p-q\mathrm{ ) rft
    shows red-single p (r+q)ft\vee red-single q (p-r)ft\vee
        (\exists\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}.red-single p p'ft\wedge red-single q q'ft}^\mp@code{r}=\mp@subsup{p}{}{\prime}-\mp@subsup{q}{}{\prime})(\mathrm{ is ?A }\vee\mathrm{ ?B
\vee ?C)
proof -
    let ?s=t\opluslt f
    from assms have f}\not=
        and lookup (p-q) ?s }\not=
        and r:r=p-q- monom-mult ((lookup (p-q) ?s) / lc f)tf
        unfolding red-single-def by auto
    from this(2) have diff:lookup p ?s \not= lookup q ?s by (simp add:lookup-minus)
    show ?thesis
    proof (cases lookup p ?s=0)
        case True
        with diff have ?s \in keys q by (simp add: in-keys-iff)
    moreover have lookup (p-q) ?s = lookup q ?s by (simp add: lookup-minus
True)
    ultimately have ?B using <f }\not=0\rangle\mathrm{ by (simp add: in-keys-iff red-single-def r
monom-mult-uminus-left)
    thus ?thesis by simp
    next
        case False
        hence ?s \in keys p by (simp add: in-keys-iff)
        show ?thesis
        proof (cases lookup q ?s = 0)
            case True
            hence lookup ( }p-q\mathrm{ ) ?s = lookup p ?s by (simp add: lookup-minus)
            hence ?A using \langlef }=0\rangle\langle?s\in\mathrm{ keys p> by (simp add: in-keys-iff red-single-def
r ~ m o n o m - m u l t - u m i n u s - l e f t )
            thus ?thesis ..
        next
            case False
            hence ?s \in keys q by (simp add: in-keys-iff)
            let ?p = p - monom-mult ((lookup p ?s)/lc f) tf
            let ?q=q-monom-mult ((lookup q ?s) / lc f)tf
            have ?C
            proof (intro exI conjI)
```

from $\langle f \neq 0\rangle\langle ? s \in$ keys $p\rangle$ show red-single $p$ ?p $f t$ by (simp add: in-keys-iff red-single-def)
next
from $\langle f \neq 0\rangle\langle ? s \in$ keys $q\rangle$ show red-single $q$ ? $q$ ft by (simp add: in-keys-iff red-single-def)
next
from $\langle f \neq 0\rangle$ have $l c f \neq 0$ by (rule lc-not- 0 )
hence eq: (lookup p?s lookup q ?s) / lc $f=$
lookup $p$ ?s / lc f-lookup q ?s / lc f by (simp add: field-simps)
show $r=? p-? q$ by (simp add: r lookup-minus eq monom-mult-dist-left-minus)
qed
thus ?thesis by simp
qed
qed
qed
lemma red-monom-mult:
assumes $a$ : red $F p q$ and $c \neq 0$
shows red $F$ (monom-mult $c$ s $p$ ) (monom-mult $c s q$ )
proof -
from red-setE[OF a] obtain $f$ and $t$ where $f \in F$ and rs: red-single $p$ qft by auto
from red-single-monom-mult $[$ OF rs $\langle c \neq 0\rangle$, of $s]$ show ?thesis by (intro red-setI[OF $\langle f \in F\rangle])$
qed
lemma red-plus-keys-disjoint:
assumes red $F p q$ and keys $p \cap$ keys $r=\{ \}$
shows red $F(p+r)(q+r)$
proof -
from assms(1) obtain $f t$ where $f \in F$ and $*$ : red-single $p q f t$ by (rule red-setE)
from this(2) have red-single $(p+r)(q+r) f t$
proof (rule red-single-plus-2)
from $*$ have lookup $q(t \oplus l t f)=0$
by (simp add: red-single-def lookup-minus lookup-monom-mult lc-def[symmetric] lc-not-0 term-simps)
hence $t \oplus l t f \notin$ keys $q$ by (simp add: in-keys-iff)
moreover have $t \oplus l t f \notin$ keys $r$
proof
assume $t \oplus l t f \in$ keys $r$
moreover from $*$ have $t \oplus l t f \in$ keys $p$ by (simp add: in-keys-iff red-single-def)
ultimately have $t \oplus l t f \in$ keys $p \cap$ keys $r$ by simp
with assms(2) show False by simp
qed
ultimately have $t \oplus l t f \notin$ keys $q \cup$ keys $r$ by simp
thus $t \oplus l t f \notin$ keys $(q+r)$
by (meson Poly-Mapping.keys-add subsetD)
qed
with $\langle f \in F\rangle$ show ?thesis by (rule red-setI)
qed
lemma red-plus:
assumes red $F p q$
obtains $s$ where $(\text { red } F)^{* *}(p+r) s$ and $(r e d F)^{* *}(q+r) s$
proof -
from red-set $E[O F$ assms $]$ obtain $f$ and $t$ where $f \in F$ and rs: red-single $p q f$ $t$ by auto
from red-single-plus[OF rs, of $r$ ] show ?thesis
proof
assume c1: red-single $(p+r)(q+r) f t$
show ?thesis
proof
from $c 1$ show $(\text { red } F)^{* *}(p+r)(q+r)$ by (intro $r$-into-rtranclp, intro red-setI[OF $\langle f \in F\rangle])$
next
show $(\text { red } F)^{* *}(q+r)(q+r)$..
qed
next
assume red-single $(q+r)(p+r) f t \vee(\exists s$. red-single $(p+r)$ sft $\wedge$ red-single $(q+r) s f t)$
thus ?thesis
proof
assume $c$ 2: red-single $(q+r)(p+r) f t$
show ?thesis
proof show $(r e d F)^{* *}(p+r)(p+r)$..
next
from $c 2$ show $(\text { red } F)^{* *}(q+r)(p+r)$ by (intro $r$-into-rtranclp, intro red-setI[OF $\langle f \in F\rangle]$ )
qed
next
assume $\exists$ s. red-single $(p+r) s f t \wedge r e d$-single $(q+r) s f t$
then obtain $s$ where s1: red-single $(p+r) s f t$ and s2: red-single $(q+r)$ $s f t$ by auto
show ?thesis
proof
from $s 1$ show $(\text { red } F)^{* *}(p+r) s$ by (intro r-into-rtranclp, intro red-setI[OF $\langle f \in F\rangle])$
next
from $s 2$ show $(\text { red } F)^{* *}(q+r) s$ by (intro r-into-rtranclp, intro red-setI[OF $\langle f \in F\rangle])$
qed
qed
qed
qed
corollary red-plus-cs:

```
    assumes red Fpq
    shows relation.cs (red F) (p+r) (q+r)
    unfolding relation.cs-def
proof -
    from assms obtain s where (red F)** (p+r)s and (red F)** (q+r)s by
(rule red-plus)
    show \existss. (red F)** (p+r)s\wedge(red F)** (q+r)s by (intro exI, intro conjI,
fact, fact)
qed
lemma red-uminus:
    assumes red Fpq
    shows red F (-p) (-q)
    using red-monom-mult[OF assms, of -1 0] by (simp add: uminus-monom-mult)
lemma red-diff:
    assumes red F (p-q)r
    obtains p' q' where (red F)** p p ' and (red F)** q q' and r= p' - q'
proof -
    from assms obtain ft where f\inF and red-single (p-q)rft by (rule
red-setE)
    from red-single-diff[OF this(2)] show ?thesis
    proof (elim disjE)
    assume red-single p (r+q)ft
    with}\langlef\inF\rangle\mathrm{ have *: red F p(r+q) by (rule red-setI)
    show ?thesis
    proof
        from * show (red F)** p (r+q) ..
    next
        show (red F)** q q ..
    qed simp
    next
    assume red-single q(p-r)ft
    with}\langlef\inF\rangle\mathrm{ have *: red Fq(p-r) by (rule red-setI)
    show ?thesis
    proof
        show (red F)** p p ..
    next
        from * show (red F)** q(p-r)..
    qed simp
    next
    assume \exists\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}.\mathrm{ . red-single p p'ft}\wedge red-single q q ' ft ^r= p' - q'
    then obtain p' q' where 1: red-single p p'ft and 2: red-single q q'ft and
r= p
    by blast
    from }\langlef\inF\rangle2\mathrm{ have red Fqq' by (rule red-setI)
    from }\langlef\inF\rangle1\mathrm{ have red F p p' by (rule red-setI)
    hence (red F)** p p'..
    moreover from <red Fqq'> have (red F)** q q' ..
```

```
    moreover note \langler= p' - q'>
    ultimately show ?thesis ..
    qed
qed
lemma red-diff-rtrancl':
    assumes (red F)** (p-q)r
    obtains p' q' where (red F)** p p ' and (red F)** q q' and r= p' - q'
    using assms
proof (induct arbitrary: thesis rule: rtranclp-induct)
    case base
    show ?case by (rule base, fact rtrancl-refl[to-pred], fact rtrancl-refl[to-pred], fact
refl)
next
    case (step y z)
    obtain p1 q1 where p1: (red F)** p p1 and q1: (red F)** qq1 and y:y=p1
- q1 by (rule step(3))
    from step(2) obtain p' q}\mp@subsup{q}{}{\prime}\mathrm{ where p}\mp@subsup{p}{}{\prime}:(\mathrm{ red F)** p1 p
and z:z=\mp@subsup{p}{}{\prime}-\mp@subsup{q}{}{\prime}
    unfolding y by (rule red-diff)
    show ?case
    proof (rule step(4))
        from p1 p' show (red F)** p p' by simp
    next
        from q1 q' show (red F)** q q' by simp
    qed fact
qed
lemma red-diff-rtrancl:
    assumes (red F)** (p-q)0
    obtains s where (red F)** ps and (red F)** qs
proof -
    from assms obtain p' }\mp@subsup{p}{}{\prime}\mathrm{ where p}\mp@subsup{p}{}{\prime}:(\mathrm{ red F}\mp@subsup{)}{}{**}p\mp@subsup{p}{}{\prime}\mathrm{ and q':(red F)** q q ' and 0
= p' - q'
    by (rule red-diff-rtrancl')
    from this(3) have q' = p
    from }\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}\mathrm{ show ?thesis unfolding }\langle\mp@subsup{q}{}{\prime}=\mp@subsup{p}{}{\prime}\rangle.
qed
corollary red-diff-rtrancl-cs:
    assumes (red F)** (p-q)0
    shows relation.cs (red F) pq
    unfolding relation.cs-def
proof -
    from assms obtain s where (red F)** ps and (red F)** qs by (rule red-diff-rtrancl)
    show \existss.(red F)** ps^(red F)** qs by (intro exI, intro conjI, fact, fact)
qed
```


### 4.3 Confluence of Reducibility

```
lemma confluent-distinct-aux:
    assumes r1: red-single p q1 f1 t1 and r2: red-single p q2 f2 t2
        and t1 \opluslt f1 < t t2 \opluslt f2 and f1 \inF and f2 }\in
    obtains s}\mathrm{ where (red F)** q1 s and (red F)** q2 s
proof -
    from r1 have f1 \not=0 and c1: lookup p (t1 \opluslt f1) \not=0
        and q1-def: q1 = p-monom-mult (lookup p (t1 \opluslt f1) / lc f1) t1 f1
        unfolding red-single-def by auto
    from r2 have f2 # 0 and c2: lookup p (t2 \opluslt f2) #=0
        and q2-def: q2 = p - monom-mult (lookup p (t2 \oplus lt f2) / lc f2) t2 f2
        unfolding red-single-def by auto
    from <t1 }\opluslt f1 < t t2 \opluslt f2>
    have lookup (monom-mult (lookup p (t1 \oplus lt f1) / lc f1) t1 f1) (t2 \oplus lt f2) = 0
        by (simp add: lookup-monom-mult-eq-zero)
    from lookup-minus[of p-t2 \opluslt f2] this have c:lookup q1 (t2 }\opluslt f2) = lookup
p(t2 \opluslt f2)
    unfolding q1-def by simp
    define q3 where q3 \equivq1 - monom-mult ((lookup q1 (t2 \oplus lt f2)) / lc f2) t2
f2
    have red-single q1 q3 f2 t2 unfolding red-single-def
    proof (rule, fact, rule)
        from c c2 show lookup q1 (t2 \oplus lt f2) #= 0 by simp
    next
        show q3 = q1 - monom-mult (lookup q1 (t2 \oplus lt f2) / lc f2) t2 f2 unfolding
q3-def ..
    qed
    hence red F q1 q3 by (intro red-setI[OF<f2 \in F>])
    hence q1q3:(red F)** q1 q3 by (intro r-into-rtranclp)
    from r1 have red F p q1 by (intro red-setI[OF<<1 \inF>])
    from red-plus[OF this, of - monom-mult ((lookup p (t2 \oplus lt f2)) / lc f2) t2 f2]
obtain s
    where r3: (red F)** (p - monom-mult (lookup p (t2 \oplus lt f2) / lc f2) t2 f2) s
    and r4:(red F)** (q1 - monom-mult (lookup p (t2 \oplus lt f2) / lc f2) t2 f2) s
by auto
    from r3 have q2s: (red F)** q2 s unfolding q2-def by simp
    from r4 c have q3s: (red F)** q3 s unfolding q3-def by simp
    show ?thesis
    proof
        from rtranclp-trans[OF q1q3 q3s] show (red F)** q1 s.
    next
        from q2s show (red F)** q2 s.
    qed
qed
lemma confluent-distinct:
assumes r1: red-single \(p\) q1 f1 t1 and r2: red-single \(p\) q2 f2 t2 and \(n e: t 1 \oplus l t f 1 \neq t 2 \oplus l t f 2\) and \(f 1 \in F\) and \(f 2 \in F\)
obtains \(s\) where \((\text { red } F)^{* *} q 1 s\) and \((\operatorname{red} F)^{* *} q 2 s\)
```

```
proof -
    from ne have t1 \opluslt f1 \prec}\mp@subsup{t}{t}{\prime2}\opluslt f2 \vee t2 \opluslt f2 < < t1 \opluslt f1 by aut
    thus ?thesis
    proof
        assume a1: t1 \opluslt f1 < t t2 \opluslt f2
        from confluent-distinct-aux[OF r1 r2 a1 <f1 \inF\rangle\langlef2 \inF>] obtain s where
            (red F)** q1 s and (red F)** q2 s .
        thus ?thesis ..
    next
        assume a2:t2 \oplus lt f2 < <t t1 \oplus lt f1
        from confluent-distinct-aux[OF r2 r1 a2 <f2 \inF\rangle\langlef1 \inF>] obtain s where
            (red F)** q1 s and (red F)** q2 s .
        thus ?thesis ..
    qed
qed
corollary confluent-same:
    assumes r1: red-single p q1 ft1 and r2: red-single pq2 f t2 and f}\in
    obtains s where (red F)** q1 s and (red F)** q2 s
proof (cases t1 = t2)
    case True
    with r1 r2 have q1 = q2 by (simp add: red-single-def)
    show ?thesis
    proof
        show (red F)** q1 q2 unfolding <q1 = q2`..
    next
        show (red F)** q2 q2 ..
    qed
next
    case False
    hence t1 \opluslt f\not= t2 \opluslt f by (simp add: term-simps)
    from r1 r2 this }\langlef\inF\rangle\langlef\inF\rangle\mathrm{ obtain s where (red F)** q1 s and (red F)**
q2 s
    by (rule confluent-distinct)
    thus ?thesis..
qed
```


### 4.4 Reducibility and Module Membership

lemma srtc-in-pmdl:
assumes relation.srtc (red F) pq
shows $p-q \in p m d l F$
using assms unfolding relation.srtc-def
proof (induct rule: rtranclp.induct)
fix $p$
show $p-p \in p m d l$ $F$ by (simp add: pmdl.span-zero)
next
fix $p r q$
assume $p r-i n: p-r \in p m d l F$ and red: red $F r q \vee \operatorname{red} F q r$

```
    from red obtain \(f\) ct where \(f \in F\) and \(q=r\) - monom-mult ct \(f\)
    proof
    assume red \(F r q\)
    from red-set \(E[O F\) this \(]\) obtain \(f t\) where \(f \in F\) and red-single \(r q f\).
        hence \(q=r\) - monom-mult (lookup \(r(t \oplus l t f) / l c f) t f\) by (simp add:
red-single-def)
    show thesis by (rule, fact, fact)
    next
    assume red \(F q r\)
    from red-setE[OF this] obtain \(f t\) where \(f \in F\) and red-single qrft.
        hence \(r=q\) - monom-mult (lookup \(q(t \oplus l t f) / l c f) t f\) by (simp add:
red-single-def)
    hence \(q=r+\) monom-mult (lookup \(q(t \oplus l t f) / l c f) t f\) by simp
    hence \(q=r\) - monom-mult ( \(-(\) lookup \(q(t \oplus l t f) / l c f)) t f\)
        using monom-mult-uminus-left \([o f-t f]\) by simp
    show thesis by (rule, fact, fact)
    qed
    hence eq: \(p-q=(p-r)+\) monom-mult \(c t f\) by simp
    show \(p-q \in p m d l F\) unfolding eq
    by (rule pmdl.span-add, fact, rule monom-mult-in-pmdl, fact)
qed
lemma in-pmdl-srtc:
    assumes \(p \in p m d l F\)
    shows relation.srtc (red \(F) p 0\)
    using assms
proof (induct p rule: pmdl-induct)
    show relation.srtc (red F) 00 unfolding relation.srtc-def ..
next
    fix \(a f c t\)
    assume \(a\)-in: \(a \in p m d l F\) and \(I H\) : relation.srtc (red \(F\) ) a 0 and \(f \in F\)
    show relation.srtc (red \(F)(a+\) monom-mult \(c t f) 0\)
    proof (cases \(c=0\) )
        assume \(c=0\)
        hence \(a+\) monom-mult ct \(f=a\) by simp
        thus ?thesis using \(I H\) by simp
    next
        assume \(c \neq 0\)
        show ?thesis
    proof (cases \(f=0\) )
        assume \(f=0\)
        hence \(a+\) monom-mult ctf \(=a\) by simp
        thus ?thesis using \(I H\) by simp
    next
        assume \(f \neq 0\)
        from \(l c-n o t-0[O F\) this \(]\) have \(l c f \neq 0\).
        have red \(F\) (monom-mult ctf) 0
        proof (intro red-setI[OF \(\langle f \in F\rangle]\) )
            from lookup-monom-mult-plus[of ctf lt f]
```

have eq: lookup (monom-mult $c t f)(t \oplus l t f)=c * l c f$ unfolding $l c$-def
show red-single (monom-mult ctf) $0 f t$ unfolding red-single-def eq proof (intro conjI, fact)
from $\langle c \neq 0\rangle\langle l c f \neq 0\rangle$ show $c * l c f \neq 0$ by simp
next
from $\langle l c f \neq 0\rangle$ show $0=$ monom-mult $c t f-$ monom-mult $(c * l c f /$ lcf) $t f$ by simp
qed
qed
from red-plus $[$ OF this, of $a]$ obtain $s$ where
s1: $(\text { red } F)^{* *}($ monom-mult $c t f+a) s$ and $s 2:(\text { red } F)^{* *}(0+a) s$.
have relation.cs $($ red $F)(a+$ monom-mult $c t f) a$ unfolding relation.cs-def proof (intro exI $[o f-s]$, intro conjI)
from $s 1$ show $(\text { red } F)^{* *}(a+$ monom-mult $c t f) s$ by (simp only: add.commute)
next
from $s 2$ show $(\text { red } F)^{* *}$ as by simp
qed
from relation.srtc-transitive $[O F$ relation.cs-implies-srtc $[O F$ this $] I H]$ show
?thesis.
qed
qed
qed
lemma red-rtranclp-diff-in-pmdl:
assumes $(\text { red } F)^{* *} p q$
shows $p-q \in p m d l F$
proof -
from assms have relation.srtc (red F) pq
by (simp add: r-into-rtranclp relation.rtc-implies-srtc)
thus ?thesis by (rule srtc-in-pmdl)
qed
corollary red-diff-in-pmdl:
assumes red $F p q$
shows $p-q \in p m d l F$
by (rule red-rtranclp-diff-in-pmdl, rule r-into-rtranclp, fact)
corollary red-rtranclp-0-in-pmdl:
assumes $(\text { red } F)^{* *} p 0$
shows $p \in p m d l F$
using assms red-rtranclp-diff-in-pmdl by fastforce
lemma pmdl-closed-red:
assumes $p m d l B \subseteq p m d l a$ and $p \in p m d l A$ and $\operatorname{red} B p q$
shows $q \in p m d l a$
proof -
have $q-p \in p m d l A$

## proof

have $p-q \in p m d l B$ by (rule red-diff-in-pmdl, fact)
hence $-(p-q) \in p m d l B$ by (rule pmdl.span-neg)
thus $q-p \in p m d l B$ by simp
qed $f a c t$
from pmdl.span-add[OF this $\langle p \in p m d l ~ A\rangle]$ show ?thesis by simp
qed

### 4.5 More Properties of red, red-single and is-red

```
lemma red-rtrancl-mult:
    assumes (red F)** p q
    shows (red F)** (monom-mult c t p)(monom-mult c t q)
proof (cases c = 0)
    case True
    have (red F)** 0 0 by simp
    thus ?thesis by (simp only: True monom-mult-zero-left)
next
    case False
    from assms show ?thesis
    proof (induct rule: rtranclp-induct)
        show (red F)** (monom-mult c t p) (monom-mult c t p) by simp
    next
        fix q0q
            assume (red F)** p q0 and red F q0 q and (red F)** (monom-mult c t p)
(monom-mult c t q0)
    show (red F)** (monom-mult c t p) (monom-mult c t q)
    proof (rule rtranclp.intros(2)[OF<(red F)** (monom-mult c t p) (monom-mult
ct q0)>])
            from red-monom-mult[OF <red F q0 q> False, of t] show red F (monom-mult
c t q0)(monom-mult ctq).
    qed
    qed
qed
corollary red-rtrancl-uminus:
    assumes (red F)** p q
    shows (red F)** (-p) (-q)
    using red-rtrancl-mult[OF assms, of -1 0] by (simp add: uminus-monom-mult)
lemma red-rtrancl-diff-induct [consumes 1, case-names base step]:
    assumes a: (red F)** (p-q)r
        and cases: P p p!!yz. [| (red F)** (p-q)z; red F z y;Pp(q+z)|] ==> P
p (q+y)
    shows P p (q+r)
    using a
proof (induct rule: rtranclp-induct)
    from cases(1) show P p (q+(p-q)) by simp
next
```

fix $y z$
assume $(\text { red } F)^{* *}(p-q) z$ red $F z$ y $P p(q+z)$
thus $P p(q+y)$ using $\operatorname{cases}(2)$ by $\operatorname{simp}$
qed
lemma red-rtrancl-diff-0-induct [consumes 1, case-names base step]:
assumes $a$ : $(\operatorname{red} F)^{* *}(p-q) 0$
and base: P p pand ind: $\bigwedge y z .\left[\mid(\operatorname{red} F)^{* *}(p-q) y ;\right.$ red Fyz;Pp(y+q)|]
$==>P p(z+q)$
shows $P p q$
proof -
from ind red-rtrancl-diff-induct[of FpquP, OF a base] have $P$ p $(0+q)$
by (simp add: ac-simps)
thus ?thesis by simp
qed
lemma is-red-union: is-red $(A \cup B) p \longleftrightarrow(i s-r e d A p \vee i s-r e d B p)$
unfolding is-red-alt red-union by auto
lemma red-single-0-lt:
assumes red-single f $0 h t$
shows $l t f=t \oplus l t h$
proof -
from red-single-nonzero $1[$ OF assms] have $f \neq 0$.
\{
assume $h \neq 0$ and neq: lookup $f(t \oplus l t h) \neq 0$ and
$e q: f=$ monom-mult (lookup $f(t \oplus l t h) / l c h) t h$
from lc-not- $0[O F\langle h \neq 0\rangle]$ have $l c h \neq 0$.
with neq have (lookup $f(t \oplus l t h) / l c h) \neq 0$ by simp
from eq lt-monom-mult $[O F$ this $\langle h \neq 0\rangle$, of $t]$ have $l t f=t \oplus l t h$ by simp
hence lt $f=t \oplus l t h$ by (simp add: ac-simps)
\}
with assms show ?thesis unfolding red-single-def by auto
qed
lemma red-single-lt-distinct-lt:
assumes rs: red-single fght and $g \neq 0$ and $l t g \neq l t f$
shows $l t f=t \oplus l t h$
proof -
from red-single-nonzero1 $[$ OF $r s]$ have $f \neq 0$.
from red-single-ord $\left[\right.$ OF rs] have $g \preceq_{p} f$ by simp
from ord-p-lt[OF this] «lt $g \neq l t f\rangle$ have $l t g \prec_{t} l t f$ by simp \{
assume $h \neq 0$ and neq: lookup $f(t \oplus l t h) \neq 0$ and
eq: $f=g+$ monom-mult (lookup $f(t \oplus l t h) / l c h) t h(i s f=g+? R)$
from lc-not- $0[O F\langle h \neq 0\rangle]$ have $l c h \neq 0$.
with neq have (lookup $f(t \oplus l t h) / l c h) \neq 0($ is $? c \neq 0)$ by simp
from eq lt-monom-mult $[O F$ this $\langle h \neq 0\rangle$, of $t]$ have $l t R$ : lt $? R=t \oplus l t h$ by simp
from monom－mult－eq－zero－iff $[$ of ？c $t h]\langle ? c \neq 0\rangle\langle h \neq 0\rangle$ have $? R \neq 0$ by auto
from lt－plus－lessE［of g］eq 〈lt $\left.g \prec_{t} l t f\right\rangle$ have $l t g \prec_{t} l t$ ？$R$ by auto
from lt－plus－eqI［OF this］eq ltR have lt $f=t \oplus l t h$ by（simp add：ac－simps） \}
with assms show ？thesis unfolding red－single－def by auto

## qed

lemma zero－reducibility－implies－lt－divisibility＇：
assumes $(\text { red } F)^{* *} f 0$ and $f \neq 0$
shows $\exists h \in F . h \neq 0 \wedge\left(l t h a d d s_{t} l t f\right)$
using assms
proof（induct rule：converse－rtranclp－induct）
case base
then show？case by simp
next

```
    case (step fg)
```

    show ?case
    proof (cases \(g=0\) )
        case True
        with step.hyps have red Ff0 by simp
        from red-setE[OF this] obtain \(h t\) where \(h \in F\) and rs: red-single f \(0 h t\) by
    auto
show ?thesis
proof
from red-single-0-lt[OF rs] have lt $h$ adds $s_{t}$ lt $f$ by (simp add: term-simps)
also from rs have $h \neq 0$ by (simp add: red-single-def)
ultimately show $h \neq 0 \wedge l t h a d d s_{t}$ lt $f$ by simp
qed (rule $\langle h \in F\rangle$ )
next
case False
show ?thesis
proof (cases lt $g=l t f$ )
case True
with False step.hyps show?thesis by simp
next
case False
from red-setE[OF〈red Ffg〉] obtain $h t$ where $h \in F$ and rs: red-single $f$
$g h t$ by auto
show ?thesis
proof
from red-single-lt-distinct-lt[OF rs $\langle g \neq 0\rangle$ False] have lt haddst $l t f$
by (simp add: term-simps)
also from rs have $h \neq 0$ by (simp add: red-single-def)
ultimately show $h \neq 0 \wedge l t h$ addst $l t f$ by simp
qed (rule $\langle h \in F\rangle$ )
qed
qed
qed

```
lemma zero-reducibility-implies-lt-divisibility:
    assumes (red F)** f0 and f}\not=
    obtains }h\mathrm{ where }h\inF\mathrm{ and }h\not=0\mathrm{ and lt h addst lt f
    using zero-reducibility-implies-lt-divisibility'[OF assms] by auto
lemma is-red-addsI:
    assumes }f\inF\mathrm{ and }f\not=0\mathrm{ and v}\in\mathrm{ keys p and lt faddst v
    shows is-red Fp
    using assms
proof (induction p rule: poly-mapping-tail-induct)
    case 0
    from }\langlev\in\mathrm{ keys 0> show ?case by auto
next
    case (tail p)
    from tail.IH[OF <f\inF\rangle\langlef}\not=0\rangle-\langlelt f addst v\rangle] have imp:v\in keys (tail p
"s-red F (tail p).
    show ?case
    proof (cases v=lt p)
        case True
        show ?thesis
        proof (rule is-red-indI1[OF}\langlef\inF\rangle\langlef\not=0\rangle\langlep\not=0\rangle]
            from <lt f addst v> True show lt f addst lt p by simp
        qed
    next
        case False
        with }\langlev\in\mathrm{ keys p><p}\not=0\rangle\mathrm{ have }v\in\mathrm{ keys (tail p)
            by (simp add: lookup-tail-2 in-keys-iff)
    from is-red-indI2[OF }\langlep\not=0\rangleimp[OF this]] show ?thesis
    qed
qed
lemma is-red-addsE':
    assumes is-red Fp
    shows }\existsf\inF.\existsv\inkeys p.f\not=0\wedgeltfaddst
    using assms
proof (induction p rule: poly-mapping-tail-induct)
    case 0
    with irred-0[of F] show ?case by simp
next
    case (tail p)
    from is-red-indE[OF〈is-red F p`] show ?case
    proof
        assume }\existsf\inF.f\not=0\wedgeltfadds\mp@subsup{s}{t}{}lt 
        then obtain f}\mathrm{ where f}\inF\mathrm{ and }f\not=0\mathrm{ and lt f addst lt p by auto
        show ?case
        proof
            show \existsv\inkeys p.f\not=0\wedgelt faddstv
            proof (intro bexI, intro conjI)
```

```
            from }\langlep\not=0\rangle\mathrm{ show lt p keys p by (metis in-keys-iff lc-def lc-not-0)
            qed (rule }\langlef=0\rangle\mathrm{ , rule <lt f addst lt p>)
        qed (rule }\langlef\inF\rangle
    next
        assume is-red F (tail p)
        from tail.IH[OF this] obtain fv
            where f\inF and f\not=0 and v-in-keys-tail: v \in keys (tail p) and lt f addst
v \mp@code { b y ~ a u t o }
    from tail.hyps v-in-keys-tail have v-in-keys: v\in keys p by (metis lookup-tail
in-keys-iff)
    show ?case
    proof
        show \exists}|\inkeys p.f\not=0\wedgelt faddst 
            by (intro bexI, intro conjI, rule }\langlef\not=0\rangle\mathrm{ , rule <lt f addst v>, rule v-in-keys)
    qed (rule}\langlef\inF`
    qed
qed
lemma is-red-addsE:
    assumes is-red F p
    obtains fv where f\inF and v\in keys p and f\not=0 and lt f addst v
    using is-red-addsE'[OF assms] by auto
lemma is-red-adds-iff:
    shows (is-red F p)\longleftrightarrow \longleftrightarrow (\existsf\inF.\existsv\inkeys p.f\not=0^lt f addst v)
    using is-red-addsE' is-red-addsI by auto
lemma is-red-subset:
    assumes red:is-red A p and sub: A\subseteqB
    shows is-red B p
proof -
    from red obtain fv}\mathrm{ where f}\inA\mathrm{ and }v\inkeys p and f\not=0 and lt f addstv 
by (rule is-red-addsE)
    show ?thesis by (rule is-red-addsI, rule, fact+)
qed
lemma not-is-red-empty: ᄀ is-red {} f
    by (simp add: is-red-adds-iff)
lemma red-single-mult-const:
    assumes red-single p qft and c\not=0
    shows red-single p q(monom-mult c 0f)t
proof -
    let ?s = t\opluslt f
    let ?f = monom-mult c 0f
    from assms(1) have f}\not=0\mathrm{ and lookup p ?s }\not=
            and q=p - monom-mult ((lookup p ?s) / lc f) t f by (simp-all add:
red-single-def)
    from this(1) assms(2) have lt:lt ?f = lt f and lc:lc ?f = c*lcf
```

```
    by (simp add: lt-monom-mult term-simps, simp)
    show ?thesis unfolding red-single-def
    proof (intro conjI)
    from }\langlef\not=0\rangle\operatorname{assms(2) show ?f }\not=0\mathrm{ by (simp add: monom-mult-eq-zero-iff)
    next
    from〈lookup p ?s \not= 0> show lookup p (t\opluslt ?f) \not=0 by (simp add:lt)
    next
    show q=p - monom-mult (lookup p (t\opluslt ?f) / lc ?f) t ?f
        by (simp add: lt monom-mult-assoc lc assms(2), fact)
    qed
qed
lemma red-rtrancl-plus-higher:
    assumes (red F)** p q and \uv.u k keys p\Longrightarrowv\inkeys r\Longrightarrowu < < v
    shows (red F)** (p+r) (q+r)
    using assms(1)
proof induct
    case base
    show ?case ..
next
    case (step y z)
    from step(1) have }y\mp@subsup{\preceq}{p}{}p\mathrm{ by (rule red-rtrancl-ord)
    hence lt y \preceq_t lt p by (rule ord-p-lt)
    from step(2) have red F}(y+r)(z+r
    proof (rule red-plus-keys-disjoint)
        show keys y \cap keys r={}
        proof (rule ccontr)
            assume keys y \cap keys r}\not={
            then obtain v}\mathrm{ where v}\mathrm{ veys y and vekeys r by auto
                    from this(1) have v \preceq_ lt y and y}\not=0\mathrm{ using lt-max by (auto simp:
in-keys-iff)
            with }\langley\mp@subsup{\preceq}{p}{}\mp@subsup{}{}{p}\rangle\mathrm{ have }p\not=0\mathrm{ using ord-p-zero-min[of y] by auto
            hence lt p k keys p by (rule lt-in-keys)
            from this }\langlev\in\mathrm{ keys }r\rangle\mathrm{ have lt p}\mp@subsup{\prec}{t}{}v\mathrm{ by (rule assms(2))
            with <lt y \preceq\preceqt lt p> have lt y }\mp@subsup{\prec}{t}{}v\mathrm{ by simp
            with }\langlev\mp@subsup{\preceq}{t}{}lty\rangle\mathrm{ show False by simp
        qed
    qed
    with step(3) show ?case ..
qed
lemma red-mult-scalar-leading-monomial: (red {f})** (p\odot monomial (lc f) (lt f))
(-p\odot tail f)
proof (cases f=0)
    case True
    show ?thesis by (simp add: True lc-def)
next
    case False
    show ?thesis
```

```
proof (induct p rule: punit.poly-mapping-tail-induct)
    case 0
    show ?case by simp
next
    case (tail p)
    from False have lc \(f \neq 0\) by (rule lc-not-0)
    from tail(1) have punit.lc \(p \neq 0\) by (rule punit.lc-not-0)
    let \(? t=\) punit.tail \(p \odot\) monomial \((l c f)(l t f)\)
    let \(? m=\) monom-mult \((\) punit.lc \(p)(\) punit.lt \(p)(\) monomial \((l c f)(l t f))\)
    from \(\langle l c f \neq 0\rangle\) have kt: keys ? \(t=(\lambda t . t \oplus l t f)\) 'keys \((\) punit.tail \(p)\)
        by (rule keys-mult-scalar-monomial-right)
    have \(k m\) : keys ? \(m=\{\) punit.lt \(p \oplus l t f\}\)
        by (simp add: keys-monom-mult \([\) OF 〈punit.lc \(p \neq 0\rangle]\langle l c f \neq 0\rangle)\)
    from tail(2) have \((\text { red }\{f\})^{* *}(? t+? m)(-\) punit.tail \(p \odot\) tail \(f+? m)\)
    proof (rule red-rtrancl-plus-higher)
        fix \(u v\)
        assume \(u \in\) keys ?t and \(v \in\) keys ?m
        from this(1) obtain \(s\) where \(s \in\) keys (punit.tail \(p\) ) and \(u: u=s \oplus l t f\)
unfolding \(k t\)..
    from this(1) have punit.tail \(p \neq 0\) and \(s \preceq\) punit.lt (punit.tail \(p\) ) using
punit.lt-max by (auto simp: in-keys-iff)
    moreover from 〈punit.tail \(p \neq 0\) 〉 have punit.lt (punit.tail \(p\) ) \(\prec\) punit.lt \(p\)
by (rule punit.lt-tail)
    ultimately have \(s \prec\) punit.lt \(p\) by simp
    moreover from \(\langle v \in\) keys \(? m\rangle\) have \(v=\) punit.lt \(p \oplus l t f\) by (simp only:
km, simp)
    ultimately show \(u \prec_{t} v\) by (simp add: u splus-mono-strict-left)
    qed
    hence \(*:(\text { red }\{f\})^{* *}(p \odot\) monomial \((l c f)(l t f))(? m-p u n i t . t a i l ~ p \odot t a i l f)\)
    by (simp add: punit.leading-monomial-tail[symmetric, of p] mult-scalar-monomial[symmetric]
                mult-scalar-distrib-right[symmetric] add.commute[of punit.tail p])
    have red \(\{f\}\) ? \(m(-(\) monomial \((\) punit.lc \(p)(\) punit.lt \(p)) \odot\) tail \(f)\) unfolding
red-singleton
    proof
    show red-single ? \(m(-(\) monomial \((\) punit.lc \(p)(\) punit.lt \(p)) \odot\) tail f) \(f(\) punit.lt
p)
    proof (simp add: red-single-def \(\langle f \neq 0\rangle k m\) lookup-monom-mult \(\langle l c f \neq 0\rangle\)
〈punit.lc \(p \neq 0\rangle\) term-simps,
                simp add: monom-mult-dist-right-minus[symmetric] mult-scalar-monomial)
            have monom-mult (punit.lc p) (punit.lt p) (monomial \((l c f)(l t f)-f)=\)
                - monom-mult (punit.lc p) (punit.lt p) (f-monomial (lc f) (lt f))
            by (metis minus-diff-eq monom-mult-uminus-right)
            also have \(\ldots=-\) monom-mult (punit.lc \(p\) ) (punit.lt p) (tail f) by (simp
only: tail-alt-2)
            finally show - monom-mult (punit.lc p) (punit.lt p) (tail f) =
                monom-mult (punit.lc \(p\) ) (punit.lt \(p\) ) (monomial (lc \(f\) ) (lt \(f\) ) -
f) by \(\operatorname{simp}\)
    qed
qed
```

```
    hence red {f} (?m + (- punit.tail p \odot tail f))
                (- (monomial (punit.lc p) (punit.lt p)) \odot tail f + (- punit.tail p
\odottail f))
    proof (rule red-plus-keys-disjoint)
        show keys ?m \cap keys (- punit.tail p \odot tail f) = {}
        proof (cases punit.tail p=0)
            case True
            show ?thesis by (simp add: True)
        next
            case False
            from tail(2) have - punit.tail p \odot tail f \preceq_ ? ?t by (rule red-rtrancl-ord)
            hence lt (- punit.tail p \odot tail f) \preceq\preceq lt ?t by (rule ord-p-lt)
            also from <lc f\not=0\rangle False have ... = punit.lt (punit.tail p) }\opluslt 
                by (rule lt-mult-scalar-monomial-right)
                also from punit.lt-tail[OF False] have ... }\mp@subsup{\prec}{t}{}\mathrm{ punit.lt p }\oplus\mathrm{ lt f by (rule
splus-mono-strict-left)
            finally have punit.lt p}\oplusltf\not\inkeys(- punit.tail p \odot tail f) using lt-gr-key
by blast
            thus ?thesis by (simp add: km)
            qed
    qed
    hence red {f} (?m - punit.tail p \odot tail f)
                (- (monomial (punit.lc p)(punit.lt p)) \odot tail f - punit.tail p \odot tail f)
        by (simp add: term-simps)
    also have ... = - p\odot tail f using punit.leading-monomial-tail[symmetric, of
p]
    by (metis (mono-tags, lifting) add-uminus-conv-diff minus-add-distrib mult-scalar-distrib-right
            mult-scalar-minus-mult-left)
    finally have red {f}(?m - punit.tail p \odot tail f) (-p\odot tail f).
    with * show ?case ..
    qed
qed
corollary red-mult-scalar-lt:
    assumes f}\not=
    shows (red {f})** (p\odot monomial c (lt f)) (monom-mult (- c / lc f) 0 (p\odot
tail f))
proof -
    from assms have lc f}\not=0\mathrm{ by (rule lc-not-0)
    hence 1:p\odot monomial c (lt f) = punit.monom-mult (c/lcf) 0p \odot monomial
(lc f) (lt f)
    by (simp add: punit.mult-scalar-monomial[symmetric] mult.commute
        mult-scalar-assoc mult-scalar-monomial-monomial term-simps)
    have 2: monom-mult (-c/lc f) 0 (p\odot tail f) = - punit.monom-mult (c / lc
f) 0p}\odot\operatorname{tail}
    by (simp add: times-monomial-left[symmetric] mult-scalar-assoc
                monom-mult-uminus-left mult-scalar-monomial)
    show ?thesis unfolding 12 by (fact red-mult-scalar-leading-monomial)
qed
```

lemma is-red-monomial-iff: is-red $F$ (monomial c $v) \longleftrightarrow(c \neq 0 \wedge(\exists f \in F . f \neq$ $\left.0 \wedge l t f a d d s_{t} v\right)$ )
by (simp add: is-red-adds-iff)
lemma is-red-monomialI:
assumes $c \neq 0$ and $f \in F$ and $f \neq 0$ and $l t f a d d s_{t} v$
shows is-red $F$ (monomial c v)
unfolding is-red-monomial-iff using assms by blast
lemma is-red-monomialD:
assumes is-red $F$ (monomial c v)
shows $c \neq 0$
using assms unfolding is-red-monomial-iff ..
lemma is-red-monomialE:
assumes is-red $F$ (monomial c $v$ )
obtains $f$ where $f \in F$ and $f \neq 0$ and $l t f a d d s_{t} v$
using assms unfolding is-red-monomial-iff by blast
lemma replace-lt-adds-stable-is-red:
assumes red: is-red $F f$ and $q \neq 0$ and lt $q$ addst $l t p$
shows is-red (insert $q(F-\{p\})) f$
proof -
from red obtain $g v$ where $g \in F$ and $g \neq 0$ and $v \in$ keys $f$ and lt $g$ adds $s_{t} v$ by (rule is-red-addsE)
show ?thesis
proof (cases $g=p$ )
case True
show ?thesis
proof (rule is-red-addsI)
show $q \in \operatorname{insert} q(F-\{p\})$ by $\operatorname{simp}$
next
have $l t q$ addst $l t p$ by fact
also have ... addst $v$ using $\left\langle l t ~ g a d d s_{t} v\right\rangle$ unfolding True .
finally show $l t q a d d s_{t} v$.
qed (fact+)
next
case False
with $\langle g \in F\rangle$ have $g \in$ insert $q(F-\{p\})$ by blast
from this $\langle g \neq 0\rangle\langle v \in$ keys $f\rangle\left\langle l t g\right.$ adds $\left.s_{t} v\right\rangle$ show ?thesis by (rule is-red-addsI)
qed
qed
lemma conversion-property:
assumes is-red $\{p\} f$ and red $\{r\} p q$
shows is-red $\{q\} f \vee$ is-red $\{r\} f$
proof -
let ? $s=l p p-l p r$
from $\langle i s$-red $\{p\} f\rangle$ obtain $v$ where $v \in k e y s f$ and lt $p$ adds $s_{t} v$ and $p \neq 0$
by (rule is-red-addsE, simp)
from red-ind $E[O F\langle r e d\{r\} \quad p \quad q\rangle]$
have $\left(r \neq 0 \wedge l t r a d d s_{t}\right.$ lt $p \wedge q=p$ - monom-mult (lc $\left.p / l c r\right)$ ?s $\left.r\right) \vee$ red $\{r\}($ tail $p)(q-$ monomial (lc $p)(l t p))$ by simp
thus ?thesis
proof
assume $r \neq 0 \wedge l t r a d d s_{t} l t p \wedge q=p-$ monom-mult (lc $p / l c r$ ) ?s $r$
hence $r \neq 0$ and $l t r$ addst ${ }_{t}$ lt $p$ by simp-all
show ?thesis by (intro disjI2, rule is-red-singleton-trans, rule $\langle i s-r e d ~\{p\} f\rangle$,

## fact+)

next
assume red $\{r\}($ tail $p)(q-$ monomial $(l c p)(l t p))\left(\right.$ is red - ? $\left.p^{\prime} ? q^{\prime}\right)$
with red-ord have ? $q^{\prime} \prec_{p} ? p^{\prime}$.
hence ? $p^{\prime} \neq 0$
and assm: $\left(? q^{\prime}=0 \vee\left(\left(l t ? q^{\prime}\right) \prec_{t}\left(l t ? p p^{\prime}\right) \vee\left(l t ? q^{\prime}\right)=\left(l t ? p^{\prime}\right)\right)\right)$
unfolding ord-strict-p-rec[of ? $q^{\prime}$ ?p $]$ by (auto simp add: Let-def lc-def)
have $l t$ ? $p^{\prime} \prec_{t} l t p$ by (rule lt-tail, fact)
let $? m=$ monomial $(l c p)(l t p)$
from monomial-0D[of lt p lc p] lc-not- $0[O F\langle p \neq 0\rangle]$ have $? m \neq 0$ by blast
have $l t ? m=l t p$ by (rule $l t$-monomial, rule lc-not- 0 , fact)
have $q \neq 0 \wedge l t q=l t p$
proof (cases ? $q^{\prime}=0$ )
case True
hence $q=$ ? $m$ by simp
with $\langle ? m \neq 0\rangle\langle l t ? m=l t p\rangle$ show ?thesis by simp
next
case False
from assm show ?thesis
proof
assume $\left(l t ? q^{\prime}\right) \prec_{t}\left(l t ? p^{\prime}\right) \vee\left(l t ? q^{\prime}\right)=\left(l t ? p^{\prime}\right)$
hence $l t ? q^{\prime} \preceq_{t} l t$ ? $p^{\prime}$ by auto
also have $\ldots \prec_{t}$ lt $p$ by fact
finally have $l t ? q^{\prime} \prec_{t}$ lt $p$.
hence $l t ? q^{\prime} \prec_{t} l t$ ? $m$ unfolding $\langle l t$ ? $m=l t ~ p\rangle$.
from lt-plus-eqI[OF this] $\langle l t ? m=l t p\rangle$ have $l t q=l t p$ by simp
show ?thesis
proof (intro conjI, rule ccontr)
assume $\neg q \neq 0$
hence $q=0$ by $\operatorname{simp}$
hence ? $q^{\prime}=-$ ?m by simp
hence $l t ? q^{\prime}=l t(-? m)$ by $\operatorname{simp}$
also have $\ldots=l t$ ? $m$ using $l t$-uminus .
finally have $l t ? q^{\prime}=l t ? m$.
with $\left\langle l t ? q^{\prime} \prec_{t} l t\right.$ ? $\left.m\right\rangle$ show False by simp qed (fact)
next
assume $? q^{\prime}=0$
with False show ?thesis ..

```
        qed
    qed
    hence q\not=0 and lt qaddst lt p by (simp-all add: term-simps)
    show ?thesis by (intro disjI1, rule is-red-singleton-trans, rule 〈is-red {p}f>,
fact+)
    qed
qed
lemma replace-red-stable-is-red:
    assumes a1: is-red Ff and a2: red (F-{p}) pq
    shows is-red (insert q(F-{p}))f(is is-red ?F'f)
proof -
    from a1 obtain g}\mathrm{ where g}\inF\mathrm{ and is-red {g}f by (rule is-red-singletonI)
    show ?thesis
    proof (cases g=p)
        case True
        from a2 obtain h where h\inF-{p} and red {h} pq unfolding red-def
by auto
    from <is-red {g} f> have is-red {p}f unfolding True .
    have is-red {q} f\veeis-red {h} f by (rule conversion-property, fact+)
    thus ?thesis
    proof
            assume is-red {q} f
            show ?thesis
            proof (rule is-red-singletonD)
                show q}\in
            qed fact
    next
            assume is-red {h}f
            show ?thesis
            proof (rule is-red-singletonD)
                from <h\inF-{p}> show h\in?F' by simp
            qed fact
    qed
    next
        case False
        show ?thesis
        proof (rule is-red-singletonD)
            from }\langleg\inF\rangle\mathrm{ False show }g\in?\mp@subsup{F}{}{\prime}\mathrm{ by blast
        qed fact
    qed
qed
lemma is-red-map-scale:
    assumes is-red F (c\cdotp)
    shows is-red Fp
proof -
    from assms obtain fu}\mathrm{ where f}\inF\mathrm{ and }u\inkeys (c\cdotp) and f\not=
        and a:lt f addst u by (rule is-red-addsE)
```

```
    from this(2) keys-map-scale-subset have u \in keys p ..
    with }\langlef\inF\rangle\langlef\not=0\rangle\mathrm{ show ?thesis using a by (rule is-red-addsI)
qed
corollary is-irred-map-scale: \negis-red F p\Longrightarrow \neg is-red F (c\cdotp)
    by (auto dest: is-red-map-scale)
lemma is-red-map-scale-iff:is-red F (c\cdotp)\longleftrightarrow(c\not=0^ is-red Fp)
proof (intro iffI conjI notI)
    assume is-red F (c\cdotp) and c=0
    thus False by (simp add: irred-0)
next
    assume is-red F (c\cdotp)
    thus is-red Fp by (rule is-red-map-scale)
next
    assume c\not=0^is-red Fp
    hence is-red F (inverse c.c.p) by (simp add: map-scale-assoc)
    thus is-red F (c\cdotp) by (rule is-red-map-scale)
qed
lemma is-red-uminus: is-red F (-p)\longleftrightarrow is-red F p
    by (auto elim!: is-red-addsE simp: keys-uminus intro: is-red-addsI)
lemma is-red-plus:
    assumes is-red F(p+q)
    shows is-red Fp\vee is-red Fq
proof -
    from assms obtain fu where f\inF and u\in keys ( }p+q)\mathrm{ and f}=
        and a:lt f addst u by (rule is-red-addsE)
    from this(2) have }u\in\mathrm{ keys }p\cup\mathrm{ keys q
        by (meson Poly-Mapping.keys-add subsetD)
    thus ?thesis
    proof
        assume u\in keys p
        with}\langlef\inF\rangle\langlef\not=0\rangle\mathrm{ have is-red F p using a by (rule is-red-addsI)
        thus ?thesis..
    next
        assume u\in keys q
        with }\langlef\inF\rangle\langlef\not=0\rangle\mathrm{ have is-red F q using a by (rule is-red-addsI)
        thus ?thesis ..
    qed
qed
lemma is-irred-plus: ᄀis-red F p\Longrightarrow\neg is-red Fq\Longrightarrow \negis-red F (p+q)
    by (auto dest: is-red-plus)
lemma is-red-minus:
    assumes is-red F (p-q)
    shows is-red Fp\veeis-red Fq
```

```
proof -
    from assms have is-red F(p+(-q)) by simp
    hence is-red Fp\vee is-red F (-q) by (rule is-red-plus)
    thus ?thesis by (simp only: is-red-uminus)
qed
lemma is-irred-minus: }\neg\mathrm{ is-red F p ב ᄀis-red F q }\Longrightarrow\neg\mathrm{ is-red F (p-q)
    by (auto dest: is-red-minus)
end
```


### 4.6 Well-foundedness and Termination

```
context gd-term
```

begin
lemma dgrad-set-le-red-single:
assumes dickson-grading $d$ and red-single $p q f t$
shows dgrad-set-le d $\{t\}$ (pp-of-term'keys $p$ )
proof (rule dgrad-set-leI, simp)
have $t$ adds $t+l p f$ by simp
with $\operatorname{assms}(1)$ have $d t \leq d(p p$-of-term $(t \oplus l t f))$
by (simp add: term-simps, rule dickson-grading-adds-imp-le)
moreover from assms(2) have $t \oplus l t f \in$ keys $p$ by (simp add: in-keys-iff
red-single-def)
ultimately show $\exists v \in k e y s p . d t \leq d(p p$-of-term $v)$..
qed
lemma dgrad-p-set-le-red-single:
assumes dickson-grading $d$ and red-single $p q f t$
shows dgrad-p-set-le $d\{q\}\{f, p\}$
proof -
let ?f $=$ monom-mult $((l o o k u p ~ p(t \oplus l t f)) / l c f) t f$
from assms(2) have $t \oplus l t f \in$ keys $p$ and $q: q=p-$ ?f by (simp-all add:
red-single-def in-keys-iff)
have dgrad-p-set-le $d\{q\}\{p, ? f\}$ unfolding $q$ by (fact dgrad-p-set-le-minus)
also have dgrad-p-set-le d ... $\{f, p\}$
proof (rule dgrad-p-set-leI-insert)
from $\operatorname{assms}(1)$ have dgrad-set-le $d$ (pp-of-term'keys ?f) (insert $t$ ( $p$ p-of-term
‘ keys $f$ ))
by (rule dgrad-set-le-monom-mult)
also have dgrad-set-le d ... (pp-of-term' (keys $f \cup$ keys $p)$ )
proof (rule dgrad-set-leI, simp)
fix $s$
assume $s=t \vee s \in p p$-of-term'keys $f$
thus $\exists u \in k e y s f \cup$ keys $p$. $d s \leq d(p p$-of-term $u)$
proof
assume $s=t$
from assms have dgrad-set-le $d\{s\}$ (pp-of-term'keys $p$ ) unfolding $\prec s=$
by (rule dgrad-set-le-red-single)
moreover have $s \in\{s\}$..
ultimately obtain $s 0$ where $s 0 \in p p$-of-term' keys $p$ and $d s \leq d s 0$ by
(rule dgrad-set-leE)
from this(1) obtain $u$ where $u \in$ keys $p$ and $s 0=p p$-of-term $u .$.
from this(1) have $u \in$ keys $f \cup$ keys $p$ by simp
with $\langle d s \leq d s 0\rangle$ show ?thesis unfolding $\langle s 0=p p$-of-term $u\rangle .$.
next
assume $s \in p p$-of-term' keys $f$
hence $s \in p p$-of-term' (keys $f \cup$ keys $p$ ) by blast
then obtain $u$ where $u \in$ keys $f \cup$ keys $p$ and $s=p p$-of-term $u$..
note this(1)
moreover have $d s \leq d s$..
ultimately show ?thesis unfolding $\langle s=p p$-of-term $u\rangle$..
qed
qed
finally show dgrad-p-set-le $d\{? f\}\{f, p\}$ by (simp add: dgrad-p-set-le-def
Keys-insert)
next
show dgrad-p-set-le $d\{p\}\{f, p\}$ by (rule dgrad-p-set-le-subset, simp)
qed
finally show ?thesis .
qed
lemma dgrad-p-set-le-red:
assumes dickson-grading $d$ and red $F p q$
shows dgrad-p-set-le $d\{q\}$ (insert $p F$ )
proof -
from assms(2) obtain $f t$ where $f \in F$ and red-single $p q f t$ by (rule red-setE)
from $\operatorname{assms}(1)$ this(2) have dgrad-p-set-le $d\{q\}\{f, p\}$ by (rule dgrad-p-set-le-red-single)
also have dgrad-p-set-le $d \ldots$ (insert $p F)$ by (rule dgrad-p-set-le-subset, auto
intro: $\langle f \in F\rangle$ )
finally show ?thesis.
qed
corollary dgrad-p-set-le-red-rtrancl:
assumes dickson-grading $d$ and $(\text { red } F)^{* *} p q$
shows dgrad-p-set-le $d\{q\}$ (insert $p F$ )
using assms(2)
proof (induct)
case base
show ?case by (rule dgrad-p-set-le-subset, simp)
next
case (step y $z$ )
from $\operatorname{assms}(1) \operatorname{step}(2)$ have dgrad-p-set-le $d\{z\}$ (insert y $F$ ) by (rule dgrad-p-set-le-red)
also have dgrad-p-set-le $d \ldots$... insert p $F$ )
proof (rule dgrad-p-set-leI-insert)
show dgrad-p-set-le d F (insert p F) by (rule dgrad-p-set-le-subset, blast)
qed fact
finally show ?case .
qed
lemma dgrad-p-set-red-single-pp:
assumes dickson-grading $d$ and $p \in d$ grad- $p$-set $d m$ and red-single $p q f t$
shows $d t \leq m$
proof -
from $\operatorname{assms}(1) \operatorname{assms}(3)$ have dgrad-set-le $d\{t\}$ ( $p p$-of-term' keys $p$ ) by (rule dgrad-set-le-red-single)
moreover have $t \in\{t\}$..
ultimately obtain $s$ where $s \in p p$-of-term' keys $p$ and $d t \leq d s$ by (rule dgrad-set-leE)
from this(1) obtain $u$ where $u \in$ keys $p$ and $s=p p$-of-term $u$..
from $\operatorname{assms}(2)$ this(1) have $d(p p$-of-term $u) \leq m$ by (rule dgrad-p-setD)
with $\langle d t \leq d s\rangle$ show ?thesis unfolding $\langle s=p p$-of-term $u\rangle$ by (rule le-trans)
qed
lemma dgrad-p-set-closed-red-single:
assumes dickson-grading $d$ and $p \in d g r a d-p$-set $d m$ and $f \in d g r a d-p$-set $d m$ and red-single $p q f t$
shows $q \in d g r a d$ - $p$-set $d m$
proof -
from dgrad-p-set-le-red-single $[O F \operatorname{assms}(1,4)]$ have $\{q\} \subseteq d g r a d-p$-set $d m$ proof (rule dgrad-p-set-le-dgrad-p-set)
from $\operatorname{assms}(2,3)$ show $\{f, p\} \subseteq d g r a d-p$-set $d m$ by simp
qed
thus?thesis by simp
qed
lemma dgrad-p-set-closed-red:
assumes dickson-grading $d$ and $F \subseteq$ dgrad- $p$-set $d m$ and $p \in d g r a d-p$-set $d m$ and red $F p q$
shows $q \in$ dgrad- $p$-set $d m$
proof -
from $\operatorname{assms}(4)$ obtain $f t$ where $f \in F$ and $*$ : red-single $p q f t$ by (rule red-setE)
from $\operatorname{assms}(2)$ this(1) have $f \in$ dgrad-p-set $d m$..
from $\operatorname{assms}(1)$ assms(3) this * show ?thesis by (rule dgrad-p-set-closed-red-single)
qed
lemma dgrad- $p$-set-closed-red-rtrancl:
assumes dickson-grading $d$ and $F \subseteq d g r a d$ - $p$-set $d m$ and $p \in d g r a d$ - $p$-set $d m$
and $(\text { red } F)^{* *} p q$
shows $q \in$ dgrad- $p$-set $d m$
using assms(4)
proof (induct)
case base
from assms(3) show ?case.

```
next
    case (step r q)
    from assms(1) assms(2) step(3) step(2) show q\indgrad-p-set d m by (rule
dgrad-p-set-closed-red)
qed
lemma red-rtrancl-repE:
    assumes dickson-grading d and G\subseteqdgrad-p-set d m and finite G and p\in
dgrad-p-set d m
    and (red G)** pr
    obtains q where p=r+(\sumg\inG.qg\odotg) and \g.qg 的彷it.dgrad-p-set
dm
    and \g.lt (qg\odotg) \preceq_ lt p
    using assms(5)
proof (induct r arbitrary: thesis)
    case base
    show ?case
    proof (rule base)
            show }p=p+(\sumg\inG.0\odotg) by sim
    qed (simp-all add: punit.zero-in-dgrad-p-set min-term-min)
next
    case (step r'r)
    from step.hyps(2) obtain gt where g\inG and rs:red-single r'r rgt by (rule
red-setE)
    from this(2) have r'=r+monomial (lookup r'}(t\opluslt g)/lcg)t\odot
        by (simp add: red-single-def mult-scalar-monomial)
    moreover define q0 where q0 = monomial (lookup r'}(t\opluslt g) / lc g)
    ultimately have r': r' = r + q0 \odot g by simp
    obtain q' where p:p= r'+(\sumg\inG. q' g\odotg) and 1:\bigwedgeg. q'g\inpunit.dgrad-p-set
dm
    and 2: \bigwedgeg.lt ( q' g\odotg) \preceq́t lt p by (rule step.hyps) blast
    define q}\mathrm{ where }q=\mp@subsup{q}{}{\prime}(g:=q0+\mp@subsup{q}{}{\prime}g
    show ?case
    proof (rule step.prems)
        from assms(3)<g\inG> have p=(r+q0\odotg)+(\mp@subsup{q}{}{\prime}g\odotg+(\sumg\inG-
{g}. q' g\odotg))
            by (simp add: p r' sum.remove)
        also have ... =r + (qg\odotg+(\sumg\inG-{g}. q' g\odotg))
            by (simp add: q-def mult-scalar-distrib-right)
        also from refl have ( }\sumg\inG-{g}.\mp@subsup{q}{}{\prime}g\odotg)=(\sumg\inG-{g}.qg\odotg
            by (rule sum.cong) (simp add: q-def)
        finally show p=r+(\sumg\inG.qg\odotg) using assms(3)<g\inG` by (simp
only: sum.remove)
    next
        fix g0
        have qg0 \in punit.dgrad-p-set d m}\wedgelt (qg0\odotg0)\preceq⿱㇒l_lt 
        proof (cases g0=g)
            case True
            have eq: q g = q0 + q' g by (simp add: q-def)
```

```
    show ?thesis unfolding True eq
    proof
        from assms(1, 2, 4) step.hyps(1) have r}\mp@subsup{r}{}{\prime}\indgrad-p-set d m
            by (rule dgrad-p-set-closed-red-rtrancl)
        with assms(1) have d t\leqm using rs by (rule dgrad-p-set-red-single-pp)
    hence q0 \in punit.dgrad-p-set d m by (simp add: q0-def punit.dgrad-p-set-def
dgrad-set-def)
    thus q0 + q' g\in punit.dgrad-p-set d m by (intro punit.dgrad-p-set-closed-plus
1)
    next
        have lt (q0\odotg+ q'g\odotg) \preceq́t ord-term-lin.max (lt (q0\odotg)) (lt (q'g
g))
            by (fact lt-plus-le-max)
            also have ... \preceq
            proof (intro ord-term-lin.max.boundedI 2)
                have lt (q0 \odot g) \preceq́t t\opluslt g by (simp add: q0-def mult-scalar-monomial
lt-monom-mult-le)
            also from rs have ... \preceq́ lt r' by (intro lt-max) (simp add: red-single-def)
            also from step.hyps(1) have ... \preceq_ lt p by (intro ord-p-lt red-rtrancl-ord)
            finally show lt (q0 \odot g) \preceq́t lt p .
            qed
        finally show lt ((q0 + q'g)\odotg)\preceq\preceq}tlt p by (simp only: mult-scalar-distrib-right)
        qed
    next
        case False
        hence qg0 = q' g0 by (simp add: q-def)
        thus ?thesis by (simp add: 1 2)
    qed
    thus qg0 f punit.dgrad-p-set d m and lt (qg0\odotg0) \preceq_ lt p by simp-all
    qed
qed
lemma is-relation-order-red:
    assumes dickson-grading d
    shows Confluence.relation-order (red F) ( }\mp@subsup{\prec}{p}{})(\mathrm{ dgrad-p-set d m)
proof
    show wfp-on ( }\mp@subsup{\prec}{p}{})(\mathrm{ dgrad-p-set d m)
    proof (rule wfp-onI-min)
        fix }x:\mp@subsup{:}{}{\prime}t=\mp@subsup{=}{0}{\prime}c\mathrm{ and }
        assume }x\inQ\mathrm{ and }Q\subseteqdgrad-p-set d 
        with assms obtain q}\mathrm{ where }q\inQ\mathrm{ and *: \y. y }\mp@subsup{\prec}{p}{}q\Longrightarrowy\not\in
        by (rule ord-p-minimum-dgrad-p-set, auto)
    from this(1) show }\existsz\inQ.\forally\indgrad-p-set d m. y \mp@subsup{\prec}{p}{}z\longrightarrowy\not\in
    proof
        from * show }\forally\indgrad-p-set d m. y \mp@subsup{\prec}{p}{}q\longrightarrowy\not\inQ by aut
    qed
    qed
next
    show red F}\leq(\mp@subsup{\prec}{p}{}\mp@subsup{)}{}{-1-1}\mathrm{ by (simp add: predicate2I red-ord)
```

```
qed (fact ord-strict-p-transitive)
```

lemma red-wf-dgrad-p-set-aux:
assumes dickson-grading $d$ and $F \subseteq d g r a d-p$-set $d m$
shows wfp-on $(\text { red } F)^{-1-1}(d g r a d-p-s e t d m)$
proof (rule wfp-onI-min)
fix $x::^{\prime} t \Rightarrow_{0}{ }^{\prime} b$ and $Q$
assume $x \in Q$ and $Q \subseteq d g r a d-p$-set $d m$
with $\operatorname{assms}(1)$ obtain $q$ where $q \in Q$ and $*: \bigwedge y . y \prec_{p} q \Longrightarrow y \notin Q$
by (rule ord- $p$-minimum-dgrad- $p$-set, auto)
from this(1) show $\exists z \in Q . \forall y \in d g r a d-p$-set $d m$. (red $F)^{-1-1} y z \longrightarrow y \notin Q$
proof
show $\forall y \in d g r a d-p$-set $d m .(\text { red } F)^{-1-1} y q \longrightarrow y \notin Q$
proof (intro ballI impI, simp)
fix $y$
assume red $F q y$
hence $y \prec_{p} q$ by (rule red-ord)
thus $y \notin Q$ by (rule *)
qed
qed
qed
lemma red-wf-dgrad-p-set:
assumes dickson-grading $d$ and $F \subseteq$ dgrad-p-set $d m$
shows wfP $(\text { red } F)^{-1-1}$
proof (rule wfI-min[to-pred])
fix $x::^{\prime} t \Rightarrow_{0}{ }^{\prime} b$ and $Q$
assume $x \in Q$
from $\operatorname{assms}(2)$ obtain $n$ where $m \leq n$ and $x \in d g r a d-p$-set $d n$ and $F \subseteq$
dgrad-p-set $d n$
by (rule dgrad-p-set-insert)
let $? Q=Q \cap$ dgrad- $p$-set $d n$
from $\operatorname{assms}(1)\langle F \subseteq$ dgrad-p-set $d n\rangle$ have $w f p$-on $(\text { red } F)^{-1-1}($ dgrad- $p$-set $d$
n)
by (rule red-wf-dgrad-p-set-aux)
moreover from $\langle x \in Q\rangle\langle x \in$ dgrad- $p$-set $d n\rangle$ have $x \in$ ? $Q$..
moreover have ? $Q \subseteq d g r a d-p$-set $d n$ by simp
ultimately obtain $z$ where $z \in ? Q$ and $*: \bigwedge y$. $(\text { red } F)^{-1-1} y z \Longrightarrow y \notin ? Q$
by (rule wfp-onE-min) blast
from this(1) have $z \in Q$ and $z \in d g r a d-p-s e t d n$ by simp-all
from this(1) show $\exists z \in Q . \forall y .(\text { red } F)^{-1-1} y z \longrightarrow y \notin Q$
proof
show $\forall y$. $(\text { red } F)^{-1-1} y z \longrightarrow y \notin Q$
proof (intro allI impI)
fix $y$
assume $(\text { red } F)^{-1-1} y z$
hence red $F z y$ by simp
with $\operatorname{assms}(1)\langle F \subseteq d g r a d-p$-set $d n\rangle\langle z \in d g r a d$ - $p$-set $d n\rangle$ have $y \in d g r a d-p$-set
$d n$

```
                by (rule dgrad-p-set-closed-red)
            moreover from <(red F) -1-1 y z` have y\not\in?Q by (rule *)
            ultimately show y}\not\inQ\mathrm{ by blast
        qed
    qed
qed
```

lemmas red-wf-finite $=$ red-wf-dgrad-p-set[OF dickson-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemma cbelow-on-monom-mult:
assumes dickson-grading $d$ and $F \subseteq$ dgrad- $p$-set $d m$ and $d t \leq m$ and $c \neq 0$
and cbelow-on (dgrad-p-set $d m)\left(\prec_{p}\right) z(\lambda a b$. red $F a b \vee$ red $F b a) p q$
shows cbelow-on (dgrad-p-set d m) ( $\prec_{p}$ ) (monom-mult ctz) $(\lambda a b$. red $F a b \vee$
red $F b a)$
(monom-mult ctp)(monom-mult ctq)
using assms(5)
proof (induct rule: cbelow-on-induct)
case base
show ?case unfolding cbelow-on-def
proof (rule disjI1, intro conjI, fact refl)
from assms(5) have $p \in d g r a d-p$-set $d m$ by (rule cbelow-on-first-in)
with $\operatorname{assms(1)} \operatorname{assms}(3)$ show monom-mult c $t p \in d g r a d-p-s e t ~ d m$ by (rule
dgrad-p-set-closed-monom-mult)
next
from assms(5) have $p \prec_{p} z$ by (rule cbelow-on-first-below)
from this assms(4) show monom-mult ct $p \prec_{p}$ monom-mult c $t z$ by (rule
ord-strict-p-monom-mult)
qed
next
case (step $q^{\prime} q$ )
let $? R=\lambda a b$. red $F a b \vee r e d F b a$
from step(5) show ?case
proof
from assms(1) assms(3) step(3) show monom-mult ct $q \in \operatorname{dgrad-p-set~d~m}$
by (rule dgrad-p-set-closed-monom-mult)
next
from step(2) red-monom-mult $[O F-\operatorname{assms}(4)]$ show $? R$ (monom-mult ct $q^{\prime}$ )
(monom-mult ct $q$ ) by auto
next
from step (4) assms(4) show monom-mult c $t q \prec_{p}$ monom-mult c $t z$ by (rule
ord-strict-p-monom-mult)
qed
qed
lemma cbelow-on-monom-mult-monomial:
assumes $c \neq 0$
and cbelow-on (dgrad-p-set d $m$ ) $\left(\prec_{p}\right)\left(\right.$ monomial $\left.c^{\prime} v\right)(\lambda a b$. red $F a b \vee$ red
Fba) $p$ q
shows cbelow-on (dgrad-p-set d $m$ ) $\left(\prec_{p}\right)($ monomial $c(t \oplus v))(\lambda a b$. red $F a b$

```
\vee red F b a) pq
proof -
    have *: f}\mp@subsup{\prec}{p}{}\mathrm{ monomial c'v ఋf }\mp@subsup{\prec}{p}{\prime}\mathrm{ monomial c (t }\oplusv)\mathrm{ for }
    proof (simp add: ord-strict-p-monomial-iff assms(1), elim conjE disjE, erule
disjI1, rule disjI2)
    assume lt f}\mp@subsup{\prec}{t}{}
    also have ... \preceq}tt\mp@code{v using local.zero-min using splus-mono-left splus-zero
by fastforce
    finally show lt f}\mp@subsup{\prec}{t}{}t\oplusv
    qed
    from assms(2) show ?thesis
    proof (induct rule: cbelow-on-induct)
    case base
    show ?case unfolding cbelow-on-def
    proof (rule disjI1, intro conjI, fact refl)
            from assms(2) show p\indgrad-p-set d m by (rule cbelow-on-first-in)
    next
            from assms(2) have p}\mp@subsup{\prec}{p}{}\mathrm{ monomial c'v}\mathrm{ by (rule cbelow-on-first-below)
            thus }p\mp@subsup{\prec}{p}{}\mathrm{ monomial c}(t\oplusv)\mathrm{ by (rule *)
        qed
    next
    case (step q' q)
    let ?R = \lambdaab. red F ab\vee red Fba
    from step(5) step(3) step(2) show ?case
    proof
            from step(4) show q}\mp@subsup{\prec}{p}{}\mathrm{ monomial c (t }\oplusv)\mathrm{ by (rule *)
    qed
    qed
qed
lemma cbelow-on-plus:
    assumes dickson-grading d and F}\subseteqdgrad-p-set d m and r dgrad-p-set d m
    and keys r\cap keys z={}
    and cbelow-on (dgrad-p-set d m) ( }\mp@subsup{\prec}{p}{})z(\lambdaab.red Fab\vee red F b a) p
    shows cbelow-on (dgrad-p-set d m) (<~) (z+r) (\lambdaa b. red F a b\vee red F b a)
(p+r) (q+r)
    using assms(5)
proof (induct rule: cbelow-on-induct)
    case base
    show ?case unfolding cbelow-on-def
    proof (rule disjI1, intro conjI, fact refl)
        from assms(5) have p\indgrad-p-set d m by (rule cbelow-on-first-in)
    from this assms(3) show }p+r\indgrad-p-set d m by (rule dgrad-p-set-closed-plus
    next
        from assms(5) have p}\mp@subsup{\prec}{p}{}z\mathrm{ by (rule cbelow-on-first-below)
        from this assms(4) show p+r < < z + r by (rule ord-strict-p-plus)
    qed
next
    case (step q' q)
```

```
    let ?RS = \lambdaa b. red F a b\vee red F b a
    let ?A = dgrad-p-set d m
    let ?R = red F
    let ?ord = (}\mp@subsup{\prec}{p}{}
    from assms(1) have ro: relation-order ?R ?ord ?A
    by (rule is-relation-order-red)
    have dw: relation.dw-closed ?R ?A
            by (rule relation.dw-closedI, rule dgrad-p-set-closed-red, rule assms(1), rule
assms(2))
    from step(2) have relation.cs (red F) ( q' +r) (q+r)
    proof
        assume red Fqq'
        hence relation.cs (red F) (q+r) (q' +r) by (rule red-plus-cs)
        thus ?thesis by (rule relation.cs-sym)
    next
        assume red F q' q
        thus ?thesis by (rule red-plus-cs)
    qed
    with ro dw have cbelow-on ?A ?ord (z+r) ?RS ( }\mp@subsup{q}{}{\prime}+r)(q+r
    proof (rule relation-order.cs-implies-cbelow-on)
        from step(1) have q' &?A by (rule cbelow-on-second-in)
        from this assms(3) show q' +r\in?A by (rule dgrad-p-set-closed-plus)
    next
        from step(3) assms(3) show q+r\in?A by (rule dgrad-p-set-closed-plus)
    next
        from step(1) have q}\mp@subsup{q}{}{\prime}\mp@subsup{\prec}{p}{}z\mathrm{ by (rule cbelow-on-second-below)
        from this assms(4) show q' +r < p}z+r by (rule ord-strict-p-plus
    next
        from step(4) assms(4) show q+r \prec}\mp@subsup{\mp@code{p}}{}{2}+r\mathrm{ by (rule ord-strict-p-plus)
    qed
    with step(5) show ?case by (rule cbelow-on-transitive)
qed
lemma is-full-pmdlI-lt-dgrad-p-set:
    assumes dickson-grading d and B\subseteqdgrad-p-set dm
    assumes }\k.k\in\mathrm{ component-of-term' Keys (B::(' }t=\mp@subsup{=}{0}{\prime}'b::field) set)
            (\existsb\inB.b}=0\wedge\mathrm{ component-of-term (lt b) =k^lpb=0)
    shows is-full-pmdl B
proof (rule is-full-pmdlI)
    fix p::'t =0 'b
    from assms(1, 2) have wfP (red B) -1-1 by (rule red-wf-dgrad-p-set)
    moreover assume component-of-term'keys p\subseteqcomponent-of-term'Keys B
    ultimately show }p\inpmdl 
    proof (induct p)
        case (less p)
        show ?case
        proof (cases p=0)
            case True
            show ?thesis by (simp add: True pmdl.span-zero)
```

next
case False
hence lt $p \in$ keys $p$ by (rule lt-in-keys)
hence component-of-term (lt p) ) component-of-term' keys p by simp
also have $\ldots \subseteq$ component-of-term'Keys $B$ by fact
finally have $\exists b \in B . b \neq 0 \wedge$ component-of-term (lt b) $=$ component-of-term $(l t p) \wedge l p b=0$
by (rule assms(3))
then obtain $b$ where $b \in B$ and $b \neq 0$ and component-of-term (lt $b$ ) $=$ component-of-term (lt p)
and $l p b=0$ by blast
from this(3, 4) have eq: lp $p \oplus l t b=l t p$ by (simp add: splus-def term-of-pair-pair)
define $q$ where $q=p$ - monom-mult (lookup $p((l p p) \oplus l t b) / l c b)(l p p)$ $b$
have red-single $p q b(l p p)$
by (auto simp: red-single-def $\langle b \neq 0\rangle q$-def eq $\langle l t p \in$ keys $p\rangle$ )
with $\langle b \in B\rangle$ have red $B p q$ by (rule red-setI)
hence $(\text { red } B)^{-1-1} q p$..
moreover have component-of-term 'keys $q \subseteq$ component-of-term'Keys $B$
proof (rule subset-trans)
from 〈red B p q〉 show component-of-term‘keys $q \subseteq$ component-of-term‘
keys $p \cup$ component-of-term ' Keys $B$
by (rule components-red-subset)
next
from less(2) show component-of-term'keys $p \cup$ component-of-term'Keys
$B \subseteq$ component-of-term ' Keys $B$
by blast
qed
ultimately have $q \in p m d l B$ by (rule less.hyps)
have $q$ + monom-mult (lookup $p((l p p) \oplus l t b) / l c b)(l p p) b \in p m d l B$
by (rule pmdl.span-add, fact, rule pmdl-closed-monom-mult, rule pmdl.span-base, fact)
thus ?thesis by (simp add: $q$-def)
qed
qed
qed
lemmas $i s$-full-pmdlI-lt-finite $=$ is-full-pmdlI-lt-dgrad-p-set[OF dickson-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
end

### 4.7 Algorithms

### 4.7.1 Function find-adds

context ordered-term
begin


```
    find-adds [] - = None
    find-adds (f#fs)u=(if f\not=0\wedgelt faddst u then Some f else find-adds fs u}
lemma find-adds-SomeD1:
    assumes find-adds fs }u=\mathrm{ Some f
    shows f}\in\mathrm{ set fs
    using assms by (induct fs, simp, simp split: if-splits)
lemma find-adds-SomeD2:
    assumes find-adds fs }u=\mathrm{ Some f
    shows }f\not=
    using assms by (induct fs, simp, simp split: if-splits)
lemma find-adds-SomeD3:
    assumes find-adds fs u=Some f
    shows lt f addst u
    using assms by (induct fs, simp, simp split: if-splits)
lemma find-adds-NoneE:
    assumes find-adds fs }u=\mathrm{ None and f}\in\mathrm{ set fs
    assumes f=0\Longrightarrow thesis and f}=0\Longrightarrow\neglt faddst u\Longrightarrow thesi
    shows thesis
    using assms
proof (induct fs arbitrary: thesis)
    case Nil
    from Nil(2) show ?case by simp
next
case (Cons a fs)
from Cons(2) have 1:a=0 \ न lt a addst u and 2: find-adds fs u = None
    by (simp-all split: if-splits)
from Cons(3) have f=a\veef\in set fs by simp
thus ?case
proof
    assume f=a
    show ?thesis
    proof (cases a=0)
                case True
                show ?thesis by (rule Cons(4), simp add: <f =a`True)
    next
                case False
                with 1 have *: ᄀ lt a addst u by simp
                show ?thesis by (rule Cons(5), simp-all add:<f =a〉* False)
        qed
    next
    assume f}\in\mathrm{ set fs
    with 2 show ?thesis
    proof (rule Cons(1))
            assume f=0
```

```
        thus ?thesis by (rule Cons(4))
    next
        assume }f\not=0\mathrm{ and }\negltfaddst 
        thus ?thesis by (rule Cons(5))
    qed
    qed
qed
lemma find-adds-SomeD-red-single:
    assumes p\not=0 and find-adds fs (lt p) = Some f
    shows red-single p(tail p-monom-mult (lc p / lc f) (lp p-lpf) (tail f))f(lp
p-lpf)
proof -
    let ?f = monom-mult (lc p/lcf) (lp p-lpf)f
    from assms(2) have f\not=0 and lt f addst lt p by (rule find-adds-SomeD2, rule
find-adds-SomeD3)
    from this(2) have eq: (lp p-lpf)\opluslt f=lt p
        by (simp add: adds-minus-splus adds-term-def term-of-pair-pair)
    from assms(1) have lc p\not=0 by (rule lc-not-0)
    moreover from }\langlef\not=0\rangle\mathrm{ have lc f}\not=0\mathrm{ by (rule lc-not-0)
    ultimately have lc p / lc f =0 by simp
    hence lt ?f = (lp p-lpf)\opluslt f by (simp add:lt-monom-mult <f \not=0`)
    hence lt-f:lt ?f = lt p by (simp only: eq)
    have lookup ?f (lt p)= lookup ?f ((lp p - lp f)\opluslt f) by (simp only: eq)
    also have ... = (lc p / lc f)* lookup f(lt f) by (rule lookup-monom-mult-plus)
    also from }\langlelcf\not=0\rangle\mathrm{ have ... = lookup p (lt p) by (simp add:lc-def)
    finally have lc-f:lookup ?f (lt p)= lookup p (lt p).
    have red-single p (p-?f) f (lp p-lpf)
    by (auto simp: red-single-def eq lc-def 〈f }\not=0\ranglelt\mathrm{ -in-keys assms(1))
    moreover have p-?f = tail p - monom-mult (lc p / lc f) (lp p - lpf) (tail
f)
    by (rule poly-mapping-eqI,
    simp add: tail-monom-mult[symmetric] lookup-minus lookup-tail-2 lt-f lc-f
split:if-split)
    ultimately show ?thesis by simp
qed
lemma find-adds-SomeD-red:
    assumes p\not=0 and find-adds fs (lt p)=Some f
    shows red (set fs) p (tail p - monom-mult (lc p / lc f) (lp p - lp f) (tail f))
proof (rule red-setI)
    from assms(2) show f\in set fs by (rule find-adds-SomeD1)
next
    from assms show red-single p (tail p - monom-mult (lc p / lc f) (lp p - lp f)
(tail f)) f(lp p-lpf)
    by (rule find-adds-SomeD-red-single)
qed
end
```


### 4.7.2 Function $t r d$

```
context gd-term
begin
```

definition trd-term :: ('a nat) $\Rightarrow\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\right.$ field $)$ list $\times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}\right.$ 'b)) $\times$

$$
\left.\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \text { list } \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}^{\prime} ' b\right)\right)\right) \text { set }
$$

where trd-term $d=\{(x, y)$. dgrad-p-set-le d (set $(f s t($ snd $x) \#$ fst $x))($ set $(f s t$ $($ snd $y) \#$ fst $y)) \wedge$ fst $($ snd $x) \prec_{p}$ fst $($ snd $\left.y)\right\}$
lemma trd-term-wf:
assumes dickson-grading d
shows wf (trd-term d)
proof (rule wfI-min)
fix $x::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.$ field $)$ list $\times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ and $Q$
assume $x \in Q$
let ? $A=\operatorname{set}(f s t(s n d x) \# f s t x)$
have finite? A ..
then obtain $m$ where $A: ? A \subseteq d g r a d-p$-set $d m$ by (rule dgrad-p-set-exhaust)
let $? B=$ dgrad- $p$-set $d m$
let ? $Q=\{q \in Q$. set $(f s t($ snd $q) \# f s t q) \subseteq ? B\}$
note assms
moreover have fst (snd $x$ ) $\in f_{s t}$ ' snd ' ? $Q$
by (rule, fact refl, rule, fact refl, simp only: mem-Collect-eq $A\langle x \in Q\rangle$ )
moreover have $f s t$ ' snd ' ? $Q \subseteq$ ? B by auto
ultimately obtain $z 0$ where $z 0 \in f s t$ 'snd '? $Q$
and $*: ~ \bigwedge y . y \prec_{p} z 0 \Longrightarrow y \notin f s t$ 'snd'? $Q$ by (rule ord- $p$-minimum-dgrad- $p$-set, blast)
from this $(1)$ obtain $z$ where $z \in\{q \in Q$. set $(f s t(s n d q) \# f s t q) \subseteq ? B\}$ and $z 0: z 0=$ fst $($ snd $z)$
by fastforce
from this(1) have $z \in Q$ and $a$ : set (fst (snd $z) \#$ fst $z) \subseteq ? B$ by simp-all
from this(1) show $\exists z \in Q . \forall y .(y, z) \in$ trd-term $d \longrightarrow y \notin Q$
proof
show $\forall y .(y, z) \in$ trd-term $d \longrightarrow y \notin Q$
proof (intro allI impI)
fix $y$
assume $(y, z) \in$ trd-term $d$
hence b: dgrad-p-set-le d (set $(f s t(s n d y) \# f s t y))(\operatorname{set}(f s t(s n d z) \# f s t z))$
and $f$ st (snd $y$ ) $\prec_{p} z 0$
by (simp-all add: trd-term-def z0)
from this(2) have $f s t$ (snd $y$ ) $\notin f s t$ ' snd' $? Q$ by (rule *)
hence $y \notin Q \vee \neg \operatorname{set}(f s t(s n d y) \# f s t y) \subseteq$ ? $B$ by auto
moreover from $b a$ have set $(f s t(s n d y) \# f s t y) \subseteq ? B$ by (rule dgrad-p-set-le-dgrad-p-set)
ultimately show $y \notin Q$ by simp
qed
qed
qed

```
function trd-aux :: (' \(\left.t \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) field \()\)
where
    trd-aux fs pr\(=\)
        (if \(p=0\) then
            \(r\)
    else
                case find-adds fs (lt p) of
                    None \(\Rightarrow\) trd-aux fs \((\) tail \(p)(r+\) monomial (lc \(p)(l t p))\)
                Some \(f \Rightarrow\) trd-aux fs (tail \(p\) - monom-mult (lc p / lc f) (lp p-lpf) (tail
f)) \(r\)
    )
    by auto
termination proof -
    from ex-dgrad obtain \(d::^{\prime} a \Rightarrow\) nat where \(d g\) : dickson-grading \(d\)..
    let ? \(R=\) trd-term \(d\)
    show ?thesis
    proof (rule, rule trd-term-wf, fact)
    fix \(f s\) and \(p r::^{\prime} t \Rightarrow_{0}{ }^{\prime} b\)
    assume \(p \neq 0\)
    show \(((f s\), tail \(p, r+\) monomial \((l c p)(l t p)), f s, p, r) \in \operatorname{trd}\)-term d
    proof (simp add: trd-term-def, rule)
            show dgrad-p-set-le d (insert (tail p) (set fs)) (insert p (set fs))
                    proof (rule dgrad-p-set-leI-insert-keys, rule dgrad-p-set-le-subset, rule sub-
set-insertI,
                            rule dgrad-set-le-subset, simp add: Keys-insert image-Un)
            have keys (tail \(p\) ) \(\subseteq\) keys \(p\) by (auto simp: keys-tail)
            hence pp-of-term' keys (tail p) \(\subseteq p p\)-of-term' keys \(p\) by (rule image-mono)
                    thus pp-of-term' keys (tail p) \(\subseteq\) pp-of-term' keys \(p \cup p p\)-of-term'Keys
(set \(f s\) ) by blast
            qed
        next
                from \(\langle p \neq 0\rangle\) show tail \(p \prec_{p} p\) by (rule tail-ord- \(p\) )
            qed
    next
    fix \(f s::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list and \(p r f::^{\prime} t \Rightarrow_{0}{ }^{\prime} b\)
    assume \(p \neq 0\) and find-adds fs (lt \(p\) ) \(=\) Some \(f\)
    hence red (set fs) \(p\) (tail \(p\) - monom-mult (lc \(p / l c f)(l p p-l p f)(t a i l f))\)
            (is red - p?q) by (rule find-adds-SomeD-red)
    show \(((f s, ? q, r), f s, p, r) \in\) trd-term \(d\)
    by (simp add: trd-term-def, rule, rule dgrad-p-set-leI-insert, rule dgrad-p-set-le-subset,
rule subset-insertI,
                rule dgrad-p-set-le-red, fact dg, fact 〈red (set fs) p ?q〉, rule red-ord, fact)
    qed
qed
definition trd :: ( \(' t \Rightarrow_{0}{ }^{\prime} b::\) field \()\) list \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\)
    where trd fs \(p=\) trd-aux fs p 0
lemma trd-aux-red-rtrancl: \((\text { red }(\text { set fs) }))^{* *} p(\) trd-aux fs \(p r-r)\)
```

```
proof (induct fs pr rule: trd-aux.induct)
    case (1 fs pr)
    show ?case
    proof (simp, split option.split, intro conjI impI allI)
    assume \(p \neq 0\) and find-adds fs (lt \(p)=\) None
    hence \(\left(\right.\) red \((\) set fs) \(){ }^{* *}(\) tail p) (trd-aux fs (tail p) ( \(r+\) monomial (lc p) (lt p))
\(-(r+\) monomial (lc p) (lt p)))
            by (rule 1 (1))
    hence \((\text { red }(\text { set fs }))^{* *}(\) tail \(p+\) monomial (lc p) (lt p))
                (trd-aux fs (tail p) (r + monomial (lc p) (lt p)) - (r + monomial (lc
p) \((\) lt \(p))+\) monomial (lc p) \((l t p))\)
    proof (rule red-rtrancl-plus-higher)
            fix \(u v\)
            assume \(u \in\) keys (tail p)
            assume \(v \in\) keys (monomial (lc p) (lt p))
            also have \(\ldots \subseteq\{l t p\}\) by (simp add: keys-monomial)
            finally have \(v=l t p\) by \(\operatorname{simp}\)
                from \(\left\langle u \in\right.\) keys (tail \(p\) ) show \(u \prec_{t} v\) unfolding \(\langle v=l t p\rangle\) by (rule
keys-tail-less-lt)
    qed
    thus \((\text { red }(\text { set } f s))^{* *} p(\) trd-aux fs \((\) tail \(p)(r+\) monomial \((l c p)(l t ~ p))-r)\)
            by (simp only: leading-monomial-tail[symmetric] add.commute[of - monomial
(lc p) (lt p)], simp)
    next
        fix \(f\)
        assume \(p \neq 0\) and find-adds fs (lt \(p\) ) \(=\) Some \(f\)
        hence (red (set fs) ) \({ }^{* *}\) (tail \(p\) - monom-mult (lc p /lc f) (lp p-lpf) (tail f))
                            (trd-aux fs (tail p-monom-mult (lc p/lcf) (lp p-lpf) (tail
f)) \(r-r\) )
            and \(*\) : red (set fs) \(p\) (tail \(p\) - monom-mult \((l c p / l c f)(l p p-l p f)(\) tail \(f))\)
            by (rule 1 (2), rule find-adds-SomeD-red)
    let \(? q=\) tail \(p-\) monom-mult \((l c p / l c f)(l p p-l p f)(\) tail \(f)\)
    from * have \((\text { red }(\text { set } f s))^{* *} p\) ? \(q\)..
    moreover have (red (set fs))** ?q (trd-aux fs ?q r -r) by fact
    ultimately show \((\text { red }(\text { set fs }))^{* *} p(\) trd-aux fs ? \(q r-r)\) by (rule rtranclp-trans)
    qed
qed
corollary trd-red-rtrancl: \((\text { red }(\text { set fs }))^{* *} p(\operatorname{trd} f s p)\)
proof -
    have \((\text { red }(\text { set fs }))^{* *} p(\) trd fs \(p-0)\) unfolding trd-def by (rule trd-aux-red-rtrancl)
    thus ?thesis by simp
qed
lemma trd-aux-irred:
    assumes \(\neg i s\)-red (set fs) \(r\)
    shows \(\neg\) is-red (set fs) (trd-aux fs pr)
    using assms
proof (induct fs prole: trd-aux.induct)
```

```
    case (1 fs pr)
    show ?case
    proof (simp add: 1(3), split option.split, intro impI conjI allI)
    assume p\not=0 and *: find-adds fs (lt p) = None
    thus ᄀ is-red (set fs) (trd-aux fs (tail p) (r + monomial (lc p) (lt p)))
    proof (rule 1(1))
        show \neg is-red (set fs) (r + monomial (lc p) (lt p))
        proof
            assume is-red (set fs) (r + monomial (lc p) (lt p))
            then obtain fu where f\in set fs and f\not=0 and u\in keys (r+monomial
(lc p) (lt p))
            and lt f addst }u\mathrm{ by (rule is-red-addsE)
            note this(3)
            also have keys (r + monomial (lc p) (lt p))\subseteq keys r \cup keys (monomial (lc
p) (lt p))
            by (rule Poly-Mapping.keys-add)
            also have .. \subseteq insert (lt p) (keys r) by auto
            finally show False
            proof
            assume u}=lt
            from *\langlef\in set fs\rangle show ?thesis
            proof (rule find-adds-NoneE)
                    assume f=0
                    with }\langlef\not=0\rangle\mathrm{ show ?thesis ..
            next
                    assume \neglt f addst lt p
                    from this <lt f addst u〉 show ?thesis unfolding <u = lt p> ..
            qed
            next
                assume u\in keys r
            from}\langlef\in\mathrm{ set fs><f}\not=0\rangle\mathrm{ this <lt f addst }u\rangle\mathrm{ have is-red (set fs)r by (rule
is-red-addsI)
            with 1(3) show ?thesis ..
            qed
        qed
        qed
    next
        fix f
    assume p\not=0 and find-adds fs (lt p)=Some f
    from this 1(3) show \neg is-red (set fs) (trd-aux fs (tail p - monom-mult (lc p
/lc f) (lp p-lpf) (tail f)) r)
        by (rule 1(2))
    qed
qed
corollary trd-irred: ᄀ is-red (set fs) (trd fs p)
    unfolding trd-def using irred-0 by (rule trd-aux-irred)
lemma trd-in-pmdl: p - (trd fs p) \in pmdl (set fs)
```

using trd-red-rtrancl by (rule red-rtranclp-diff-in-pmdl)
lemma pmdl-closed-trd:
assumes $p \in p m d l B$ and set $f s \subseteq p m d l B$
shows (trd fs $p) \in p m d l B$
proof -
from $\operatorname{assms}(2)$ have $p m d l(s e t f s) \subseteq p m d l \operatorname{B}$ by (rule pmdl.span-subset-spanI)
with trd-in-pmdl have $p-t r d f s p \in p m d l B$..
with $\operatorname{assms}(1)$ have $p-(p-t r d f s p) \in p m d l B$ by (rule pmdl.span-diff)
thus?thesis by simp
qed
end
end

## 5 Gröbner Bases and Buchberger's Theorem

theory Groebner-Bases<br>imports Reduction<br>begin

This theory provides the main results about Gröbner bases for modules of multivariate polynomials.

```
context gd-term
begin
definition crit-pair :: \(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) field \() \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right)\)
    where crit-pair p \(q=\)
            (if component-of-term (lt p) = component-of-term (lt q) then
            (monom-mult \((1 / l c p)((l c s(l p p)(l p q))-(l p p))(\) tail \(p)\),
            monom-mult \((1 / l c q)((l c s(l p p)(l p q))-(l p q))(\) tail q) \()\)
        else (0, 0))
definition crit-pair-cbelow-on :: ('a nat) \(\Rightarrow\) nat \(\Rightarrow\left(' t \Rightarrow{ }_{0}{ }^{\prime} b::\right.\) field \()\) set \(\Rightarrow(' t\)
\(\left.\Rightarrow{ }_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\) bool
    where crit-pair-cbelow-on d m Fpqu
                                    cbelow-on (dgrad-p-set d m) \(\left(\prec_{p}\right)\)
                            (monomial 1 (term-of-pair (lcs (lp p) (lp q), component-of-term
\((\) (t \(p)))\) )
                            \((\lambda a b\). red \(F a b \vee \operatorname{red} F b a)(f s t(\) crit-pair \(p q))(\) snd (crit-pair
\(p q)\) )
definition spoly \(::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) field \()\)
    where spoly \(p q=(\) let \(v 1=l t p ; v 2=l t q\) in
                            if component-of-term v1 = component-of-term v2 then
                            let t1 \(=p p\)-of-term v1; t2 \(=\) pp-of-term v2; l=lcs t1 t2 in
                            (monom-mult \((1 /\) lookup \(p\) v1) \((l-t 1) p)-(\) monom-mult (1
/ lookup q v2) \((l-t 2) q\) )
```

definition (in ordered-term) is-Groebner-basis :: ( $' t \nRightarrow_{0}{ }^{\prime} b::$ field $)$ set $\Rightarrow$ bool where is-Groebner-basis $F \equiv$ relation.is-ChurchRosser (red F)

### 5.1 Critical Pairs and S-Polynomials

lemma crit-pair-same: fst (crit-pair p p) $=$ snd $($ crit-pair p $p)$
by (simp add: crit-pair-def)
lemma crit-pair-swap: crit-pair p $q=($ snd $($ crit-pair $q$ p), fst $(\operatorname{crit}-$ pair $q p))$ by (simp add: crit-pair-def lcs-comm)
lemma crit-pair-zero [simp]: fst (crit-pair $0 q)=0$ and snd $($ crit-pair p 0$)=0$ by (simp-all add: crit-pair-def)
lemma dgrad-p-set-le-crit-pair-zero: dgrad-p-set-le d \{fst (crit-pair p 0) \} \{p\} proof (simp add: crit-pair-deflt-def[of 0] lcs-comm lcs-zero dgrad-p-set-le-def Keys-insert min-term-def term-simps, intro conjI impI dgrad-set-leI)
fix $s$
assume $s \in$ pp-of-term' keys (monom-mult (1 / lc p) 0 (tail p))
then obtain $v$ where $v \in$ keys (monom-mult ( $1 / l c p$ ) $0($ tail $p)$ ) and $s=$
pp-of-term $v$..
from this $(1)$ keys-monom-mult-subset have $v \in(\oplus) 0$ ' keys (tail p) ..
hence $v \in$ keys (tail p) by (simp add: image-iff term-simps)
hence $v \in$ keys $p$ by (simp add: keys-tail)
hence $s \in p p$-of-term ' keys $p$ by (simp add: $\langle s=p p$-of-term $v\rangle$ )
moreover have $d s \leq d s$..
ultimately show $\exists t \in p p$-of-term' keys $p$. $d s \leq d t$..
qed $\operatorname{simp}$
lemma dgrad-p-set-le-fst-crit-pair:
assumes dickson-grading d
shows dgrad-p-set-le $d\{f$ st (crit-pair $p q)\}\{p, q\}$
proof (cases $q=0$ )
case True
have dgrad-p-set-le d $\{$ fst (crit-pair $p q)\}\{p\}$ unfolding True
by (fact dgrad-p-set-le-crit-pair-zero)
also have dgrad-p-set-le $d \ldots\{p, q\}$ by (rule dgrad-p-set-le-subset, simp)
finally show ?thesis .
next
case False
show ?thesis
proof (cases $p=0$ )
case True
have dgrad-p-set-le d \{fst (crit-pair p q)\} \{q\}
by (simp add: True dgrad-p-set-le-def dgrad-set-le-def)
also have dgrad-p-set-le $d \ldots\{p, q\}$ by (rule dgrad-p-set-le-subset, simp)
finally show ?thesis .

```
    next
        case False
    show ?thesis
    proof (simp add: dgrad-p-set-le-def Keys-insert crit-pair-def, intro conjI impI)
        define t where t=lcs (lp p) (lpq) - lp p
        let ?m = monom-mult (1 / lc p) t (tail p)
        from assms have dgrad-set-le d (pp-of-term'keys ?m) (insert t (pp-of-term
`keys (tail p)))
            by (rule dgrad-set-le-monom-mult)
        also have dgrad-set-le d ... (pp-of-term'(keys p \cup keys q))
        proof (rule dgrad-set-leI, simp)
            fix }
            assume s=t\vees\inpp-of-term`keys(tail p)
            thus \existsv\inkeys p\cup keys q. ds\leqd(pp-of-term v)
            proof
                assume s=t
                from assms have ds\leqord-class.max (d (lp p)) (d (lp q))
                    unfolding <s = t\rangle t-def by (rule dickson-grading-lcs-minus)
                hence ds\leqd(lp p)\veeds\leqd(lpq) by auto
                thus?thesis
                proof
                    from}\langlep\not=0\rangle\mathrm{ have lt p keys p by (rule lt-in-keys)
                    hence lt p}\in\mathrm{ keys }p\cup\mathrm{ keys q by simp
                    moreover assume d s\leqd (lp p)
                    ultimately show ?thesis ..
                next
                from }\langleq\not=0\rangle\mathrm{ have lt q| keys q by (rule lt-in-keys)
                hence lt q\in keys p\cup keys q by simp
                    moreover assume ds\leqd(lpq)
                    ultimately show ?thesis ..
                qed
            next
                assume s f pp-of-term`keys (tail p)
                hence s\inpp-of-term'(keys p\cup keys q) by (auto simp: keys-tail)
                then obtain v}\mathrm{ where v}v\inkeys p\cupkeys q and s=pp-of-term v ..
                note this(1)
                moreover have ds\leqd (pp-of-term v) by (simp add: <s = pp-of-term v`)
                ultimately show ?thesis ..
            qed
        qed
        finally show dgrad-set-le d (pp-of-term' keys ?m) (pp-of-term'(keys p U
keys q)) .
    qed (rule dgrad-set-leI, simp)
    qed
qed
lemma dgrad-p-set-le-snd-crit-pair:
assumes dickson-grading d
shows dgrad-p-set-le d {snd (crit-pair p q)} {p,q}
```

by (simp add: crit-pair-swap[of p] insert-commute[of p q], rule dgrad-p-set-le-fst-crit-pair, fact)
lemma dgrad-p-set-closed-fst-crit-pair:
assumes dickson-grading $d$ and $p \in d g r a d-p$-set $d m$ and $q \in d g r a d$ - $p$-set $d m$ shows $f s t$ (crit-pair p $q$ ) $\in$ dgrad-p-set $d m$
proof -
from dgrad-p-set-le-fst-crit-pair $[O F \operatorname{assms}(1)]$ have $\{$ fst (crit-pair $p q)\} \subseteq$ dgrad- $p$-set d $m$
proof (rule dgrad-p-set-le-dgrad-p-set)
from $\operatorname{assms}(2,3)$ show $\{p, q\} \subseteq d g r a d-p$-set $d m$ by simp
qed
thus ?thesis by simp
qed
lemma dgrad-p-set-closed-snd-crit-pair:
assumes dickson-grading $d$ and $p \in$ dgrad- $p$-set $d m$ and $q \in d g r a d$ - $p$-set $d m$ shows snd (crit-pair $p q) \in$ dgrad- $p$-set $d m$ by (simp add: crit-pair-swap[of p q], rule dgrad-p-set-closed-fst-crit-pair, fact+)
lemma fst-crit-pair-below-lcs:
fst (crit-pair $p q$ ) $\prec_{p}$ monomial 1 (term-of-pair (lcs (lp p) (lp q), component-of-term
(lt p)))
proof (cases tail $p=0$ )
case True
thus ?thesis by (simp add: crit-pair-def ord-strict-p-monomial-iff)
next
case False
let $? t 1=l p p$
let $? t 2=l p q$
from False have $p \neq 0$ by auto
hence lc $p \neq 0$ by (rule lc-not- 0 )
hence 1 / lc $p \neq 0$ by $\operatorname{simp}$
from this False have lt (monom-mult $(1 /$ lc $p)(l c s$ ?t1 ?t2 - ?t1 $)($ tail $p))=$ (lcs ? t1 ?t2 - ? t1 $) \oplus l t($ tail p)
by (rule lt-monom-mult)
also from lt-tail[OF False] have $\ldots \prec_{t}($ lcs ? $t 1$ ? t2 - ? t1 $) \oplus$ lt $p$
by (rule splus-mono-strict)
also from adds-lcs have $\ldots=$ term-of-pair (lcs ?t1 ?t2, component-of-term (lt p))
by ( simp add: adds-lcs adds-minus splus-def)
finally show ?thesis by (auto simp add: crit-pair-def ord-strict-p-monomial-iff) qed
lemma snd-crit-pair-below-lcs:
snd (crit-pair p q) $\prec_{p}$ monomial 1 (term-of-pair (lcs (lp p) (lp q), compo-nent-of-term (lt p)))
proof (cases component-of-term (lt p) $=$ component-of-term (lt q))
case True

```
    show ?thesis
    by (simp add: True crit-pair-swap[of p] lcs-comm[of lp p], fact fst-crit-pair-below-lcs)
next
    case False
    show ?thesis by (simp add: crit-pair-def False ord-strict-p-monomial-iff)
qed
lemma crit-pair-cbelow-same:
    assumes dickson-grading d and p\indgrad-p-set d m
    shows crit-pair-cbelow-on d m F p p
proof (simp add: crit-pair-cbelow-on-def crit-pair-same cbelow-on-def term-simps,
intro disjI1 conjI)
    from assms(1) assms(2) assms(2) show snd (crit-pair p p) \indgrad-p-set d m
        by (rule dgrad-p-set-closed-snd-crit-pair)
next
    from snd-crit-pair-below-lcs[of p p] show snd (crit-pair p p) \prec}\mp@subsup{}{p}{}\mathrm{ monomial 1 (lt
p)
    by (simp add: term-simps)
qed
lemma crit-pair-cbelow-distinct-component:
    assumes component-of-term (lt p) \not= component-of-term (lt q)
    shows crit-pair-cbelow-on d m F p q
    by (simp add: crit-pair-cbelow-on-def crit-pair-def assms cbelow-on-def
        ord-strict-p-monomial-iff zero-in-dgrad-p-set)
lemma crit-pair-cbelow-sym:
    assumes crit-pair-cbelow-on d m F pq
    shows crit-pair-cbelow-on d m F q p
proof (cases component-of-term (lt q) = component-of-term (lt p))
    case True
    from assms show ?thesis
    proof (simp add: crit-pair-cbelow-on-def crit-pair-swap[of p q] lcs-comm True,
                elim cbelow-on-symmetric)
            show symp (\lambdaab.red Fabv red F b a) by (simp add: symp-def)
    qed
next
    case False
    thus ?thesis by (rule crit-pair-cbelow-distinct-component)
qed
lemma crit-pair-cs-imp-crit-pair-cbelow-on:
    assumes dickson-grading d and F\subseteqdgrad-p-set d m and p}\indgrad-p-set d m
        and q}\indgrad-p-set d m
        and relation.cs (red F) (fst (crit-pair p q)) (snd (crit-pair p q))
    shows crit-pair-cbelow-on d m Fpq
proof -
    from assms(1) have relation-order (red F) ( }\mp@subsup{\prec}{p}{})(\mathrm{ dgrad-p-set d m) by (rule
is-relation-order-red)
```

```
    moreover have relation.dw-closed (red F) (dgrad-p-set d m)
    by (rule relation.dw-closedI, rule dgrad-p-set-closed-red, rule assms(1), rule
assms(2))
    moreover note assms(5)
    moreover from assms(1) assms(3) assms(4) have fst (crit-pair p q) \in dgrad-p-set
dm
    by (rule dgrad-p-set-closed-fst-crit-pair)
    moreover from assms(1) assms(3) assms(4) have snd (crit-pair p q) \indgrad-p-set
dm
    by (rule dgrad-p-set-closed-snd-crit-pair)
    moreover note fst-crit-pair-below-lcs snd-crit-pair-below-lcs
    ultimately show ?thesis unfolding crit-pair-cbelow-on-def by (rule relation-order.cs-implies-cbelow-on)
qed
lemma crit-pair-cbelow-mono:
    assumes crit-pair-cbelow-on d m F pq and F\subseteqG
    shows crit-pair-cbelow-on d m G p q
    using assms(1) unfolding crit-pair-cbelow-on-def
proof (induct rule: cbelow-on-induct)
    case base
    show ?case by (simp add: cbelow-on-def, intro disjI1 conjI, fact+)
next
    case (step b c)
    from step(2) have red Gbc\vee red Gcb using red-subset[OF - assms(2)] by
blast
    from step(5) step(3) this step(4) show ?case ..
qed
lemma lcs-red-single-fst-crit-pair:
    assumes p\not=0 and component-of-term (lt p)=component-of-term (lt q)
    defines t1 \equivlp p
    defines t2 \equivlpq
    shows red-single (monomial (- 1) (term-of-pair (lcs t1 t2, component-of-term
(lt p))))
    (fst (crit-pair p q)) p(lcs t1 t2 - t1)
proof -
    let ?l = term-of-pair (lcs t1 t2, component-of-term (lt p))
    from assms(1) have lc p\not=0 by (rule lc-not-0)
    have lt p addst ?l by (simp add: adds-lcs adds-term-def t1-def term-simps)
    hence eq1: (lcs t1 t2 - t1) \oplus lt p=?l
        by (simp add: adds-lcs adds-minus splus-def t1-def)
    with assms(1) show ?thesis
    proof (simp add: crit-pair-def red-single-def assms(2))
        have eq2: monomial (- 1) ?l = monom-mult (- (1 / lc p)) (lcs t1 t2 - t1)
(monomial (lc p) (lt p))
            by (simp add: monom-mult-monomial eq1 <lc p\not=0>)
            show monom-mult (1 / lc p) (lcs (lp p) (lp q) - lp p) (tail p)=
                monomial (- 1) (term-of-pair (lcs t1 t2, component-of-term (lt q))) -
monom-mult (- (1 / lc p)) (lcs t1 t2 - t1) p
```

```
    apply (simp add: t1-def t2-def monom-mult-dist-right-minus tail-alt-2 monom-mult-uminus-left)
    by (metis assms(2) eq2 monom-mult-uminus-left t1-def t2-def)
    qed
qed
```

corollary lcs-red-single-snd-crit-pair:
assumes $q \neq 0$ and component-of-term (lt $p$ ) $=$ component-of-term (lt $q$ )
defines $t 1 \equiv l p p$
defines $t 2 \equiv l p q$
shows red-single (monomial (-1) (term-of-pair (lcs t1 t2, component-of-term
(lt p))))
(snd (crit-pair p q)) q (lcs t1 t2 - t2)
by (simp add: crit-pair-swap[of p q] lcs-comm[of lp p]assms(2) t1-def t2-def,
rule lcs-red-single-fst-crit-pair, simp-all add: assms(1, 2))
lemma GB-imp-crit-pair-cbelow-dgrad-p-set:
assumes dickson-grading $d$ and $F \subseteq$ dgrad-p-set $d m$ and is-Groebner-basis $F$
assumes $p \in F$ and $q \in F$ and $p \neq 0$ and $q \neq 0$
shows crit-pair-cbelow-on $d m F p q$
proof (cases component-of-term (lt p) $=$ component-of-term (lt q))
case True
from $\operatorname{assms}(1,2)$ show ?thesis
proof (rule crit-pair-cs-imp-crit-pair-cbelow-on)
from $\operatorname{assms}(4,2)$ show $p \in d g r a d-p-s e t d m .$.
next
from $\operatorname{assms}(5,2)$ show $q \in d g r a d-p$-set $d m$..
next
let $? c p=$ crit-pair $p q$
let ?l = monomial ( -1 ) (term-of-pair (lcs (lp p) (lp q), component-of-term (lt
p)))
from $\operatorname{assms}(4)$ lcs-red-single-fst-crit-pair $[O F \operatorname{assms}(6)$ True $]$ have red $F$ ?l (fst
? $c p)$
by (rule red-setI)
hence 1: $(\text { red } F)^{* *}$ ?l (fst ? cp) ..
from assms(5) lcs-red-single-snd-crit-pair $[O F \operatorname{assms}(7)$ True $]$ have red $F$ ?l
(snd ?.cp)
by (rule red-setI)
hence 2: $(\text { red } F)^{* *}$ ?l (snd ?cp) ..
from assms(3) have relation.is-confluent-on (red F) UNIV
by (simp only: is-Groebner-basis-def relation.confluence-equiv-ChurchRosser[symmetric]
relation.is-confluent-def)
from this 12 show relation.cs (red F) (fst ?cp) (snd ?cp)
by (simp add: relation.is-confluent-on-def)
qed
next
case False
thus ?thesis by (rule crit-pair-cbelow-distinct-component)
qed

```
lemma spoly-alt:
    assumes \(p \neq 0\) and \(q \neq 0\)
    shows spoly \(p\) q \(=\) fst (crit-pair \(p q)-\) snd (crit-pair \(p q)\)
proof (cases component-of-term (lt p) \(=\) component-of-term (lt q))
    case ec: True
    show ?thesis
    proof (rule poly-mapping-eqI, simp only: lookup-minus)
        fix \(v\)
    define \(t 1\) where \(t 1=l p p\)
    define \(t 2\) where \(t 2=l p q\)
    let ? l = lcs t1 t2
    let ?lv \(=\) term-of-pair (?l, component-of-term (lt p))
    let ? \(c p=\) crit-pair \(p q\)
    let \(? a=\lambda x\). monom-mult \((1 / l c p)(? l-t 1) x\)
    let \(? b=\lambda x\). monom-mult \((1 / l c q)(? l-t 2) x\)
    have l-1: \((? l-t 1) \oplus l t p=? l v\) by (simp add: adds-lcs adds-minus splus-def
t1-def)
    have l-2: \((? l-t 2) \oplus l t q=? l v\) by (simp add: ec adds-lcs-2 adds-minus splus-def
t2-def)
    show lookup (spoly p \(q\) ) \(v=\) lookup (fst ? \(c p\) ) \(v-\) lookup (snd ? \(c p) v\)
    proof (cases \(v=\) ?lv)
        case True
        have \(v-1: v=(? l-t 1) \oplus l t p\) by \((\) simp add: True \(l-1)\)
        from \(\langle p \neq 0\rangle\) have \(l t p \in\) keys \(p\) by (rule lt-in-keys)
        hence \(v-2: v=(? l-t 2) \oplus l t q\) by \((s i m p\) add: True \(l-2)\)
        from \(\langle q \neq 0\rangle\) have \(l t q \in\) keys \(q\) by (rule lt-in-keys)
        from \(\langle l t p \in\) keys \(p\rangle\) have lookup (?a \(p\) ) \(v=1\)
            by (simp add: in-keys-iff \(v\)-1 lookup-monom-mult lc-def term-simps)
        also from \(\langle l t q \in\) keys \(q\rangle\) have \(\ldots=\operatorname{lookup}(? b q) v\)
            by (simp add: in-keys-iff v-2 lookup-monom-mult lc-def term-simps)
        finally have lookup (spoly \(p q\) ) \(v=0\)
            by (simp add: spoly-def ec Let-def t1-def t2-def lookup-minus lc-def)
        moreover have lookup (fst ?cp) \(v=0\)
        by (simp add: crit-pair-def ec v-1 lookup-monom-mult t1-def t2-def term-simps,
                simp only: not-in-keys-iff-lookup-eq-zero[symmetric] keys-tail, simp)
    moreover have lookup (snd ?cp) \(v=0\)
    by (simp add: crit-pair-def ec v-2 lookup-monom-mult t1-def t2-def term-simps,
                simp only: not-in-keys-iff-lookup-eq-zero[symmetric] keys-tail, simp)
    ultimately show?thesis by simp
next
    case False
    have lookup (?a (tail p)) v=lookup (?a p) v
    proof (cases ?l - t1 addsp \(v\) )
        case True
        then obtain \(u\) where \(v: v=(? l-t 1) \oplus u\)..
        have \(u \neq l t p\)
        proof
            assume \(u=l t p\)
            hence \(v=\) ?lv by (simp add: v l-1)
```

```
            with }\langlev\not=\mathrm{ ?lv> show False ..
            qed
            thus ?thesis by (simp add: v lookup-monom-mult lookup-tail-2 term-simps)
            next
            case False
            thus ?thesis by (simp add:lookup-monom-mult)
    qed
    moreover have lookup (?b (tail q)) v= lookup (?b q) v
    proof (cases ?l - t2 addsp v)
            case True
            then obtain u where v:v=(?l - t2) \oplusu..
            have u\not=lt q
            proof
            assume u}=lt
            hence v=?lv by (simp add: v l-2)
            with }\langlev\not=\mathrm{ ?lv> show False ..
            qed
            thus ?thesis by (simp add: v lookup-monom-mult lookup-tail-2 term-simps)
    next
            case False
            thus ?thesis by (simp add: lookup-monom-mult)
            qed
            ultimately show ?thesis
            by (simp add: ec spoly-def crit-pair-def lookup-minus t1-def t2-def Let-def
lc-def)
            qed
    qed
next
    case False
    show ?thesis by (simp add: spoly-def crit-pair-def False)
qed
lemma spoly-same: spoly p p=0
    by (simp add: spoly-def)
lemma spoly-swap: spoly p q = - spoly q p
    by (simp add: spoly-def lcs-comm Let-def)
lemma spoly-red-zero-imp-crit-pair-cbelow-on:
    assumes dickson-grading d and F\subseteqdgrad-p-set d m and p}\indgrad-p-set d m
        and q\indgrad-p-set d m and p\not=0 and q\not=0 and (red F)** (spoly pq)0
    shows crit-pair-cbelow-on d m Fpq
proof -
    from assms(7) have relation.cs (red F) (fst (crit-pair p q)) (snd (crit-pair p q))
        unfolding spoly-alt[OF assms(5) assms(6)] by (rule red-diff-rtrancl-cs)
    with assms(1) assms(2) assms(3) assms(4) show ?thesis by (rule crit-pair-cs-imp-crit-pair-cbelow-on)
qed
lemma dgrad-p-set-le-spoly-zero: dgrad-p-set-le d {spoly p 0} {p}
```

proof (simp add: term-simps spoly-def lt-def[of 0] lcs-comm lcs-zero dgrad-p-set-le-def Keys-insert

> Let-def min-term-def lc-def[symmetric], intro conjI impI dgrad-set-leI)

## fix $s$

assume $s \in p p$-of-term'keys (monom-mult ( $1 / l c p$ ) $0 p$ )
then obtain $u$ where $u \in$ keys (monom-mult ( $1 / l c p$ ) $0 p$ ) and $s=p p$-of-term $u$.. from this(1) keys-monom-mult-subset have $u \in(\oplus) 0$ ' keys $p$.. hence $u \in$ keys $p$ by (simp add: image-iff term-simps)
hence $s \in p p$-of-term' keys $p$ by (simp add: $\langle s=p p$-of-term $u\rangle$ )
moreover have $d s \leq d s$..
ultimately show $\exists t \in p p$-of-term' keys $p . d s \leq d t$..
qed $\operatorname{simp}$
lemma dgrad-p-set-le-spoly: assumes dickson-grading $d$
shows dgrad-p-set-le $d\{$ spoly $p q\}\{p, q\}$
proof (cases $p=0$ )
case True
have dgrad-p-set-le d \{spoly $p q\}\{$ spoly $q 0\}$ unfolding True spoly-swap[of $0 q]$ by (fact dgrad-p-set-le-uminus)
also have dgrad-p-set-le $d \ldots\{q\}$ by (fact dgrad-p-set-le-spoly-zero)
also have dgrad-p-set-le $d \ldots\{p, q\}$ by (rule dgrad-p-set-le-subset, simp)
finally show ?thesis.
next
case False
show ?thesis
proof (cases $q=0$ )
case True
have dgrad-p-set-le $d\{$ spoly $p q\}\{p\}$ unfolding True by (fact dgrad-p-set-le-spoly-zero) also have dgrad-p-set-le $d \ldots\{p, q\}$ by (rule dgrad-p-set-le-subset, simp) finally show ?thesis.
next
case False
have dgrad-p-set-le $d\{$ spoly $p q\}\{$ fst (crit-pair $p q$ ), snd (crit-pair $p q)\}$
unfolding spoly-alt $[O F\langle p \neq 0\rangle$ False $]$ by (rule dgrad-p-set-le-minus)
also have dgrad-p-set-le d ... $\{p, q\}$
proof (rule dgrad-p-set-leI-insert)
from assms show dgrad-p-set-le d $\{$ fst (crit-pair p q) $\}\{p, q\}$
by (rule dgrad-p-set-le-fst-crit-pair)
next
from assms show dgrad-p-set-le $d$ \{snd (crit-pair $p q)\}\{p, q\}$
by (rule dgrad-p-set-le-snd-crit-pair)
qed
finally show ?thesis .
qed
qed
lemma dgrad-p-set-closed-spoly:

```
    assumes dickson-grading d and p\indgrad-p-set d m and q\indgrad-p-set d m
    shows spoly p q\indgrad-p-set d m
proof -
    from dgrad-p-set-le-spoly[OF assms(1)] have {spoly p q} \subseteq dgrad-p-set d m
    proof (rule dgrad-p-set-le-dgrad-p-set)
    from assms(2, 3) show {p,q}\subseteqdgrad-p-set d m by simp
    qed
    thus ?thesis by simp
qed
lemma components-spoly-subset: component-of-term' keys (spoly p q) \subseteq compo-
nent-of-term 'Keys {p,q}
    unfolding spoly-def Let-def
proof (split if-split, intro conjI impI)
    define c where c=(1 / lookup p (lt p))
    define d}\mathrm{ where d}=(1/\mathrm{ lookup q (lt q))
    define s}\mathrm{ where s=lcs (lp p) (lpq) - lp p
    define t where t=lcs (lp p) (lpq) - lpq
    show component-of-term'keys (monom-mult cs p - monom-mult d t q) \subseteq
component-of-term'Keys {p,q}
    proof
    fix }
    assume k \in component-of-term`keys(monom-mult c s p - monom-mult d t
q)
    then obtain v}\mathrm{ where vekeys (monom-mult c s p-monom-mult d t q) and
k:k= component-of-term v ..
    from this(1) keys-minus have v\inkeys (monom-mult c s p) \cup keys (monom-mult
d t q) ..
    thus }k\in\mathrm{ component-of-term'Keys {p,q}
    proof
            assume v \in keys (monom-mult c s p)
            from this keys-monom-mult-subset have v\in(\oplus) s'keys p ..
            then obtain u where }u\in\mathrm{ keys p and v: v=s }\oplusu.
            have u\in Keys {p,q} by (rule in-KeysI, fact, simp)
            moreover have k= component-of-term u by (simp add: v k term-simps)
            ultimately show ?thesis by simp
    next
                assume v\in keys (monom-mult d t q)
                from this keys-monom-mult-subset have v\in(\oplus)t'keys q..
                then obtain }u\mathrm{ where }u\in\mathrm{ keys q and v:v=tөu..
            have}u\in\mathrm{ Keys {p,q} by (rule in-KeysI, fact, simp)
            moreover have k= component-of-term u by (simp add: v k term-simps)
            ultimately show ?thesis by simp
        qed
    qed
qed simp
lemma pmdl-closed-spoly:
    assumes p
```

```
    shows spoly p q \in pmdl F
proof (cases component-of-term (lt p) = component-of-term (lt q))
    case True
    show ?thesis
    by (simp add: spoly-def True Let-def, rule pmdl.span-diff,
        (rule pmdl-closed-monom-mult, fact)+)
next
    case False
    show ?thesis by (simp add: spoly-def False pmdl.span-zero)
qed
```


## 5．2 Buchberger＇s Theorem

Before proving the main theorem of Gröbner bases theory for S－polynomials， as is usually done in textbooks，we first prove it for critical pairs：a set $F$ yields a confluent reduction relation if the critical pairs of all $p \in F$ and $q \in F$ can be connected below the least common sum of the leading power－products of $p$ and $q$ ．The reason why we proceed in this way is that it becomes much easier to prove the correctness of Buchberger＇s second criterion for avoiding useless pairs．

```
lemma crit-pair-cbelow-imp-confluent-dgrad-p-set:
    assumes \(d g\) : dickson-grading \(d\) and \(F \subseteq d g r a d\) - \(p\)-set \(d m\)
    assumes main: \(\bigwedge p q . p \in F \Longrightarrow q \in F \Longrightarrow p \neq 0 \Longrightarrow q \neq 0 \Longrightarrow\) crit-pair-cbelow-on
\(d m F p q\)
    shows relation.is-confluent-on (red F) (dgrad-p-set \(d m)\)
proof -
    let ? \(A=\) dgrad- \(p\)-set \(d m\)
    let \(? R=\) red \(F\)
    let ? \(R S=\lambda a b\). red \(F a b \vee \operatorname{red} F b a\)
    let ?ord \(=\left(\prec_{p}\right)\)
    from \(d g\) have ro: Confluence.relation-order ?R ?ord?A
        by (rule is-relation-order-red)
    have dw: relation.dw-closed ?R ?A
        by (rule relation.dw-closedI, rule dgrad-p-set-closed-red, rule dg, rule assms(2))
    show ?thesis
    proof (rule relation-order.loc-connectivity-implies-confluence, fact ro)
        show is-loc-connective-on ?A ?ord ?R unfolding is-loc-connective-on-def
    proof (intro ballI allI impI)
        fix \(a b 1 b 2\) :: ' \(t \Rightarrow_{0}{ }^{\prime} b\)
        assume \(a \in\) ? \(A\)
        assume ? R a b1 \(\wedge\) ? R a b2
        hence ?R a b1 and ?R a b2 by simp-all
        hence \(b 1 \in ? A\) and \(b 2 \in ? A\) and ?ord b1 \(a\) and ?ord b2 a
            using red-ord dgrad-p-set-closed-red[OF dg assms(2) \(\langle a \in\) ? A〉] by blast+
        from this(1) this(2) have \(b 1-b 2 \in ? A\) by (rule dgrad-p-set-closed-minus)
        from «red \(F\) a b1〉 obtain \(f 1\) and \(t 1\) where \(f 1 \in F\) and \(r 1\) : red-single a b1
f1 t1 by (rule red-setE)
        from 〈red \(F\) a b2〉 obtain f2 and \(t 2\) where \(f 2 \in F\) and \(r 2\) : red-single a b2
```

f2 t2 by（rule red－setE）
from $r 1 r 2$ have $f 1 \neq 0$ and $f 2 \neq 0$ by（simp－all add：red－single－def）
hence $l c 1: l c f 1 \neq 0$ and $l c 2: l c f 2 \neq 0$ using $l c-n o t-0$ by auto
show cbelow－on ？A ？ord $a(\lambda a b$ ．？$R a b \vee ? R b a) b 1 b 2$
proof（cases t1 $\oplus l t f 1=t 2 \oplus l t f 2)$
case False
from confluent－distinct $[$ OF r1 r2 False $\langle f 1 \in F\rangle\langle f 2 \in F\rangle]$ obtain $s$ where $s 1:(\text { red } F)^{* *} b 1 \mathrm{~s}$ and $\mathrm{s} 2:(\operatorname{red} F)^{* *} b 2 \mathrm{~s}$ ．
have relation．cs ？$R$ b1 b2 unfolding relation．cs－def by（intro exI conjI， fact s1，fact s2）
from ro dw this $\langle b 1 \in ? A\rangle\langle b 2 \in ? A\rangle\langle ? o r d$ b1 a〉〈？ord b2 a〉 show ？thesis by（rule relation－order．cs－implies－cbelow－on）
next
case True
hence ec：component－of－term（lt f1）$=$ component－of－term（lt f2）
by（metis component－of－term－splus）
let ？$l 1=l p f 1$
let $? 12=\operatorname{lp} f 2$
define $v$ where $v \equiv t 2 \oplus l t$ f2
define $l$ where $l \equiv l c s ? l 1$ ？ 12
define $a^{\prime}$ where $a^{\prime}=$ except $a\{v\}$
define $m a$ where $m a=$ monomial（lookup a $v$ ）$v$
have $v$－alt：$v=t 1 \oplus l t f 1$ by（simp only：True $v$－def）
have $a=m a+a^{\prime}$ unfolding ma－def $a^{\prime}$－def by（fact plus－except）
have comp－f1：component－of－term（lt f1）＝component－of－term $v$ by（simp
add：v－alt term－simps）
have ？l1 adds $l$ unfolding $l$－def by（rule adds－lcs）
have ？12 adds $l$ unfolding $l$－def by（rule adds－lcs－2）
have ？l1 adds $s_{p}(t 1 \oplus l t f 1)$ by（simp add：adds－pp－splus term－simps）
hence ？l1 adds $s_{p} v$ by（simp add：v－alt）
have ？12 adds $s_{p} v$ by（simp add：v－def adds－pp－splus term－simps）
from 〈？l1 addsp $v\rangle\langle ? l 2$ addsp $v\rangle$ have $l$ adds $s_{p} v$ by（simp add：l－def adds－pp－def lcs－adds）
have $p p$－of－term $(v \ominus ? l 1)=t 1$ by（simp add：v－alt term－simps $)$
with $\left\langle l a d d s_{p} v\right\rangle\langle ? l 1$ adds $l\rangle$ have $t f 1^{\prime}:$ pp－of－term $((l-? l 1) \oplus(v \ominus l))=$ t1
by（simp add：minus－splus－sminus－cancel）
hence tf1：$((p p-o f-t e r m v)-l)+(l-? l 1)=t 1$ by $($ simp add：add．commute term－simps）
have pp－of－term $(v \ominus$ ？l2 $)=t 2$ by（simp add：$v$－def term－simps $)$
with $\left\langle l a d d s_{p} v\right\rangle\langle ? l 2$ adds $l\rangle$ have tf2＇：pp－of－term $((l-$ ？l2 $) \oplus(v \ominus l))=$
by（simp add：minus－splus－sminus－cancel）
hence $t f 2:((p p-o f-t e r m v)-l)+(l-$ ？l2 $)=t 2$ by $($ simp add：add．commute term－simps）
let ？$c a=$ lookup a $v$
let $? v=p p$－of－term $v-l$
have $? v+l=p p$－of－term $v$ using $\left\langle l a d d s_{p} v\right\rangle$ adds－minus adds－pp－def by
blast
from tf1' have ?v adds t1 unfolding pp-of-term-splus add.commute[of $l$ ?l1] pp-of-term-sminus
using addsI by blast
with $d g$ have $d ? v \leq d t 1$ by (rule dickson-grading-adds-imp-le)
also from $d g\langle a \in$ ?A〉r1 have $\ldots \leq m$ by (rule dgrad-p-set-red-single-pp)
finally have $d ? v \leq m$.
from $r 2$ have $? c a \neq 0$ by (simp add: red-single-def $v$-def)
hence - ? $c a \neq 0$ by simp
from $r 1$ have $b 1=a-$ monom-mult (?ca / lc f1) t1 f1 by (simp add: red-single-def $v$-alt)
also have $\ldots=$ monom-mult $(-$ ?ca) ?v $($ fst (crit-pair f1 f2 $))+a^{\prime}$
proof (simp add: $a^{\prime}$-def ec crit-pair-def l-def[symmetric] monom-mult-assoc $t f 1$,
rule poly-mapping-eqI, simp add: lookup-add lookup-minus)

## fix $u$

show lookup a $u$ - lookup (monom-mult (?ca / lc f1) t1 f1) $u=$
lookup (monom-mult (- (?ca / lc f1)) t1 (tail f1)) $u+$ lookup (except $a\{v\}) u$
proof (cases $u=v$ )
case True
show ?thesis
by (simp add: True lookup-except v-alt lookup-monom-mult lookup-tail-2 lc-def[symmetric] lc1 term-simps)
next
case False
hence $u \notin\{v\}$ by simp
moreover
\{
assume $t 1 a d d s_{p} u$
hence $t 1 \oplus(u \ominus t 1)=u$ by (simp add: adds-pp-sminus)
hence $u \ominus t 1 \neq l t f 1$ using False $v$-alt by auto
hence lookup f1 $(u \ominus t 1)=$ lookup (tail f1) $(u \ominus t 1)$ by (simp add:
lookup-tail-2)
\}
ultimately show ?thesis using False by (simp add: lookup-except lookup-monom-mult)
qed
qed
finally have b1: b1 = monom-mult $\left(-\right.$ ?ca) ?v $($ fst $($ crit-pair f1 f2 $))+a^{\prime}$.
from $r 2$ have $b 2=a-$ monom-mult (?ca / lc f2) t2 f2
by (simp add: red-single-def $v$-def True)
also have $\ldots=$ monom-mult $(-$ ?ca) ?v (snd (crit-pair f1 f2) $)+a^{\prime}$
proof (simp add: $a^{\prime}$-def ec crit-pair-def l-def[symmetric] monom-mult-assoc tf2,
rule poly－mapping－eqI，simp add：lookup－add lookup－minus）
fix $u$
show lookup a $u$－lookup（monom－mult（？ca／lc f2）t2 f2）$u=$ lookup（monom－mult（－（？ca／lc f2））t2（tail f2））$u+$ lookup（except $a\{v\}) u$
proof（cases $u=v$ ）
case True
show ？thesis
by（simp add：True lookup－except v－def lookup－monom－mult lookup－tail－2
lc－def［symmetric］lc2 term－simps）
next
case False
hence $u \notin\{v\}$ by simp
moreover
\｛
assume t2 $a d d s_{p} u$
hence $t 2 \oplus(u \ominus t 2)=u$ by（simp add：adds－pp－sminus）
hence $u \ominus$ t2 $\neq l t$ f2 using False $v$－def by auto
hence lookup f2 $(u \ominus$ t2 $)=$ lookup（tail f2）$(u \ominus$ t2）by（simp add：
lookup－tail－2）
\}
ultimately show ？thesis using False by（simp add：lookup－except lookup－monom－mult）
qed
qed
finally have b2：b2＝monom－mult $\left(-\right.$ ？ca）？$v($ snd $($ crit－pair f1 f2 $))+a^{\prime}$
let ？lv $=$ term－of－pair（ $l$ ，component－of－term（lt f1））
from $\langle f 1 \in F\rangle\langle f \mathcal{2} \in F\rangle\langle f 1 \neq 0\rangle\langle f 2 \neq 0\rangle$ have crit－pair－cbelow－on $d m F$ f1 f2 by（rule main）
hence cbelow－on ？A ？ord（monomial 1 ？lv）？RS（fst（crit－pair f1 f2））（snd （crit－pair f1 f2））
by（simp only：crit－pair－cbelow－on－def l－def）
with $d g$ assms（2）$\langle d ? v \leq m\rangle\langle-? c a \neq 0\rangle$
have cbelow－on ？A ？ord（monom－mult（－？ca）？v（monomial 1 ？lv））？RS
（monom－mult（－？ca）？v（fst（crit－pair f1 f2）））
（monom－mult（－？ca）？v（snd（crit－pair f1 f2）））
by（rule cbelow－on－monom－mult）
hence cbelow－on ？A ？ord（monomial（－？ca）v）？RS
（monom－mult（ - ？ca）？v（fst（crit－pair f1 f2）））
（monom－mult（ - ？ca）？v（snd（crit－pair f1 f2）））
by（simp add：monom－mult－monomial $\prec(p p-o f-t e r m ~ v-l)+l=p p$－of－term v〉 splus－def comp－f1 term－simps）
with 〈？$c a \neq 0$ 〉 have cbelow－on ？A ？ord（monomial ？ca $(0 \oplus v)$ ）？RS
（monom－mult $(-? c a)$ ？v $(f s t($ crit－pair f1 f2）$))$（monom－mult（ - ？ca）
？v（snd（crit－pair f1 f2）））
by（rule cbelow－on－monom－mult－monomial）
hence cbelow－on？A ？ord ma？RS
(monom-mult (-?ca) ?v (fst (crit-pair f1 f2))) (monom-mult (-?ca) ?v ( snd (crit-pair f1 f2)) )
by (simp add: ma-def term-simps) with $d g \operatorname{assms}(2)$-show cbelow-on ?A ? ord $a$ ?RS b1 b2 unfolding $\left\langle a=m a+a^{\prime}\right\rangle b 1$ b2 proof (rule cbelow-on-plus)
show $a^{\prime} \in$ ? $A$
by (rule, simp add: $a^{\prime}$-def keys-except, erule conjE, intro dgrad-p-setD, rule $\langle a \in d g r a d-p$-set $d m\rangle)$
next
show keys $a^{\prime} \cap$ keys $m a=\{ \}$ by (simp add: ma-def $a^{\prime}$-def keys-except) qed
qed
qed
qed fact
qed
corollary crit-pair-cbelow-imp-GB-dgrad-p-set:
assumes dickson-grading $d$ and $F \subseteq d g r a d-p$-set $d m$
assumes $\bigwedge p q . p \in F \Longrightarrow q \in F \Longrightarrow p \neq 0 \Longrightarrow q \neq 0 \Longrightarrow$ crit-pair-cbelow-on $d m F p q$
shows is-Groebner-basis F
unfolding is-Groebner-basis-def
proof (rule relation.confluence-implies-ChurchRosser,
simp only: relation.is-confluent-def relation.is-confluent-on-def, intro ballI allI impI)
fix $a b 1$ b2
assume $a:(\text { red } F)^{* *}$ a b1 $\wedge(\text { red } F)^{* *}$ a b2
from $\operatorname{assms}(2)$ obtain $n$ where $m \leq n$ and $a \in d g r a d-p-s e t d n$ and $F \subseteq$ dgrad-p-set $d n$
by (rule dgrad-p-set-insert)
\{
fix $p q$
assume $p \in F$ and $q \in F$ and $p \neq 0$ and $q \neq 0$
hence crit-pair-cbelow-on dm Fpq by (rule assms(3))
from this dgrad-p-set-subset[OF $\langle m \leq n\rangle$ ] have crit-pair-cbelow-on d n F p q
unfolding crit-pair-cbelow-on-def by (rule cbelow-on-mono)
\}
with $\operatorname{assms}(1)\langle F \subseteq$ dgrad- $p$-set $d n\rangle$ have relation.is-confluent-on (red $F$ ) (dgrad-p-set d $n$ )
by (rule crit-pair-cbelow-imp-confluent-dgrad-p-set)
from this $\langle a \in d$ grad-p-set $d n\rangle$ have $\forall b 1$ b2. $(\text { red } F)^{* *} a b 1 \wedge(\text { red } F)^{* *} a b 2$ $\longrightarrow$ relation.cs (red F) b1 b2
unfolding relation.is-confluent-on-def ..
with $a$ show relation.cs (red $F$ ) b1 b2 by blast
qed
corollary Buchberger-criterion-dgrad-p-set:
assumes dickson-grading $d$ and $F \subseteq d g r a d-p$-set $d m$

```
    assumes \(\bigwedge p q . p \in F \Longrightarrow q \in F \Longrightarrow p \neq 0 \Longrightarrow q \neq 0 \Longrightarrow p \neq q \Longrightarrow\)
                            component-of-term \((l t p)=\) component-of-term \((l t q) \Longrightarrow(\) red
\(F)^{* *}(\) spoly \(p\) q) 0
    shows is-Groebner-basis F
    using assms(1) assms(2)
proof (rule crit-pair-cbelow-imp-GB-dgrad-p-set)
    fix \(p q\)
    assume \(p \in F\) and \(q \in F\) and \(p \neq 0\) and \(q \neq 0\)
    from this(1, 2) assms(2) have \(p: p \in d g r a d-p\)-set \(d m\) and \(q: q \in d g r a d-p\)-set \(d\)
\(m\) by auto
    show crit-pair-cbelow-on d m Fpq
    proof (cases \(p=q\) )
        case True
    from \(\operatorname{assms}(1) q\) show ?thesis unfolding True by (rule crit-pair-cbelow-same)
    next
        case False
        show ?thesis
    proof (cases component-of-term (lt p) \(=\) component-of-term (lt q))
        case True
        from \(\operatorname{assms}(1) \operatorname{assms}(2) p q\langle p \neq 0\rangle\langle q \neq 0\rangle\) show crit-pair-cbelow-on \(d m\)
F p q
    proof (rule spoly-red-zero-imp-crit-pair-cbelow-on)
            from \(\langle p \in F\rangle\langle q \in F\rangle\langle p \neq 0\rangle\langle q \neq 0\rangle\langle p \neq q\rangle\) True show (red \(F\) )** (spoly
p q) 0
                by (rule assms(3))
            qed
    next
            case False
            thus ?thesis by (rule crit-pair-cbelow-distinct-component)
        qed
    qed
qed
lemmas Buchberger-criterion-finite \(=\) Buchberger-criterion-dgrad-p-set[OF dick-son-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemma (in ordered-term) GB-imp-zero-reducibility: assumes is-Groebner-basis \(G\) and \(f \in p m d l G\)
    shows \((\text { red } G)^{* *} f 0\)
proof -
    from in-pmdl-srtc \([O F\langle f \in p m d l G\rangle]\langle i s\)-Groebner-basis \(G\rangle\) have relation.cs (red
G) \(f 0\)
    unfolding is-Groebner-basis-def relation.is-ChurchRosser-def by simp
    then obtain \(s\) where \(\mathrm{rfs}:(\operatorname{red} G)^{* *} f s\) and r0s: \((\text { red } G)^{* *} 0 s\) unfolding
relation.cs-def by auto
    from rtrancl- \(0[O F r 0 s]\) and \(r f s\) show ?thesis by simp
qed
lemma (in ordered-term) GB-imp-reducibility:
```

assumes is-Groebner-basis $G$ and $f \neq 0$ and $f \in \operatorname{pmdl} G$
shows is-red $G f$
using assms by (meson GB-imp-zero-reducibility is-red-def relation.rtrancl-is-final)
lemma is-Groebner-basis-empty: is-Groebner-basis \{\}
by (rule Buchberger-criterion-finite, rule, simp)
lemma is-Groebner-basis-singleton: is-Groebner-basis $\{f\}$
by (rule Buchberger-criterion-finite, simp, simp add: spoly-same)

### 5.3 Buchberger's Criteria for Avoiding Useless Pairs

Unfortunately, the product criterion is only applicable to scalar polynomials.
lemma (in gd-powerprod) product-criterion:
assumes dickson-grading $d$ and $F \subseteq$ punit.dgrad-p-set $d m$ and $p \in F$ and $q \in$ F
and $p \neq 0$ and $q \neq 0$ and gcs (punit.lt $p)($ punit.lt $q)=0$
shows punit.crit-pair-cbelow-on d m F p q
proof -
let ?lt $=$ punit.lt $p$
let $? l q=$ punit.lt $q$
let $? l=l c s ? l t ? l q$
define $s$ where $s=$ punit.monom-mult ( $-1 /($ punit.lc $p *$ punit.lc $q)$ ) 0
(punit.tail $p$ * punit.tail $q$ )
from $\operatorname{assms}(7)$ have $? l=? l t+? l q$ by (metis add-cancel-left-left gcs-plus-lcs)
hence $? l-? l t=? l q$ and $? l-? l q=? l t$ by simp-all
have (punit.red $\{q\})^{* *}($ punit.tail $p *($ monomial (1 / punit.lc $p)($ punit.lt $\left.q))\right)$ (punit.monom-mult $(-(1 /$ punit.lc $p) /$ punit.lc $q) 0$ (punit.tail $p *$ punit.tail q))
unfolding punit-mult-scalar[symmetric] using $\langle q \neq 0\rangle$ by (rule punit.red-mult-scalar-lt)
moreover have punit.monom-mult $(1 /$ punit.lc $p)($ punit.lt $q)($ punit.tail $p)=$ punit.tail $p *($ monomial (1 / punit.lc $p)($ punit.lt $q))$
by (simp add: times-monomial-left[symmetric])
ultimately have (punit.red $\{q\})^{* *}($ fst (punit.crit-pair $\left.p q)\right) s$
by (simp add: punit.crit-pair-def〈?l - ?lt $=$ ?lq> s-def)
moreover from $\langle q \in F\rangle$ have $\{q\} \subseteq F$ by simp
ultimately have 1 : (punit.red $F)^{* *}(f s t($ punit.crit-pair $p q)) s$ by (rule punit.red-rtrancl-subset)
have (punit.red $\{p\})^{* *}($ punit.tail $q *($ monomial $(1 /$ punit.lc $q)($ punit.lt $p)))$ (punit.monom-mult (- (1 / punit.lc q) / punit.lc p) 0 (punit.tail $q *$ punit.tail p))
unfolding punit-mult-scalar[symmetric] using $\langle p \neq 0\rangle$ by (rule punit.red-mult-scalar-lt)
hence (punit.red $\{p\})^{* *}($ snd (punit.crit-pair $\left.p q)\right) s$
by (simp add: punit.crit-pair-def $\langle ? l-? l q=? l t\rangle s$-def mult.commute fip:
times-monomial-left)
moreover from $\langle p \in F\rangle$ have $\{p\} \subseteq F$ by simp
ultimately have 2: $(\text { punit.red } F)^{* *}($ snd (punit.crit-pair $\left.p q)\right) s$ by (rule punit.red-rtrancl-subset)
note $\operatorname{assms}(1) \operatorname{assms}(2)$
moreover from $\langle p \in F\rangle\langle F \subseteq$ punit.dgrad-p-set $d m\rangle$ have $p \in$ punit.dgrad-p-set $d m$..
moreover from $\langle q \in F\rangle\langle F \subseteq$ punit.dgrad-p-set $d m\rangle$ have $q \in$ punit.dgrad- $p$-set $d m$..
moreover from 12 have relation.cs (punit.red $F)$ (fst (punit.crit-pair p q)) (snd (punit.crit-pair p q))
unfolding relation.cs-def by blast
ultimately show ?thesis by (rule punit.crit-pair-cs-imp-crit-pair-cbelow-on)
qed
lemma chain-criterion:
assumes dickson-grading $d$ and $F \subseteq$ dgrad- $p$-set $d m$ and $p \in F$ and $q \in F$
and $p \neq 0$ and $q \neq 0$ and $l p$ radds lcs ( $l p p)(l p q)$
and component-of-term (lt r) $=$ component-of-term (lt $p$ )
and pr:crit-pair-cbelow-on d m F pr and rq: crit-pair-cbelow-on d m Fr q
shows crit-pair-cbelow-on d m F p q
proof (cases component-of-term (lt p) $=$ component-of-term $(l t q))$
case True
with $\operatorname{assms}(8)$ have comp-r: component-of-term (lt r) $=$ component-of-term (lt
q) by $\operatorname{simp}$
let ? $A=d g r a d-p$-set $d m$
let $? R S=\lambda a b$. red $F a b \vee \operatorname{red} F b a$
let ? $l t=l p p$
let $? l q=l p q$
let $? l r=l p r$
let ?ltr = lcs ?lt ?lr
let $? l r q=l c s ? l r ? l q$
let $? l t q=l c s ? l t ? l q$
from $\langle p \in F\rangle\langle F \subseteq$ dgrad- $p$-set $d m\rangle$ have $p \in$ dgrad- $p$-set $d m$..
from this $\langle p \neq 0\rangle$ have $d$ ?lt $\leq m$ by (rule dgrad- $p$-setD-lp)
from $\langle q \in F\rangle\langle F \subseteq$ dgrad- $p$-set $d m\rangle$ have $q \in$ dgrad- $p$-set $d m$..
from this $\langle q \neq 0\rangle$ have $d ? l q \leq m$ by (rule dgrad- $p$-setD-lp)
from $\operatorname{assms}(1)$ have $d ? l t q \leq$ ord-class.max $(d ? l t)$ ( $d$ ?lq) by (rule dick-son-grading-lcs)
also from $\langle d ? l t \leq m\rangle\langle d ? l q \leq m\rangle$ have $\ldots \leq m$ by $\operatorname{simp}$
finally have $d ? l t q \leq m$.
from adds-lcs 〈?lr adds ?ltq〉 have ?ltr adds ?ltq by (rule lcs-adds)
then obtain $u p$ where ? $1 t q=$ ? $1 t r+u p .$.
hence up1: ?ltq - ?lt $=u p+(? l t r-? l t)$ and $u p 2: u p+(? l t r-? l r)=? l t q-$ ?lr
by (metis add.commute adds-lcs minus-plus, metis add.commute adds-lcs-2 minus-plus)
have fst-pq: fst (crit-pair p q) $=$ monom-mult 1 up $(f s t(c r i t-p a i r ~ p r))$
by (simp add: crit-pair-def monom-mult-assoc up1 True comp-r)
from $\operatorname{assms}(1) \operatorname{assms}(2)--p r$
have cbelow-on ?A $\left(\prec_{p}\right)$ (monom-mult 1 up (monomial 1 (term-of-pair (?ltr,

```
component-of-term (lt p))))) ?RS
            (fst (crit-pair p q) ) (monom-mult 1 up (snd (crit-pair pr)))
    unfolding fst-pq crit-pair-cbelow-on-def
    proof (rule cbelow-on-monom-mult)
        from \(\langle d ? l t q \leq m\rangle\) show \(d\) up \(\leq m\) by (simp add: 〈?ltq \(=?\) ?ltr + up〉dick-
son-gradingD1[OF assms(1)])
    qed simp
    hence 1: cbelow-on ? A \(\left(\prec_{p}\right)\) (monomial 1 (term-of-pair (?ltq, component-of-term
(lt p)))) ?RS
    \((f s t(\) crit-pair \(p q))(\) monom-mult 1 up (snd (crit-pair pr)))
    by (simp add: monom-mult-monomial «?ltq \(=\) ? ltr + up〉 add.commute splus-def
term-simps)
    from〈?lr adds ?ltq〉 adds-lcs-2 have ?lrq adds ?ltq by (rule lcs-adds)
    then obtain \(u q\) where \(? l t q=?!r q+u q .\).
    hence \(u q 1: ? l t q-? l q=u q+(? l r q-? l q)\) and \(u q 2: u q+(? l r q-? l r)=? l t q\)
- ?lr
    by (metis add.commute adds-lcs-2 minus-plus, metis add.commute adds-lcs mi-
nus-plus)
    have eq: monom-mult \(1 u q(f s t(\) crit-pair \(r q))=\) monom-mult 1 up (snd (crit-pair
\(p r)\) )
            by (simp add: crit-pair-def monom-mult-assoc up2 uq2 True comp-r)
    have snd-pq: snd (crit-pair p q) = monom-mult 1 uq (snd (crit-pair r q))
        by (simp add: crit-pair-def monom-mult-assoc uq1 True comp-r)
    from \(\operatorname{assms}(1) \operatorname{assms}(2)--r q\)
    have cbelow-on ?A \(\left(\prec_{p}\right)\) (monom-mult 1 uq (monomial 1 (term-of-pair (?lrq,
component-of-term (lt p))))) ?RS
                    (monom-mult 1 uq (fst (crit-pair r q))) (snd (crit-pair p q))
    unfolding snd-pq crit-pair-cbelow-on-def assms(8)
    proof (rule cbelow-on-monom-mult)
        from \(\langle d ? l t q \leq m\rangle\) show \(d u q \leq m\) by (simp add: <?ltq \(=? l r q+u q\rangle\) dick-
son-gradingD1[OF \(\operatorname{assms}(1)])\)
    qed simp
    hence cbelow-on ?A \(\left(\prec_{p}\right)\) (monomial 1 (term-of-pair (?ltq, component-of-term
(lt p)))) ?RS
                                    (monom-mult \(1 u q(f s t(\) crit-pair \(r q)))(\) snd \((\) crit-pair \(p q))\)
    by (simp add: monom-mult-monomial \(\langle ? l t q=? l r q+u q\rangle\) add.commute splus-def
term-simps)
    hence cbelow-on ?A \(\left(\prec_{p}\right)\) (monomial 1 (term-of-pair (?ltq, component-of-term
(lt p)))) ?RS
                                    (monom-mult 1 up (snd (crit-pair pr))) (snd (crit-pair p q))
    by (simp only: eq)
    with 1 show ?thesis unfolding crit-pair-cbelow-on-def by (rule cbelow-on-transitive)
next
    case False
    thus ?thesis by (rule crit-pair-cbelow-distinct-component)
qed
```


### 5.4 Weak and Strong Gröbner Bases

```
lemma ord-p-wf-on:
    assumes dickson-grading d
    shows wfp-on (}\mp@subsup{\prec}{p}{})(dgrad-p-set d m
proof (rule wfp-onI-min)
    fix }x::'t=>\mp@subsup{|}{0}{\prime}b\mathrm{ and }
    assume }x\inQ\mathrm{ and Q}\subseteq\mathrm{ dgrad-p-set d m
    with assms obtain z where z\inQ and *: \y. y \precp z\Longrightarrowy\not\inQ
        by (rule ord-p-minimum-dgrad-p-set, blast)
    from this(1) show }\existsz\inQ.\forally\indgrad-p-set d m. y \mp@subsup{\prec}{p}{}z\longrightarrowy\not\in
    proof
        show }\forally\indgrad-p-set d m. y \mp@subsup{\prec}{p}{}z\longrightarrowy\not\existsQ by (intro ballI impI*
    qed
qed
```

lemma is-red-implies-0-red-dgrad-p-set:
assumes dickson-grading $d$ and $B \subseteq$ dgrad- $p$-set $d m$
assumes $p m d l B \subseteq p m d l A$ and $\bigwedge q . q \in p m d l A \Longrightarrow q \in d g r a d-p-s e t d m \Longrightarrow$
$q \neq 0 \Longrightarrow$ is-red $B q$
and $p \in p m d l A$ and $p \in d g r a d-p$-set $d m$
shows $(\text { red } B)^{* *}$ p 0
proof -
from ord- $p$-wf-on $[O F \operatorname{assms}(1)] \operatorname{assms}(6,5)$ show ?thesis
proof (induction p rule: wfp-on-induct)
case (less p)
show ?case
proof (cases $p=0$ )
case True
thus?thesis by simp
next
case False
from $\operatorname{assms}(4)[O F \operatorname{less}(3,1)$ False $]$ obtain $q$ where redpq: red $B$ p $q$ un-
folding is-red-alt ..
with $\operatorname{assms}$ (1) assms(2) less(1) have $q \in d g r a d-p$-set $d m$ by (rule dgrad-p-set-closed-red)
moreover from redpq have $q \prec_{p} p$ by (rule red-ord)
moreover from $\langle p m d l B \subseteq p m d l A\rangle\langle p \in p m d l A\rangle\langle r e d B p q\rangle$ have $q \in$
pmdl $A$
by (rule pmdl-closed-red)
ultimately have $(\text { red } B)^{* *} q 0$ by (rule less(2))
show ?thesis by (rule converse-rtranclp-into-rtranclp, rule redpq, fact)
qed
qed
qed
lemma is-red-implies-0-red-dgrad-p-set':
assumes dickson-grading $d$ and $B \subseteq d g r a d-p$-set $d m$
assumes $p m d l B \subseteq p m d l A$ and $\bigwedge q . q \in p m d l A \Longrightarrow q \neq 0 \Longrightarrow i s$-red $B q$
and $p \in p m d l A$

```
    shows (red B)** p 0
proof -
    from assms(2) obtain n where m\leqn and p\indgrad-p-set d n and B:B\subseteq
dgrad-p-set d n
    by (rule dgrad-p-set-insert)
    from ord-p-wf-on[OF assms(1)] this(2) assms(5) show ?thesis
    proof (induction p rule: wfp-on-induct)
    case (less p)
    show ?case
    proof (cases p=0)
        case True
        thus ?thesis by simp
    next
        case False
        from assms(4)[OF< < \in (pmdl A)> False] obtain q where redpq: red B p q
unfolding is-red-alt ..
            with assms(1) B<p \indgrad-p-set d n> have q\indgrad-p-set d n by (rule
dgrad-p-set-closed-red)
            moreover from redpq have q}\mp@subsup{\prec}{p}{}p\mathrm{ by (rule red-ord)
            moreover from <pmdl B\subseteqpmdl A\rangle\langlep\inpmdl A\rangle\langlered B p q> have q\in
pmdl A
            by (rule pmdl-closed-red)
            ultimately have (red B)** q 0 by (rule less(2))
            show ?thesis by (rule converse-rtranclp-into-rtranclp, rule redpq, fact)
    qed
    qed
qed
lemma pmdl-eqI-adds-lt-dgrad-p-set:
    fixes G::(' }t=\mp@subsup{=}{0}{\prime}'b::{feld) se
    assumes dickson-grading d and G\subseteqdgrad-p-set d m and B\subseteqdgrad-p-set d m
and pmdl G\subseteqpmdl B
    assumes }\bigwedgef.f\inpmdl B\Longrightarrowf\indgrad-p-set d m\Longrightarrowf\not=0\Longrightarrow(\existsg\inG.
# 0^lt g addst lt f)
    shows pmdl G = pmdl B
proof
    show pmdl B\subseteqpmdl G
    proof (rule pmdl.span-subset-spanI, rule)
        fix }
        assume }p\in
        hence }p\inpmdl B and p\indgrad-p-set d m by (rule pmdl.span-base,rule
intro assms(3))
    with assms(1, 2, 4)- have (red G)** p 0
    proof (rule is-red-implies-0-red-dgrad-p-set)
            fix f
            assume f\inpmdl B and f\indgrad-p-set d m and f}=
            hence (\existsg\inG.g\not=0\wedgelt gaddst
            then obtain g}\mathrm{ where g}\inG\mathrm{ and }g\not=0\mathrm{ and lt g addst lt f by blast
            thus is-red Gf using <f \not=0\rangle is-red-indI1 by blast
```

```
    qed
    thus }p\inpmdl G by (rule red-rtranclp-0-in-pmdl)
    qed
qed fact
lemma pmdl-eqI-adds-lt-dgrad-p-set':
    fixes G::(' }t=\mp@subsup{=}{0}{\prime}'b::field) se
    assumes dickson-grading d and G\subseteqdgrad-p-set d m and pmdl G\subseteqpmdl B
    assumes }\f.f\inpmdl B\Longrightarrowf\not=0\Longrightarrow(\existsg\inG.g\not=0\wedgelt gaddst lt f
    shows pmdl G = pmdl B
proof
    show pmdl B\subseteqpmdl G
    proof
        fix p
        assume p f pmdl B
    with assms(1, 2, 3) - have (red G)** p 0
    proof (rule is-red-implies-0-red-dgrad-p-set')
            fix f
            assume f}\inpmdl B and f\not=
            hence (\existsg\inG.g\not=0^lt gaddst lt f) by (rule assms(4))
            then obtain g}\mathrm{ where g}\inG\mathrm{ and }g\not=0\mathrm{ and lt g addst lt f by blast
            thus is-red Gf using <f \not=0\rangle is-red-indI1 by blast
    qed
```



```
    qed
qed fact
lemma GB-implies-unique-nf-dgrad-p-set:
    assumes dickson-grading d and G\subseteqdgrad-p-set d m
    assumes isGB: is-Groebner-basis G
    shows }\exists\mathrm{ ! h. (red G)** fh}\wedge\negis-red G
proof -
    from assms(1) assms(2) have wfP (red G) -1-1 by (rule red-wf-dgrad-p-set)
    then obtain h}\mathrm{ where ftoh: (red G)** fh and irredh: relation.is-final (red G)h
        by (rule relation.wf-imp-nf-ex)
    show ?thesis
    proof
        from ftoh and irredh show (red G)** f h}\wedge\negis-red G h by (simp add
is-red-def)
    next
        fix }\mp@subsup{h}{}{\prime
        assume (red G)** f h'^\neg is-red G h'
        hence ftoh':(red G)** f h' and irredh': relation.is-final (red G) h' by (simp-all
add: is-red-def)
    show }\mp@subsup{h}{}{\prime}=
    proof (rule relation.ChurchRosser-unique-final)
    from isGB show relation.is-ChurchRosser (red G) by (simp only: is-Groebner-basis-def)
    qed fact+
qed
```


## qed

lemma translation-property':
assumes $p \neq 0$ and red- $p-0:(\text { red } F)^{* *} p 0$
shows is-red $F(p+q) \vee$ is-red $F q$
proof (rule disjCI)
assume not-red: $\neg$ is-red $F q$
from red- $p-0\langle p \neq 0\rangle$ obtain $f$ where $f \in F$ and $f \neq 0$ and $l t$-adds: lt $f$ adds $s_{t}$ lt $p$
by (rule zero-reducibility-implies-lt-divisibility)
show is-red $F(p+q)$
proof (cases $q=0$ )
case True
with is-red-indI1[OF $\langle f \in F\rangle\langle f \neq 0\rangle\langle p \neq 0\rangle l t$-adds] show ?thesis by simp next
case False
from not-red is-red-addsI[OF $\langle f \in F\rangle\langle f \neq 0\rangle$-lt-adds, of $q]$ have $\neg l t p \in$ (keys q) by blast
hence lookup $q$ (lt $p$ ) $=0$ by (simp add: in-keys-iff)
with $l t$-in-keys $[O F\langle p \neq 0\rangle]$ have lt $p \in(k e y s(p+q))$ unfolding in-keys-iff by (simp add: lookup-add)
from is-red-addsI[OF $\langle f \in F\rangle\langle f \neq 0\rangle$ this lt-adds] show ?thesis. qed
qed
lemma translation-property: assumes $p \neq q$ and red- 0 : $(\text { red } F)^{* *}(p-q) 0$ shows is-red $F p \vee i s$-red $F q$
proof -
from $\langle p \neq q\rangle$ have $p-q \neq 0$ by simp
from translation-property' $[$ OF this red- 0 , of $q]$ show ?thesis by simp
qed
lemma weak-GB-is-strong-GB-dgrad-p-set:
assumes dickson-grading $d$ and $G \subseteq$ dgrad- $p$-set $d m$
assumes $\Lambda f . f \in \operatorname{pmdl} G \Longrightarrow f \in$ dgrad-p-set $d m \Longrightarrow(\text { red } G)^{* *} f 0$
shows is-Groebner-basis $G$
using $\operatorname{assms}(1,2)$
proof (rule Buchberger-criterion-dgrad-p-set)
fix $p q$
assume $p \in G$ and $q \in G$
hence $p \in p m d l G$ and $q \in p m d l G$ by (auto intro: pmdl.span-base)
hence spoly $p \quad q \in p m d l G$ by (rule pmdl-closed-spoly)
thus $(\text { red } G)^{* *}($ spoly $p q) 0$
proof (rule assms(3))
note assms (1)
moreover from $\langle p \in G\rangle \operatorname{assms}(2)$ have $p \in d g r a d-p$-set $d m$..
moreover from $\langle q \in G\rangle \operatorname{assms}(2)$ have $q \in$ dgrad- $p$-set $d m$..
ultimately show spoly $p q \in d g r a d-p$-set $d m$ by (rule dgrad-p-set-closed-spoly)
qed
qed
lemma weak-GB-is-strong-GB:
assumes $\wedge f . f \in(p m d l G) \Longrightarrow(\text { red } G)^{* *} f 0$
shows is-Groebner-basis $G$
unfolding is-Groebner-basis-def
proof (rule relation.confluence-implies-ChurchRosser,
simp add: relation.is-confluent-def relation.is-confluent-on-def, intro allI impI,
erule conjE)
fix $f p q$
assume $(\text { red } G)^{* *} f p$ and $(\text { red } G)^{* *} f q$
hence relation.srtc (red $G$ ) $p q$
by (meson relation.rtc-implies-srtc relation.srtc-symmetric relation.srtc-transitive)
hence $p-q \in p m d l G$ by (rule srtc-in-pmdl)
hence $(\text { red } G)^{* *}(p-q) 0$ by (rule assms)
thus relation.cs (red $G$ ) $p q$ by (rule red-diff-rtrancl-cs)
qed
corollary GB-alt-1-dgrad-p-set:
assumes dickson-grading $d$ and $G \subseteq$ dgrad-p-set $d m$
shows is-Groebner-basis $G \longleftrightarrow(\forall f \in$ pmdl $G . f \in$ dgrad-p-set $d m \longrightarrow$ (red $G)^{* *} f 0$ )
using weak-GB-is-strong-GB-dgrad-p-set[OF assms] GB-imp-zero-reducibility by blast
corollary GB-alt-1: is-Groebner-basis $G \longleftrightarrow\left(\forall f \in\right.$ pmdl $\left.G .(\text { red } G)^{* *} f 0\right)$
using weak-GB-is-strong-GB GB-imp-zero-reducibility by blast
lemma isGB-I-is-red:
assumes dickson-grading $d$ and $G \subseteq$ dgrad-p-set $d m$
assumes $\bigwedge f . f \in p m d l G \Longrightarrow f \in$ dgrad- $p$-set $d m \Longrightarrow f \neq 0 \Longrightarrow$ is-red $G f$
shows is-Groebner-basis $G$
unfolding GB-alt-1-dgrad-p-set[OF $\operatorname{assms}(1,2)]$
proof (intro ballI impI)
fix $f$
assume $f \in p m d l G$ and $f \in d g r a d-p$-set $d m$
with assms(1, 2) subset-refl assms(3) show (red G) ${ }^{* *}$ f 0
by (rule is-red-implies-0-red-dgrad-p-set)
qed
lemma GB-alt-2-dgrad-p-set:
assumes dickson-grading $d$ and $G \subseteq$ dgrad-p-set $d m$
shows is-Groebner-basis $G \longleftrightarrow(\forall f \in$ pmdl $G . f \neq 0 \longrightarrow$ is-red $G f)$

## proof

assume is-Groebner-basis $G$
show $\forall f \in p m d l G . f \neq 0 \longrightarrow$ is-red $G f$
proof (intro ballI, intro impI)
fix $f$

```
        assume f\in(pmdl G) and f}\not=
        show is-red Gf by (rule GB-imp-reducibility, fact+)
    qed
next
    assume a2: }\forallf\inpmdl G.f\not=0\longrightarrowis-red G f
    show is-Groebner-basis G unfolding GB-alt-1
    proof
        fix f
        assume f}\inpmdl 
        from assms show (red G)** f0
        proof (rule is-red-implies-0-red-dgrad-p-set')
            fix q
            assume q\inpmdl G and q}\not=
            thus is-red Gq by (rule a2[rule-format])
        qed (fact subset-refl, fact)
    qed
qed
lemma GB-adds-lt:
    assumes is-Groebner-basis G}\mathrm{ and }f\in\mathrm{ pmdl G and f}\not=
    obtains g}\mathrm{ where }g\inG\mathrm{ and }g\not=0\mathrm{ and lt g addst lt f
proof -
    from assms(1) assms(2) have (red G)** f 0 by (rule GB-imp-zero-reducibility)
    show ?thesis by (rule zero-reducibility-implies-lt-divisibility, fact+)
qed
lemma isGB-I-adds-lt:
    assumes dickson-grading d and G}\subseteqdgrad-p-set d m
    assumes \f.f\inpmdl G\Longrightarrowf\indgrad-p-set d m\Longrightarrowf\not=0\Longrightarrow(\existsg\inG.g
# 0^lt g addst lt f)
    shows is-Groebner-basis G
    using assms(1, 2)
proof (rule isGB-I-is-red)
    fix f
    assume f\inpmdl G and f\indgrad-p-set dm and f}=
    hence ( }\existsg\inG.g\not=0\wedgelt g addst lt f) by (rule assms(3)
    then obtain g}\mathrm{ where g}\inG\mathrm{ and }g\not=0\mathrm{ and lt g addst lt f by blast
    thus is-red Gf using <f \not=0\rangle is-red-indI1 by blast
qed
lemma GB-alt-3-dgrad-p-set:
    assumes dickson-grading d and G\subseteqdgrad-p-set d m
    shows is-Groebner-basis }G\longleftrightarrow(\forallf\inpmdl G.f\not=0\longrightarrow(\existsg\inG.g\not=0\wedgel
g addst lt f))
        (is ?L }\longleftrightarrow\mathrm{ ? R)
proof
    assume ?L
    show ?R
    proof (intro ballI impI)
```

fix $f$
assume $f \in p m d l G$ and $f \neq 0$
with $\langle ? L\rangle$ obtain $g$ where $g \in G$ and $g \neq 0$ and $l t g$ adds $s_{t}$ lt $f$ by (rule GB-adds-lt)
thus $\exists g \in G . g \neq 0 \wedge l t g a d d s_{t} l t f$ by blast
qed
next
assume ? $R$
show ?L unfolding GB-alt-2-dgrad-p-set[OF assms]
proof (intro ballI impI)
fix $f$
assume $f \in p m d l G$ and $f \neq 0$
with $\langle ? R\rangle$ have $\left(\exists g \in G . g \neq 0 \wedge l t g a d d s_{t} l t f\right)$ by blast
then obtain $g$ where $g \in G$ and $g \neq 0$ and $l t g$ adds $s_{t}$ lt $f$ by blast
thus is-red $G f$ using $\langle f \neq 0\rangle$ is-red-indI1 by blast
qed
qed
lemma GB-insert:
assumes is-Groebner-basis $G$ and $f \in$ pmdl $G$
shows is-Groebner-basis (insert f $G$ )
using assms unfolding GB-alt-1
by (metis insert-subset pmdl.span-insert-idI red-rtrancl-subset subsetI)
lemma GB-subset:
assumes is-Groebner-basis $G$ and $G \subseteq G^{\prime}$ and $p m d l ~ G^{\prime}=p m d l G$
shows is-Groebner-basis $G^{\prime}$
using assms(1) unfolding GB-alt-1 using assms(2) assms(3) red-rtrancl-subset by blast
lemma (in ordered-term) GB-remove-0-stable-GB:
assumes is-Groebner-basis $G$
shows is-Groebner-basis $(G-\{0\})$
using assms by (simp only: is-Groebner-basis-def red-minus-singleton-zero)
lemmas is-red-implies-0-red-finite $=$ is-red-implies-0-red-dgrad-p-set' $[$ OF dickson-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemmas GB-implies-unique-nf-finite $=$ GB-implies-unique-nf-dgrad-p-set[OF dick-son-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemmas GB-alt-2-finite $=$ GB-alt-2-dgrad-p-set[OF dickson-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemmas GB-alt-3-finite $=$ GB-alt-3-dgrad-p-set[OF dickson-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemmas pmdl-eqI-adds-lt-finite $=p m d l-e q I-a d d s-l t-d g r a d-p-s e t '[O F ~ d i c k s o n-g r a d i n g-d g r a d-d u m m y ~$ dgrad-p-set-exhaust-expl]

### 5.5 Alternative Characterization of Gröbner Bases via Representations of S-Polynomials

```
definition spoly-rep :: ( \({ }^{\prime} a \Rightarrow\) nat \() \Rightarrow\) nat \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) set \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow{ }_{0}\right.\)
\(' b::\) field \() \Rightarrow\) bool
    where spoly-rep dmGg1g2 \(\longleftrightarrow\left(\exists q\right.\). spoly \(g 1 g 2=\left(\sum g \in G . q g \odot g\right) \wedge\)
                        ( \(\forall g . q g \in\) punit.dgrad-p-set d \(m \wedge\)
                                    \(\left(q g \odot g \neq 0 \longrightarrow l t(q g \odot g) \prec_{t}\right.\) term-of-pair (lcs (lpg1) (lp
g2),
                                    component-of-term (lt g2)))))
```

lemma spoly-rep I:
spoly g1 g2 $=\left(\sum g \in G . q g \odot g\right) \Longrightarrow(\bigwedge g . q g \in$ punit.dgrad-p-set d $m) \Longrightarrow$
$\left(\bigwedge g . q g \odot g \neq 0 \Longrightarrow l t(q g \odot g) \prec_{t}\right.$ term-of-pair (lcs (lpg1) (lp g2),
component-of-term (lt g2))) $\Longrightarrow$
spoly-rep $d m$ g1 g2
by (auto simp: spoly-rep-def)
lemma spoly-repI-zero:
assumes spoly g1 g2 $=0$
shows spoly-rep d m G g1 g2
proof (rule spoly-repI)
show spoly g1 g2 $=\left(\sum g \in G .0 \odot g\right)$ by (simp add: assms $)$
qed (simp-all add: punit.zero-in-dgrad-p-set)
lemma spoly-repE:
assumes spoly-rep d m G g1 g2
obtains $q$ where spoly $g 1$ g2 $=\left(\sum g \in G . q g \odot g\right)$ and $\wedge g . q g \in$ punit.dgrad-p-set
$d m$
and $\bigwedge g . q g \odot g \neq 0 \Longrightarrow l t(q g \odot g) \prec_{t}$ term-of-pair (lcs (lp g1) (lp g2),
component-of-term (lt g2))
using assms by (auto simp: spoly-rep-def)
corollary isGB-D-spoly-rep:
assumes dickson-grading $d$ and is-Groebner-basis $G$ and $G \subseteq$ dgrad-p-set $d m$
and finite $G$
and $g 1 \in G$ and $g 2 \in G$ and $g 1 \neq 0$ and $g 2 \neq 0$
shows spoly-rep d m G g1 g2
proof (cases spoly g1 g2 $=0$ )
case True
thus ?thesis by (rule spoly-repI-zero)
next
case False
let ?v $=$ term-of-pair (lcs (lp g1) (lp g2), component-of-term (lt g1))
let $? h=$ crit-pair g1 g2
from $\operatorname{assms}(7,8)$ have eq: spoly g1 g2 $=f s t ? h+(-$ snd $? h)$ by (simp add:
spoly-alt)
have $f$ st ? $h \prec_{p}$ monomial 1 ?v by (fact fst-crit-pair-below-lcs)
hence $d 1: f s t ? h=0 \vee l t(f s t ? h) \prec_{t} ? v$ by (simp only: ord-strict- $p$-monomial-iff)
have snd ? $h \prec_{p}$ monomial 1 ?v by (fact snd-crit-pair-below-lcs)
hence $d 2:$ snd $? h=0 \vee l t(-s n d ? h) \prec_{t} ? v$ by (simp only: ord-strict-p-monomial-iff lt-uminus)
note $\operatorname{assms}(1)$
moreover from $\operatorname{assms}(5,3)$ have $g 1 \in d g r a d-p$-set $d m$..
moreover from $\operatorname{assms}(6,3)$ have $g 2 \in d g r a d-p$-set $d m$..
ultimately have spoly g1 g2 $\in$ dgrad-p-set $d m$ by (rule dgrad-p-set-closed-spoly)
from assms(5) have $g 1 \in p m d l G$ by (rule pmdl.span-base)
moreover from $\operatorname{assms}(6)$ have $g 2 \in p m d l G$ by (rule pmdl.span-base)
ultimately have spoly g1 g2 $\operatorname{pmdl} G$ by (rule pmdl-closed-spoly)
with assms(2) have (red G)** (spoly g1 g2) 0 by (rule GB-imp-zero-reducibility)
with $\operatorname{assms}(1,3,4)<$ spoly $-\in$ dgrad-p-set $->$ obtain $q$
where 1: spoly g1 $g 2=0+\left(\sum g \in G . q g \odot g\right)$ and 2: $\bigwedge g . q g \in$ punit.dgrad- $p$-set $d m$
and $\bigwedge g$. lt $(q g \odot g) \preceq_{t} l t($ spoly $g 1 g 2)$ by (rule red-rtrancl-repE) blast
show ?thesis
proof (rule spoly-repI)
fix $g$
note $\left\langle l t(q g \odot g) \preceq_{t} l t(\right.$ spoly g1 g2) $)$
also from $d 1$ have $l t$ (spoly g1 g2) $\prec_{t}$ ?v
proof
assume $f s t ? h=0$
hence eq: spoly g1 g2 $=-$ snd ? $h$ by (simp add: eq)
also from $d 2$ have $l t \ldots \prec_{t}$ ?v
proof
assume snd $? h=0$
with False show ?thesis by (simp add: eq)
qed
finally show ?thesis .
next
assume $*: l t(f s t ? h) \prec_{t}$ ?v
from d2 show ?thesis
proof
assume snd $? h=0$
with $*$ show ?thesis by (simp add: eq)
next
assume $* *$ : lt $(-$ snd ? $h) \prec_{t}$ ?v
have lt (spoly g1 g2) $\preceq_{t}$ ord-term-lin.max $(l t(f s t ~ ? h))(l t(-$ snd ?h))
unfolding $e q$
by (fact lt-plus-le-max)
also from $* * *$ have $\ldots \prec_{t}$ ? $v$ by (simp only: ord-term-lin.max-less-iff-conj)
finally show ?thesis .
qed
qed
also from False have $\ldots=$ term-of-pair (lcs (lp g1) (lp g2), component-of-term (lt g2))
by (simp add: spoly-def Let-def split: if-split-asm)
finally show $l t(q g \odot g) \prec_{t}$ term-of-pair (lcs (lp g1) (lp g2), component-of-term (lt g2)).
qed (simp-all add: 1 2)

## qed

The finiteness assumption on $G$ in the following theorem could be dropped, but it makes the proof a lot easier (although it is still fairly complicated).

```
lemma isGB-I-spoly-rep:
    assumes dickson-grading d and G\subseteqdgrad-p-set d m and finite G
        and \g1 g2.g1 }\G\Longrightarrowg2\inG\Longrightarrowg1\not=0\Longrightarrowg2\not=0\Longrightarrow spoly g1 g2 # =
0\Longrightarrow spoly-rep d m G g1 g2
    shows is-Groebner-basis G
proof (rule ccontr)
    assume ᄀ is-Groebner-basis G
    then obtain p where p\inpmdl G and p-in: p\indgrad-p-set d m and }\neg\mathrm{ (red
G)** p 0
    by (auto simp: GB-alt-1-dgrad-p-set[OF assms(1, 2)])
    from }\neg \s\mathrm{ -Groebner-basis }G`\mathrm{ have G}={}\mathrm{ by (auto simp: is-Groebner-basis-empty)
    obtain r where p-red: (red G)** pr and r-irred: ᄀ is-red Gr
    proof -
    define }A\mathrm{ where }A={q.(\operatorname{red}G\mp@subsup{)}{}{**}pq
    from assms(1, 2) have wfP (red G) -1-1 by (rule red-wf-dgrad-p-set)
    moreover have p\inA by (simp add: A-def)
    ultimately obtain r where r}\inA\mathrm{ and r-min: }\z.(\mathrm{ red G) -1-1 zr mz#
A
        by (rule wfE-min[to-pred]) blast
    show ?thesis
    proof
        from }\langler\inA\rangle\mathrm{ show *: (red G)** pr by (simp add: A-def)
        show \neg is-red Gr
        proof
            assume is-red Gr
            then obtain z where (red G) r z by (rule is-redE)
            hence (red G)
            hence z}\not\inA\mathrm{ by (rule r-min)
            hence }\neg(\mathrm{ red G)** pz by (simp add: A-def)
            moreover from * 〈(red G) r z> have (red G)** p z ..
            ultimately show False ..
        qed
    qed
qed
from assms(1, 2) p-in p-red have r-in: r\indgrad-p-set d m by (rule dgrad-p-set-closed-red-rtrancl)
from p-red \prec\neg(red G)** p 0` have r\not= 0 by blast
from p-red have p-r\in pmdl G by (rule red-rtranclp-diff-in-pmdl)
with }<p\inpmdl G` have p-(p-r)\inpmdl G by (rule pmdl.span-diff
hence }r\inpmdl G by sim
with assms(3) obtain q0 where r: r=(\sumg\inG.q0 g\odotg) by (rule pmdl.span-finiteE)
from assms(3) have finite (q0'G) by (rule finite-imageI)
then obtain m0 where q0' G\subseteq punit.dgrad-p-set d m0 by (rule punit.dgrad-p-set-exhaust)
```


have dgrad-p-set $d m \subseteq d g r a d-p$-set $d m^{\prime}$ by (rule dgrad-p-set-subset) (simp add: $m^{\prime}$-def)
with $\operatorname{assms}(2)$ have $G$-sub: $G \subseteq$ dgrad-p-set $d m^{\prime}$ by (rule subset-trans)
have punit.dgrad-p-set $d m 0 \subseteq$ punit.dgrad-p-set $d m^{\prime}$
by (rule punit.dgrad-p-set-subset) (simp add: $m^{\prime}$-def)
with $\left\langle q 0^{\prime} G \subseteq->\right.$ have $q 0^{\prime} G \subseteq$ punit.dgrad-p-set $d m^{\prime}$ by (rule subset-trans)
define $m l t$ where $m l t=(\lambda q$. ord-term-lin.Max $(l t '\{q g \odot g \mid g . g \in G \wedge q g$ $\odot g \neq 0\})$ )
define mnum where mnum $=(\lambda q$. card $\{g \in G . q g \odot g \neq 0 \wedge l t(q g \odot g)=$ mlt qu)
define rel where rel $=\left(\lambda q 1\right.$ q2. mlt $q 1 \prec_{t}$ mlt $q 2 \vee(m l t q 1=m l t q 2 \wedge m n u m$ q1 < mnum q2))
define rel-dom where rel-dom $=\left\{q . q^{\prime} G \subseteq\right.$ punit.dgrad-p-set $d m^{\prime} \wedge r=$ $\left.\left(\sum g \in G . q g \odot g\right)\right\}$
have mlt-in: mlt $q \in l t$ ' $\{q g \odot g \mid g . g \in G \wedge q g \odot g \neq 0\}$ if $q \in$ rel-dom for $q$
unfolding mlt-def
proof (rule ord-term-lin.Max-in, simp-all add: assms(3), rule ccontr)
assume $\nexists g . g \in G \wedge q g \odot g \neq 0$
hence $q g \odot g=0$ if $g \in G$ for $g$ using that by simp
with that have $r=0$ by (simp add: rel-dom-def)
with $\langle r \neq 0\rangle$ show False ..
qed
have rel-dom-dgrad-set: pp-of-term ' mlt 'rel-dom $\subseteq$ dgrad-set d m'
proof (rule subsetI, elim imageE)
fix $q v t$
assume $q \in$ rel-dom and $v: v=m l t q$ and $t: t=p p$-of-term $v$
from this(1) have $v \in l t$ ' $\{q g \odot g \mid g . g \in G \wedge q g \odot g \neq 0\}$ unfolding $v$ by (rule mlt-in)
then obtain $g$ where $g \in G$ and $q g \odot g \neq 0$ and $v: v=l t(q g \odot g)$ by blast
from this(2) have $q g \neq 0$ and $g \neq 0$ by auto
hence $v=$ punit.lt $(q g) \oplus l t g$ unfolding $v$ by (rule lt-mult-scalar)
hence $t=$ punit.lt $(q g)+l p g$ by (simp add: tpp-of-term-splus)
also from $\operatorname{assms}(1)$ have $d \ldots=\operatorname{ord}$-class.max $(d($ punit.lt $(q g)))(d(l p g))$ by (rule dickson-gradingD1)
also have $\ldots \leq m^{\prime}$
proof (rule max.boundedI)
from $\langle g \in G\rangle\left\langle q \in\right.$ rel-dom> have $q g \in$ punit.dgrad-p-set $d m^{\prime}$ by (auto simp: rel-dom-def)
moreover from $\langle q g \neq 0\rangle$ have punit.lt $(q g) \in$ keys $(q g)$ by (rule punit.lt-in-keys)
ultimately show $d$ (punit.lt $(q g)) \leq m^{\prime}$ by (rule punit.dgrad-p-setD[simplified]) next
from $\langle g \in G\rangle G$-sub have $g \in d g r a d-p$-set $d m^{\prime}$..
moreover from $\langle g \neq 0\rangle$ have $l t g \in$ keys $g$ by (rule lt-in-keys)

```
            ultimately show d (lp g) \leq m' by (rule dgrad-p-setD)
    qed
    finally show }t\indgrad-set d m' by (simp add: dgrad-set-def
    qed
    obtain q}\mathrm{ where q}\in\mathrm{ rel-dom and q-min: \ \q'.rel q' }q\Longrightarrow\mp@subsup{q}{}{\prime}\not\in\mathrm{ rel-dom
    proof -
    from <q0' }G\subseteq\mathrm{ punit.dgrad-p-set d m'> have q0 }\in\mathrm{ rel-dom by (simp add:
rel-dom-def r)
    hence mlt q0 \in mlt'rel-dom by (rule imageI)
    with assms(1) obtain }u\mathrm{ where }u\inmlt'rel-dom and u-min: \w. w < < u
"w\not\in mlt`rel-dom
            using rel-dom-dgrad-set by (rule ord-term-minimum-dgrad-set) blast
    from this(1) obtain q}\mp@subsup{q}{}{\prime}\mathrm{ where q}\mp@subsup{q}{}{\prime}\in\mathrm{ rel-dom and u:u=mlt q'..
    hence q'\in rel-dom \cap {q. mlt q=u} (is - \in?A) by simp
    hence mnum q' }\in\mathrm{ mnum'?A by (rule imageI)
    with wf[to-pred] obtain k where k m mum'?A and k-min: \bigwedgel.l < k\Longrightarrowl
&mnum'?A
            by (rule wfE-min[to-pred]) blast
    from this(1) obtain q'" where q'|}\mp@subsup{q}{}{\prime\prime}\in\mathrm{ rel-dom and mlt': mlt }\mp@subsup{q}{}{\prime\prime}=u\mathrm{ and }k:
=mnum q"
            by blast
    from this(1) show ?thesis
    proof
            fix q0
            assume rel q0 q"
            show q0 # rel-dom
            proof
            assume q0 \in rel-dom
            from <rel q0 q'\ show False unfolding rel-def
            proof (elim disjE conjE)
                assume mlt q0 \prec}\mp@subsup{t}{tlt ql'}{\prime\prime
                    hence mlt q0 #mlt 'rel-dom unfolding mlt' by (rule u-min)
                    moreover from «q0 \in rel-dom> have mlt q0 \in mlt` rel-dom by (rule
imageI)
                    ultimately show ?thesis ..
            next
                    assume mlt q0 = mlt q"
                    with «q0 \in rel-dom` have q0 \in?A by (simp add: mlt'")
                    assume mnum q0 < mnum q'
                    hence mnum q0 \not\inmnum'?A unfolding k[symmetric] by (rule k-min)
                    with }\langleq0\in?A`\mathrm{ show ?thesis by blast
            qed
        qed
    qed
qed
from this(1) have q-in: \g.g\inG\Longrightarrowqg\in punit.dgrad-p-set d m'
    and}r:r=(\sumg\inG.qg\odotg) by (auto simp: rel-dom-def
```

define $v$ where $v=m l t q$
from $\langle q \in$ rel-dom〉 have $v \in l t ‘\{q g \odot g \mid g . g \in G \wedge q g \odot g \neq 0\}$ unfolding $v$-def
by (rule mlt-in)
then obtain $g 1$ where $g 1 \in G$ and $q g 1 \odot g 1 \neq 0$ and $v 1: v=l t(q g 1 \odot$ g1) by blast
moreover define $M$ where $M=\{g \in G . q g \odot g \neq 0 \wedge l t(q g \odot g)=v\}$
ultimately have $g 1 \in M$ by $\operatorname{simp}$
have $v$-max: lt $(q g \odot g) \prec_{t} v$ if $g \in G$ and $g \notin M$ and $q g \odot g \neq 0$ for $g$
proof -
from that have $l t(q g \odot g) \neq v$ by (auto simp: $M$-def)
moreover have $l t(q g \odot g) \preceq_{t} v$ unfolding $v$-def mlt-def
by (rule ord-term-lin.Max-ge) (auto simp: assms(3) $\langle q g \odot g \neq 0\rangle$ intro!:
imageI $\langle g \in G\rangle)$
ultimately show?thesis by simp
qed
from $\langle q g 1 \odot g 1 \neq 0\rangle$ have $q g 1 \neq 0$ and $g 1 \neq 0$ by auto
hence $v 1^{\prime}: v=$ punit.lt $(q g 1) \oplus l t g 1$ unfolding $v 1$ by (rule lt-mult-scalar)
have $M-\{g 1\} \neq\{ \}$
proof
assume $M-\{g 1\}=\{ \}$
have $v \in$ keys $(q g 1 \odot g 1)$ unfolding $v 1$ using $\langle q g 1 \odot g 1 \neq 0\rangle$ by (rule lt-in-keys)
moreover have $v \notin$ keys $\left(\sum g \in G-\{g 1\} . q g \odot g\right)$
proof
assume $v \in$ keys $\left(\sum g \in G-\{g 1\} . q g \odot g\right)$
also have $\ldots \subseteq(\bigcup g \in G-\{g 1\}$. keys $(q g \odot g))$ by (fact keys-sum-subset)
finally obtain $g$ where $g \in G-\{g 1\}$ and $v \in$ keys $(q g \odot g)$..
from this(2) have $q g \odot g \neq 0$ and $v \preceq_{t} l t(q g \odot g)$ by (auto intro: lt-max-keys)
from $\langle g \in G-\{g 1\}\rangle\langle M-\{g 1\}=\{ \}\rangle$ have $g \in G$ and $g \notin M$ by blast+ hence $l t(q g \odot g) \prec_{t} v$ by (rule $v$-max) fact
with $\left\langle v \preceq_{t} \rightarrow\right.$ show False by simp
qed
ultimately have $v \in \operatorname{keys}\left(q g 1 \odot g 1+\left(\sum g \in G-\{g 1\} . q g \odot g\right)\right)$ by (rule in-keys-plusI1)
also from $\langle g 1 \in G\rangle \operatorname{assms}(3)$ have $\ldots=$ keys $r$ by (simp add: $r$ sum.remove)
finally have $v \in$ keys $r$.
with $\langle g 1 \in G\rangle\langle g 1 \neq 0\rangle$ have $i s$-red $G r$ by (rule is-red-addsI) (simp add: v1' term-simps)
with r-irred show False ..
qed
then obtain $g_{2}$ where $g 2 \in M$ and $g 1 \neq g 2$ by blast
from this (1) have $g_{2} \in G$ and $q g_{2} \odot g_{2} \neq 0$ and $v 2: v=l t\left(q g_{2} \odot g_{2}\right)$ by (simp-all add: M-def)
from this(2) have $q g^{2} \neq 0$ and $g 2 \neq 0$ by auto
hence $v^{\prime} 2^{\prime}: v=$ punit.lt $(q g 2) \oplus l t ~ g 2$ unfolding $v 2$ by (rule $l t$-mult-scalar)
hence component-of-term (punit.lt ( $q$ g1) $\oplus$ lt g1) $=$ component-of-term (punit.lt $(q g 2) \oplus l t g 2)$

```
    by (simp only: v1' flip: v2')
    hence cmp-eq: component-of-term (lt g1) \(=\) component-of-term (lt g2) by (simp
add: term-simps)
    have \(M \subseteq G\) by (simp add: \(M\)-def)
    have \(r=q g 1 \odot g 1+\left(\sum g \in G-\{g 1\} . q g \odot g\right)\)
    using \(\operatorname{assms}(3)\langle g 1 \in G\rangle\) by (simp add: \(r\) sum.remove)
    also have \(\ldots=q g 1 \odot g 1+q g 2 \odot g 2+\left(\sum g \in G-\{g 1\}-\{g 2\} . q g \odot g\right)\)
    using \(\operatorname{assms}(3)\langle g 2 \in G\rangle\langle g 1 \neq g 2\rangle\)
    by (metis (no-types, lifting) add.assoc finite-Diff insert-Diff insert-Diff-single
insert-iff
sum.insert-remove)
    finally have \(r: r=q g 1 \odot g 1+q g 2 \odot g 2+\left(\sum g \in G-\{g 1, g 2\} . q g \odot g\right)\)
    by (simp flip: Diff-insert2)
    let ?l \(=l c s(l p g 1)(l p g 2)\)
    let ?v \(=\) term-of-pair (?l, component-of-term (lt g2))
    have lp g1 adds lp (q g1 \(\odot g 1)\) by (simp add: v1' pp-of-term-splus fip: v1)
    moreover have lp g2 adds lp (q g1 \(\odot\) g1) by (simp add: v2' pp-of-term-splus
flip: v1)
    ultimately have l-adds: ?l adds lp (q g1 \(\odot\) g1) by (rule lcs-adds)
    have spoly-rep dmag1 g2
    proof (cases spoly g1 g2 \(=0\) )
        case True
        thus ?thesis by (rule spoly-repI-zero)
    next
        case False
        with \(\langle g 1 \in G\rangle\langle g 2 \in G\rangle\langle g 1 \neq 0\rangle\langle g 2 \neq 0\rangle\) show ?thesis by (rule assms(4))
    qed
    then obtain \(q^{\prime}\) where spoly: spoly \(g 1 g 2=\left(\sum g \in G \cdot q^{\prime} g \odot g\right)\)
    and \(\bigwedge g . q^{\prime} g \in\) punit.dgrad-p-set \(d m\) and \(\bigwedge g . q^{\prime} g \odot g \neq 0 \Longrightarrow l t\left(q^{\prime} g \odot g\right)\)
\(\prec_{t}\) ? \(v\)
    by (rule spoly-repE) blast
    note this(2)
    also have punit.dgrad-p-set \(d m \subseteq\) punit.dgrad-p-set \(d m^{\prime}\)
    by (rule punit.dgrad-p-set-subset) (simp add: \(m^{\prime}\)-def)
    finally have \(q^{\prime}\)-in: \(\bigwedge g . q^{\prime} g \in\) punit.dgrad- \(p\)-set \(d m^{\prime}\).
    define \(m u\) where \(m u=\) monomial \((l c(q g 1 \odot g 1))(l p(q g 1 \odot g 1)-? l)\)
    define \(m u 1\) where \(m u 1=\) monomial \((1 / l c g 1)(? l-l p g 1)\)
    define mu2 where mu2 \(=\) monomial \((1 / l c\) g2 \()(? l-l p g 2)\)
    define \(q^{\prime \prime}\) where \(q^{\prime \prime}=\left(\lambda g . q g+m u * q^{\prime} g\right)\)
                                    \(\left(g 1:=\right.\) punit.tail \((q g 1)+m u * q^{\prime} g 1, g 2:=q g 2+m u * q^{\prime}\)
\(g 2+m u * m u 2)\)
    from \(\langle q g 1 \odot g 1 \neq 0\rangle\) have \(m u \neq 0\) by (simp add: mu-def monomial-0-iff
lc-eq-zero-iff)
    from \(\langle g 1 \neq 0\rangle l\)-adds have \(m u\)-times-mu1: \(m u * m u 1=\) monomial (punit.lc \((q\)
g1)) (punit.lt (q g1))
```

by (simp add: mu-def mu1-def times-monomial-monomial lc-mult-scalar lc-eq-zero-iff minus-plus-minus-cancel adds-lcs v1' pp-of-term-splus flip: v1)
from l-adds have mu-times-mu2: mu* mu2 $=$ monomial $(l c(q g 1 \odot g 1) / l c$ g2) (punit.lt ( $q$ g2))
by (simp add: mu-def mu2-def times-monomial-monomial lc-mult-scalar mi-nus-plus-minus-cancel adds-lcs-2 v2' pp-of-term-splus flip: v1)
have mu1 $\odot g 1-m u 2 \odot g 2=$ spoly $g 1$ g2
by (simp add: spoly-def Let-def cmp-eq lc-def mult-scalar-monomial mu1-def mu2-def)
also have $\ldots=q^{\prime} g 1 \odot g 1+\left(\sum g \in G-\{g 1\} \cdot q^{\prime} g \odot g\right)$
using $\operatorname{assms}(3)\langle g 1 \in G\rangle$ by (simp add: spoly sum.remove)
also have $\ldots=q^{\prime} g 1 \odot g 1+q^{\prime} g 2 \odot g 2+\left(\sum g \in G-\{g 1\}-\{g 2\} . q^{\prime} g \odot g\right)$
using $\operatorname{assms}(3)\langle g 2 \in G\rangle\langle g 1 \neq g 2\rangle$
by (metis (no-types, lifting) add.assoc finite-Diff insert-Diff insert-Diff-single insert-iff
sum.insert-remove)
finally have $\left(q^{\prime} g 1-m u 1\right) \odot g 1+\left(q^{\prime} g 2+m u 2\right) \odot g 2+\left(\sum g \in G-\{g 1\right.$, $\left.g 2\} \cdot q^{\prime} g \odot g\right)=0$
by (simp add: algebra-simps fip: Diff-insert2)
hence $0=m u \odot\left(\left(q^{\prime} g 1-m u 1\right) \odot g 1+\left(q^{\prime} g 2+m u 2\right) \odot g 2+\left(\sum g \in G-\right.\right.$ $\left.\{g 1, g 2\} . q^{\prime} g \odot g\right)$ ) by simp
also have $\ldots=\left(m u * q^{\prime} g 1-m u * m u 1\right) \odot g 1+\left(m u * q^{\prime} g 2+m u * m u 2\right)$ - $g_{2}+$

$$
\left(\sum g \in G-\{g 1, g 2\} \cdot\left(m u * q^{\prime} g\right) \odot g\right)
$$

by (simp add: mult-scalar-distrib-left sum-mult-scalar-distrib-left distrib-left right-diff-distrib
flip: mult-scalar-assoc)
finally have $r=r+\left(m u * q^{\prime} g 1-m u * m u 1\right) \odot g 1+\left(m u * q^{\prime} g 2+m u *\right.$ mu2) $\odot g 2+$

$$
\left(\sum g \in G-\{g 1, g 2\} \cdot\left(m u * q^{\prime} g\right) \odot g\right) \text { by } \operatorname{simp}
$$

also have $\ldots=\left(q g 1-m u * m u 1+m u * q^{\prime} g 1\right) \odot g 1+\left(q g 2+m u * q^{\prime} g 2\right.$ $+m u * m u 2) \odot g 2+$

$$
\left(\sum g \in G-\{g 1, g 2\} .\left(q g+m u * q^{\prime} g\right) \odot g\right)
$$

by (simp add: $r$ algebra-simps flip: sum.distrib)
also have $q g 1-m u * m u 1=$ punit.tail $(q g 1)$
by (simp only: mu-times-mu1 punit.leading-monomial-tail diff-eq-eq add.commute[of punit.tail (qg1)])
finally have $r=q^{\prime \prime} g 1 \odot g 1+q^{\prime \prime} g 2 \odot g 2+\left(\sum g \in G-\{g 1\}-\{g 2\} \cdot q^{\prime \prime} g\right.$ $\odot g)$
using $\langle g 1 \neq g 2\rangle$ by (simp add: $q^{\prime \prime}$-def flip: Diff-insert2)
also from $\langle$ finite $G\rangle\langle g 1 \neq g 2\rangle\langle g 1 \in G\rangle\langle g 2 \in G\rangle$ have $\ldots=\left(\sum g \in G . q^{\prime \prime} g\right.$ $\odot g)$
by (simp add: sum.remove) (metis (no-types, lifting) finite-Diff insert-Diff insert-iff sum.remove)
finally have $r: r=\left(\sum g \in G . q^{\prime \prime} g \odot g\right)$.
have 1: lt $\left(\left(m u * q^{\prime} g\right) \odot g\right) \prec_{t} v$ if $\left(m u * q^{\prime} g\right) \odot g \neq 0$ for $g$
proof -
from that have $q^{\prime} g \odot g \neq 0$ by (auto simp: mult-scalar-assoc)
hence $*: l t\left(q^{\prime} g \odot g\right) \prec_{t}$ ?v by fact
from $\left\langle q^{\prime} g \odot g \neq 0\right\rangle\langle m u \neq 0\rangle$ have $l t\left(\left(m u * q^{\prime} g\right) \odot g\right)=(l p(q g 1 \odot g 1)$
$-? l) \oplus l t\left(q^{\prime} g \odot g\right)$
by (simp add: mult-scalar-assoc lt-mult-scalar) (simp add: mu-def punit.lt-monomial monomial-0-iff)
also from $*$ have $\ldots \prec_{t}(l p(q g 1 \odot g 1)-? l) \oplus ? v$ by (rule splus-mono-strict)
also from $l$-adds have $\ldots=v$ by (simp add: splus-def minus-plus term-simps v1' flip: cmp-eq v1)
finally show ?thesis .
qed
have 2: lt $\left(q^{\prime \prime} g 1 \odot g 1\right) \prec_{t} v$ if $q^{\prime \prime} g 1 \odot g 1 \neq 0$ using that
proof (rule lt-less)
fix $u$
assume $v \preceq_{t} u$
have $u \notin$ keys $\left(q^{\prime \prime} g 1 \odot g 1\right)$
proof
assume $u \in$ keys $\left(q^{\prime \prime} g 1 \odot g 1\right)$
also from $\langle g 1 \neq g 2\rangle$ have $\ldots=$ keys $\left(\left(\right.\right.$ punit.tail $\left.(q g 1)+m u * q^{\prime} g 1\right) \odot$
g1)
by (simp add: $q^{\prime \prime}$-def)
also have $\ldots \subseteq$ keys $($ punit.tail $(q g 1) \odot g 1) \cup$ keys $\left(\left(m u * q^{\prime} g 1\right) \odot g 1\right)$
unfolding mult-scalar-distrib-right by (fact Poly-Mapping.keys-add)
finally show False
proof
assume $u \in$ keys (punit.tail $(q g 1) \odot g 1)$
hence $u \preceq_{t} l t$ (punit.tail ( $q$ g1) $\odot$ g1) by (rule lt-max-keys)
also have $\ldots \preceq_{t}$ punit.lt (punit.tail $(q$ g1)) $\oplus l t ~ g 1$
by (metis in-keys-mult-scalar-le lt-def lt-in-keys min-term-min)
also have $\ldots \prec_{t}$ punit.lt ( $q$ g1) $\oplus l t$ g1
proof (intro splus-mono-strict-left punit.lt-tail notI)
assume punit.tail ( $q$ g1) $=0$
with $\langle u \in$ keys (punit.tail $(q g 1) \odot g 1)\rangle$ show False by simp
qed
also have $\ldots=v$ by (simp only: v1')
finally show ?thesis using $\left\langle v \preceq_{t} u\right\rangle$ by simp
next
assume $u \in$ keys $\left(\left(m u * q^{\prime} g 1\right) \odot g 1\right)$
hence $\left(m u * q^{\prime} g 1\right) \odot g 1 \neq 0$ and $u \preceq_{t} l t\left(\left(m u * q^{\prime} g 1\right) \odot g 1\right)$ by (auto intro: lt-max-keys)
note this(2)
also from $\left\langle\left(m u * q^{\prime} g 1\right) \odot g 1 \neq 0\right\rangle$ have $l t\left(\left(m u * q^{\prime} g 1\right) \odot g 1\right) \prec_{t} v$ by
(rule 1)
finally show ?thesis using $\left\langle v \preceq_{t} u\right\rangle$ by simp
qed
qed
thus lookup $\left(q^{\prime \prime} g 1 \odot g 1\right) u=0$ by (simp add: in-keys-iff)
qed
have 3: lt $\left(q^{\prime \prime} g 2 \odot g 2\right) \preceq_{t} v$ proof (rule lt-le)
fix $u$
assume $v \prec_{t} u$
have $u \notin$ keys $\left(q^{\prime \prime} g 2 \odot g 2\right)$
proof
assume $u \in$ keys $\left(q^{\prime \prime} g 2 \odot g 2\right)$
also have $\ldots=$ keys $\left(\left(q g_{2}+m u * q^{\prime} g 2+m u * m u 2\right) \odot g_{2}\right)$ by (simp
add: $q^{\prime \prime}$-def)
also have $\ldots \subseteq$ keys $\left(q g 2 \odot g 2+\left(m u * q^{\prime} g 2\right) \odot g 2\right) \cup$ keys $((m u * m u 2)$
$\odot g 2)$
unfolding mult-scalar-distrib-right by (fact Poly-Mapping.keys-add)
finally show False
proof
assume $u \in$ keys $\left(q g_{2} \odot g_{2}+\left(m u * q^{\prime} g 2\right) \odot g_{2}\right)$
also have $\ldots \subseteq$ keys $(q g 2 \odot g 2) \cup$ keys $\left(\left(m u * q^{\prime} g 2\right) \odot g 2\right)$ by $(f a c t$
Poly-Mapping.keys-add)
finally show ?thesis
proof
assume $u \in$ keys $\left(q g_{2} \odot g^{2}\right)$
hence $u \preceq_{t} l t(q g 2 \odot g 2)$ by (rule lt-max-keys)
with $\left\langle v \prec_{t} u\right\rangle$ show ?thesis by (simp add: v2)
next
assume $u \in$ keys $\left(\left(m u * q^{\prime} g 2\right) \odot g 2\right)$
hence $\left(m u * q^{\prime} g 2\right) \odot g 2 \neq 0$ and $u \preceq_{t} l t\left(\left(m u * q^{\prime} g 2\right) \odot g 2\right)$ by (auto intro: lt-max-keys)
note this(2)
also from $\left\langle\left(m u * q^{\prime} g 2\right) \odot g \mathcal{2} \neq 0\right.$ ไ have $l t\left(\left(m u * q^{\prime} g 2\right) \odot g 2\right) \prec_{t} v$ by (rule 1)
finally show ?thesis using $\left\langle v \prec_{t} u\right\rangle$ by simp
qed
next
assume $u \in$ keys $((m u * m u 2) \odot g 2)$
hence $(m u * m u 2) \odot g_{2} \neq 0$ and $u \preceq_{t} l t((m u * m u 2) \odot g 2)$ by (auto intro: lt-max-keys)
from this(1) have $(m u * m u 2) \neq 0$ by auto
note $\left\langle u \preceq_{t}\right.$-〉
also from $\langle m u * m u 2 \neq 0\rangle\langle g 2 \neq 0\rangle$ have $l t((m u * m u 2) \odot g 2)=$ punit.lt $(q g 2) \oplus l t g 2$
by (simp add: lt-mult-scalar) (simp add: mu-times-mu2 punit.lt-monomial monomial-0-iff)
finally show ?thesis using $\left\langle v \prec_{t} u\right\rangle$ by (simp add: v2 $\left.{ }^{\prime}\right)$
qed
qed
thus lookup $\left(q^{\prime \prime} g 2 \odot g 2\right) u=0$ by (simp add: in-keys-iff $)$ qed
have 4: lt $\left(q^{\prime \prime} g \odot g\right) \preceq_{t} v$ if $g \in M$ for $g$

```
proof (cases \(g \in\{g 1, g 2\}\) )
    case True
    hence \(g=g 1 \vee g=g 2\) by simp
    thus ?thesis
    proof
        assume \(g=g 1\)
        show ?thesis
        proof (cases \(\left.q^{\prime \prime} g 1 \odot g 1=0\right)\)
            case True
            thus ?thesis by (simp add: \(\langle g=g 1\rangle\) min-term-min)
        next
            case False
            hence \(l t\left(q^{\prime \prime} g \odot g\right) \prec_{t} v\) unfolding \(\langle g=g 1\rangle\) by (rule 2)
            thus ?thesis by simp
        qed
    next
            assume \(g=g 2\)
            with 3 show?thesis by simp
        qed
next
    case False
    hence \(q^{\prime \prime}: q^{\prime \prime} g=q g+m u * q^{\prime} g\) by (simp add: \(q^{\prime \prime}\)-def)
    show ?thesis
    proof (rule lt-le)
        fix \(u\)
        assume \(v \prec_{t} u\)
        have \(u \notin\) keys \(\left(q^{\prime \prime} g \odot g\right)\)
        proof
            assume \(u \in\) keys \(\left(q^{\prime \prime} g \odot g\right)\)
            also have \(\ldots \subseteq\) keys \((q g \odot g) \cup\) keys \(\left(\left(m u * q^{\prime} g\right) \odot g\right)\)
            unfolding \(q^{\prime \prime}\) mult-scalar-distrib-right by (fact Poly-Mapping.keys-add)
            finally show False
            proof
            assume \(u \in\) keys \((q g \odot g)\)
            hence \(u \preceq_{t} l t(q g \odot g)\) by (rule lt-max-keys)
            with \(\langle g \in M\rangle\left\langle v \prec_{t} u\right\rangle\) show ?thesis by (simp add: M-def)
            next
            assume \(u \in\) keys \(\left(\left(m u * q^{\prime} g\right) \odot g\right)\)
            hence \(\left(m u * q^{\prime} g\right) \odot g \neq 0\) and \(u \preceq_{t} l t\left(\left(m u * q^{\prime} g\right) \odot g\right)\) by (auto intro:
lt-max-keys)
                    note this(2)
                    also from \(\left\langle\left(m u * q^{\prime} g\right) \odot g \neq 0\right\rangle\) have \(l t\left(\left(m u * q^{\prime} g\right) \odot g\right) \prec_{t} v\) by \((r u l e\)
1)
                    finally show ?thesis using \(\left\langle v \prec_{t} u\right\rangle\) by simp
                    qed
        qed
        thus lookup \(\left(q^{\prime \prime} g \odot g\right) u=0\) by (simp add: in-keys-iff)
        qed
qed
```

have 5:lt $\left(q^{\prime \prime} g \odot g\right) \prec_{t} v$ if $g \in G$ and $g \notin M$ and $q^{\prime \prime} g \odot g \neq 0$ for $g$ using that(3)
proof (rule lt-less)
fix $u$
assume $v \preceq_{t} u$
from that(2) $\langle g 1 \in M\rangle\langle g 2 \in M\rangle$ have $g \neq g 1$ and $g \neq g 2$ by blast+
hence $q^{\prime \prime}: q^{\prime \prime} g=q g+m u * q^{\prime} g$ by (simp add: $q^{\prime \prime}$-def)
have $u \notin$ keys $\left(q^{\prime \prime} g \odot g\right)$
proof
assume $u \in$ keys $\left(q^{\prime \prime} g \odot g\right)$
also have $\ldots \subseteq$ keys $(q g \odot g) \cup$ keys $\left(\left(m u * q^{\prime} g\right) \odot g\right)$
unfolding $q^{\prime \prime}$ mult-scalar-distrib-right by (fact Poly-Mapping.keys-add)
finally show False
proof
assume $u \in$ keys $(q g \odot g)$
hence $q g \odot g \neq 0$ and $u \preceq_{t} l t(q g \odot g)$ by (auto intro: lt-max-keys)
note this(2)
also from that (1, 2) $\langle q g \odot g \neq 0\rangle$ have $\ldots \prec_{t} v$ by (rule $v$-max)
finally show ?thesis using $\left\langle v \preceq_{t} u\right\rangle$ by simp
next
assume $u \in$ keys $\left(\left(m u * q^{\prime} g\right) \odot g\right)$
hence $\left(m u * q^{\prime} g\right) \odot g \neq 0$ and $u \preceq_{t} l t\left(\left(m u * q^{\prime} g\right) \odot g\right)$ by (auto intro:
lt-max-keys)
note this(2)
also from $\left\langle\left(m u * q^{\prime} g\right) \odot g \neq 0\right\rangle$ have $l t\left(\left(m u * q^{\prime} g\right) \odot g\right) \prec_{t} v$ by $(r u l e$
1)
finally show ?thesis using $\left\langle v \preceq_{t} u\right\rangle$ by simp qed
qed
thus lookup $\left(q^{\prime \prime} g \odot g\right) u=0$ by (simp add: in-keys-iff)
qed
define $u$ where $u=m l t q^{\prime \prime}$
have $u$-in: $u \in l t$ ' $\left\{q^{\prime \prime} g \odot g \mid g . g \in G \wedge q^{\prime \prime} g \odot g \neq 0\right\}$ unfolding u-def mlt-def
proof (rule ord-term-lin.Max-in, simp-all add: assms(3), rule ccontr)
assume $\nexists g . g \in G \wedge q^{\prime \prime} g \odot g \neq 0$
hence $q^{\prime \prime} g \odot g=0$ if $g \in G$ for $g$ using that by simp
hence $r=0$ by (simp add: $r$ )
with $\langle r \neq 0\rangle$ show False ..
qed
have $u$-max: lt $\left(q^{\prime \prime} g \odot g\right) \preceq_{t} u$ if $g \in G$ for $g$
proof (cases $q^{\prime \prime} g \odot g=0$ )
case True
thus ?thesis by (simp add: min-term-min)
next
case False
show ?thesis unfolding $u$-def mlt-def
by (rule ord-term-lin.Max-ge) (auto simp: assms(3) False intro!: imageI $\langle g \in$ $G 〉)$

## qed

have $q^{\prime \prime} \in$ rel-dom
proof (simp add: rel-dom-def r, intro subsetI, elim imageE)
fix $g$
assume $g \in G$
from assms(1) l-adds have $d(l p(q g 1 \odot g 1)-? l) \leq d(l p(q g 1 \odot g 1))$
by (rule dickson-grading-minus)
also have $\ldots=d$ (punit.lt ( $q$ g1) $+l p g 1$ ) by (simp add: v1' term-simps flip: v1)
also from $\operatorname{assms}(1)$ have $\ldots=$ ord-class.max $(d($ punit.lt $(q g 1)))(d(l p g 1))$ by (rule dickson-gradingD1)
also have $\ldots \leq m^{\prime}$
proof (rule max.boundedI)
from $\langle g 1 \in G\rangle$ have $q g 1 \in$ punit.dgrad-p-set $d m^{\prime}$ by (rule $q$-in)
moreover from $\langle q g 1 \neq 0\rangle$ have punit.lt $(q g 1) \in$ keys $(q g 1)$ by (rule punit.lt-in-keys)
ultimately show $d$ (punit.lt ( $q$ g1 $)$ ) $\leq m^{\prime}$ by (rule punit.dgrad-p-setD[simplified $]$ )
next
from $\langle g 1 \in G\rangle G$-sub have $g 1 \in d g r a d-p$-set $d m^{\prime} .$.
moreover from $\langle g 1 \neq 0\rangle$ have lt g1 $\in$ keys g1 by (rule lt-in-keys)
ultimately show $d(l p g 1) \leq m^{\prime}$ by (rule dgrad-p-setD)
qed
finally have $d 1: d(l p(q g 1 \odot g 1)-? l) \leq m^{\prime}$.
have $d(? l-l p g 2) \leq$ ord-class.max $(d(l p g 2))(d(l p g 1))$
unfolding lcs-comm[of lp g1] using assms(1) by (rule dickson-grading-lcs-minus)
also have $\ldots \leq m^{\prime}$
proof (rule max.boundedI)
from $\langle g 2 \in G\rangle G$-sub have $g 2 \in \operatorname{dgrad}$ - $p$-set $d m^{\prime} .$.
moreover from $\langle g 2 \neq 0\rangle$ have lt g2 $k$ keys $g 2$ by (rule lt-in-keys)
ultimately show $d(l p g 2) \leq m^{\prime}$ by (rule dgrad- $p$-setD)
next
from $\langle g 1 \in G\rangle G$-sub have $g 1 \in$ dgrad-p-set $d m^{\prime} .$.
moreover from $\langle g 1 \neq 0\rangle$ have $l t ~ g 1 \in$ keys $g 1$ by (rule lt-in-keys)
ultimately show $d$ (lp g1) $\leq m^{\prime}$ by (rule dgrad-p-setD)
qed
finally have mu2: mu2 $\in$ punit.dgrad-p-set $d m^{\prime}$
by (simp add: mu2-def punit.dgrad-p-set-def dgrad-set-def)
fix $z$
assume $z: z=q^{\prime \prime} g$
have $g=g 1 \vee g=g 2 \vee(g \neq g 1 \wedge g \neq g 2)$ by blast
thus $z \in$ punit.dgrad- $p$-set $d m^{\prime}$
proof (elim disjE conjE)
assume $g=g 1$
with $\langle g 1 \neq g 2\rangle$ have $q^{\prime \prime} g=$ punit.tail $(q g 1)+m u * q^{\prime} g 1$ by (simp add: $q^{\prime \prime}$-def)
also have $\ldots \in$ punit.dgrad-p-set $d m^{\prime}$ unfolding $m u$-def times-monomial-left by (intro punit.dgrad-p-set-closed-plus punit.dgrad-p-set-closed-tail

```
            punit.dgrad-p-set-closed-monom-mult d1 assms(1) q-in q'-in <g1 \in
```

$G 〉)$
finally show ?thesis by (simp only: z)
next
assume $g=g 2$
hence $q^{\prime \prime} g=q g^{2}+m u * q^{\prime} g 2+m u * m u 2$ by (simp add: $q^{\prime \prime}$-def)
also have ... $\in$ punit.dgrad-p-set $d m^{\prime}$ unfolding mu-def times-monomial-left
by (intro punit.dgrad-p-set-closed-plus punit.dgrad-p-set-closed-monom-mult
d1 mu2 $q$-in $q^{\prime}$-in $\operatorname{assms}(1)\langle g 2 \in G 〉)$
finally show? ?thesis by (simp only: $z$ )
next
assume $g \neq g 1$ and $g \neq g^{2}$
hence $q^{\prime \prime} g=q g+m u * q^{\prime} g$ by (simp add: $q^{\prime \prime}$-def)
also have $\ldots \in$ punit.dgrad-p-set $d m^{\prime}$ unfolding mu-def times-monomial-left
by (intro punit.dgrad-p-set-closed-plus punit.dgrad-p-set-closed-monom-mult
$d 1 \operatorname{assms}(1) q$-in $q^{\prime}$-in $\left.\prec g \in G 〉\right)$
finally show ?thesis by (simp only: z)
qed
qed
with $q$-min have $\neg$ rel $q^{\prime \prime} q$ by blast
hence $v \preceq_{t} u$ and $u \neq v \vee$ mnum $q \leq$ mnum $q^{\prime \prime}$ by (auto simp: v-def $u$-def
rel-def)
moreover have $u \preceq_{t} v$
proof -
from $u$-in obtain $g$ where $g \in G$ and $q^{\prime \prime} g \odot g \neq 0$ and $u: u=l t\left(q^{\prime \prime} g \odot\right.$
g) by blast
show ?thesis
proof (cases $g \in M$ )
case True
thus ?thesis unfolding $u$ by (rule 4)
next
case False
with $\langle g \in G\rangle$ have $l t\left(q^{\prime \prime} g \odot g\right) \prec_{t} v$ using $\left\langle q^{\prime \prime} g \odot g \neq 0\right\rangle$ by (rule 5)
thus ?thesis by (simp add: u)
qed
qed
ultimately have $u-v: u=v$ and mnum $q \leq m n u m q^{\prime \prime}$ by simp-all
note this(2)
also have mnum $q^{\prime \prime}<$ card $M$ unfolding mnum-def
proof (rule psubset-card-mono)
from $\langle M \subseteq G\rangle\langle$ finite $G\rangle$ show finite $M$ by (rule finite-subset)
next
have $\left\{g \in G . q^{\prime \prime} g \odot g \neq 0 \wedge l t\left(q^{\prime \prime} g \odot g\right)=v\right\} \subseteq M-\{g 1\}$
proof
fix $g$
assume $g \in\left\{g \in G . q^{\prime \prime} g \odot g \neq 0 \wedge l t\left(q^{\prime \prime} g \odot g\right)=v\right\}$
hence $g \in G$ and $q^{\prime \prime} g \odot g \neq 0$ and $l t\left(q^{\prime \prime} g \odot g\right)=v$ by simp-all
with 25 show $g \in M-\{g 1\}$ by blast
qed

```
    also from <g1 \inM> have }\ldots\subsetM\mathrm{ by blast
    finally show }{g\inG.\mp@subsup{q}{}{\prime\prime}g\odotg\not=0\wedgelt(\mp@subsup{q}{}{\prime\prime}g\odotg)=mlt \mp@subsup{q}{}{\prime\prime}}\subset
        by (simp only: u-v flip: u-def)
    qed
    also have ... = mnum q by (simp only: M-def mnum-def v-def)
    finally show False ..
qed
```


### 5.6 Replacing Elements in Gröbner Bases

```
lemma replace-in-dgrad-p-set:
    assumes \(G \subseteq d g r a d-p\)-set \(d m\)
    obtains \(n\) where \(q \in d g r a d-p\)-set \(d n\) and \(G \subseteq\) dgrad-p-set \(d n\)
        and insert \(q(G-\{p\}) \subseteq\) dgrad- \(p\)-set \(d n\)
proof -
    from assms obtain \(n\) where \(m \leq n\) and 1: \(q \in d g r a d-p\)-set \(d n\) and 2: \(G \subseteq\)
dgrad-p-set dn
            by (rule dgrad-p-set-insert)
    from this(2, 3) have insert \(q(G-\{p\}) \subseteq d g r a d-p-\) set \(d n\) by auto
    with 12 show ?thesis ..
qed
lemma GB-replace-lt-adds-stable-GB-dgrad-p-set:
    assumes dickson-grading \(d\) and \(G \subseteq d g r a d-p\)-set \(d m\)
    assumes isGB: is-Groebner-basis \(G\) and \(q \neq 0\) and \(q: q \in(p m d l G)\) and \(l t q\)
addst lt \(p\)
    shows is-Groebner-basis (insert \(q(G-\{p\})\) )(is is-Groebner-basis ? \(G^{\prime}\) )
proof -
    from \(\operatorname{assms}(2)\) obtain \(n\) where 1: \(G \subseteq d g r a d-p-s e t d n\) and 2: ? \(G^{\prime} \subseteq d g r a d-p\)-set
\(d n\)
    by (rule replace-in-dgrad-p-set)
    from isGB show ?thesis unfolding GB-alt-3-dgrad-p-set[OF assms(1) 1] GB-alt-3-dgrad-p-set[OF
\(\operatorname{assms}(1)\) 2]
    proof (intro ballI impI)
        fix \(f\)
        assume \(f 1: f \in\left(p m d l ? G^{\prime}\right)\) and \(f \neq 0\)
            and \(a 1: \forall f \in p m d l G . f \neq 0 \longrightarrow\left(\exists g \in G . g \neq 0 \wedge l t g a d d s_{t} l t f\right)\)
    from f1 pmdl.replace-span[OF q, of p] have \(f \in p m d l ~ G\)..
    from a1 [rule-format, OF this \(\langle f \neq 0\rangle\) ] obtain \(g\) where \(g \in G\) and \(g \neq 0\) and
lt \(g\) adds \(s_{t} l t f\) by auto
    show \(\exists g \in ? G^{\prime} . g \neq 0 \wedge l t g a d d s_{t} l t f\)
    proof (cases \(g=p\) )
            case True
            show ?thesis
            proof
            from «lt \(q\) adds \({ }_{t}\) lt \(\left.p\right\rangle\) have \(l t q a d d s_{t}\) lt \(g\) unfolding True.
            also have ... adds \({ }_{t}\) lt \(f\) by fact
            finally have \(l t q\) adds \(l t f\).
            with \(\langle q \neq 0\rangle\) show \(q \neq 0 \wedge l t q a d d s_{t} l t f\)..
```

```
            next
                show q\in?G' by simp
            qed
    next
            case False
            show ?thesis
            proof
                show g\not=0\wedgelt g addst lt f by (rule, fact+)
            next
                from }\langleg\inG\rangle\mathrm{ False show }g\in?G\mp@subsup{G}{}{\prime}\mathrm{ by blast
            qed
        qed
    qed
qed
lemma GB-replace-lt-adds-stable-pmdl-dgrad-p-set:
    assumes dickson-grading d and G\subseteq dgrad-p-set d m
    assumes isGB: is-Groebner-basis G and q\not=0 and q\inpmdl G and lt qaddst
lt p
    shows pmdl (insert q (G-{p})) = pmdl G (is pmdl ?G' = pmdl G)
proof (rule, rule pmdl.replace-span, fact, rule)
    fix f
    assume f}\inpmdl 
    note assms(1)
    moreover from assms(2) obtain n where ? G'G dgrad-p-set d n by (rule
replace-in-dgrad-p-set)
    moreover have is-Groebner-basis ?G' by (rule GB-replace-lt-adds-stable-GB-dgrad-p-set,
fact+)
    ultimately have }\exists\mathrm{ ! h. (red ? G')** fh^ᄀis-red ?G' h by (rule GB-implies-unique-nf-dgrad-p-set)
    then obtain h}\mathrm{ where ftoh: (red ?G}\mp@subsup{G}{}{\prime}\mp@subsup{)}{}{**}fh\mathrm{ and irredh: ᄀis-red? ?G'h by auto
    have }\neg\mathrm{ is-red Gh
    proof
        assume is-red Gh
        have is-red ?G' h by (rule replace-lt-adds-stable-is-red, fact+)
        with irredh show False ..
    qed
    have f-h\inpmdl ?G' by (rule red-rtranclp-diff-in-pmdl, rule ftoh)
    have f-h\inpmdl G by (rule, fact, rule pmdl.replace-span, fact)
    from pmdl.span-diff[OF this <f \in pmdl G〉] have -h\in pmdl G by simp
    from pmdl.span-neg[OF this] have h\inpmdl G by simp
    with isGB<\neg is-red G h` have h=0 using GB-imp-reducibility by auto
    with ftoh have (red? ?G')** f 0 by simp
    thus f\inpmdl? ?G' by (simp add: red-rtranclp-0-in-pmdl)
qed
lemma GB-replace-red-stable-GB-dgrad-p-set:
    assumes dickson-grading d and G\subseteqdgrad-p-set d m
    assumes isGB: is-Groebner-basis G and p\inG and q: red (G-{p}) pq
    shows is-Groebner-basis (insert q(G-{p}))(is is-Groebner-basis ?G')
```

```
proof -
    from assms(2) obtain n where 1:G\subseteqdgrad-p-set d n and 2:?G'\subseteqdgrad-p-set
d n
        by (rule replace-in-dgrad-p-set)
    from isGB show ?thesis unfolding GB-alt-2-dgrad-p-set[OF assms(1) 1] GB-alt-2-dgrad-p-set[OF
assms(1) 2]
    proof (intro ballI impI)
        fix f
        assume f1:f\in(pmdl ?G') and f}\not=
            and a1:\forallf\inpmdl G.f\not=0\longrightarrow is-red Gf
        have q}\inpmdl 
        proof (rule pmdl-closed-red, rule pmdl.span-mono)
            from pmdl.span-superset }\langlep\inG\rangle\mathrm{ show }p\inpmdl G ..
        next
            show G-{p}\subseteqG by (rule Diff-subset)
        qed (rule q)
        from f1 pmdl.replace-span[OF this, of p] have f\inpmdl G ..
        have is-red Gf by (rule a1 [rule-format], fact+)
        show is-red? G' f by (rule replace-red-stable-is-red, fact+)
    qed
qed
lemma GB-replace-red-stable-pmdl-dgrad-p-set:
    assumes dickson-grading d and G\subseteqdgrad-p-set d m
    assumes isGB: is-Groebner-basis G and p\inG and ptoq: red (G-{p})pq
    shows pmdl (insert q(G-{p})) = pmdl G (is pmdl ?G' = -)
proof -
    from }\langlep\inG\ranglepmdl.span-superset have p f pmdl G ..
    have q}\operatorname{pmdl}
    by (rule pmdl-closed-red, rule pmdl.span-mono, rule Diff-subset, rule }\langlep\inpmd
G>, rule ptoq)
    show ?thesis
    proof (rule, rule pmdl.replace-span, fact, rule)
        fix f
        assume f}\inpmdl 
        note assms(1)
        moreover from assms(2) obtain n where ? G' }\subseteq\mathrm{ dgrad-p-set d n by (rule
replace-in-dgrad-p-set)
    moreover have is-Groebner-basis ?G' by (rule GB-replace-red-stable-GB-dgrad-p-set,
fact+)
    ultimately have }\exists\mathrm{ ! h. (red ?G}\mp@subsup{G}{}{\prime}\mp@subsup{)}{}{**}fh\wedge\negis-red ? G' h by (rule GB-implies-unique-nf-dgrad-p-set)
    then obtain h}\mathrm{ where ftoh:(red ?G')** f h and irredh: ᄀ is-red ? G' h by auto
    have ᄀ is-red G h
    proof
            assume is-red Gh
            have is-red? ?G'h by (rule replace-red-stable-is-red, fact+)
            with irredh show False ..
    qed
    have f-h\inpmdl? ?G' by (rule red-rtranclp-diff-in-pmdl, rule ftoh)
```

```
    have f-h\in pmdl G by (rule, fact, rule pmdl.replace-span, fact)
    from pmdl.span-diff[OF this <f \in pmdl G`] have -h\inpmdl G by simp
    from pmdl.span-neg[OF this] have h\inpmdl G by simp
    with isGB<\neg is-red G h> have h=0 using GB-imp-reducibility by auto
    with ftoh have (red? 'G ')** f 0 by simp
    thus f}\in\mathrm{ pmdl ?G' by (simp add: red-rtranclp-0-in-pmdl)
    qed
qed
lemma GB-replace-red-rtranclp-stable-GB-dgrad-p-set:
    assumes dickson-grading d and G\subseteqdgrad-p-set d m
    assumes isGB: is-Groebner-basis G and p\inG and ptoq: (red (G-{p}))** p
q
    shows is-Groebner-basis (insert q(G-{p}))
    using ptoq
proof (induct q rule: rtranclp-induct)
    case base
    from isGB<p}\inG\rangle\mathrm{ show ?case by (simp add: insert-absorb)
next
    case (step y z)
    show ?case
    proof (cases y = p)
        case True
        from assms(1) assms(2) isGB <p\inG> show ?thesis
        proof (rule GB-replace-red-stable-GB-dgrad-p-set)
            from <red (G-{p}) y z` show red (G-{p}) pz unfolding True.
        qed
    next
        case False
        show ?thesis
            proof (cases y \inG)
                case True
                with }\langley\not=p\rangle\mathrm{ have }y\inG-{p} (is - \in? ?G') by blas
                hence insert y (G-{p})=? G' by auto
                with step (3) have is-Groebner-basis ?G' by simp
                from }\langley\in?\mp@subsup{?}{}{\prime}\ranglepmdl.span-superset have y pmdl ?G' ..
                have z f pmdl? 'G' by (rule pmdl-closed-red, rule subset-refl, fact+)
                show is-Groebner-basis (insert z ?G') by (rule GB-insert, fact+)
            next
                case False
                from assms(2) obtain n where insert y (G-{p})\subseteqdgrad-p-set d n
                    by (rule replace-in-dgrad-p-set)
            from assms(1) this step(3) have is-Groebner-basis (insert z (insert y (G -
{p}) - {y}))
                proof (rule GB-replace-red-stable-GB-dgrad-p-set)
                    from <red (G-{p}) y z` False show red ((insert y (G-{p})) - {y})
yz by simp
        qed simp
        moreover from False have ... = (insert z (G-{p})) by simp
```

```
            ultimately show ?thesis by simp
            qed
    qed
qed
lemma GB-replace-red-rtranclp-stable-pmdl-dgrad-p-set:
    assumes dickson-grading d and G\subseteqdgrad-p-set d m
    assumes isGB: is-Groebner-basis G and p\inG and ptoq: (red (G-{p}))** p
q
    shows pmdl (insert q (G-{p})) = pmdl G
    using ptoq
proof (induct q rule: rtranclp-induct)
    case base
    from }\langlep\inG\rangle\mathrm{ show ?case by (simp add: insert-absorb)
next
    case (step y z)
    show ?case
    proof (cases y=p)
        case True
        from assms(1) assms(2) isGB <p\inG〉 step(2) show ?thesis unfolding True
        by (rule GB-replace-red-stable-pmdl-dgrad-p-set)
    next
    case False
    have gb: is-Groebner-basis (insert y (G-{p}))
        by (rule GB-replace-red-rtranclp-stable-GB-dgrad-p-set, fact+)
    show ?thesis
    proof (cases y \inG)
        case True
        with }\langley\not=p\rangle\mathrm{ have }y\inG-{p} (is - \in?G') by blas
        hence eq: insert y?G' =? 'G' by auto
        from }\langley\in?,\mp@subsup{G}{}{\prime}> have y\inpmdl ?G' by (rule pmdl.span-base)
        have z\inpmdl ?G' by (rule pmdl-closed-red, rule subset-refl, fact+)
        hence pmdl (insert z ?G') = pmdl ?G' by (rule pmdl.span-insert-idI)
        also from step(3) have ... = pmdl G by (simp only: eq)
        finally show ?thesis .
    next
        case False
        from assms(2) obtain n where 1: insert y (G-{p})\subseteqdgrad-p-set d n
                by (rule replace-in-dgrad-p-set)
    from False have pmdl (insert z (G-{p}))=pmdl (insert z (insert y (G-
{p}) - {y}))
            by auto
        also from assms(1) 1 gb have ... = pmdl (insert y (G-{p}))
        proof (rule GB-replace-red-stable-pmdl-dgrad-p-set)
            from step(2) False show red ((insert y (G-{p})) - {y}) y z by simp
        qed simp
        also have ... = pmdl G by fact
        finally show ?thesis.
    qed
```

qed
qed
lemmas GB-replace-lt-adds-stable-GB-finite $=$
GB-replace-lt-adds-stable-GB-dgrad-p-set[OF dickson-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemmas GB-replace-lt-adds-stable-pmdl-finite $=$
GB-replace-lt-adds-stable-pmdl-dgrad-p-set[OF dickson-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemmas $G B$-replace-red-stable-GB-finite $=$
GB-replace-red-stable-GB-dgrad-p-set[OF dickson-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemmas GB-replace-red-stable-pmdl-finite $=$
GB-replace-red-stable-pmdl-dgrad-p-set[OF dickson-grading-dgrad-dummy dgrad-p-set-exhaust-expl]
lemmas $G B$-replace-red-rtranclp-stable-GB-finite $=$
GB-replace-red-rtranclp-stable-GB-dgrad-p-set[OF dickson-grading-dgrad-dummy
dgrad-p-set-exhaust-expl]
lemmas GB-replace-red-rtranclp-stable-pmdl-finite $=$
GB-replace-red-rtranclp-stable-pmdl-dgrad-p-set[OF dickson-grading-dgrad-dummy
dgrad-p-set-exhaust-expl]

### 5.7 An Inconstructive Proof of the Existence of Finite Gröbner Bases

lemma ex-finite-GB-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term' Keys $F$ ) and $F \subseteq$ dgrad-p-set d m
obtains $G$ where $G \subseteq$ dgrad-p-set $d m$ and finite $G$ and is-Groebner-basis $G$
and $p m d l ~ G=p m d l ~ F$
proof -
define $S$ where $S=\{l t f \mid f . f \in \operatorname{pmdl} F \wedge f \in \operatorname{dgrad-p-set} d m \wedge f \neq 0\}$
note assms (1)
moreover from - assms(2) have finite (component-of-term'S)
proof (rule finite-subset)
have component-of-term' $S \subseteq$ component-of-term' Keys (pmdl F)
by (rule image-mono, rule, auto simp add: S-def intro!: in-KeysI lt-in-keys)
thus component-of-term' $S \subseteq$ component-of-term' Keys $F$ by (simp only:
components-pmdl)
qed
moreover have $p p$-of-term ' $S \subseteq d g r a d$-set $d m$
proof
fix $s$
assume $s \in p p$-of-term ' $S$
then obtain $u$ where $u \in S$ and $s=p p$-of-term $u$..
from this(1) obtain $f$ where $f \in \operatorname{pmdl} F \wedge f \in$ dgrad-p-set $d m \wedge f \neq 0$ and $u: u=l t f$
unfolding $S$-def by blast
from this (1) have $f \in d$ grad- $p$-set $d m$ and $f \neq 0$ by simp-all
have $u \in$ keys $f$ unfolding $u$ by (rule lt-in-keys, fact)
with $\langle f \in d g r a d-p-s e t d m\rangle$ have $d(p p$-of-term $u) \leq m$ unfolding $u$ by (rule dgrad-p-setD)
thus $s \in d g r a d-s e t d m$ by (simp add: $\langle s=p p$-of-term $u\rangle d g r a d-s e t-d e f)$
qed
ultimately obtain $T$ where finite $T$ and $T \subseteq S$ and $*: \bigwedge s . s \in S \Longrightarrow(\exists t \in T$. $t a d d s_{t} s$ )
by (rule ex-finite-adds-term, blast)
define crit where crit $=(\lambda t f . f \in p m d l F \wedge f \in \operatorname{dgrad}$ - $p$-set $d m \wedge f \neq 0 \wedge t$ $=l t f$ )
have ex-crit: $t \in T \Longrightarrow(\exists f$. crit $t f)$ for $t$
proof -
assume $t \in T$
from this $\langle T \subseteq S\rangle$ have $t \in S$..
then obtain $f$ where $f \in p m d l F \wedge f \in d g r a d$ - $p$-set $d m \wedge f \neq 0$ and $t=l t f$
unfolding $S$-def by blast
thus $\exists f$. crit $t f$ unfolding crit-def by blast
qed
define $G$ where $G=(\lambda t$. SOME g. crit $t g)$ ' $T$
have $G: g \in G \Longrightarrow g \in \operatorname{pmdl} F \wedge g \in$ dgrad- $p$-set $d m \wedge g \neq 0$ for $g$
proof -
assume $g \in G$
then obtain $t$ where $t \in T$ and $g: g=(S O M E h$. crit $t h)$ unfolding $G$-def
have crit $t g$ unfolding $g$ by (rule someI-ex, rule ex-crit, fact)
thus $g \in$ pmdl $F \wedge g \in$ dgrad- $p$-set $d m \wedge g \neq 0$ by (simp add: crit-def)
qed
have $* *: t \in T \Longrightarrow(\exists g \in G$. lt $g=t)$ for $t$
proof -
assume $t \in T$
define $g$ where $g=($ SOME h. crit $t h)$
from $\langle t \in T\rangle$ have $g \in G$ unfolding $g$-def $G$-def by blast
thus $\exists g \in G$. lt $g=t$
proof
have crit $t g$ unfolding $g$-def by (rule someI-ex, rule ex-crit, fact)
thus $l t g=t$ by (simp add: crit-def)
qed
qed
have adds: $f \in$ pmdl $F \Longrightarrow f \in$ dgrad-p-set $d m \Longrightarrow f \neq 0 \Longrightarrow(\exists g \in G . g \neq 0$
$\left.\wedge l t g a d d s_{t} l t f\right)$ for $f$
proof -
assume $f \in p m d l F$ and $f \in d g r a d-p$-set $d m$ and $f \neq 0$
hence lt $f \in S$ unfolding $S$-def by blast
hence $\exists t \in T$. $t$ adds $s_{t}(l t f)$ by (rule *)
then obtain $t$ where $t \in T$ and $t a d d s_{t}(l t f)$..
from this(1) have $\exists g \in G$. lt $g=t$ by (rule $* *$ )
then obtain $g$ where $g \in G$ and $l t g=t$..
show $\exists g \in G . g \neq 0 \wedge l t g$ adds $s_{t} l t f$
proof (intro bexI conjI)
from $G[O F\langle g \in G\rangle]$ show $g \neq 0$ by (elim conjE)
next
from $\left\langle t\right.$ adds $\left.s_{t} l t f\right\rangle$ show $l t ~ g a d d s_{t} l t f$ by (simp only: $\left.\langle l t ~ g=t\rangle\right)$
qed fact

```
qed
have sub1: pmdl G\subseteqpmdl F
proof (rule pmdl.span-subset-spanI, rule)
    fix g
    assume g}\in
    from G[OF this] show g\inpmdl F ..
qed
have sub2: G\subseteqdgrad-p-set d m
proof
    fix g
    assume g}\in
    from G[OF this] show g\indgrad-p-set d m}\mathrm{ by (elim conjE)
qed
show ?thesis
proof
    from〈finite T〉 show finite G unfolding G-def ..
next
    from assms(1) sub2 adds show is-Groebner-basis G
    proof (rule isGB-I-adds-lt)
        fix f
        assume f}\inpmdl 
        from this sub1 show f}\in\mathrm{ pmdl F ..
    qed
next
    show pmdl G = pmdl F
    proof
        show pmdl F\subseteqpmdl G
        proof (rule pmdl.span-subset-spanI, rule)
            fix f
            assume f}\in
            hence f}\in\mathrm{ pmdl F by (rule pmdl.span-base)
            from}\langlef\inF\rangle\operatorname{assms(3) have f\indgrad-p-set d m ..
            with assms(1) sub2 sub1 - <f f pmdl F> have (red G)** f 0
            proof (rule is-red-implies-0-red-dgrad-p-set)
                    fix }
                    assume q\inpmdl F and q\indgrad-p-set dm and q}=
                    hence ( }\existsg\inG.g\not=0\wedgelt gaddst lt q) by (rule adds
                    then obtain g}\mathrm{ where g}\inG\mathrm{ and }g\not=0\mathrm{ and lt g addst lt q by blast
                    thus is-red Gqusing <q\not=0\rangle is-red-indI1 by blast
                qed
                thus f}\in\mathrm{ pmdl G by (rule red-rtranclp-0-in-pmdl)
            qed
    qed fact
next
    show G\subseteqdgrad-p-set d m
    proof
            fix g
        assume g}\in
        hence g}\in\mathrm{ pmdl F}\wedgeg\indgrad-p-set d m\wedgeg\not=0 by (rule G
```

```
        thus g}\indgrad-p-set d m by (elim conjE
    qed
    qed
qed
```

The preceding lemma justifies the following definition.
definition some- $G B::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ set $\Rightarrow\left(' t \nRightarrow_{0}{ }^{\prime} b::\right.$ field $)$ set
where some-GB $F=(S O M E G$. finite $G \wedge i s$-Groebner-basis $G \wedge p m d l G=$ pmdl F)
lemma some-GB-props-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set d m
shows finite (some-GBF) $\wedge$ is-Groebner-basis (some-GB F) $\wedge$ pmdl (some-GB
$F)=p m d l F$
proof -
from assms obtain $G$ where finite $G$ and is-Groebner-basis $G$ and pmdl $G=$ pmdl F by (rule ex-finite-GB-dgrad-p-set)
hence finite $G \wedge i s$-Groebner-basis $G \wedge p m d l G=p m d l F$ by simp
thus finite (some-GBF) $\wedge$ is-Groebner-basis (some-GB F) $\wedge$ pmdl (some-GB
$F)=p m d l F$
unfolding some-GB-def by (rule someI)
qed
lemma finite-some-GB-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set d $m$
shows finite (some-GB F)
using some-GB-props-dgrad-p-set[OF assms] ..
lemma some-GB-isGB-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set d m
shows is-Groebner-basis (some-GB F)
using some-GB-props-dgrad-p-set[OF assms] by (elim conjE)
lemma some-GB-pmdl-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set d m
shows pmdl (some-GB F) $=$ pmdl $F$
using some-GB-props-dgrad-p-set[OF assms] by (elim conjE)
lemma finite-imp-finite-component-Keys:
assumes finite $F$
shows finite (component-of-term 'Keys F)
by (rule finite-imageI, rule finite-Keys, fact)
lemma finite-some-GB-finite: finite $F \Longrightarrow$ finite (some-GB F)
by (rule finite-some-GB-dgrad-p-set, rule dickson-grading-dgrad-dummy,
erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma some-GB-isGB-finite: finite $F \Longrightarrow$ is-Groebner-basis (some-GB F) by (rule some-GB-isGB-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma some-GB-pmdl-finite: finite $F \Longrightarrow p m d l($ some-GB $F)=p m d l F$ by (rule some-GB-pmdl-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)

Theory Buchberger implements an algorithm for effectively computing Gröbner bases.

### 5.8 Relation red-supset

The following relation is needed for proving the termination of Buchberger's algorithm (i.e. function $g b$-schema-aux).

```
definition red-supset::(' }t>
    where red-supset A B}\equiv(\existsp.is-red A p\wedge\neg is-red B p)\wedge(\forallp. is-red B p
is-red A p)
lemma red-supsetE:
    assumes }A\sqsupsetp
    obtains p where is-red A p and }\negis-red B 
proof -
    from assms have \existsp. is-red A p\wedge\neg is-red B p by (simp add: red-supset-def)
    from this obtain p where is-red A p and }\neg\mathrm{ is-red B p by auto
    thus ?thesis ..
qed
lemma red-supsetD:
    assumes a1:A\sqsupsetpB and a2: is-red B p
    shows is-red A p
proof -
    from assms have }\forallp\mathrm{ . is-red B p }\longrightarrow\mathrm{ is-red A p by (simp add: red-supset-def)
    hence is-red B p\longrightarrowis-red A p ..
    from a2 this show ?thesis by simp
qed
lemma red-supsetI [intro]:
    assumes }\bigwedgeq. is-red Bq\Longrightarrowis-red A q and is-red A p and \negis-red B 
    shows }A\sqsupsetp
    unfolding red-supset-def using assms by auto
lemma red-supset-insertI:
    assumes }x\not=0\mathrm{ and }\negis-red A x
    shows (insert x A) \sqsupsetpA
```

```
proof
    fix q
    assume is-red A q
    thus is-red (insert x A) q unfolding is-red-alt
    proof
        fix }
        assume red A q a
        from red-unionI2[OF this, of {x}] have red (insert x A) q a by simp
        show \existsqa. red (insert x A) q qa
        proof
            show red (insert x A) q a by fact
        qed
    qed
next
    show is-red (insert x A) x unfolding is-red-alt
    proof
        from red-unionI1[OF red-self[OF<x\not=0\rangle], of A] show red (insert x A) x 0
by simp
    qed
next
    show ᄀ is-red A x by fact
qed
lemma red-supset-transitive:
    assumes }A\sqsupsetpB\mathrm{ and B}\sqsupsetp
    shows }A\sqsupsetp
proof -
    from assms(2) obtain p where is-red B p and \negis-red C p by (rule red-supsetE)
    show ?thesis
    proof
        fix q
        assume is-red Cq
        with assms(2) have is-red B q by (rule red-supsetD)
        with assms(1) show is-red A q by (rule red-supsetD)
    next
        from assms(1)<is-red B p〉 show is-red A p by (rule red-supsetD)
    qed fact
qed
lemma red-supset-wf-on:
    assumes dickson-grading d and finite K
    shows wfp-on ( }\existsp)(\mathrm{ Pow (dgrad-p-set d m) }\cap{F\mathrm{ . component-of-term`Keys F
\subseteq K \} )
proof (rule wfp-onI-chain, rule, erule exE)
    let ?A = dgrad-p-set d m
    fix f::nat }=>((\mp@subsup{'}{}{\prime}t=0 'b) set
    assume \foralli.fi\in Pow?A \cap{F.component-of-term`Keys F\subseteqK} ^f(Suc i)
|fi
    hence a1-subset: fi\subseteq?A and comp-sub: component-of-term 'Keys (fi)\subseteqK
```

```
    and a1: f(Suc i)}\sqsupsetpfi\mathrm{ for i by simp-all
have a1-trans: i<j\Longrightarrowfj\sqsupsetpfi for ij
proof -
    assume i<j
    thus fj\sqsupsetpfi
    proof (induct j)
        case 0
        thus ?case by simp
    next
        case (Suc j)
        from Suc(2) have i=j\veei<j by auto
        thus ?case
        proof
            assume i=j
            show ?thesis unfolding <i=j〉 by (fact a1)
        next
            assume i<j
            from a1 have f(Suc j) }\existspfj\mathrm{ .
            also from <i< j〉 have ... }\exists\textrm{pfi}\mathrm{ by (rule Suc(1))
            finally(red-supset-transitive) show ?thesis.
        qed
    qed
qed
have a2: \existsp\inf(Suc i). \existsq.is-red {p} q\wedge ᄀis-red (fi) q for i
proof -
    from a1 have f(Suc i)}\sqsupsetpfi
    then obtain q where red: is-red (f (Suc i)) q and irred: \negis-red (f i)q
        by (rule red-supsetE)
    from red obtain p where p\inf(Suci) and is-red {p}q by (rule is-red-singletonI)
```



```
    proof
        show \exists q. is-red {p} q\wedge\negis-red (fi)q
        proof (intro exI, intro conjI)
            show is-red {p} q by fact
        qed (fact)
    next
        show }p\inf(Suc i) by fac
    qed
qed
let ?P = \lambdai p. p \in (f (Suc i))})\wedge(\existsq. is-red {p} q\wedge\neg is-red (f i) q)
define g}\mathrm{ where g 三 \i::nat. (SOME p. ?P i p)
have a3: ?P i (gi) for i
proof -
    from a2[of i] obtain gi where gi\inf(Suc i) and \existsq.is-red {gi} q}\wedge\neg \s-re
(f i) q ..
    show ?thesis unfolding g-def by (rule someI[of - gi], intro conjI, fact+)
```

qed

```
have \(a \nmid: i<j \Longrightarrow \neg l t(g i) a d d s_{t}(l t(g j))\) for \(i j\)
proof
    assume \(i<j\) and \(a d d s\) : lt ( \(g i\) ) addst \(l t(g j)\)
    from \(a 3\) have \(\exists q\). is-red \(\{g j\} q \wedge \neg i s\)-red \((f j) q .\).
    then obtain \(q\) where redj: is-red \(\{g j\} q\) and \(\neg i s\)-red \((f j) q\) by auto
    have \(*\) : \(\neg i s\)-red \((f(S u c i)) q\)
    proof -
        from \(\langle i<j\rangle\) have \(i+1<j \vee i+1=j\) by auto
        thus ?thesis
        proof
            assume \(i+1<j\)
            from red-supsetD[OF a1-trans[rule-format, OF this], of \(q]\langle\neg i s\)-red \((f j) q\rangle\)
                show ?thesis by auto
        next
                assume \(i+1=j\)
                thus ?thesis using \(\prec \neg\) is-red \((f j) q\rangle\) by simp
        qed
    qed
    from \(a 3\) have \(g i \in f(i+1)\) and redi: \(\exists q\). is-red \(\{g i\} q \wedge \neg i s\)-red \((f i) q\)
by simp-all
    have \(\neg i s\)-red \(\{g i\} q\)
    proof
        assume is-red \(\{g i\} q\)
        from is-red-singletonD[OF this \(\langle g i \in f(i+1)\rangle] *\) show False by simp
    qed
    have \(g i \neq 0\)
    proof -
        from redi obtain \(q 0\) where \(i s\)-red \(\{g i\} q 0\) by auto
        from is-red-singleton-not- 0 [OF this] show ?thesis.
    qed
```

    from \(\left\langle\neg i s\right.\)-red \(\left\{\begin{array}{lll}g & i\} & q\rangle \\ \text { is-red-singleton-trans }[O F & \text { redj } a d d s & \\ g & i \neq 0\rangle] \text { show }, ~\end{array}\right.\)
    False by simp
qed
from - assms(2) have a5: finite (component-of-term'range (lt $\circ g)$ )
proof (rule finite-subset)
show component-of-term ' range ( $l t \circ g$ ) $\subseteq K$
proof (rule, elim imageE, simp)
fix $i$
from $a 3$ have $g i \in f(S u c i)$ and $\exists q$. is-red $\{g i\} q \wedge \neg i s$-red $(f i) q$ by
simp-all
from this(2) obtain $q$ where is-red $\{g i\} q$ by auto
hence $g i \neq 0$ by (rule is-red-singleton-not- 0 )
hence $l t(g i) \in$ keys ( $g i$ ) by (rule lt-in-keys)
hence component-of-term (lt ( $g i$ ) ) component-of-term'keys ( $g i$ ) by simp
also have $\ldots \subseteq$ component-of-term' Keys ( $f$ (Suc i))
by (rule image-mono, rule keys-subset-Keys, fact)

```
        also have ... \subseteqK by (fact comp-sub)
    finally show component-of-term (lt (gi)) \inK.
    qed
    qed
    have a6: pp-of-term'range (lt \circ g)\subseteqdgrad-set d m
    proof (rule, elim imageE, simp)
    fix i
    from a3 have gi\inf(Suc i) and \existsq.is-red {gi} q^\negis-red (fi)q by
simp-all
    from this(2) obtain q}\mathrm{ where is-red {g i} q by auto
    hence gi\not=0 by (rule is-red-singleton-not-0)
    from a1-subset <g i\inf(Suc i)\rangle have gi\in?A ..
    from this }\langlegi\not=0\rangle\mathrm{ have d (lp (gi)) sm by (rule dgrad-p-setD-lp)
    thus lp (gi)\indgrad-set d m by (rule dgrad-setI)
    qed
    from assms(1) a5 a6 obtain ij where i<j and (lt \circg) i addst (lt \circg) j by
(rule Dickson-termE)
    from this a4[OF<i<j`] show False by simp
qed
end
lemma in-lex-prod-alt:
    (x,y)\inr<*lex*> s\longleftrightarrow(((fst x),(fst y)) \inr\vee (fst x = fst y ^ ((snd x), (snd
y)) \ins))
    by (metis in-lex-prod prod.collapse prod.inject surj-pair)
```


### 5.9 Context od-term

context od-term
begin
lemmas red-wf $=$ red-wf-dgrad-p-set[OF dickson-grading-zero subset-dgrad-p-set-zero]
lemmas Buchberger-criterion = Buchberger-criterion-dgrad-p-set[OF dickson-grading-zero subset-dgrad-p-set-zero]
end
end

## 6 A General Algorithm Schema for Computing Gröbner Bases

theory Algorithm-Schema<br>imports General Groebner-Bases<br>begin

This theory formalizes a general algorithm schema for computing Gröbner bases, generalizing Buchberger's original critical-pair/completion algorithm. The algorithm schema depends on several functional parameters that can be instantiated by a variety of concrete functions. Possible instances yield Buchberger's algorithm, Faugère's F4 algorithm, and (as far as we can tell) even his F5 algorithm.

## 6.1 processed

definition minus-pairs (infixl $\left.-_{p} 65\right)$ where minus-pairs $A B=A-(B \cup$ prod.swap ' B)
definition Int-pairs (infixl $\left.\cap_{p} 65\right)$ where Int-pairs $A B=A \cap(B \cup$ prod.swap - B)
definition in-pair (infix $\left.\in_{p} 50\right)$ where in-pair $p A \longleftrightarrow(p \in A \cup$ prod.swap' $A)$
definition subset-pairs (infix $\left.\subseteq_{p} 50\right)$ where subset-pairs $A B \longleftrightarrow\left(\forall x . x \in_{p} A\right.$
$\longrightarrow x \in \in_{p} B$ )
abbreviation not-in-pair (infix $\notin p 50)$ where not-in-pair $p A \equiv \neg p \in_{p} A$
lemma in-pair-alt: $p \in_{p} A \longleftrightarrow(p \in A \vee$ prod.swap $p \in A)$
by (metis (mono-tags, lifting) UnCI UnE image-iff in-pair-def prod.collapse swap-simp)
lemma in-pair-iff: $(a, b) \in_{p} A \longleftrightarrow((a, b) \in A \vee(b, a) \in A)$
by (simp add: in-pair-alt)
lemma in-pair-minus-pairs [simp]: $p \in_{p} A-_{p} B \longleftrightarrow\left(p \in_{p} A \wedge p \notin p B\right)$
by (metis Diff-iff in-pair-def in-pair-iff minus-pairs-def prod.collapse)
lemma in-minus-pairs [simp]: $p \in A-{ }_{p} B \longleftrightarrow\left(p \in A \wedge p \nexists_{p} B\right)$
by (metis Diff-iff in-pair-def minus-pairs-def)
lemma in-pair-Int-pairs [simp]: $p \in_{p} A \cap_{p} B \longleftrightarrow\left(p \in_{p} A \wedge p \in_{p} B\right)$
by (metis (no-types, opaque-lifting) Int-iff Int-pairs-def in-pair-alt in-pair-def old.prod.exhaust swap-simp)
lemma in-pair-Un [simp]: $p \in_{p} A \cup B \longleftrightarrow\left(p \in_{p} A \vee p \in_{p} B\right)$
by (metis (mono-tags, lifting) UnE UnI1 UnI2 image-Un in-pair-def)
lemma in-pair-trans [trans]:
assumes $p \in_{p} A$ and $A \subseteq B$
shows $p \in_{p} B$
using assms by (auto simp: in-pair-def)
lemma in-pair-same [simp]: $p \in_{p} A \times A \longleftrightarrow p \in A \times A$
by (auto simp: in-pair-def)
lemma subset-pairsI [intro]:
assumes $\bigwedge x . x \in_{p} A \Longrightarrow x \in_{p} B$

```
    shows A\subseteq}\mp@subsup{\}{p}{}
    unfolding subset-pairs-def using assms by blast
lemma subset-pairsD [trans]:
    assumes }x\mp@subsup{\in}{p}{}A\mathrm{ and }A\mp@subsup{\subseteq}{p}{}
    shows }x\mp@subsup{\in}{p}{}
    using assms unfolding subset-pairs-def by blast
definition processed :: (' }a\times\mp@subsup{}{}{\prime}a)=>\mp@subsup{}{}{\prime}a\mathrm{ list }=>(\mp@subsup{}{}{\prime}a\times'a) list => boo
    where processed p xs ps \longleftrightarrowp\in set xs }\times\mathrm{ set xs }\wedgep\not\mp@subsup{\exists}{p}{}\mathrm{ set ps
lemma processed-alt:
    processed (a,b) xs ps \longleftrightarrow ((a\in set xs)})\wedge(b\in\mathrm{ set xs )}\wedge(a,b)\not\inp\mathrm{ set ps)
    unfolding processed-def by auto
lemma processedI:
    assumes }a\in\mathrm{ set xs and b}\in\mathrm{ set xs and (a,b) &p set ps
    shows processed (a,b) xs ps
    unfolding processed-alt using assms by simp
lemma processedD1:
    assumes processed (a,b) xs ps
    shows a\in set xs
    using assms by (simp add: processed-alt)
lemma processedD2:
    assumes processed (a,b) xs ps
    shows b}\in\mathrm{ set xs
    using assms by (simp add: processed-alt)
lemma processedD3:
    assumes processed (a,b) xs ps
    shows (a,b)\not\inp set ps
    using assms by (simp add: processed-alt)
lemma processed-Nil: processed (a,b) xs []\longleftrightarrow(a\in set xs ^b\in set xs)
    by (simp add: processed-alt in-pair-iff)
lemma processed-Cons:
    assumes processed (a,b) xs ps
        and a1: a=p\Longrightarrowb=q\Longrightarrow thesis
        and a2: a=q\Longrightarrowb=p\Longrightarrow thesis
        and a3: processed (a,b) xs ((p,q) # ps)\Longrightarrow thesis
    shows thesis
proof -
    from assms(1) have a\inset xs and b\in set xs and (a,b)\not\inp set ps
        by (simp-all add: processed-alt)
    show ?thesis
    proof (cases (a,b) = (p,q))
```

```
    case True
    hence }a=p\mathrm{ and }b=q\mathrm{ by simp-all
    thus ?thesis by (rule a1)
    next
    case False
    with}\langle(a,b)\not\inp\mathrm{ set ps> have *: (a,b) & set ((p,q) # ps) by (auto simp:
in-pair-iff)
    show ?thesis
    proof (cases (b,a)=(p,q))
            case True
            hence }a=q\mathrm{ and }b=p\mathrm{ by simp-all
            thus ?thesis by (rule a2)
    next
            case False
            with <(a,b) \not\inp set ps> have (b,a) & set ((p,q) # ps) by (auto simp:
in-pair-iff)
            with * have (a,b)\not\not二p set ((p,q) # ps) by (simp add: in-pair-iff)
            with }\langlea\in\mathrm{ set xs><b set xs> have processed (a,b) xs ((p,q) # ps)
            by (rule processedI)
        thus ?thesis by (rule a3)
    qed
    qed
qed
lemma processed-minus:
    assumes processed (a,b) xs (ps -- qs)
        and a1:(a,b)\in \in set qs \Longrightarrow thesis
        and a2: processed (a,b) xs ps\Longrightarrow thesis
    shows thesis
proof -
    from assms(1) have a\in set xs and b\in set xs and (a,b)\not\inp set (ps--qs)
        by (simp-all add: processed-alt)
    show ?thesis
    proof (cases (a,b) \inp set qs)
        case True
        thus ?thesis by (rule a1)
    next
        case False
        with «(a,b)\not\not=p set (ps--qs)〉 have (a,b)\not\inp set ps
            by (auto simp: set-diff-list in-pair-iff)
        with }\langlea\in\mathrm{ set xs><b fet xs> have processed (a,b) xs ps
            by (rule processedI)
        thus ?thesis by (rule a2)
    qed
qed
```


### 6.2 Algorithm Schema

```
6.2.1 const-lt-component
context ordered-term
begin
definition const-lt-component :: (' }t=\mp@subsup{=}{0}{\prime}'b::zero) = 'k optio
    where const-lt-component p=
        (let v =lt p in if pp-of-term v=0 then Some (component-of-term
v) else None)
lemma const-lt-component-SomeI:
    assumes lp p=0 and component-of-term (lt p) =cmp
    shows const-lt-component p = Some cmp
    using assms by (simp add: const-lt-component-def)
lemma const-lt-component-SomeD1:
    assumes const-lt-component p = Some cmp
    shows lp p=0
    using assms by (simp add: const-lt-component-def Let-def split: if-split-asm)
lemma const-lt-component-SomeD2:
    assumes const-lt-component p = Some cmp
    shows component-of-term (lt p) = cmp
    using assms by (simp add: const-lt-component-def Let-def split: if-split-asm)
lemma const-lt-component-subset:
    const-lt-component'(B - {0}) - {None }\subseteqSome'component-of-term'Keys
B
proof
    fix }
    assume k const-lt-component '(B-{0}) - {None }
    hence k\in const-lt-component '( }B-{0})\mathrm{ and }k\not=\mathrm{ None by simp-all
    from this(1) obtain p where p\inB-{0} and k=const-lt-component p ..
    moreover from }\langlek\not=None\rangle obtain k' where k=Some k' by blas
    ultimately have const-lt-component p=Some k' and p\inB and p\not=0 by
simp-all
    from this(1) have component-of-term (lt p) =k' by (rule const-lt-component-SomeD2)
    moreover have lt p Keys B by (rule in-KeysI, rule lt-in-keys, fact+)
    ultimately have k' component-of-term ' Keys B by fastforce
    thus k}\in\mathrm{ Some 'component-of-term 'Keys B by (simp add: < k=Some k'〉)
qed
corollary card-const-lt-component-le:
    assumes finite B
    shows card (const-lt-component '(B - {0}) - {None}) \leqcard (component-of-term
' Keys B)
proof (rule surj-card-le)
    show finite (component-of-term'Keys B)
```

```
    by (intro finite-imageI finite-Keys, fact)
next
    show const-lt-component'(B-{0})-{None }\subseteqSome'component-of-term'
Keys B
    by (fact const-lt-component-subset)
qed
end
```


### 6.2.2 Type synonyms

type-synonym $\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata' $=\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right) \times{ }^{\prime} c$
type-synonym $\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata $=\left({ }^{\prime} a \neq{ }_{0}{ }^{\prime} b\right) \times n a t \times{ }^{\prime} c$
type-synonym $\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair $=\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata $\times\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata
type-synonym $\left({ }^{\prime} a, ' b,{ }^{\prime} c,{ }^{\prime} d\right)$ sel $T=\left({ }^{\prime} a,{ }^{\prime} b, ' c\right)$ pdata list $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$

$$
\left({ }^{\prime} a,^{\prime} b,{ }^{\prime} c\right) \text { pdata-pair list } \Rightarrow n a t \times{ }^{\prime} d \Rightarrow\left({ }^{\prime} a,^{\prime} b,{ }^{\prime} c\right)
$$

pdata-pair list
type-synonym $\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right)$ compl $T=\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
$\left({ }^{\prime} a, ' b,{ }^{\prime} c\right)$ pdata-pair list $\Rightarrow\left({ }^{\prime} a,^{\prime} b,{ }^{\prime} c\right)$ pdata-pair list
$\Rightarrow$
nat $\times{ }^{\prime} d \Rightarrow\left(\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right) p d a t a{ }^{\prime}\right.$ list $\left.\times{ }^{\prime} d\right)$
type-synonym $\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right)$ ap $T=\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
$\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair list $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
$n a t \times{ }^{\prime} d \Rightarrow$
( $\left.{ }^{\prime} a, ~ ' b,{ }^{\prime} c\right)$ pdata-pair list
type-synonym $\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right) a b T=\left({ }^{\prime} a, ~ ' b,{ }^{\prime} c\right)$ pdata list $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
$\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow n a t \times{ }^{\prime} d \Rightarrow\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list

### 6.2.3 Specification of the selector parameter

definition sel-spec :: ('a, 'b, 'c, 'd) selT $\Rightarrow$ bool
where sel-spec sel $\longleftrightarrow$
( $\forall$ gs bs ps data. ps $\neq[] \longrightarrow($ sel gs bs ps data $\neq[] \wedge$ set (sel gs bs ps data) $\subseteq$ set $p s)$ )
lemma sel-specI:
assumes $\bigwedge$ gs bs ps data. ps $\neq[] \Longrightarrow$ (sel gs bs ps data $\neq[] \wedge$ set (sel gs bs ps data) $\subseteq$ set $p s$ )
shows sel-spec sel
unfolding sel-spec-def using assms by blast
lemma sel-specD1:
assumes sel-spec sel and $p s \neq[]$
shows sel gs bs ps data $\neq[]$
using assms unfolding sel-spec-def by blast

## lemma sel-specD2:

assumes sel-spec sel and $p s \neq[]$
shows set (sel gs bs ps data) $\subseteq$ set ps
using assms unfolding sel-spec-def by blast

### 6.2.4 Specification of the add-basis parameter

definition $a b$-spec $::\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right) a b T \Rightarrow$ bool
where $a b$-spec $a b \longleftrightarrow$
$(\forall$ gs bs ns data. $n s \neq[] \longrightarrow$ set $(a b$ gs bs ns data $)=$ set bs $\cup$ set $n s) \wedge$ $(\forall$ gs bs data. ab gs bs [] data $=b s)$
lemma ab-specI:
assumes $\bigwedge$ gs bs ns data. $n s \neq[] \Longrightarrow$ set (ab gs bs ns data) $=$ set $b s \cup$ set $n s$ and $\bigwedge g s$ bs data. ab gs bs [] data $=b s$
shows $a b-$ spec $a b$
unfolding ab-spec-def using assms by blast
lemma ab-specD1:
assumes $a b$-spec $a b$
shows set (ab gs bs ns data) $=$ set $b s \cup$ set ns
using assms unfolding ab-spec-def by (metis empty-set sup-bot.right-neutral)
lemma ab-specD2:
assumes $a b$-spec $a b$
shows ab gs bs [] data $=b s$
using assms unfolding ab-spec-def by blast

### 6.2.5 Specification of the add-pairs parameter

$$
\begin{aligned}
& \text { definition unique-idx }::\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right) \text { pdata list } \Rightarrow\left(\text { nat } \times{ }^{\prime} d\right) \Rightarrow \text { bool } \\
& \text { where unique-idx bs data } \longleftrightarrow \\
& (\forall f \in \text { set bs. } \forall g \in \text { set bs. } f \text { st }(\text { snd } f)=\text { sst }(\text { snd } g) \longrightarrow f=g) \wedge \\
& (\forall f \in \text { set bs. fst }(\text { snd } f)<\text { fst data })
\end{aligned}
$$

lemma unique-idxI:
assumes $\bigwedge f g . f \in$ set $b s \Longrightarrow g \in$ set $b s \Longrightarrow f s t($ snd $f)=f s t($ snd $g) \Longrightarrow f=g$ and $\bigwedge f . f \in$ set $b s \Longrightarrow f s t($ snd $f)<f s t$ data
shows unique-idx bs data
unfolding unique-idx-def using assms by blast
lemma unique-idxD1:
assumes unique-idx bs data and $f \in$ set bs and $g \in$ set bs and $f s t(s n d f)=f s t$
(snd g)
shows $f=g$
using assms unfolding unique-idx-def by blast
lemma unique-idxD2:
assumes unique-idx bs data and $f \in$ set bs
shows $f$ st $($ snd $f)<f$ st data

```
    using assms unfolding unique-idx-def by blast
lemma unique-idx-Nil: unique-idx [] data
    by (simp add: unique-idx-def)
lemma unique-idx-subset:
    assumes unique-idx bs data and set bs'}\subseteq\mathrm{ set bs
    shows unique-idx bs' data
proof (rule unique-idxI)
    fix fg
    assume f}\in\mathrm{ set bs'' and g}\in\mathrm{ set bs'
    with assms have unique-idx bs data and f\in set bs and g\in set bs by auto
    moreover assume fst (sndf)=fst (snd g)
    ultimately show f=g by (rule unique-idxD1)
next
    fix f
    assume f}\in\mathrm{ set bs'
    with assms(2) have f\in set bs by auto
    with assms(1) show fst (snd f)< fst data by (rule unique-idxD2)
qed
context gd-term
begin
definition ap-spec :: ('t, 'b::field, 'c, 'd) apT => bool
    where ap-spec ap \longleftrightarrow (\forallgs bs ps hs data.
        set (ap gs bs ps hs data)\subseteq set ps U(set hs \times(set gs U set bs U set hs )})
        (}\forallBdm.\forallh\inset hs. \forallg\inset gs \cup set bs \cup set hs. dickson-grading d \longrightarrow
            set gs \cup set bs \cup set hs\subseteqB\longrightarrowfst' B\subseteq dgrad-p-set d m}
            set ps\subseteq set bs }\times(\mathrm{ set gs U set bs) }\longrightarrowunique-idx (gs @ bs @ hs)data \longrightarrow
            is-Groebner-basis (fst'set gs) \longrightarrowh\not=g\longrightarrowfst h\not=0\longrightarrowfst g\not=0\longrightarrow
            (\forallab. (a,b) \inp set (ap gs bs ps hs data) \longrightarrowfst a\not=0\longrightarrowfst b\not=0\longrightarrow
                crit-pair-cbelow-on d m (fst' B) (fst a) (fst b))}
        (\forallab.a\in set gs \cup set bs\longrightarrowb\in set gs \cup set bs \longrightarrowfst a\not=0 \longrightarrow fst b\not=
O\longrightarrow
            crit-pair-cbelow-on d m (fst' B) (fst a) (fst b)) \longrightarrow
            crit-pair-cbelow-on d m (fst' B) (fst h) (fst g)) ^
        (\forallBdm.}\forallh g. dickson-grading d \longrightarrow
            set gs \cup set bs \cup set hs \subseteqB\longrightarrowfst' B\subseteq dgrad-p-set d m\longrightarrow
            set ps\subseteq set bs }\times(\mathrm{ set gs U set bs )}\longrightarrow(\mathrm{ set gs U set bs ) }\cap\mathrm{ set hs ={} }
            unique-idx (gs @ bs @ hs)data \longrightarrow is-Groebner-basis (fst'set gs)\longrightarrow
            h\not=g\longrightarrowfst h\not=0\longrightarrowfst g\not=0\longrightarrow
    (h,g)\in set ps - }\mp@subsup{p}{0}{}\mathrm{ set (ap gs bs ps hs data) }
    (\forallab. (a,b) \inp set (ap gs bs ps hs data) \longrightarrow(a,b)\inp set hs }\times(\mathrm{ set gs U
set bs \cup set hs) \longrightarrow
                fst a\not=0\longrightarrow fst b\not=0\longrightarrowcrit-pair-cbelow-on d m (fst' B) (fst a)
(fst b))}
    crit-pair-cbelow-on d m (fst ' B) (fst h) (fst g)))
```

Informally, ap-spec ap means that, for suitable arguments $g s, b s, p s$ and $h s$,
the value of $a p g s b s p s h s$ is a list of pairs $p s^{\prime}$ such that for every element $(a, b)$ missing in $p s^{\prime}$ there exists a set of pairs $C$ by reference to which ( $a$, $b)$ can be discarded, i.e. as soon as all critical pairs of the elements in $C$ can be connected below some set $B$, the same is true for the critical pair of $(a, b)$.

```
lemma ap-specI:
    assumes \(\bigwedge\) gs bs ps hs data. set (ap gs bs ps hs data) \(\subseteq\) set \(p s \cup(\) set hs \(\times\) (set
\(g s \cup\) set \(b s \cup\) set \(h s))\)
    assumes \(\bigwedge\) gs bs ps hs data \(B d m h g\). dickson-grading \(d \Longrightarrow\)
        set gs \(\cup\) set bs \(\cup\) set \(h s \subseteq B \Longrightarrow f s t\) ' \(B \subseteq\) dgrad- \(p\)-set \(d m \Longrightarrow\)
        \(h \in\) set \(h s \Longrightarrow g \in\) set \(g s \cup\) set \(b s \cup\) set \(h s \Longrightarrow\)
        set \(p s \subseteq\) set \(b s \times(\) set \(g s \cup\) set \(b s) \Longrightarrow u n i q u e-i d x(g s @ b s @ h s) d a t a\)
\(\Longrightarrow\)
        is-Groebner-basis \((f s t\) ' set \(g s) \Longrightarrow h \neq g \Longrightarrow f s t h \neq 0 \Longrightarrow f s t g \neq 0\)
\(\Longrightarrow\)
    \(\left(\bigwedge a b .(a, b) \in_{p}\right.\) set \((a p\) gs bs ps hs data) \(\Longrightarrow f s t a \neq 0 \Longrightarrow f s t b \neq 0\)
\(\Longrightarrow\)
            crit-pair-cbelow-on d m \(\left.\left(f_{s t}{ }^{\prime} B\right)(f s t a)(f s t ~ b)\right) \Longrightarrow\)
```

        \((\bigwedge a b . a \in \operatorname{set} g s \cup\) set \(b s \Longrightarrow b \in\) set \(g s \cup\) set \(b s \Longrightarrow f s t a \neq 0 \Longrightarrow\)
    fst $b \neq 0 \Longrightarrow$
crit-pair-cbelow-on d $m(f s t$ ' $B)(f s t ~ a)(f s t ~ b)) \Longrightarrow$
crit-pair-cbelow-on d m (fst ' B) (fst h) (fst g)
assumes $\bigwedge$ gs bs ps hs data $B d m h g$. dickson-grading $d \Longrightarrow$
set $g s \cup$ set $b s \cup$ set $h s \subseteq B \Longrightarrow f s t$ ' $B \subseteq$ dgrad- $p$-set $d m \Longrightarrow$
set $p s \subseteq$ set $b s \times($ set $g s \cup$ set $b s) \Longrightarrow($ set gs $\cup$ set $b s) \cap$ set $h s=\{ \}$
$\Longrightarrow$
unique-idx (gs @ bs @ hs) data $\Longrightarrow$ is-Groebner-basis (fst'set gs) $\Longrightarrow$
$h \neq g \Longrightarrow$
fst $h \neq 0 \Longrightarrow f s t g \neq 0 \Longrightarrow(h, g) \in$ set $p s-_{p}$ set (ap gs bs ps hs data)
$\Longrightarrow$
$\left(\bigwedge a b .(a, b) \in_{p}\right.$ set $(a p$ gs bs ps hs data $) \Longrightarrow(a, b) \in_{p}$ set $h s \times($ set
$g s \cup$ set $b s \cup$ set $h s) \Longrightarrow$
fst $a \neq 0 \Longrightarrow$ fst $b \neq 0 \Longrightarrow$ crit-pair-cbelow-on d $m$ ( $f s t$ ' $B)(f s t$
a) $(f s t b)) \Longrightarrow$
crit-pair-cbelow-on d m (fst ' B) (fst h) (fst g)
shows ap-spec ap
unfolding ap-spec-def
apply (intro allI conjI impI)
subgoal by (rule assms(1))
subgoal by (intro balli impI, rule assms(2), blast+)
subgoal by (rule assms(3), blast+)
done
lemma ap-specD1:
assumes ap-spec ap
shows set $(a p$ gs bs ps hs data) $\subseteq$ set $p s \cup($ set $h s \times($ set $g s \cup$ set bs $\cup$ set hs $))$ using assms unfolding ap-spec-def by (elim allE conjE) (assumption)

```
lemma ap-specD2:
    assumes ap-spec ap and dickson-grading d and set gs }\cup\mathrm{ set bs }\cup\mathrm{ set hs}\subseteq
        and fst' }B\subseteqdgrad-p-set dm and (h,g)\inp set hs \times (set gs \cup set bs \cup set hs
    and set ps\subseteq set bs\times(set gs \cup set bs) and unique-idx (gs @ bs @ hs)data
    and is-Groebner-basis (fst' set gs) and h\not=g and fst h\not=0 and fst g\not=0
    and \ab. (a,b)\inp set (ap gs bs ps hs data) \Longrightarrowfst a\not=0\Longrightarrowfst b\not=0\Longrightarrow
                crit-pair-cbelow-on d m (fst ' B) (fst a) (fst b)
    and \bigwedgeab.a\in set gs \cup set bs \Longrightarrowb\in set gs \cup set bs \Longrightarrowfst a\not=0\Longrightarrowfstb
#= \Longrightarrow
                                    crit-pair-cbelow-on d m (fst ' B) (fst a) (fst b)
    shows crit-pair-cbelow-on d m (fst ' B) (fst h) (fst g)
proof -
    from assms(5) have (h,g)\in set hs }\times(\mathrm{ set gs U set bs U set hs)}\vee\mp@code{(g,h)\in set
hs\times(set gs \cup set bs \cup set hs)
    by (simp only: in-pair-iff)
    thus ?thesis
    proof
        assume (h,g) \in set hs \times (set gs U set bs U set hs)
        hence}h\in\mathrm{ set hs and g}\mathrm{ set gs U set bs U set hs by simp-all
    from assms(1)[unfolded ap-spec-def, rule-format, of gs bs ps hs data] assms(2-4)
this assms (6-)
    show ?thesis by metis
    next
    assume (g,h)\in set hs \times (set gs U set bs U set hs)
    hence g}\in\mathrm{ set hs and h}\in\mathrm{ set gs U set bs U set hs by simp-all
    hence crit-pair-cbelow-on d m (fst ' B) (fst g) (fst h)
            using assms(1)[unfolded ap-spec-def, rule-format, of gs bs ps hs data]
                assms(2,3,4,6,7,8,10,11,12,13) assms(9)[symmetric]
            by metis
    thus ?thesis by (rule crit-pair-cbelow-sym)
    qed
qed
lemma ap-specD3:
assumes ap-spec ap and dickson-grading d and set gs \(\cup\) set bs \(\cup\) set \(h s \subseteq B\)
            and fst' B\subseteqdgrad-p-set d m}\mathrm{ and set ps }\subseteq\mathrm{ set bs }\times(\mathrm{ set gs U set bs)
            and (set gs \cup set bs) \cap set hs ={} and unique-idx (gs @ bs @ hs)data
    and is-Groebner-basis (fst' set gs) and h\not=g and fst h\not=0 and fst g\not=0
    and (h,g) \inp set ps - 
    and \ab.a cet hs \Longrightarrowb\in set gs \cup set bs \cup set hs \Longrightarrow(a,b)\inp set (ap gs
bs ps hs data) \Longrightarrow
                    fst a\not=0\Longrightarrow fst b}=0\Longrightarrow\mathrm{ crit-pair-cbelow-on d m (fst' B) (fst a)
(fst b)
    shows crit-pair-cbelow-on d m (fst ' B) (fst h) (fst g)
proof -
    have *: crit-pair-cbelow-on d m (fst ' B) (fst a) (fst b)
    if 1:(a,b)\in \in set (ap gs bs ps hs data) and 2: (a,b)\in 
bs\cup set hs)
    and 3: fst }a\not=0\mathrm{ and 4: fst b}=0\mathrm{ for a b
```

```
    proof -
    from 2 have (a,b) fet hs \times (set gs U set bs \cup set hs ) \vee (b,a) f set hs }
(set gs U set bs U set hs)
    by (simp only: in-pair-iff)
    thus ?thesis
    proof
            assume (a,b) \in set hs \times (set gs \cup set bs \cup set hs)
            hence }a\in\mathrm{ set hs and b}\mathrm{ set gs U set bs U set hs by simp-all
            thus ?thesis using 134 by (rule assms(13))
    next
            assume (b,a) \in set hs \times (set gs U set bs \cup set hs)
            hence b\in set hs and a\in set gs U set bs \cup set hs by simp-all
            moreover from 1 have (b,a)\inp set (ap gs bs ps hs data) by (auto simp:
in-pair-iff)
            ultimately have crit-pair-cbelow-on d m (fst` B) (fst b) (fst a) using 4 3
by (rule assms(13))
            thus ?thesis by (rule crit-pair-cbelow-sym)
            qed
    qed
    from assms(12) have (h,g)\in set ps - 
                                    (g,h)\in set ps - -p set (ap gs bs ps hs data) by (simp only:
in-pair-iff)
    thus ?thesis
    proof
        assume (h,g)\in set ps - }\mp@subsup{p}{\mathrm{ set (ap gs bs ps hs data)}}{\mathrm{ sem}
    with assms(1)[unfolded ap-spec-def, rule-format, of gs bs ps hs data] assms(2-11)
        show ?thesis using assms(10) * by metis
    next
        assume (g,h)\in set ps -p set (ap gs bs ps hs data)
    with assms(1)[unfolded ap-spec-def, rule-format, of gs bs ps hs data] assms(2-11)
        have crit-pair-cbelow-on d m (fst'B) (fst g) (fst h) using assms(10) * by
metis
    thus ?thesis by (rule crit-pair-cbelow-sym)
    qed
qed
lemma ap-spec-Nil-subset:
    assumes ap-spec ap
    shows set (ap gs bs ps [] data)\subseteq set ps
    using ap-specD1[OF assms] by fastforce
lemma ap-spec-fst-subset:
    assumes ap-spec ap
    shows fst'set (ap gs bs ps hs data)\subseteq fst ' set ps U set hs
proof -
    from ap-specD1[OF assms]
    have fst'set (ap gs bs ps hs data)\subseteq fst'( set ps U set hs \times (set gs U set bs U
set hs))
    by (rule image-mono)
```

```
        thus ?thesis by auto
qed
lemma ap-spec-snd-subset:
    assumes ap-spec ap
    shows snd'set (ap gs bs ps hs data)\subseteq snd ' set ps U set gs U set bs U set hs
proof -
    from ap-specD1[OF assms]
    have snd'set (ap gs bs ps hs data) \subseteq snd '(set ps U set hs \times (set gs U set bs
~st hs))
            by (rule image-mono)
    thus ?thesis by auto
qed
lemma ap-spec-inE:
    assumes ap-spec ap and (p,q)\in set (ap gs bs ps hs data)
    assumes 1: (p,q) \in set ps\Longrightarrow thesis
    assumes 2: p}\in\mathrm{ set hs "q set gs U set bs U set hs # thesis
    shows thesis
proof -
    from assms(2) ap-specD1[OF assms(1)] have (p,q) \in set ps U set hs \times (set gs
\cupset bs \cup set hs) ..
    thus ?thesis
    proof
            assume (p,q)\in set ps
            thus ?thesis by (rule 1)
    next
            assume (p,q)\in set hs \times (set gs U set bs U set hs)
            hence p\in set hs and q\in set gs U set bs U set hs by blast+
            thus ?thesis by (rule 2)
    qed
qed
lemma subset-Times-ap:
    assumes ap-spec ap and ab-spec ab and set ps\subseteq set bs \times (set gs U set bs)
    shows set (ap gs bs (ps-- sps) hs data)\subseteq set (ab gs bs hs data) }\times(\mathrm{ set gs U
set (ab gs bs hs data))
proof
    fix pq
    assume (p,q)\in set (ap gs bs (ps-- sps) hs data)
    with assms(1) show (p,q)\in set (ab gs bs hs data) }\times(\mathrm{ set gs U set (ab gs bs hs
data))
    proof (rule ap-spec-inE)
            assume (p,q) \in set (ps-- sps)
            hence (p,q)\in set ps by (simp add: set-diff-list)
            from this assms(3) have (p,q) \in set bs \times (set gs U set bs)..
            hence p}\in\mathrm{ set bs and q}\mathrm{ set gs U set bs by blast+
            thus ?thesis by (auto simp add: ab-specD1[OF assms(2)])
    next
```

```
    assume p\in set hs and q\in set gs U set bs U set hs
    thus ?thesis by (simp add: ab-specD1[OF assms(2)])
    qed
qed
```


### 6.2.6 Function args-to-set

definition args-to-set :: ('t, 'b::field, 'c) pdata list $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\times\left({ }^{\prime} t,{ }^{\prime} b\right.$, $\left.{ }^{\prime} c\right)$ pdata-pair list $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)$ set
where args-to-set $x=f s t$ ' $($ set $(f s t x) \cup$ set $(f s t($ snd $x)) \cup f s t$ ' set (snd (snd $x)) \cup \operatorname{snd} ' \operatorname{set}(\operatorname{snd}(\operatorname{snd} x)))$
lemma args-to-set-alt:

$$
\operatorname{args-to-set}(g s, b s, p s)=f s t \text { ' set } g s \cup f s t \text { ' set } b s \cup f s t \text { ' fst' set } p s \cup f s t \text { ' snd' }
$$

$$
\text { set } p s
$$

by (simp add: args-to-set-def image-Un)
lemma args-to-set-subset-Times:
assumes set $p s \subseteq$ set $b s \times($ set gs $\cup$ set $b s)$
shows args-to-set $(g s, b s, p s)=f s t$ ' set $g s \cup f s t$ 'set bs
unfolding args-to-set-alt using assms by auto
lemma args-to-set-subset:
assumes ap-spec $a p$ and $a b$-spec $a b$
shows args-to-set (gs, ab gs bs hs data, ap gs bs ps hs data) $\subseteq$

$$
f s t \text { ' }(\text { set } g s \cup \text { set } b s \cup f s t \text { ' set } p s \cup \text { snd' set } p s \cup \text { set } h s)(\text { is } ? l \subseteq f s t \text { ' }
$$

? $r$ )
proof (simp only: args-to-set-alt Un-subset-iff, intro conjI image-mono)
show set (ab gs bs hs data) $\subseteq$ ? $r$ by (auto simp add: ab-specD1[OF assms(2)])
next
from $\operatorname{assms}(1)$ have $f s t$ ' set (ap gs bs ps hs data) $\subseteq f s t$ ' set $p s \cup$ set hs
by (rule ap-spec-fst-subset)
thus $f s t$ ' set (ap gs bs ps hs data) $\subseteq$ ? $r$ by blast
next
from $\operatorname{assms}(1)$ have snd' set (ap gs bs ps hs data) $\subseteq$ snd' set $p s \cup$ set gs $\cup$ set bs $\cup$ set hs
by (rule ap-spec-snd-subset)
thus snd' set (ap gs bs ps hs data) $\subseteq$ ? ? by blast
qed blast
lemma args-to-set-alt2:
assumes ap-spec $a p$ and $a b-s p e c a b$ and set $p s \subseteq$ set $b s \times($ set gs $\cup$ set bs)
shows args-to-set (gs, ab gs bs hs data, ap gs bs (ps--sps) hs data) $=$ $f s t$ ' $($ set $g s \cup$ set $b s \cup$ set $h s)($ is ?l $=f s t$ ' ? $r)$
proof
from $\operatorname{assms}(1,2)$ have $? l \subseteq f s t$ ' $($ set $g s \cup$ set $b s \cup f s t$ ' set $(p s--s p s) \cup$ snd ' set $(p s--s p s) \cup$ set $h s)$
by (rule args-to-set-subset)
also have $\ldots \subseteq f s t$ ' ? $r$

```
    proof (rule image-mono)
    have set gs U set bs Ufst'set (ps-- sps) U snd'set (ps-- sps) U set hs\subseteq
                set gs U set bs Ufst'set ps U snd' set ps U set hs by (auto simp:
set-diff-list)
    also from assms(3) have \ldots. \subseteq?r by fastforce
    finally show set gs U set bs Ufst'set (ps-- sps) U snd'set (ps -- sps) U
set hs \subseteq? ?r .
    qed
    finally show ?l \subseteqfst ' ?r .
next
    from assms(2) have eq: set (ab gs bs hs data) = set bs U set hs by (rule
ab-specD1)
    have fst ' ?r }\subseteqfst 'set gs \cupfst' set (ab gs bs hs data) unfolding eq usin
assms(3)
    by fastforce
    also have ...\subseteq?l unfolding args-to-set-alt by fastforce
    finally show fst'?r }\subseteq\mathrm{ ?l.
qed
lemma args-to-set-subset1:
    assumes set gs1 \subseteq set gs2
    shows args-to-set (gs1,bs, ps)\subseteqargs-to-set (gs2, bs, ps)
    using assms by (auto simp add: args-to-set-alt)
lemma args-to-set-subset2:
    assumes set bs1\subseteq set bs2
    shows args-to-set (gs, bs1, ps)\subseteqargs-to-set (gs, bs2, ps)
    using assms by (auto simp add: args-to-set-alt)
lemma args-to-set-subset3:
    assumes set ps1\subseteq set ps2
    shows args-to-set (gs,bs, ps1) \subseteqargs-to-set (gs, bs, ps2)
    using assms unfolding args-to-set-alt by blast
```

6.2.7 Functions count-const-lt-components, count-rem-comps and full-gb definition rem-comps-spec :: ('t, 'b::zero, 'c) pdata list $\Rightarrow$ nat $\times$ ' $d \Rightarrow$ bool where rem-comps-spec bs data $\longleftrightarrow$ (card (component-of-term' Keys (fst'set $b s))=$

$$
\text { fst data }+ \text { card (const-lt-component ' (fst ' set bs - }
$$

$$
\{0\})-\{\text { None }\}))
$$

definition count-const-lt-components :: ('t, 'b::zero, 'c) pdata' list $\Rightarrow$ nat where count-const-lt-components $h s=$ length (remdups (filter $(\lambda x . x \neq$ None) (map (const-lt-component $\circ$ fst) hs)))
definition count-rem-components :: ('t, 'b::zero, 'c) pdata' list $\Rightarrow$ nat where count-rem-components $b s=$ length (remdups (map component-of-term (Keys-to-list (map fst bs)))) -

$$
\text { count-const-lt-components }[b \leftarrow b s . f s t b \neq 0]
$$

lemma count-const-lt-components-alt:
count-const-lt-components $h s=$ card (const-lt-component'fst'set hs - \{None $\}$ )
by (simp add: count-const-lt-components-def card-set[symmetric] set-diff-eq im-age-comp del: not-None-eq)
lemma count-rem-components-alt:
count-rem-components bs + card (const-lt-component' $(f s t$ ' set bs $-\{0\})-$
$\{$ None $\}$ ) $=$
card (component-of-term ' Keys (fst'set bs))
proof -
have eq: fst' $\{x \in$ set bs. fst $x \neq 0\}=f s t$ ' set bs $-\{0\}$ by fastforce
have card (const-lt-component' $(f s t '$ set bs $-\{0\})-\{$ None $\}) \leq$ card (component-of-term
‘Keys ( $f s t$ ' set bs))
by (rule card-const-lt-component-le, rule finite-imageI, fact finite-set)
thus ?thesis
by (simp add: count-rem-components-def card-set[symmetric] set-Keys-to-list count-const-lt-components-alt eq)
qed
lemma rem-comps-spec-count-rem-components: rem-comps-spec bs (count-rem-components $b s$, data)
by (simp only: rem-comps-spec-def fst-conv count-rem-components-alt)
definition full-gb :: ('t, 'b, 'c) pdata list $\Rightarrow\left(' t\right.$, 'b::zero-neq-one, $\left.{ }^{\prime} c:: d e f a u l t\right)$ pdata list
where full-gb bs $=\operatorname{map}(\lambda k .($ monomial $1($ term-of-pair $(0, k)), 0$, default $))$
(remdups (map component-of-term (Keys-to-list (map fst bs))))
lemma fst-set-full-gb:
fst'set $($ full-gb bs $)=(\lambda v$. monomial $1($ term-of-pair ( 0 , component-of-term $v))$ )
'Keys ( $f s t$ ' set bs)
by (simp add: full-gb-def set-Keys-to-list image-comp)
lemma Keys-full-gb:
Keys $(f s t$ 'set $($ full-gb bs $))=(\lambda v$. term-of-pair ( 0 , component-of-term $v)$ )'Keys
(fst' set bs)
by (auto simp add: fst-set-full-gb Keys-def image-image)
lemma pps-full-gb: pp-of-term'Keys $(f s t '$ set $(f u l l-g b b s)) \subseteq\{0\}$
by (simp add: Keys-full-gb image-comp image-subset-iff term-simps)
lemma components-full-gb:
component-of-term 'Keys $(f s t$ ' set $(f u l l-g b b s))=$ component-of-term 'Keys $(f s t$ ' set bs)
by (simp add: Keys-full-gb image-comp, rule image-cong, fact refl, simp add: term-simps)

```
lemma full-gb-is-full-pmdl: is-full-pmdl (fst'set (full-gb bs))
    for bs::('t, 'b::field, 'c::default) pdata list
proof (rule is-full-pmdlI-lt-finite)
    from finite-set show finite (fst'set (full-gb bs)) by (rule finite-imageI)
next
    fix }
    assume k\in component-of-term'Keys (fst'set (full-gb bs))
        then obtain v}\mathrm{ where v}\in\mathbb{Keys}(fst'set (full-gb bs)) and k:k=compo
nent-of-term v ..
    from this(1) obtain b}\mathrm{ where b fst'set (full-gb bs) and v}\mathrm{ ' keys b by (rule
in-KeysE)
    from this(1) obtain u where }u\inKeys(fst'set bs) and b:b= monomial 1
(term-of-pair (0, component-of-term u))
            unfolding fst-set-full-gb ..
    have lt:lt b = term-of-pair (0, component-of-term u) by (simp add: blt-monomial)
    from }\langlev\in\mathrm{ keys b> have v: v = term-of-pair (0, component-of-term u) by (simp
add: b)
    show \existsb\infst'set (full-gb bs). b\not=0^ component-of-term (lt b)=k^lpb=0
    proof (intro bexI conjI)
        show b\not=0 by (simp add: b monomial-0-iff)
    next
        show component-of-term (lt b) =k by (simp add: lt term-simps k v)
    next
        show lp b = 0 by (simp add:lt term-simps)
    qed fact
qed
```

In fact, is-full-pmdl (fst'set (full-gb ?bs)) also holds if 'b is no field.

```
lemma full-gb-isGB: is-Groebner-basis (fst'set (full-gb bs))
proof (rule Buchberger-criterion-finite)
    from finite-set show finite ( \(f s t\) 'set (full-gb bs)) by (rule finite-imageI)
next
    fix \(p q::{ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\)
    assume \(p \in f s t\) 'set (full-gb bs)
    then obtain \(v\) where \(p: p=\) monomial 1 (term-of-pair ( 0 , component-of-term
v))
    unfolding fst-set-full-gb ..
    hence \(l t\) : component-of-term (lt p) \(=\) component-of-term \(v\) by (simp add: lt-monomial
term-simps)
    assume \(q \in f s t\) 'set (full-gb bs)
    then obtain \(u\) where \(q: q=\) monomial 1 (term-of-pair ( 0 , component-of-term
u))
    unfolding fst-set-full-gb ..
    hence \(l q\) : component-of-term ( \(l t q\) ) = component-of-term \(u\) by (simp add: lt-monomial
term-simps)
    assume component-of-term (lt p) = component-of-term (lt q)
    hence component-of-term \(v=\) component-of-term \(u\) by (simp only: lt lq)
    hence \(p=q\) by (simp only: \(p q\) )
    moreover assume \(p \neq q\)
```

ultimately show $(\text { red }(f s t \text { ' set }(f u l l-g b b s)))^{* *}($ spoly p q) 0 by (simp only:) qed

### 6.2.8 Specification of the completion parameter

definition compl-struct :: ('t, 'b::field, 'c, 'd) complT $\Rightarrow$ bool where compl-struct compl $\longleftrightarrow$
( $\forall$ gs bs ps sps data. sps $\neq[] \longrightarrow$ set sps $\subseteq$ set $p s \longrightarrow$
( $\forall$ d. dickson-grading $d \longrightarrow$ dgrad-p-set-le d (fst' (set (fst (compl gs bs (ps -- sps) sps data) $)$ ))
$($ args-to-set $(g s, b s, p s))) \wedge$
component-of-term'Keys (fst' (set (fst (compl gs bs (ps -- sps) sps (ata) ) )) $\subseteq$

$$
\text { component-of-term ‘ Keys (args-to-set }(g s, b s, p s)) \wedge
$$

$0 \notin f s t$ ' set (fst (compl gs bs (ps -- sps) sps data) ) $\wedge$
( $\forall h \in \operatorname{set}(f s t(c o m p l ~ g s ~ b s ~(p s--s p s) ~ s p s ~ d a t a)) . ~ \forall b \in s e t ~ g s ~ \cup s e t ~ b s$. $\left.\left.f s t b \neq 0 \longrightarrow \neg l t(f s t b) a d d s_{t} l t(f s t h)\right)\right)$
lemma compl-structI:
assumes $\bigwedge d$ gs bs ps sps data. dickson-grading $d \Longrightarrow$ sps $\neq[] \Longrightarrow$ set sps $\subseteq$ set $p s \Longrightarrow$
dgrad-p-set-le d (fst' $(\operatorname{set}(f s t($ compl gs bs (ps -- sps) sps data $))))$ (args-to-set (gs, bs, ps))
assumes $\bigwedge$ gs bs ps sps data. sps $\neq[] \Longrightarrow$ set $s p s \subseteq$ set $p s \Longrightarrow$
component-of-term 'Keys (fst' (set (fst (compl gs bs (ps -- sps) sps
data) $))) \subseteq$

> component-of-term' Keys (args-to-set (gs, bs, ps))
assumes $\bigwedge$ gs bs ps sps data. sps $\neq[] \Longrightarrow$ set sps $\subseteq$ set $p s \Longrightarrow 0 \notin f s t$ 'set (fst (compl gs bs (ps -- sps) sps data))
assumes $\bigwedge$ gs bs ps sps $h b$ data. sps $\neq[] \Longrightarrow$ set sps $\subseteq$ set $p s \Longrightarrow h \in \operatorname{set}(f s t$
(compl gs bs (ps -- sps) sps data)) $\Longrightarrow$
$b \in$ set $g s \cup$ set $b s \Longrightarrow f s t b \neq 0 \Longrightarrow \neg l t(f s t b) a d d s_{t} l t(f s t h)$
shows compl-struct compl
unfolding compl-struct-def using assms by auto
lemma compl-structD1:
assumes compl-struct compl and dickson-grading $d$ and $s p s \neq[]$ and set sps $\subseteq$ set ps
shows dgrad-p-set-le d (fst' (set (fst (compl gs bs (ps -- sps) sps data)))) (args-to-set (gs, bs, ps))
using assms unfolding compl-struct-def by blast
lemma compl-structD2:
assumes compl-struct compl and sps $\neq[]$ and set sps $\subseteq$ set ps
shows component-of-term' Keys (fst' (set (fst (compl gs bs (ps -- sps) sps data) $))) \subseteq$

```
            component-of-term`Keys (args-to-set (gs, bs, ps))
```

using assms unfolding compl-struct-def by blast
lemma compl-structD3:
assumes compl-struct compl and sps $\neq[]$ and set sps $\subseteq$ set ps
shows $0 \notin f s t$ 'set ( $f s t$ (compl gs bs ( $p s--$ sps) sps data))
using assms unfolding compl-struct-def by blast
lemma compl-structD4:
assumes compl-struct compl and sps $\neq[]$ and set sps $\subseteq$ set ps and $h \in \operatorname{set}(f s t(c o m p l$ gs bs (ps - - sps) sps data)) and $b \in$ set $g s \cup$ set bs and fst $b \neq 0$
shows $\neg l t(f s t b) a d d s_{t} l t(f s t h)$
using assms unfolding compl-struct-def by blast
definition struct-spec :: ('t, 'b::field, ' $\left.c,{ }^{\prime} d\right)$ selT $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right) a p T \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b\right.$, $\left.{ }^{\prime} c,{ }^{\prime} d\right) a b T \Rightarrow$

$$
\left(' t,{ }^{\prime} b,{ }^{\prime} c,,^{\prime} d\right) \operatorname{compl} T \Rightarrow \text { bool }
$$

where struct-spec sel ap ab compl $\longleftrightarrow$ (sel-spec sel $\wedge$ ap-spec $a p \wedge a b$-spec $a b \wedge$ compl-struct compl)
lemma struct-specI:
assumes sel-spec sel and ap-spec ap and ab-spec ab and compl-struct compl shows struct-spec sel ap ab compl
unfolding struct-spec-def using assms by (intro conjI)
lemma struct-specD1:
assumes struct-spec sel ap ab compl
shows sel-spec sel
using assms unfolding struct-spec-def by (elim conjE)
lemma struct-specD2:
assumes struct-spec sel ap ab compl
shows ap-spec ap
using assms unfolding struct-spec-def by (elim conjE)
lemma struct-specD3:
assumes struct-spec sel ap ab compl
shows $a b$-spec $a b$
using assms unfolding struct-spec-def by (elim conjE)
lemma struct-specD4:
assumes struct-spec sel ap ab compl
shows compl-struct compl
using assms unfolding struct-spec-def by (elim conjE)
lemmas struct-specD = struct-specD1 struct-specD2 struct-specD3 struct-specD4
definition compl-pmdl $::(' t$, 'b::field, ' $c$, 'd) complT $\Rightarrow$ bool
where compl-pmdl compl $\longleftrightarrow$
( $\forall$ gs bs ps sps data. is-Groebner-basis $(f s t$ ' set gs) $\longrightarrow$ sps $\neq[] \longrightarrow$ set $s p s \subseteq$ set $p s \longrightarrow$

```
    unique-idx (gs @ bs)data \longrightarrow
    fst'(set (fst (compl gs bs (ps -- sps) sps data))) \subseteq pmdl (args-to-set
(gs,bs,ps)))
lemma compl-pmdlI:
    assumes \bigwedgegs bs ps sps data. is-Groebner-basis (fst'set gs)\Longrightarrow sps # [] \Longrightarrow set
sps}\subseteq\mathrm{ set ps }
    unique-idx (gs @ bs) data \Longrightarrow
    fst '(set (fst (compl gs bs (ps -- sps) sps data))) \subseteq pmdl (args-to-set
(gs,bs, ps))
    shows compl-pmdl compl
    unfolding compl-pmdl-def using assms by blast
lemma compl-pmdlD:
    assumes compl-pmdl compl and is-Groebner-basis (fst ' set gs)
        and sps \not=[] and set sps\subseteq set ps and unique-idx (gs @ bs) data
    shows fst ' (set (fst (compl gs bs (ps -- sps) sps data))) \subseteqpmdl (args-to-set
(gs, bs, ps))
    using assms unfolding compl-pmdl-def by blast
definition compl-conn :: ('t, 'b::field, 'c, 'd) complT = bool
    where compl-conn compl \longleftrightarrow
        (\foralldm gs bs ps sps p q data. dickson-grading d \longrightarrowfst' set gs }\subseteq\mathrm{ dgrad-p-set
dm\longrightarrow
            is-Groebner-basis (fst' set gs) \longrightarrow fst' set bs \subseteqdgrad-p-set d m}
    set ps\subseteq set bs }\times(\mathrm{ set gs }\cup\mathrm{ set bs )}\longrightarrow\mathrm{ sps }\not=[]\longrightarrow\mathrm{ set sps }\subseteq\mathrm{ set ps }
        unique-idx (gs @ bs)data \longrightarrow(p,q)\in set sps \longrightarrow fst p}=0=0\longrightarrowfst q
# 0\longrightarrow
        crit-pair-cbelow-on d m (fst '(set gs \cup set bs) \cupfst'set (fst (compl gs
bs (ps -- sps) sps data))) (fst p) (fst q))
```

Informally, compl-conn compl means that, for suitable arguments $g s, b s, p s$ and sps, the value of compl gs bs ps sps is a list hs such that the critical pairs of all elements in sps can be connected modulo set gs $\cup$ set bs $\cup$ set hs.
lemma compl-connI:
assumes $\bigwedge d m$ gs bs ps sps $p q$ data. dickson-grading $d \Longrightarrow f s t$ 'set $g s \subseteq$ dgrad-p-set $d m \Longrightarrow$
is-Groebner-basis (fst' set gs) $\Longrightarrow$ fst' set bs $\subseteq$ dgrad- $p$-set d $m \Longrightarrow$
set $p s \subseteq$ set $b s \times($ set $g s \cup$ set $b s) \Longrightarrow$ sps $\neq[] \Longrightarrow$ set sps $\subseteq$ set $p s \Longrightarrow$ unique-idx $(g s @ b s)$ data $\Longrightarrow(p, q) \in$ set sps $\Longrightarrow$ fst $p \neq 0 \Longrightarrow f$ st $q \neq$
$0 \Longrightarrow$
crit-pair-cbelow-on $d m(f s t$ ' set $g s \cup$ set bs $) \cup f s t$ ' set (fst (compl gs
$b s(p s--s p s)$ sps data) ) ) (fst p) (fst q)
shows compl-conn compl
unfolding compl-conn-def using assms by presburger
lemma compl-connD:
assumes compl-conn compl and dickson-grading $d$ and $f s t$ 'set gs $\subseteq d g r a d-p$-set

```
\(d m\)
    and \(i s\)-Groebner-basis ( \(f\) st ' set gs) and \(f\) st' set bs \(\subseteq\) dgrad-p-set d m
    and set \(p s \subseteq\) set \(b s \times(\) set gs \(\cup\) set bs \()\) and sps \(\neq[]\) and set sps \(\subseteq\) set ps
    and unique-idx (gs @ bs) data and \((p, q) \in\) set sps and fst \(p \neq 0\) and \(f\) st \(q \neq\)
0
    shows crit-pair-cbelow-on d \(m\) (fst ' (set gs \(\cup\) set bs) \(\cup f s t\) ' set (fst (compl gs bs
( \(p s--\) sps) sps data) ) ) (fst p) (fst q)
    using assms unfolding compl-conn-def Un-assoc by blast
```


### 6.2.9 Function gb-schema-dummy

definition (in -) add-indices :: (('a, 'b, 'c) pdata' list $\left.\times{ }^{\prime} d\right) \Rightarrow\left(n a t \times{ }^{\prime} d\right) \Rightarrow\left(\left({ }^{\prime} a\right.\right.$, 'b, 'c) pdata list $\times n a t \times{ }^{\prime} d$ )
where [code del]: add-indices ns data $=$
 $n s)$, snd $n s$ )
lemma (in -) add-indices-code [code]:
add-indices ( $n s$, data $)\left(n\right.$, data $\left.^{\prime}\right)=(\operatorname{map}-i d x(\lambda(h, d) i .(h, i, d)) n s n, n+$ length $n s$, data)
by (simp add: add-indices-def case-prod-beta')
lemma $f$ st-add-indices: map $f s t(f s t(a d d-i n d i c e s ~ n s ~ d a t a '))=m a p ~ f s t(f s t n s)$
by (simp add: add-indices-def map-map-idx map-idx-no-idx)
corollary fst-set-add-indices: $f s t$ ' set $(f s t($ add-indices ns data' $))=f s t$ ' set $(f s t$ $n s)$
using fst-add-indices by (metis set-map)
lemma in-set-add-indicesE:
assumes $f \in \operatorname{set}(f s t$ (add-indices aux data))
obtains $i$ where $i<$ length $\left(f_{s t}\right.$ aux $)$ and $f=\left(f_{s t}((f s t ~ a u x)!i)\right.$, fst data $+i$,
snd $(($ fst aux $)!i))$
proof -
let $? h s=$ fst (add-indices aux data)
from assms obtain $i$ where $i<l e n g t h$ ?hs and $f=? h s!i$ by (metis in-set-conv-nth)
from this(1) have $i<$ length (fst aux) by (simp add: add-indices-def)
hence $? h s!i=(f s t((f s t ~ a u x)!i)$, fst data $+i$, snd $((f$ fst aux $)!i))$
unfolding add-indices-def fst-conv by (rule map-idx-nth)
hence $f=\left(f_{s t}((f s t ~ a u x)!i), f s t ~ d a t a+i\right.$, snd $\left.((f s t ~ a u x)!i)\right)$ by (simp add: $\langle f$ $=$ ?hs ! $i\rangle$ )
with $\langle i<$ length (fst aux)〉 show ?thesis ..
qed
definition gb-schema-aux-term1 :: ((('t, 'b::field, ' $c)$ pdata list $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair list) $\times$
(('t, 'b, 'c) pdata list $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair list)) set where gb-schema-aux-term1 $=\left\{\left(a, b::\left(' t,^{\prime} b,{ }^{\prime} c\right)\right.\right.$ pdata list $)$. $(f s t$ 'set $a) \sqsupset p(f s t$ set b) $\}<* l e x *>$

```
(measure (card \circ set))
```

definition gb-schema-aux-term2 ::
$\left({ }^{\prime} a \Rightarrow\right.$ nat $) \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b:::\right.$ field, $\left.{ }^{\prime} c\right)$ pdata list $\Rightarrow\left(\left(\left(^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)\right.\right.$ pdata list $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair list) $\times$
$\left(\left(' t,{ }^{\prime} b, ' c\right)\right.$ pdata list $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair list)) set where gb-schema-aux-term2 d gs $=\{(a, b)$. dgrad-p-set-le d (args-to-set $(g s, a))$ $($ args-to-set $(g s, b)) \wedge$
component-of-term ' Keys (args-to-set $(g s, a)) \subseteq$ component-of-term
‘Keys (args-to-set $(g s, b))\}$
definition gb-schema-aux-term where gb-schema-aux-term dgs=gb-schema-aux-term1 $\cap$ gb-schema-aux-term2 d gs
gb-schema-aux-term is needed for proving termination of function gb-schema-aux.
lemma gb-schema-aux-term1-wf-on:
assumes dickson-grading $d$ and finite $K$
shows wfp-on $(\lambda x y .(x, y) \in g b$-schema-aux-term1)
$\left\{x::\left(\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)\right.\right.$ pdata list $) \times\left(\left(\left({ }^{\prime} t\right.\right.\right.$, 'b:: field, 'c) pdata-pair list $\left.)\right)$.
args-to-set $(g s, x) \subseteq$ dgrad- $p$-set $d m \wedge$ component-of-term'Keys
$(\operatorname{args}-$ to-set $(g s, x)) \subseteq K\}$
proof (rule wfp-onI-min)
let $? B=$ dgrad- $p$-set $d m$
let $? A=\left\{x::\left(\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)\right.\right.$ pdata list $) \times\left(\left(\left({ }^{\prime} t,{ }^{\prime} b,^{\prime} c\right)\right.\right.$ pdata-pair list $\left.)\right)$.
args-to-set $(g s, x) \subseteq ? B \wedge$ component-of-term'Keys (args-to-set (gs,
$x)) \subseteq K\}$
let ? $C=P o w ? B \cap\{F$. component-of-term'Keys $F \subseteq K\}$
have $A$-sub-Pow: (image fst)'set'fst'?A ? $C$
proof
fix $x$
assume $x \in$ (image fst)' set'fst'?A
then obtain $x 1$ where $x 1 \in$ set ' $f s t$ ' ? A and $x: x=f_{s t}{ }^{\prime} x 1$ by auto
from this(1) obtain $x 2$ where $x 2 \in f_{s t}$ '? $A$ and $x 1: x 1=$ set $x 2$ by auto
from this (1) obtain $x 3$ where $x 3 \in ? A$ and $x 2: x 2=$ fst $x 3$ by auto
from this(1) have args-to-set ( $g s, x 3$ ) $\subseteq$ ? $B$ and component-of-term' Keys
$($ args-to-set $(g s, x 3)) \subseteq K$
by simp-all
thus $x \in$ ? $C$ by (simp add: args-to-set-def $x$ x1 x2 image-Un Keys-Un)
qed
fix $x Q$
assume $x \in Q$ and $Q \subseteq ? A$
have $Q$-sub- $A$ : (image fst) 'set 'fst' $Q \subseteq$ (image fst) 'set'fst '?A
by ((rule image-mono) + , fact)
from assms have wfp-on $(\sqsupset p)$ ? $C$ by (rule red-supset-wf-on)
moreover have $f s t$ ' set $(f s t x) \in($ image fst)' set ' $f s t$ ' $Q$
by (rule, fact refl, rule, fact refl, rule, fact refl, simp add: $\langle x \in Q\rangle)$
moreover from $Q$-sub-A $A$-sub-Pow have (image fst)' set' fst' $Q \subseteq$ ?C by
(rule subset-trans)
ultimately obtain $z 1$ where $z 1 \in$ (image $f s t$ )' set ' $f s t$ ' $Q$
and 2: $\bigwedge y . y \sqsupset p z 1 \Longrightarrow y \notin($ image fst) 'set 'fst' $Q$ by (rule wfp-onE-min, auto)
from this(1) obtain $x 1$ where $x 1 \in Q$ and $z 1: z 1=f s t$ 'set ( $f s t x 1$ ) by auto
let $? Q 2=\{q \in Q . f s t ' \operatorname{set}(f s t q)=z 1\}$
have snd $x 1 \in$ snd'? $Q 2$ by (rule, fact refl, simp add: $\langle x 1 \in Q\rangle z 1$ )
with wf-measure obtain $z 2$ where $z 2 \in$ snd'? Q2
and 3: $\bigwedge y .(y, z 2) \in$ measure $($ card $\circ$ set $) \Longrightarrow y \notin$ snd '?Q2
by (rule wfE-min, blast)
from this(1) obtain $z$ where $z \in ? Q 2$ and $z 2: z 2=\operatorname{snd} z .$.
from this(1) have $z \in Q$ and eq1: fst'set $(f s t z)=z 1$ by blast +
from this (1) show $\exists z \in Q . \forall y \in ? A .(y, z) \in$ gb-schema-aux-term1 $\longrightarrow y \notin Q$
proof
show $\forall y \in ? A .(y, z) \in$ gb-schema-aux-term $1 \longrightarrow y \notin Q$
proof (intro ballI impI)
fix $y$
assume $y \in ? A$
assume $(y, z) \in g b$-schema-aux-term1
hence $\left(f_{s t}\right.$ 'set $(f s t y) \sqsupset p z 1 \vee(f s t y=f s t z \wedge($ snd $y, z 2) \in$ measure (card - set)))
by (simp add: gb-schema-aux-term1-def eq1 [symmetric] z2 in-lex-prod-alt)
thus $y \notin Q$
proof (elim disjE conjE)
assume $f s t$ ' set $(f s t y) \sqsupset p z 1$
hence $f s t$ ' set (fst y) $\notin$ (image fst)' set 'fst' $Q$ by (rule 2)
thus ?thesis by auto
next
assume (snd $y, z 2) \in$ measure $($ card $\circ$ set)
hence snd $y \notin$ snd ' ?Q2 by (rule 3)
hence $y \notin$ ?Q2 by blast
moreover assume fst $y=f s t z$
ultimately show ?thesis by (simp add: eq1)
qed
qed
qed
qed
lemma gb-schema-aux-term-wf:
assumes dickson-grading $d$
shows $w f$ (gb-schema-aux-term d gs)
proof (rule wfI-min)
fix $x::\left(\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)\right.$ pdata list $) \times\left(\left({ }^{\prime} t,{ }^{\prime} b,^{\prime} c\right)\right.$ pdata-pair list $)$ and $Q$
assume $x \in Q$
let $? A=\operatorname{args}$-to-set $(g s, x)$
have finite ? A by (simp add: args-to-set-def)
then obtain $m$ where $A: ? A \subseteq d g r a d-p$-set $d m$ by (rule dgrad-p-set-exhaust)
define $K$ where $K=$ component-of-term'Keys ?A
from 〈finite? $A\rangle$ have finite $K$ unfolding $K$-def by (rule finite-imp-finite-component-Keys)

```
    let ?B = dgrad-p-set d m
    let ?Q}={q\inQ.args-to-set (gs,q)\subseteq?B\wedge component-of-term'Keys(args-to-set
(gs,q))\subseteqK}
    from assms <finite K> have wfp-on (\lambdax y. (x,y) \in gb-schema-aux-term1)
                            {x. args-to-set (gs,x)\subseteq?B ^ component-of-term'Keys (args-to-set
(gs,x))\subseteqK}
    by (rule gb-schema-aux-term1-wf-on)
    moreover from }\langlex\inQ>A have x\in?Q by (simp add:K-def
    moreover have ?Q \subseteq{x. args-to-set (gs, x)\subseteq?B ^ component-of-term'Keys
(args-to-set (gs,x))\subseteqK} by auto
    ultimately obtain z where z\in?Q
    and *: \y. (y,z) f gb-schema-aux-term1 \Longrightarrowy &?Q by (rule wfp-onE-min,
blast)
    from this(1) have z & Q and a: args-to-set (gs,z)\subseteq?B and b: compo-
nent-of-term' Keys (args-to-set (gs,z))\subseteqK
    by simp-all
    from this(1) show \existsz\inQ.\forally.(y,z)\ingb-schema-aux-term d gs \longrightarrowy\not\inQ
    proof
        show }\forally.(y,z)\ingb-schema-aux-term d gs \longrightarrowy\not\in
    proof (intro allI impI)
            fix y
            assume (y,z) \ingb-schema-aux-term d gs
            hence (y,z)\ingb-schema-aux-term1 and (y,z)\ingb-schema-aux-term2 d gs
                by (simp-all add: gb-schema-aux-term-def)
            from this(2) have dgrad-p-set-le d (args-to-set (gs, y)) (args-to-set (gs,z))
                    and comp-sub: component-of-term ' Keys (args-to-set (gs,y))\subseteq compo-
nent-of-term' Keys (args-to-set (gs,z))
                by (simp-all add: gb-schema-aux-term2-def)
            from this(1)<args-to-set (gs,z)\subseteq?B〉 have args-to-set (gs,y)\subseteq?B
                by (rule dgrad-p-set-le-dgrad-p-set)
            moreover from comp-sub b have component-of-term' Keys (args-to-set (gs,
y))\subseteqK
                by (rule subset-trans)
            moreover from «(y,z)\ingb-schema-aux-term1> have y &?Q by (rule *)
            ultimately show y}\not\inQ\mathrm{ by simp
        qed
    qed
qed
lemma dgrad-p-set-le-args-to-set-ab:
assumes dickson-grading \(d\) and ap-spec ap and ab-spec \(a b\) and compl-struct compl
assumes sps \(\neq[]\) and set sps \(\subseteq\) set ps and \(h s=\) fst (add-indices (compl gs bs ( \(p s--\) sps) sps data) data)
shows dgrad-p-set-le d (args-to-set (gs, ab gs bs hs data', ap gs bs (ps -- sps) hs data')) (args-to-set (gs, bs, ps))
(is dgrad-p-set-le - ?l ?r)
proof -
have dgrad-p-set-le d?l
```

$(f s t$ ' $($ set $g s \cup$ set $b s \cup f s t$ ' set $(p s--s p s) \cup$ snd' set $(p s--s p s) \cup$ set $h s)$ )
by (rule dgrad-p-set-le-subset, rule args-to-set-subset[OF $\operatorname{assms}(2,3)])$
also have dgrad-p-set-le $d \ldots$ ?. r unfolding image-Un
proof (intro dgrad-p-set-leI-Un)
show dgrad-p-set-le d (fst'set gs) (args-to-set (gs, bs, ps))
by (rule dgrad-p-set-le-subset, auto simp add: args-to-set-def)
next
show dgrad-p-set-le d (fst'set bs) (args-to-set (gs, bs, ps))
by (rule dgrad-p-set-le-subset, auto simp add: args-to-set-def)
next
show dgrad-p-set-le d (fst'fst'set (ps -- sps)) (args-to-set (gs, bs, ps))
by (rule dgrad-p-set-le-subset, auto simp add: args-to-set-def set-diff-list)
next
show dgrad-p-set-le d ( $f s t$ ' snd' set ( $p s--\operatorname{sps}$ )) (args-to-set ( $g s, b s, p s)$ )
by (rule dgrad-p-set-le-subset, auto simp add: args-to-set-def set-diff-list)
next
from $\operatorname{assms}(4,1,5,6)$ show dgrad-p-set-le d (fst' set hs) (args-to-set (gs, bs, $p s)$ )
unfolding $\operatorname{assms}(7)$ fst-set-add-indices by (rule compl-structD1)
qed
finally show ?thesis.
qed
corollary dgrad-p-set-le-args-to-set-struct:
assumes dickson-grading $d$ and struct-spec sel ap ab compl and $p s \neq[]$
assumes $\mathrm{sps}=$ sel gs bs ps data and hs $=$ fst (add-indices (compl gs bs (ps -sps) sps data) data)
shows dgrad-p-set-le d (args-to-set (gs, ab gs bs hs data', ap gs bs (ps -- sps) hs data')) (args-to-set (gs, bs, ps))
proof -
from assms(2) have sel: sel-spec sel and $a p: a p-s p e c ~ a p$ and $a b: a b-s p e c ~ a b$
and compl: compl-struct compl by (rule struct-specD)+
from sel $\operatorname{assms}(3)$ have $s p s \neq[]$ and set sps $\subseteq$ set ps
unfolding assms(4) by (rule sel-specD1, rule sel-specD2)
from assms(1) ap ab compl this assms(5) show?thesis by (rule dgrad-p-set-le-args-to-set-ab)
qed
lemma components-subset-ab:
assumes ap-spec ap and ab-spec $a b$ and compl-struct compl
assumes sps $\neq[]$ and set sps $\subseteq$ set ps and $h s=$ fst (add-indices (compl gs bs ( $p s--$ sps) sps data) data)
shows component-of-term 'Keys (args-to-set (gs, ab gs bs hs data', ap gs bs (ps -- sps $) h s$ data')) $\subseteq$ component-of-term'Keys (args-to-set $(g s, b s, p s))($ is ?l $\subseteq ? r)$
proof -
have ?l $\subseteq$ component-of-term'Keys $(f s t$ ' (set gs $\cup$ set $b s \cup f s t$ 'set ( $p s--$ sps $) \cup$ snd'set $(p s--s p s) \cup$ set $h s))$
by (rule image-mono, rule Keys-mono, rule args-to-set-subset[OF $\operatorname{assms}(1,2)])$
also have $\ldots \subseteq$ ?r unfolding image-Un Keys-Un Un-subset-iff proof (intro conjI)
show component-of-term'Keys (fst'set gs) $\subseteq$ component-of-term' Keys (args-to-set (gs, bs, ps))
by (rule image-mono, rule Keys-mono, auto simp add: args-to-set-def) next
show component-of-term' Keys (fst'set bs) $\subseteq$ component-of-term' Keys (args-to-set (gs,bs,ps))
by (rule image-mono, rule Keys-mono, auto simp add: args-to-set-def)
next
show component-of-term ' Keys $(f s t$ ' $f s t$ ' set $(p s--s p s)) \subseteq$ component-of-term
'Keys (args-to-set ( $g s, b s, p s$ ))
by (rule image-mono, rule Keys-mono, auto simp add: set-diff-list args-to-set-def)
next
show component-of-term' Keys $(f s t$ ' snd'set $(p s--s p s)) \subseteq$ compo-nent-of-term' Keys (args-to-set (gs, bs, ps))
by (rule image-mono, rule Keys-mono, auto simp add: args-to-set-def set-diff-list)
next
from $\operatorname{assms}(3,4,5)$ show component-of-term'Keys $(f s t '$ set hs $) \subseteq$ compo-nent-of-term' Keys (args-to-set ( $g s, b s, p s$ ))
unfolding assms(6) fst-set-add-indices by (rule compl-structD2)
qed
finally show?thesis .
qed
corollary components-subset-struct:
assumes struct-spec sel ap ab compl and ps $\neq[]$
assumes $s p s=$ sel gs bs ps data and $h s=$ fst (add-indices (compl gs bs (ps -sps) sps data) data)
shows component-of-term 'Keys (args-to-set (gs, ab gs bs hs data', ap gs bs (ps -- sps) hs data')) $\subseteq$

```
component-of-term' Keys (args-to-set (gs, bs, ps))
```

proof -
from assms(1) have sel: sel-spec sel and $a p$ : ap-spec $a p$ and $a b: a b$-spec $a b$ and compl: compl-struct compl by (rule struct-specD)+
from sel assms(2) have sps $\neq[]$ and set sps $\subseteq$ set ps unfolding assms(3) by (rule sel-specD1, rule sel-specD2)
from ap ab compl this assms(4) show ?thesis by (rule components-subset-ab) qed
corollary components-struct:
assumes struct-spec sel $a p$ ab compl and $p s \neq[]$ and set $p s \subseteq$ set $b s \times($ set gs $\cup$ set $b s$ )
assumes $s p s=$ sel gs bs ps data and $h s=f s t$ (add-indices (compl gs bs (ps -sps) sps data) data)
shows component-of-term 'Keys (args-to-set (gs, ab gs bs hs data', ap gs bs (ps -- sps) hs data')) =

```
component-of-term'Keys (args-to-set (gs,bs,ps)) (is ?l = ?r)
```

proof

```
    from assms(1, 2, 4, 5) show ?l }\subseteq?r\mathrm{ by (rule components-subset-struct)
next
    from assms(1) have ap: ap-spec ap and ab: ab-spec ab and compl: compl-struct
compl
    by (rule struct-specD)+
    from ap ab assms(3)
    have sub: set (ap gs bs (ps-- sps) hs data')\subseteq set (ab gs bs hs data') }\times(\mathrm{ set gs
Ust (ab gs bs hs data'))
    by (rule subset-Times-ap)
    show ?r }\subseteq\mathrm{ ?l
        by (simp add: args-to-set-subset-Times[OF sub] args-to-set-subset-Times[OF
assms(3)] ab-specD1[OF ab],
            rule image-mono, rule Keys-mono, blast)
qed
lemma struct-spec-red-supset:
    assumes struct-spec sel ap ab compl and ps }\not=[] and sps = sel gs bs ps data
    and hs =fst (add-indices (compl gs bs (ps-- sps) sps data) data) and hs \not=
[]
    shows (fst'set (ab gs bs hs data')) \sqsupsetp(fst'set bs)
proof -
    from assms(5) have set hs }\not={}\mathrm{ by simp
    then obtain h' where h'\in set hs by fastforce
    let ?h=fst h'
    let ?m}=\mathrm{ monomial (lc ?h) (lt ?h)
    from <h'\in set hs> have h-in:?h \in fst' set hs by simp
    hence ?h \in fst ' set (fst (compl gs bs (ps -- sps) sps data))
        by (simp only: assms(4) fst-set-add-indices)
    then obtain }\mp@subsup{h}{}{\prime\prime}\mathrm{ where }\mp@subsup{h}{}{\prime\prime}-in:\mp@subsup{h}{}{\prime\prime}\in\operatorname{set}(fst(compl gs bs (ps-- sps) sps data))
        and ?h = fst h" ..
    from assms(1) have sel: sel-spec sel and ap: ap-spec ap and ab: ab-spec ab
        and compl: compl-struct compl by (rule struct-specD)+
    from sel assms(2) have sps }\not=[]\mathrm{ and set sps }\subseteq\mathrm{ set ps unfolding assms(3)
        by (rule sel-specD1, rule sel-specD2)
    from h-in compl-structD3[OF compl this] have ?h }\not=0\mathrm{ unfolding assms(4)
fst-set-add-indices
    by metis
    show ?thesis
    proof (simp add: ab-specD1[OF ab] image-Un, rule)
    fix q
    assume is-red (fst'set bs)q
    moreover have fst' set bs\subseteqfst' set bs \cupfst' set hs by simp
    ultimately show is-red (fst 'set bs \cupfst' set hs) q by (rule is-red-subset)
    next
    from <?h\not=0` have lc ?h f 0 by (rule lc-not-0)
    moreover have ?h }\in{?h}.
            ultimately have is-red {?h} ?m using〈?h f= 0〉 adds-term-refl by (rule
is-red-monomialI)
    moreover have {?h}\subseteqfst'set bs \cupfst' set hs using h-in by simp
```

```
        ultimately show is-red (fst'set bs Ufst'set hs) ?m by (rule is-red-subset)
        next
        show \neg is-red (fst ' set bs) ?m
    proof
        assume is-red (fst ' set bs) ?m
        then obtain b' where b}\mp@subsup{b}{}{\prime}\infst' set bs and b' =0 and lt b' adds\mp@subsup{s}{t}{}lt ?
        by (rule is-red-monomialE)
    from this(1) obtain b}\mathrm{ where b set bs and }\mp@subsup{b}{}{\prime}:\mp@subsup{b}{}{\prime}=fst b ..
    from this(1) have b\in set gs U set bs by simp
    from }\langle\mp@subsup{b}{}{\prime}\not=0\rangle\mathrm{ have fst b}=0\mathrm{ by (simp add: b}\mathrm{ )
```



```
lt (fst b) addst lt ?h
            unfolding <?h = fst h'l}\mp@subsup{}{}{\prime\prime}>\mathrm{ by (rule compl-structD4)
            from this «lt b' addst lt ?h> show False by (simp add: b')
        qed
    qed
qed
lemma unique-idx-append:
    assumes unique-idx gs data and (hs,data') = add-indices aux data
    shows unique-idx (gs @ hs)data'
proof -
    from assms(2) have hs: hs = fst (add-indices aux data) and data':data' = snd
(add-indices aux data)
    by (metis fst-conv, metis snd-conv)
    have len: length hs = length (fst aux) by (simp add: hs add-indices-def)
    have eq: fst data' = fst data + length hs by (simp add:data' add-indices-def hs)
    show ?thesis
    proof (rule unique-idxI)
        fix fg
        assume f\inset (gs @ hs) and g}\operatorname{get (gs @ hs)
        hence d1:f\in set gs U set hs and d2:g\in set gs U set hs by simp-all
    assume id-eq: fst (snd f) =fst (snd g)
    from d1 show f}=
    proof
            assume f}\in\mathrm{ set gs
            from d2 show ?thesis
            proof
                assume g}\in\mathrm{ set gs
                from assms(1)<f\in set gs> this id-eq show ?thesis by (rule unique-idxD1)
            next
                assume g}\in\mathrm{ set hs
                then obtain j where g=(fst (fst aux!j), fst data + j, snd (fst aux!j))
unfolding hs
                by (rule in-set-add-indicesE)
                hence fst (snd g) = fst data + j by simp
                moreover from assms(1)<f\in set gs〉 have fst (snd f)< fst data
                by (rule unique-idxD2)
                ultimately show ?thesis by (simp add: id-eq)
```

```
        qed
    next
        assume f}\in\mathrm{ set hs
        then obtain i where f:f=(fst (fst aux!i),fst data + i, snd (fst aux!i))
unfolding hs
        by (rule in-set-add-indicesE)
    hence *: fst (snd f) = fst data + i by simp
    from d2 show ?thesis
    proof
        assume g}\in\mathrm{ set gs
        with assms(1) have fst (snd g)< fst data by (rule unique-idxD2)
        with * show ?thesis by (simp add: id-eq)
    next
        assume g}\in\mathrm{ set hs
        then obtain j where g: g=(fst (fst aux ! j), fst data + j, snd (fst aux !
j)) unfolding hs
                by (rule in-set-add-indicesE)
            hence fst (snd g)=fst data + j by simp
        with * have i=j by (simp add: id-eq)
        thus ?thesis by (simp add: fg)
        qed
    qed
    next
        fix f
        assume f\inset (gs @ hs)
        hence f\in set gs U set hs by simp
        thus fst (snd f) < fst data'
        proof
            assume f}\in\mathrm{ set gs
            with assms(1) have fst (snd f)< fst data by (rule unique-idxD2)
            also have ... \leqfst data' by (simp add: eq)
            finally show ?thesis.
    next
            assume f}\in\mathrm{ set hs
            then obtain i where i< length (fst aux)
                and f}=(fst(fst aux ! i), fst data + i, snd (fst aux ! i)) unfolding hs
                by (rule in-set-add-indicesE)
            from this(2) have fst (snd f) = fst data +i by simp
            also from <i< length (fst aux)> have ... < fst data + length (fst aux) by
simp
            finally show ?thesis by (simp only: eq len)
        qed
    qed
qed
corollary unique-idx-ab:
    assumes ab-spec ab and unique-idx (gs @ bs)data and (hs,data')=add-indices
aux data
    shows unique-idx (gs @ ab gs bs hs data') data'
```

```
proof -
    from assms(2, 3) have unique-idx ((gs @ bs) @ hs) data' by (rule unique-idx-append)
    thus ?thesis by (simp add: unique-idx-def ab-specD1[OF assms(1)])
qed
lemma rem-comps-spec-struct:
    assumes struct-spec sel ap ab compl and rem-comps-spec (gs @ bs)data and ps
# []
    and set ps\subseteq(set bs)}\times(\mathrm{ set gs U set bs) and sps = sel gs bs ps (snd data)
    and aux = compl gs bs (ps-- sps) sps (snd data) and (hs,data') =add-indices
aux (snd data)
    shows rem-comps-spec (gs @ ab gs bs hs data}) (fst data - count-const-lt-components
(fst aux), data')
proof -
    from assms(1) have sel: sel-spec sel and ap: ap-spec ap and ab: ab-spec ab and
compl: compl-struct compl
    by (rule struct-specD)+
    from ap ab assms(4)
    have sub: set (ap gs bs (ps-- sps) hs data')\subseteq set (ab gs bs hs data') }\times(\mathrm{ set gs
Ust (ab gs bs hs data'))
    by (rule subset-Times-ap)
    have hs: hs = fst (add-indices aux (snd data)) by (simp add: assms(7)[symmetric])
    from sel assms(3) have sps \not=[] and set sps \subseteqset ps unfolding assms(5)
        by (rule sel-specD1, rule sel-specD2)
    have eq0:fst'set (fst aux) - {0} = fst'set (fst aux)
        by (rule Diff-triv, simp add: Int-insert-right assms(6), rule compl-structD3,
fact+)
    have component-of-term 'Keys (fst'set (gs @ ab gs bs hs data'))=
                            component-of-term'Keys (args-to-set (gs, ab gs bs hs data', ap gs bs (ps
-- sps) hs data'))
    by (simp add: args-to-set-subset-Times[OF sub] image-Un)
    also from assms(1, 3, 4, 5) hs
    have ... = component-of-term ' Keys (args-to-set (gs, bs, ps)) unfolding assms(6)
        by (rule components-struct)
    also have ... = component-of-term 'Keys (fst 'set (gs @ bs))
        by (simp add: args-to-set-subset-Times[OF assms(4)] image-Un)
    finally have eq:component-of-term'Keys (fst'set (gs@ ab gs bs hs data'))=
                            component-of-term'Keys (fst'set (gs @ bs)).
    from assms(2)
    have eq2:card (component-of-term'Keys (fst'set (gs @ bs))) =
                fst data + card (const-lt-component'(fst'set (gs @ bs) - {0}) -
{None})(is ?a = - + ?b)
        by (simp only: rem-comps-spec-def)
    have eq3:card (const-lt-component'(fst'set (gs @ ab gs bs hs data') - {0}) -
{None})=
                        ?b}+\mathrm{ count-const-lt-components (fst aux) (is ?c = -)
        proof (simp add: ab-specD1[OF ab] image-Un Un-assoc[symmetric] Un-Diff
count-const-lt-components-alt
        hs fst-set-add-indices eq0, rule card-Un-disjoint)
```

show finite（const－lt－component＇$(f s t$＇set gs $-\{0\})-\{$ None $\} \cup($ const－lt－component ＇$(f s t$＇set bs $-\{0\})-\{$ None $\}))$
by（intro finite－UnI finite－Diff finite－imageI finite－set）
next
show finite（const－lt－component＇fst＇set（fst aux）－\｛None $\}$ ）
by（rule finite－Diff，intro finite－imageI，fact finite－set）
next
have（const－lt－component＇$(f s t$＇$($ set $g s \cup$ set bs $)-\{0\})-\{N o n e\}) \cap$
$($ const－lt－component＇fst＇set $($ fst aux $)-\{$ None $\})=$
（const－lt－component＇$(f s t$＇$($ set gs $\cup$ set bs $)-\{0\}) \cap$
const－lt－component＇$f s t$＇set（fst aux））$-\{$ None $\}$ by blast
also have $\ldots=\{ \}$
proof（simp，rule，simp，elim conjE）
fix $k$
assume $k \in$ const－lt－component＇（fst＇（set gs $\cup$ set bs）$-\{0\})$
then obtain $b$ where $b \in$ set $g s \cup$ set bs and fst $b \neq 0$ and $k 1: k=$ const－lt－component（fst b）
by blast
assume $k \in$ const－lt－component＇fst＇set（fst aux）
then obtain $h$ where $h \in \operatorname{set}(f s t ~ a u x)$ and $k 2: k=$ const－lt－component（ $f s t$
$h)$ by blast
show $k=$ None
proof（rule ccontr，simp，elim exE）
fix $k^{\prime}$
assume $k=$ Some $k^{\prime}$
hence $l p(f s t b)=0$ and component－of－term $(l t(f s t b))=k^{\prime}$ unfolding $k 1$
by（rule const－lt－component－SomeD1，rule const－lt－component－SomeD2）
moreover from $\langle k=S o m e ~ k\rangle$ have $l p(f s t h)=0$ and component－of－term $(l t(f s t h))=k^{\prime}$
unfolding $k 2$ by（rule const－lt－component－SomeD1，rule const－lt－component－SomeD2）
ultimately have $l t(f s t b)$ addst $l t(f s t h)$ by（simp add：adds－term－def）
moreover from compl $\langle s p s \neq[]\rangle\langle$ set sps $\subseteq$ set ps〉〈h $\in$ set $(f s t$ aux $)\rangle\langle b$ $\in$ set gs $\cup$ set bs〉〈fst $b \neq 0\rangle$
have $\neg l t(f s t b) a d d s_{t} l t(f s t h)$ unfolding assms（6）by（rule compl－structD4）
ultimately show False by simp
qed
qed
finally show（const－lt－component＇（fst＇set gs $-\{0\})-\{$ None $\} \cup$（const－lt－component
＇$($ fst＇set bs $-\{0\})-\{$ None $\})) \cap$
（const－lt－component＇fst＇set（fst aux）$-\{$ None $\})=\{ \}$ by（simp only：
Un－Diff image－Un）
qed
have $? c \leq$ ？a unfolding eq［symmetric］
by（rule card－const－lt－component－le，rule finite－imageI，fact finite－set）
hence le：count－const－lt－components（fst aux）$\leq$ fst data by（simp only：eq2 eq3）
show ？thesis by（simp only：rem－comps－spec－def eq eq2 eq3，simp add：le）
qed
lemma pmdl－struct：

```
    assumes struct-spec sel ap ab compl and compl-pmdl compl and is-Groebner-basis
(fst'set gs)
    and ps\not=[] and set ps\subseteq(set bs)\times(set gs \cup set bs) and unique-idx (gs @ bs)
(snd data)
    and sps = sel gs bs ps (snd data) and aux = compl gs bs (ps -- sps) sps (snd
data)
        and (hs,data') = add-indices aux (snd data)
    shows pmdl (fst`set (gs @ ab gs bs hs data'))=pmdl (fst`set (gs @ bs))
proof -
    have hs: hs = fst (add-indices aux (snd data)) by (simp add: assms(9)[symmetric])
    from assms(1) have sel: sel-spec sel and ab: ab-spec ab by (rule struct-specD)+
    have eq: fst'(set gs \cup set (ab gs bs hs data')) = fst' (set gs U set bs) \cupfst' set
hs
        by (auto simp add: ab-specD1[OF ab])
    show ?thesis
    proof (simp add: eq, rule)
        show pmdl (fst ' (set gs \cup set bs) \cupfst' set hs)\subseteqpmdl (fst '(set gs U set bs))
        proof (rule pmdl.span-subset-spanI, simp only: Un-subset-iff, rule)
            show fst'(set gs U set bs)\subseteqpmdl (fst '(set gs U set bs))
                by (fact pmdl.span-superset)
        next
            from sel assms(4) have sps }\not=[] \mathrm{ and set sps }\subseteq\mathrm{ set ps
                unfolding assms(7) by (rule sel-specD1, rule sel-specD2)
            with assms(2, 3) have fst'set hs \subseteqpmdl (args-to-set (gs,bs,ps))
            unfolding hs assms(8) fst-set-add-indices using assms(6) by (rule compl-pmdlD)
            thus fst' set hs\subseteqpmdl (fst'(set gs \cup set bs))
                by (simp only: args-to-set-subset-Times[OF assms(5)] image-Un)
        qed
    next
        show pmdl (fst' (set gs \cup set bs))\subseteqpmdl (fst '(set gs U set bs) Ufst'set hs)
            by (rule pmdl.span-mono, blast)
    qed
qed
```

lemma discarded-subset:
assumes $a b$-spec $a b$
and $D^{\prime}=D \cup\left(\right.$ set $h s \times($ set gs $\cup$ set $b s \cup$ set $h s) \cup$ set $(p s--s p s)-{ }_{p}$ set (ap gs bs (ps -- sps) hs data'))
and set $p s \subseteq$ set $b s \times($ set $g s \cup$ set $b s)$ and $D \subseteq($ set $g s \cup$ set bs $) \times($ set $g s \cup$ set bs)
shows $D^{\prime} \subseteq($ set $g s \cup$ set $(a b$ gs bs hs data' $)) \times($ set gs $\cup$ set $(a b$ gs bs hs data' $))$ proof -
from assms(1) have eq: set ( $a b$ gs bs hs data') = set $b s \cup$ set hs by (rule ab-specD1)
from $\operatorname{assms}(4)$ have $D \subseteq($ set $g s \cup($ set $b s \cup$ set $h s)) \times($ set $g s \cup($ set $b s \cup$ set $h s)$ ) by fastforce
moreover have set $h s \times($ set gs $\cup$ set $b s \cup$ set hs $) \cup$ set $(p s--s p s)-{ }_{p}$ set (ap gs bs (ps -- sps) hs data') $\subseteq$
$($ set $g s \cup($ set $b s \cup$ set $h s)) \times($ set $g s \cup($ set $b s \cup$ set $h s))($ is $? l \subseteq$ ? $r)$
proof (rule subset-trans)
show $? l \subseteq$ set $h s \times($ set $g s \cup$ set $b s \cup$ set $h s) \cup$ set $(p s--s p s)$
by (simp add: minus-pairs-def)
next
have set $h s \times($ set gs $\cup$ set bs $\cup$ set $h s) \subseteq$ ?r by fastforce
moreover have set $(p s--s p s) \subseteq$ ?r
proof (rule subset-trans)
show set ( $p s--s p s$ ) $\subseteq$ set ps by (auto simp: set-diff-list)
next
from assms(3) show set $p s \subseteq$ ?r by fastforce
qed
ultimately show set $h s \times($ set $g s \cup$ set $b s \cup$ set $h s) \cup$ set $(p s--s p s) \subseteq$ ?r by (rule Un-least)
qed
ultimately show ?thesis unfolding eq assms(2) by (rule Un-least)
qed
lemma compl-struct-disjoint:
assumes compl-struct compl and sps $\neq[]$ and set sps $\subseteq$ set $p s$
shows $f s t$ ' set ( $f s t($ compl gs bs (ps -- sps) sps data $)) \cap$ fst' (set gs $\cup$ set bs)
$=\{ \}$
proof (rule, rule)
fix $x$
assume $x \in f s t$ 'set $(f s t($ compl gs bs (ps -- sps) sps data) $) \cap f s t$ ' (set gs $\cup$ set bs)
hence $x$-in: $x \in f s t$ ' set ( $f s t$ (compl gs bs (ps-- sps) sps data)) and $x \in f s t$ '
( set gs $\cup$ set bs)
by simp-all
 and $x 1: x=f s t h .$.
from compl-structD3[OF assms, of gs bs data] $x$-in have $x \neq 0$ by auto
from $\langle x \in f s t$ ' (set gs $\cup$ set $b s)\rangle$ obtain $b$ where $b$-in: $b \in$ set $g s \cup$ set $b s$ and $x 2: x=f s t b .$.
from $\langle x \neq 0\rangle$ have $f s t b \neq 0$ by (simp add: x2)
with assms $h$-in $b$-in have $\neg l t(f s t b)$ adds $s_{t} l t(f s t h)$ by (rule compl-structD4)
hence $\neg$ lt $x$ addst $l t x$ by (simp add: $x 1$ [symmetric] x2)
from this adds-term-refl show $x \in\}$..
qed $\operatorname{simp}$

## context

fixes sel::('t, 'b::field, 'c::default, 'd) selT and ap::('t, 'b, 'c, 'd) apT and $a b::\left(' t, ' b,{ }^{\prime} c,{ }^{\prime} d\right) a b T$ and compl::('t, 'b, 'c, 'd) complT and $g s::\left(' t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list
begin
function (domintros) gb-schema-dummy :: nat $\times n a t \times{ }^{\prime} d \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair set $\Rightarrow$

$$
\left(' t,{ }^{\prime} b,,^{\prime} c\right) p d a t a \text { list } \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,^{\prime} c\right) \text { pdata-pair list } \Rightarrow
$$

```
                (('t, 'b, 'c) pdata list }\times(\mp@subsup{}{}{\prime}t,\mp@subsup{,}{}{\prime}b,'c) pdata-pair set
    where
    gb-schema-dummy data D bs ps=
        (if ps = [] then
        (gs @bs,D)
        else
            (let sps = sel gs bs ps (snd data);ps0 = ps -- sps;aux = compl gs bs
ps0 sps (snd data);
            remcomps = fst (data) - count-const-lt-components (fst aux) in
            (if remcomps =0 then
                    (full-gb (gs @ bs),D)
                    else
                    let (hs,data') = add-indices aux (snd data) in
                        gb-schema-dummy (remcomps,data')
                            (D\cup((set hs \times (set gs \cup set bs U set hs ) \cup set (ps -- sps)) - 
set (ap gs bs ps0 hs data')))
                (ab gs bs hs data') (ap gs bs ps0 hs data')
            )
        )
        )
    by pat-completeness auto
lemma gb-schema-dummy-domI1:gb-schema-dummy-dom (data, D, bs, [])
    by (rule gb-schema-dummy.domintros, simp)
lemma gb-schema-dummy-domI2:
    assumes struct-spec sel ap ab compl
    shows gb-schema-dummy-dom (data, D, args)
proof -
    from assms have sel: sel-spec sel and ap: ap-spec ap and ab:ab-spec ab by (rule
struct-specD)+
    from ex-dgrad obtain d::'a m nat where dg: dickson-grading d ..
    let ?R = (gb-schema-aux-term d gs)
    from dg have wf ?R by (rule gb-schema-aux-term-wf)
    thus ?thesis
    proof (induct args arbitrary: data D rule: wf-induct-rule)
    fix x data D
    assume IH: \bigwedgey data' D'. (y,x) \in?R \Longrightarrowgb-schema-dummy-dom (data', D',
y)
    obtain bs ps where x: x=(bs,ps) by (meson case-prodE case-prodI2)
    show gb-schema-dummy-dom (data, D, x) unfolding x
    proof (rule gb-schema-dummy.domintros)
            fix rc0 n0 data0 hs n1 data1
            assume ps \not= []
                and hs-data': (hs,n1, data1) = add-indices (compl gs bs (ps -- sel gs bs
ps (n0, data0))
                                    (sel gs bs ps (n0, data0)) (n0, data0)) (n0,
data0)
```

            and data: data \(=(r c 0, n 0, d a t a 0)\)
    define sps where sps $=$ sel gs bs ps (n0, data0)
define $d_{a t a}{ }^{\prime}$ where $d a t a^{\prime}=(n 1$, data1 $)$
define $D^{\prime}$ where $D^{\prime}=D \cup$
$\left(\right.$ set $h s \times($ set $g s \cup$ set $b s \cup$ set $h s) \cup$ set $(p s--s p s)-{ }_{p}$
set (ap gs bs (ps-- sps) hs data'))
define $r c$ where $r c=r c 0$ - count-const-lt-components (fst (compl gs bs (ps -- sel gs bs ps (n0, data0))
(sel gs bs ps (n0, data0)) (n0,
data0)))
from $h s$-data' have $h s: h s=f s t$ (add-indices (compl gs bs (ps -- sps) sps (snd data)) (snd data))
unfolding sps-def data snd-conv by (metis fstI)
show gb-schema-dummy-dom ((rc, data $), D^{\prime}$, ab gs bs hs data' , ap gs bs (ps -- sps) hs data')
proof (rule IH, simp add: x gb-schema-aux-term-def gb-schema-aux-term1-def gb-schema-aux-term2-def, intro conjI)
show fst ' set (ab gs bs hs data') $\sqsupset p$ fst ' set bs $\vee$ ab gs bs hs data' $=$ bs $\wedge$ card $($ set (ap gs bs $(p s--$ sps $) h s d a t a \prime))<$ card (set ps)
proof (cases hs = [])
case True
have $a b$ gs bs hs data' $=b s \wedge$ card $\left(\right.$ set (ap gs bs $\left.\left.(p s--s p s) h s d a t a{ }^{\prime}\right)\right)$
$<\operatorname{card}$ (set ps)
proof (simp only: True, rule)
from $a b$ show $a b$ gs $b s[] d a t a^{\prime}=b s$ by (rule $\left.a b-s p e c D 2\right)$
next
from sel $\langle p s \neq[]\rangle$ have $s p s \neq[]$ and set $s p s \subseteq$ set $p s$
unfolding sps-def by (rule sel-specD1, rule sel-specD2)
moreover from sel-specD1 $[$ OF sel $\langle p s \neq[]\rangle]$ have set sps $\neq\{ \}$ by (simp add: sps-def)
ultimately have set $p s \cap$ set sps $\neq\{ \}$ by (simp add: inf.absorb-iff2)
hence set $(p s--s p s) \subset$ set ps unfolding set-diff-list by fastforce
hence $\operatorname{card}($ set $(p s--s p s))<\operatorname{card}($ set ps) by (simp add: psub-set-card-mono)
moreover have card (set (ap gs bs (ps-- sps) [] data')) $\leq \operatorname{card}$ (set ( $p s--s p s)$ )
by (rule card-mono, fact finite-set, rule ap-spec-Nil-subset, fact ap)
ultimately show card (set (ap gs bs (ps--sps) [] data')) < card (set $p s)$ by $\operatorname{simp}$
qed
thus ?thesis ..
next
case False
with assms $\langle p s \neq[]\rangle$ sps-def hs have $f s t$ ‘set (ab gs bs hs data') $\sqsupset p$ fst‘ set bs
unfolding data snd-conv by (rule struct-spec-red-supset)
thus ?thesis..
qed
next

```
            from dg assms <ps \not= []> sps-def hs
            show dgrad-p-set-le d (args-to-set (gs, ab gs bs hs data', ap gs bs (ps --
sps) hs data')) (args-to-set (gs, bs, ps))
            unfolding data snd-conv by (rule dgrad-p-set-le-args-to-set-struct)
            next
            from assms <ps \not= []> sps-def hs
            show component-of-term' Keys (args-to-set (gs,ab gs bs hs data', ap gs bs
(ps -- sps) hs data'))\subseteq
                component-of-term'Keys (args-to-set (gs, bs, ps))
            unfolding data snd-conv by (rule components-subset-struct)
        qed
    qed
    qed
qed
```

lemmas gb-schema-dummy-simp $=$ gb-schema-dummy.psimps[OF gb-schema-dummy-domI2]
lemma gb-schema-dummy-Nil [simp]: gb-schema-dummy data D bs []$=(g s$ @ bs, D)
by (simp add: gb-schema-dummy.psimps[OF gb-schema-dummy-domI1])
lemma gb-schema-dummy-not-Nil:
assumes struct-spec sel ap ab compl and ps $\neq[]$
shows gb-schema-dummy data $D$ bs ps $=$
(let sps $=$ sel gs bs ps (snd data) ; ps0 $=$ ps -- sps; aux $=$ compl gs bs
ps0 sps (snd data);
remcomps $=$ fst $($ data $)-$ count-const-lt-components $($ fst aux $)$ in (if remcomps $=0$ then
(full-gb (gs @ bs), D)
else
let $\left(h s, d^{\prime} a^{\prime}\right)=$ add-indices aux (snd data) in gb-schema-dummy (remcomps, data')
$(D \cup(($ set $h s \times($ set $g s \cup$ set $b s \cup$ set $h s) \cup$ set $(p s--s p s))-p$
set (ap gs bs ps0 hs data')))
(ab gs bs hs data') (ap gs bs ps0 hs data')
)
)
by (simp add: gb-schema-dummy-simp[OF assms(1)] assms(2))
lemma gb-schema-dummy-induct [consumes 1, case-names base rec1 rec2]:
assumes struct-spec sel ap ab compl
assumes base: $\bigwedge$ bs data D. P data D bs [] (gs @ bs, D)
and rec1: $\bigwedge b s$ ps sps data $D . p s \neq[] \Longrightarrow$ sps $=$ sel gs bs ps (snd data) $\Longrightarrow$

$$
f s t(\text { data }) \leq \text { count-const-lt-components }(\text { fst }(\text { compl gs bs }(p s--s p s)
$$

sps $($ snd data $))) \Longrightarrow$
$P$ data $D$ bs ps (full-gb (gs @ bs), D)
and rec2: $\bigwedge$ bs ps sps aux hs rc data data ${ }^{\prime} D D^{\prime} . p s \neq[] \Longrightarrow$ sps $=$ sel gs bs ps (snd data) $\Longrightarrow$

$$
a u x=\text { compl gs bs }(p s-- \text { sps }) \text { sps }(\text { snd data }) \Longrightarrow\left(h s, d a t a^{\prime}\right)=
$$

```
add-indices aux (snd data) \Longrightarrow
    rc= fst data - count-const-lt-components (fst aux) \Longrightarrow0<rc\Longrightarrow
    D'}=(D\cup((set hs \times (set gs \cup set bs \cup set hs )\cup set (ps -- sps)) 
- p set (ap gs bs (ps -- sps) hs data'})))
    P (rc,data') D' (ab gs bs hs data) (ap gs bs (ps -- sps) hs data')
    (gb-schema-dummy (rc, data') D' (ab gs bs hs data') (ap gs bs (ps
-- sps) hs data')) \Longrightarrow
                    P data D bs ps (gb-schema-dummy (rc, data') D'(ab gs bs hs data')
(ap gs bs (ps -- sps) hs data'))
    shows P data D bs ps (gb-schema-dummy data D bs ps)
proof -
    from assms(1) have gb-schema-dummy-dom (data, D, bs, ps) by (rule gb-schema-dummy-domI2)
    thus ?thesis
    proof (induct data D bs ps rule: gb-schema-dummy.pinduct)
        case (1 data D bs ps)
        show ?case
        proof (cases ps=[])
            case True
            show ?thesis by (simp add: True, rule base)
    next
                case False
                show ?thesis
        proof (simp only:gb-schema-dummy-not-Nil[OF assms(1) False] Let-def split:
if-split, intro conjI impI)
            define sps where sps=sel gs bs ps (snd data)
            assume fst data - count-const-lt-components (fst (compl gs bs (ps -- sps)
sps (snd data))) = 0
            hence fst data \leq count-const-lt-components (fst (compl gs bs (ps -- sps)
sps (snd data)))
                    by simp
                    with False sps-def show P data D bs ps (full-gb (gs @ bs),D) by (rule
rec1)
    next
            define sps where sps=sel gs bs ps (snd data)
            define aux where aux = compl gs bs (ps-- sps) sps (snd data)
            define hs where hs = fst (add-indices aux (snd data))
            define data' where data' = snd (add-indices aux (snd data))
            define rc where rc = fst data - count-const-lt-components (fst aux)
            define }\mp@subsup{D}{}{\prime}\mathrm{ where }\mp@subsup{D}{}{\prime}=(D\cup((set hs \times (set gs \cup set bs U set hs ) U set (p
-- sps)) - 
            have eq: add-indices aux (snd data) = (hs, data') by (simp add: hs-def
data'-def)
            assume rc\not=0
            hence 0<rc by simp
            show P data D bs ps
                (case add-indices aux (snd data) of
                (hs, data') }
                    gb-schema-dummy (rc,data')
                    (D\cup(set hs \times (set gs U set bs U set hs ) U set (ps-- sps) - -p set (ap
```

```
gs bs (ps -- sps) hs data')))
                    (ab gs bs hs data') (ap gs bs (ps -- sps) hs data'))
            unfolding eq prod.case D'-def[symmetric] using False sps-def aux-def
eq[symmetric] rc-def <0 < rc> D'-def
            proof (rule rec2)
            show P (rc,data') D' (ab gs bs hs data') (ap gs bs (ps-- sps) hs data')
                    (gb-schema-dummy (rc, data') D' (ab gs bs hs data') (ap gs bs (ps
-- sps) hs data'))
                    unfolding D'-def using False sps-def refl aux-def rc-def \langlerc \not=0\rangle
eq[symmetric] refl
                by (rule 1)
            qed
        qed
    qed
    qed
qed
lemma fst-gb-schema-dummy-dgrad-p-set-le:
    assumes dickson-grading d and struct-spec sel ap ab compl
    shows dgrad-p-set-le d (fst'set (fst (gb-schema-dummy data D bs ps))) (args-to-set
(gs, bs, ps))
    using assms(2)
proof (induct rule: gb-schema-dummy-induct)
    case (base bs data D)
    show ?case by (simp add: args-to-set-def, rule dgrad-p-set-le-subset, fact sub-
set-refl)
next
    case (rec1 bs ps sps data D)
    show ?case
    proof (cases fst' set gs Ufst'set bs\subseteq{0})
        case True
        hence Keys (fst 'set (gs @ bs))={} by (auto simp add: image-Un Keys-def)
        hence component-of-term'Keys (fst'set (full-gb (gs @ bs)))={}
            by (simp add: components-full-gb)
    hence Keys (fst'set (full-gb (gs @ bs)))={} by simp
    thus ?thesis by (simp add: dgrad-p-set-le-def dgrad-set-le-def)
    next
    case False
    from pps-full-gb have dgrad-set-le d (pp-of-term'Keys (fst'set (full-gb (gs @
bs)))) {0}
        by (rule dgrad-set-le-subset)
    also have dgrad-set-le d ... (pp-of-term'Keys (args-to-set (gs, bs, ps)))
    proof (rule dgrad-set-leI, simp)
        from False have Keys (args-to-set (gs, bs, ps)) \not={}
            by (simp add: args-to-set-alt Keys-Un, metis Keys-not-empty singletonI
subsetI)
        then obtain v}\mathrm{ where v}\mathrm{ ve Keys (args-to-set (gs,bs,ps)) by blast
                moreover have d 0 \leq d (pp-of-term v) by (simp add: assms(1) dick-
son-grading-adds-imp-le)
```

```
            ultimately show \existst\inKeys(args-to-set (gs,bs,ps)).d 0 \leqd (pp-of-term t)
..
    qed
    finally show ?thesis by (simp add: dgrad-p-set-le-def)
    qed
next
    case (rec2 bs ps sps aux hs rc data data' D D')
    from rec2(4) have hs = fst (add-indices (compl gs bs (ps -- sps) sps (snd
data)) (snd data))
    unfolding rec2(3) by (metis fstI)
    with assms rec2(1, 2)
    have dgrad-p-set-le d (args-to-set (gs, ab gs bs hs data', ap gs bs (ps -- sps) hs
data')) (args-to-set (gs, bs, ps))
    by (rule dgrad-p-set-le-args-to-set-struct)
    with rec2(8) show ?case by (rule dgrad-p-set-le-trans)
qed
lemma fst-gb-schema-dummy-components:
    assumes struct-spec sel ap ab compl and set ps\subseteq(set bs) }\times(\mathrm{ set gs U set bs)
    shows component-of-term 'Keys (fst'set (fst (gb-schema-dummy data D bs ps)))
=
    component-of-term` Keys (args-to-set (gs, bs, ps))
    using assms
proof (induct rule: gb-schema-dummy-induct)
    case (base bs data D)
    show ?case by (simp add: args-to-set-def)
next
    case (rec1 bs ps sps data D)
    have component-of-term 'Keys (fst 'set (full-gb (gs @ bs))) =
        component-of-term'Keys (fst'set (gs@ bs)) by (fact components-full-gb)
    also have ... = component-of-term'Keys (args-to-set (gs, bs, ps))
        by (simp add: args-to-set-subset-Times[OF rec1.prems] image-Un)
    finally show ?case by simp
next
    case (rec2 bs ps sps aux hs rc data data' D D')
    from assms(1) have ap: ap-spec ap and ab: ab-spec ab by (rule struct-specD)+
    from this rec2.prems
    have sub: set (ap gs bs (ps -- sps) hs data')\subseteq set (ab gs bs hs data') }\times(\mathrm{ set gs
 set (ab gs bs hs data'))
            by (rule subset-Times-ap)
    from recZ(4) have hs:hs=fst (add-indices (compl gs bs (ps-- sps) sps (snd
data)) (snd data))
            unfolding rec2(3) by (metis fstI)
    have component-of-term 'Keys (args-to-set (gs, ab gs bs hs data', ap gs bs (ps
-- sps) hs data')) =
            component-of-term'Keys (args-to-set (gs,bs,ps))(is ?l = ?r)
    proof
    from assms(1) recZ(1, 2) hs show ?l }\subseteq\mathrm{ ?r by (rule components-subset-struct)
    next
```

```
    show ?r }\subseteq\mathrm{ ?l
    by (simp add: args-to-set-subset-Times[OF rec2.prems] args-to-set-alt2[OF ap
ab rec2.prems] image-Un,
    rule image-mono, rule Keys-mono, blast)
    qed
    with rec2.hyps(8)[OF sub] show ?case by (rule trans)
qed
lemma fst-gb-schema-dummy-pmdl:
    assumes struct-spec sel ap ab compl and compl-pmdl compl and is-Groebner-basis
(fst'set gs)
    and set ps\subseteq set bs \times (set gs U set bs) and unique-idx (gs @ bs) (snd data)
    and rem-comps-spec (gs @ bs)data
    shows pmdl (fst'set (fst (gb-schema-dummy data D bs ps))) = pmdl (fst'set
(gs @ bs))
proof -
    from assms(1) have sel: sel-spec sel and ap:ap-spec ap and ab:ab-spec ab and
compl: compl-struct compl
    by (rule struct-specD)+
    from assms(1, 4, 5, 6) show ?thesis
    proof (induct bs ps rule: gb-schema-dummy-induct)
        case (base bs data D)
        show ?case by simp
    next
        case (rec1 bs ps sps data D)
        define aux where aux = compl gs bs (ps -- sps) sps (snd data)
    define data' where data' = snd (add-indices aux (snd data))
    define hs where hs = fst (add-indices aux (snd data))
    have hs-data': (hs, data') = add-indices aux (snd data) by (simp add: hs-def
data'-def)
    have eq: set (gs @ ab gs bs hs data) = set (gs @ bs @ hs) by (simp add:
ab-specD1[OF ab])
    from sel rec1(1) have sps }\not=[]\mathrm{ and set sps }\subseteq\mathrm{ set ps unfolding rec1(2)
        by (rule sel-specD1, rule sel-specD2)
    from full-gb-is-full-pmdl have pmdl (fst'set (full-gb (gs @ bs))) = pmdl (fst
` set (gs @ ab gs bs hs data'))
    proof (rule is-full-pmdl-eq)
            show is-full-pmdl (fst'set (gs @ ab gs bs hs data'))
            proof (rule is-full-pmdlI-lt-finite)
                from finite-set show finite (fst'set (gs @ ab gs bs hs data')) by (rule
finite-imageI)
            next
                fix }
                assume k \in component-of-term 'Keys (fst'set (gs @ ab gs bs hs data'))
                hence Some k \inSome'component-of-term'Keys (fst'set (gs @ ab gs bs
hs data')) by simp
                also have ... = const-lt-component '(fst'set (gs @ ab gs bs hs data') -
{0})-{None} (is ?A = ?B)
    proof (rule card-seteq[symmetric])
```

show finite ? A by (intro finite-imageI finite-Keys, fact finite-set)

## next

have rem-comps-spec ( $g s @ a b$ gs bs hs data') (fst data - count-const-lt-components (fst aux), data')
using assms(1) rec1.prems(3) rec1.hyps(1) rec1.prems(1) rec1.hyps(2) aux-def hs-data'
by (rule rem-comps-spec-struct)
also have $\ldots=\left(0\right.$, data $\left.^{\prime}\right)$ by (simp add: aux-def rec1.hyps(3))
finally have card (const-lt-component ' $(f s t$ 'set ( $g s$ @ ab gs bs hs data') $-\{0\})-\{$ None $\})=$ card (component-of-term'Keys (fst'set (gs @ ab gs bs hs data' $)$ ))
by (simp add: rem-comps-spec-def)
also have $\ldots=\operatorname{card}$ (Some 'component-of-term'Keys (fst'set (gs @ ab gs bs hs data')))
by (rule card-image[symmetric], simp)
finally show card ? $A \leq$ card ? $B$ by simp
qed (fact const-lt-component-subset)
finally have Some $k \in$ const-lt-component' (fst'set (gs @ab gs bs hs $\left.\left.d a t a^{\prime}\right)-\{0\}\right)$
by simp
then obtain $b$ where $b \in f s t$ ' set ( $g s$ @ ab gs bs hs data') and $b \neq 0$
and $*$ : const-lt-component $b=$ Some $k$ by fastforce
show $\exists b \in f s t$ 'set (gs @ab gs bs hs data'). $b \neq 0 \wedge$ component-of-term (lt b) $=k \wedge l p b=0$
proof (intro bexI conjI)
from $*$ show component-of-term (lt b) $=k$ by (rule const-lt-component-SomeD2)
next
from $*$ show $l p b=0$ by (rule const-lt-component-SomeD1)
qed fact +
qed
next
from compl $\langle s p s \neq[]\rangle\langle s e t ~ s p s \subseteq$ set $p s\rangle$
have component-of-term' Keys (fst' set hs) $\subseteq$ component-of-term' Keys (args-to-set ( $g s, b s, p s)$ )
unfolding hs-def aux-def fst-set-add-indices by (rule compl-structD2)
hence sub: component-of-term 'Keys (fst' set hs) $\subseteq$ component-of-term' Keys (fst'set (gs @ bs))
by (simp add: args-to-set-subset-Times[OF rec1 .prems(1)] image-Un)
have component-of-term'Keys (fst'set (full-gb (gs @ bs))) = component-of-term 'Keys (fst'set (gs @ bs)) by (fact components-full-gb)
also have $\ldots=$ component-of-term 'Keys (fst'set ((gs @ bs) @ hs ))
by (simp only: set-append $[$ of - hs] image-Un Keys-Un Un-absorb2 sub)
finally show component-of-term'Keys $(f s t$ ' set $(f u l l-g b(g s @ b s)))=$
component-of-term 'Keys (fst'set (gs @ ab gs bs hs data'))
by (simp only: eq append-assoc)
qed
also have $\ldots=p m d l(f s t$ ' set ( $g s$ @ bs))
using $\operatorname{assms}(1,2,3) \operatorname{rec1} . \operatorname{hyps(1)} \operatorname{rec} 1 . \operatorname{prems}(1,2) \operatorname{rec} 1 . h y p s(2) a u x-d e f$

```
hs-data'
            by (rule pmdl-struct)
    finally show?case by simp
    next
    case (rec2 bs ps sps aux hs rc data data' D D')
    from rec2(4) have hs: hs = fst (add-indices aux (snd data)) by (metis fstI)
    have pmdl (fst'set (fst (gb-schema-dummy (rc, data') D' (ab gs bs hs data')
(ap gs bs (ps -- sps) hs data')))) =
                pmdl (fst'set (gs @ ab gs bs hs data'))
    proof (rule rec2.hyps(8))
            from ap ab rec2.prems(1)
            show set (ap gs bs (ps-- sps) hs data')\subseteq set (ab gs bs hs data') }\times(\mathrm{ set gs
set (ab gs bs hs data'))
            by (rule subset-Times-ap)
    next
            from ab rec2.prems(2) rec2(4) show unique-idx (gs @ ab gs bs hs data') (snd
(rc, data'))
            unfolding snd-conv by (rule unique-idx-ab)
    next
    show rem-comps-spec (gs @ ab gs bs hs data') (rc,data') unfolding rec2.hyps(5)
                using assms(1) rec2.prems(3) rec2.hyps(1) rec2.prems(1) rec2.hyps(2, 3,
4)
            by (rule rem-comps-spec-struct)
    qed
    also have ... = pmdl (fst'set (gs @ bs))
        using assms(1, 2, 3) rec2.hyps(1) rec2.prems(1, 2) rec2.hyps(2, 3, 4) by
(rule pmdl-struct)
    finally show ?case.
    qed
qed
lemma snd-gb-schema-dummy-subset:
    assumes struct-spec sel ap ab compl and set ps\subseteq set bs \times (set gs \cup set bs)
    and D\subseteq(set gs U set bs)}\times(\mathrm{ set gs U set bs) and res = gb-schema-dummy
data D bs ps
    shows snd res \subseteqset (fst res) }\times\mathrm{ set (fst res )}\vee(\existsxs.fst (res)=full-gb xs
    using assms
proof (induct data D bs ps rule: gb-schema-dummy-induct)
    case (base bs data D)
    from base(2) show ?case by (simp add: base(3))
next
    case (rec1 bs ps sps data D)
    have \existsxs.fst res = full-gb xs by (auto simp: rec1(6))
    thus ?case ..
next
    case (rec2 bs ps sps aux hs rc data data' D D')
    from assms(1) have ab:ab-spec ab and ap:ap-spec ap by (rule struct-specD)+
    from - - rec2.prems(3) show ?case
    proof (rule rec2.hyps(8))
```

```
    from ap ab rec2.prems(1)
    show set (ap gs bs (ps -- sps) hs data')\subseteq set (ab gs bs hs data') }\times(\mathrm{ set gs }
set (ab gs bs hs data'))
        by (rule subset-Times-ap)
    next
    from ab rec2.hyps(7) rec2.prems(1) rec2.prems(2)
    show }\mp@subsup{D}{}{\prime}\subseteq(\mathrm{ set gs U set (ab gs bs hs data})) \times(set gs \cup set (ab gs bs hs data'))
        by (rule discarded-subset)
    qed
qed
lemma gb-schema-dummy-connectible1:
    assumes struct-spec sel ap ab compl and compl-conn compl and dickson-grading
d
    and fst' set gs \subseteqdgrad-p-set d m and is-Groebner-basis (fst' set gs)
    and fst' set bs \subseteqdgrad-p-set d m
    and set ps\subseteq set bs }\times(\mathrm{ set gs }\cup\mathrm{ set bs)
    and unique-idx (gs @ bs) (snd data)
    and }\bigwedgepq.processed (p,q)(gs@ @s) ps\Longrightarrow(p,q)\not\inpD\Longrightarrowfst p\not=0\Longrightarrowfs
q\not=0\Longrightarrow
            crit-pair-cbelow-on d m (fst ' (set gs U set bs)) (fst p) (fst q)
    and }\neg(\existsxs.fst (gb-schema-dummy data D bs ps) = full-gb xs
    assumes f}\in\operatorname{set (fst (gb-schema-dummy data D bs ps))
    and g\in set (fst (gb-schema-dummy data D bs ps))
    and (f,g)\not\inp snd (gb-schema-dummy data D bs ps)
    and fst f}\not=0\mathrm{ and fst g}\not=
    shows crit-pair-cbelow-on d m (fst'set (fst (gb-schema-dummy data D bs ps)))
(fst f) (fst g)
    using assms(1, 6, 7, 8, 9, 10, 11, 12, 13)
proof (induct data D bs ps rule: gb-schema-dummy-induct)
    case (base bs data D)
    show ?case
    proof (cases f Get gs)
    case True
    show ?thesis
    proof (cases g set gs)
        case True
        note assms(3, 4, 5)
        moreover from <f \in set gs\rangle have fst f\infst' set gs by simp
        moreover from }\langleg\in\mathrm{ set gs` have fst g}\infst' set gs by sim
        ultimately have crit-pair-cbelow-on d m (fst'set gs) (fst f) (fst g)
            using assms(14, 15) by (rule GB-imp-crit-pair-cbelow-dgrad-p-set)
        moreover have fst'set gs \subseteqfst'set (fst (gs @ bs,D)) by auto
        ultimately show ?thesis by (rule crit-pair-cbelow-mono)
    next
        case False
            from this base(6, 7) have processed (g, f) (gs @ bs) [] by (simp add:
processed-Nil)
        moreover from base.prems(8) have (g,f)\not\inp D by (simp add: in-pair-iff)
```

ultimately have crit-pair-cbelow-on d m(fst'set (gs @ bs)) (fst g) (fst f)
using $\langle f s t g \neq 0\rangle\langle f s t f \neq 0\rangle$ unfolding set-append by (rule base(4))
thus ?thesis unfolding fst-conv by (rule crit-pair-cbelow-sym)
qed
next
case False
from this base $(6,7)$ have processed $(f, g)(g s @ b s)[]$ by (simp add: pro-cessed-Nil)
moreover from base.prems (8) have $(f, g) \not \notin p D$ by simp
ultimately show ?thesis unfolding fst-conv set-append using $\langle f s t f \neq 0\rangle\langle f s t$ $g \neq 0>$ by (rule base(4))
qed
next
case (rec1 bs ps sps data D)
from rec1.prems(5) show ?case by auto
next
case (rec2 bs ps sps aux hs rc data data' $D D^{\prime}$ )
from rec2.hyps(4) have $h s$ : $h s=$ fst (add-indices aux (snd data)) by (metis fstI)
from $\operatorname{assms}(1)$ have sel: sel-spec sel and $a p: a p$-spec $a p$ and $a b: a b$-spec $a b$ and compl: compl-struct compl
by (rule struct-specD1, rule struct-specD2, rule struct-specD3, rule struct-specD4)
from sel rec2.hyps(1) have sps $\neq[]$ and set sps $\subseteq$ set ps
unfolding rec2.hyps(2) by (rule sel-specD1, rule sel-specD2)
from ap ab rec2.prems(2) have ap-sub: set (ap gs bs (ps--sps) hs data') $\subseteq$ set $(a b$ gs bs hs data') $) \times($ set gs $\cup$ set $(a b$ gs bs hs data'))
by (rule subset-Times-ap)
have $n s$-sub: $f s t$ ' set $h s \subseteq d g r a d-p$-set $d m$
proof (rule dgrad-p-set-le-dgrad-p-set)
from compl $\operatorname{assms}(3)\langle s p s \neq[]\rangle\langle s e t$ sps $\subseteq$ set $p s\rangle$
show dgrad-p-set-le d (fst' set hs) (args-to-set (gs, bs, ps))
unfolding hs rec2.hyps(3) fst-set-add-indices by (rule compl-structD1)
next
from $\operatorname{assms}(4)$ rec2.prems (1) show args-to-set (gs, bs, ps) $\subseteq$ dgrad-p-set d m by (simp add: args-to-set-subset-Times[OF rec2.prems(2)])
qed
with rec2.prems(1) have ab-sub: fst'set (ab gs bs hs data') $\subseteq$ dgrad-p-set d $m$ by (auto simp add: ab-specD1[OF ab])
have cpq: $(p, q) \in_{p}$ set sps $\Longrightarrow$ fst $p \neq 0 \Longrightarrow$ fst $q \neq 0 \Longrightarrow$
crit-pair-cbelow-on d $m\left(f_{s t}\right.$ ' (set gs $\cup$ set $(a b$ gs bs hs data' $\left.\left.)\right)\right)(f s t p)$
$(f s t q)$ for $p q$
proof -
assume $(p, q) \in_{p}$ set sps and fst $p \neq 0$ and $f$ st $q \neq 0$
from this $(1)$ have $(p, q) \in$ set sps $\vee(q, p) \in$ set sps by (simp only: in-pair-iff)
hence crit-pair-cbelow-on d $m(f s t$ ' (set gs $\cup$ set bs) $\cup f s t$ ' set (fst (compl gs
$b s(p s--s p s) s p s(s n d$ data $))))$
(fst p) (fst q)
proof

```
    assume (p,q)\in set sps
    from assms(2, 3, 4, 5) rec2.prems(1, 2) <sps \not= []><set sps \subseteq set ps`
rec2.prems(3) this
    fst p}\not=0\rangle\langlefst q\not=0\rangle\mathrm{ show ?thesis by (rule compl-connD)
    next
            assume (q, p)\in set sps
            from assms(2, 3, 4, 5) rec2.prems(1, 2) <sps }\not=[]\<set sps \subseteq set ps
rec2.prems(3) this
            <st q}=0\rangle\langlefst p\not=0
            have crit-pair-cbelow-on d m (fst '(set gs U set bs)\cup fst' set (fst (compl gs
        bs (ps -- sps) sps (snd data))))
            (fst q) (fst p) by (rule compl-connD)
            thus ?thesis by (rule crit-pair-cbelow-sym)
    qed
    thus crit-pair-cbelow-on d m (fst' (set gs U set (ab gs bs hs data'))) (fst p) (fst
q)
            by (simp add: ab-specD1[OF ab] hs rec2.hyps(3) fst-set-add-indices image-Un
Un-assoc)
    qed
    from ab-sub ap-sub - - rec2.prems(5, 6, 7, 8) show ?case
    proof (rule rec2.hyps(8))
    from ab rec2.prems(3) rec2(4) show unique-idx (gs @ ab gs bs hs data') (snd
(rc, data`))
            unfolding snd-conv by (rule unique-idx-ab)
    next
    fix p q :: ('t,'b,'c) pdata
    define ps' where ps' = ap gs bs (ps-- sps) hs data'
    assume fst p\not=0 and fst q\not=0 and (p,q)\not\inp D'
    assume processed (p,q) (gs @ ab gs bs hs data') ps'
    hence p-in: p set gs \cup set bs \cup set hs and q-in:q\in set gs U set bs U set hs
            and (p,q)\not\inp set ps' by (simp-all add: processed-alt ab-specD1[OF ab])
    from this(3)<(p,q)\not\inp D'〉 have (p,q)\not\inp D and ( }p,q)\not\inp set (ps -- sps
        and (p,q)\not\inp set hs \times (set gs U set bs \cup set hs)
        by (auto simp: in-pair-iff rec2.hyps(7) ps'-def)
    from this(3) p-in q-in have p\in set gs \cup set bs and q\in set gs \cup set bs
            by (meson SigmaI UnE in-pair-iff)+
    show crit-pair-cbelow-on d m(fst'(set gs U set (ab gs bs hs data'))) (fst p)
(fst q)
    proof (cases component-of-term (lt (fst p)) = component-of-term (lt (fst q)))
            case True
            show ?thesis
            proof (cases (p,q)\in \in set sps)
            case True
            from this <fst p\not=0\rangle\langlefst q\not=0\rangle show ?thesis by (rule cpq)
            next
                case False
                with}<(p,q)\not\inp\mathrm{ set (ps -- sps)> have ( }p,q)\not\inp\mathrm{ set ps
                by (auto simp: in-pair-iff set-diff-list)
```

```
    with}\langlep\in\mathrm{ set gs U set bs><qG set gs U set bs> have processed ( }p,q\mathrm{ ) (gs @
bs) ps
            by (simp add: processed-alt)
            from this «(p,q)\not\inp D>\langlefst p\not=0\rangle\langlefst q\not=0\rangle
            have crit-pair-cbelow-on d m (fst'( set gs U set bs)) (fst p) (fst q)
            by (rule rec2.prems(4))
            moreover have fst'(set gs U set bs)\subseteqfst'(set gs U set (ab gs bs hs
data'))
                    by (auto simp: ab-specD1[OF ab])
            ultimately show ?thesis by (rule crit-pair-cbelow-mono)
            qed
    next
            case False
            thus ?thesis by (rule crit-pair-cbelow-distinct-component)
    qed
    qed
qed
lemma gb-schema-dummy-connectible2:
assumes struct-spec sel ap ab compl and compl-conn compl and dickson-grading \(d\)
    and fst ' set gs \subseteqdgrad-p-set d m and is-Groebner-basis (fst' set gs)
    and fst' set bs\subseteqdgrad-p-set d m
    and set ps\subseteq set bs \times (set gs U set bs) and D\subseteq(set gs U set bs) }\times(\mathrm{ set gs U
set bs)
    and set ps }\mp@subsup{\cap}{p}{}D={}\mathrm{ and unique-idx (gs @ bs) (snd data)
    and }\bigwedgeBab. set gs \cup set bs\subseteqB\Longrightarrowfst' B\subseteqdgrad-p-set d m \Longrightarrow (a,b)\in
D\Longrightarrow
fst }a\not=0\Longrightarrow\mathrm{ fst }b\not=0
```



```
                        fst }x\not=0\Longrightarrow\mathrm{ fst }y\not=0\Longrightarrow\mathrm{ crit-pair-cbelow-on d m (fst'B) (fst x)
(fst y))\Longrightarrow
            crit-pair-cbelow-on d m (fst ` B) (fst a) (fst b)
    and }\bigwedgexy.x\in\operatorname{set}(fst (gb-schema-dummy data D bs ps)) \Longrightarrowy\inset (fs
(gb-schema-dummy data D bs ps))\Longrightarrow
            (x,y)\not\inp snd (gb-schema-dummy data D bs ps)\Longrightarrowfst }x\not=0\Longrightarrowfst
\not=0\Longrightarrow
    crit-pair-cbelow-on d m (fst'set (fst (gb-schema-dummy data D bs ps)))
(fst x) (fst y)
    and }\neg(\existsxs.fst (gb-schema-dummy data D bs ps)= full-gb xs
    assumes (f,g)\inp snd (gb-schema-dummy data D bs ps)
    and fst f}\not=0\mathrm{ and fst g}=
    shows crit-pair-cbelow-on d m (fst'set (fst (gb-schema-dummy data D bs ps)))
(fst f) (fst g)
    using assms(1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)
proof (induct data D bs ps rule: gb-schema-dummy-induct)
    case (base bs data D)
    have set gs \cup set bs\subseteq set (fst (gs @ bs,D)) by simp
    moreover from assms(4) base.prems(1) have fst'set (fst (gs @ bs,D))\subseteq
```

```
dgrad-p-set d m by auto
    moreover from base.prems(9) have (f,g) \inp D by simp
    moreover note assms(15, 16)
    ultimately show ?case
    proof (rule base.prems(6))
        fix }x
    assume x\in set gs U set bs and y set gs U set bs and (x,y)\not\inp D
    hence }x\in\operatorname{set}(fst(gs@bs,D))\mathrm{ and }y\in\operatorname{set}(fst (gs @bs,D)) and (x,y)\not\in
snd (gs @ bs,D)
            by simp-all
    moreover assume fst x\not=0 and fst y}\not=
    ultimately show crit-pair-cbelow-on d m (fst'set (fst (gs @ bs,D))) (fst x)
(fst y)
            by (rule base.prems(7))
    qed
next
    case (rec1 bs ps sps data D)
    from rec1.prems(8) show ?case by auto
next
    case (rec2 bs ps sps aux hs rc data data' D D')
    from rec2.hyps(4) have hs: hs = fst (add-indices aux (snd data)) by (metis fstI)
    from assms(1) have sel: sel-spec sel and ap: ap-spec ap and ab: ab-spec ab
        and compl: compl-struct compl by (rule struct-specD)+
    let ?X = set (ps-- sps) U set hs \times (set gs U set bs U set hs )
    from sel rec2.hyps(1) have sps }\not=[]\mathrm{ and set sps }\subseteq\mathrm{ set ps
    unfolding rec2.hyps(2) by (rule sel-specD1, rule sel-specD2)
    have fst' set hs \cap fst'(set gs U set bs)={}
        unfolding hs fst-set-add-indices rec2.hyps(3) using compl «sps \not= []〉<set sps
\subseteq \mp@code { s e t ~ p s > }
    by (rule compl-struct-disjoint)
    hence disj1:(set gs \cup set bs) \cap set hs = {} by fastforce
    have disj2: set (ap gs bs (ps-- sps) hs data') \cap }\mp@subsup{\cap}{p}{\prime}\mp@subsup{D}{}{\prime}={
    proof (rule, rule)
    fix }x
    assume (x,y)\in set (ap gs bs (ps-- sps)hs data') \cap 焐'
        hence (x,y)\in \in set (ap gs bs (ps -- sps) hs data') \cap }\mp@subsup{\cap}{p}{}\mp@subsup{D}{}{\prime}\mathrm{ by (simp add:
in-pair-alt)
    hence 1: (x,y)\inp set (ap gs bs (ps-- sps)hs data') and (x,y) \inp D' by
simp-all
    hence (x,y) \inp D by (simp add: rec2.hyps(7))
    from this rec2.prems(3) have }x\in\mathrm{ set gs }\cup\mathrm{ set bs and y set gs U set bs
        by (auto simp: in-pair-iff)
    from 1 ap-specD1[OF ap] have (x,y)\in \inp ?X by (rule in-pair-trans)
    thus (x,y)\in{} unfolding in-pair-Un
    proof
```

```
        assume (x,y)\inp set (ps-- sps)
        also have ...\subseteq set ps by (auto simp: set-diff-list)
        finally have (x,y)\inf set ps }\mp@subsup{\cap}{p}{}D\mathrm{ using < (x,y) Єp D> by simp
        also have ... ={} by (fact rec2.prems(4))
        finally show ?thesis by (simp add: in-pair-iff)
    next
        assume (x,y)\inp set hs \times (set gs U set bs U set hs)
        hence }x\in\mathrm{ set hs }\veey\in\mathrm{ set hs by (auto simp: in-pair-iff)
        thus ?thesis
        proof
            assume x f set hs
            with }\langlex\in\mathrm{ set gs U set bs> have x ( set gs U set bs) }\cap\mathrm{ set hs ..
            thus ?thesis by (simp add: disj1)
    next
        assume y f set hs
        with }\langley\in\mathrm{ set gs U set bs` have y ( set gs U set bs) \ set hs ..
        thus ?thesis by (simp add: disj1)
    qed
    qed
qed simp
have hs-sub: fst ' set hs \subseteqdgrad-p-set d m
proof (rule dgrad-p-set-le-dgrad-p-set)
    from compl assms(3)\langlesps \not=[]〉\langleset sps\subseteq set ps〉
    show dgrad-p-set-le d (fst'set hs) (args-to-set (gs, bs, ps))
    unfolding hs rec2.hyps(3) fst-set-add-indices by (rule compl-structD1)
next
    from assms(4) rec2.prems(1) show args-to-set (gs, bs, ps)\subseteqdgrad-p-set d m
        by (simp add: args-to-set-subset-Times[OF rec2.prems(2)])
qed
with rec2.prems(1) have ab-sub: fst' set (ab gs bs hs data')\subseteq dgrad-p-set d m
    by (auto simp add: ab-specD1[OF ab])
moreover from ap ab rec2.prems(2)
have ap-sub: set (ap gs bs (ps -- sps) hs data')\subseteq set (ab gs bs hs data') }\times(\mathrm{ set
gs \cup set (ab gs bs hs data'))
    by (rule subset-Times-ap)
    moreover from ab rec2.hyps(7) rec2.prems(2) rec2.prems(3)
    have D'\subseteq(set gs U set (ab gs bs hs data'))}\times(\mathrm{ set gs U set (ab gs bs hs data'))
    by (rule discarded-subset)
moreover note disj2
    moreover from ab rec2.prems(5) rec2.hyps(4) have uid:unique-idx (gs @ ab
gs bs hs data') (snd (rc, data'))
    unfolding snd-conv by (rule unique-idx-ab)
ultimately show ?case using - - rec2.prems(8, 9, 10, 11)
```

```
    proof (rule rec2.hyps(8), simp only: ab-specD1[OF ab] Un-assoc[symmetric])
```

    define \(p s^{\prime}\) where \(p s^{\prime}=a p\) gs \(b s(p s--s p s) h s d a t a{ }^{\prime}\)
    fix \(B a b\)
    assume \(B\)-sup: set \(g s \cup\) set \(b s \cup\) set \(h s \subseteq B\)
    hence set \(g s \cup\) set \(b s \subseteq B\) and set \(h s \subseteq B\) by simp-all
    assume \((a, b) \in_{p} D^{\prime}\)
    hence ab-cases: \((a, b) \in_{p} D \vee(a, b) \in_{p}\) set \(h s \times(\) set \(g s \cup\) set \(b s \cup\) set \(h s)-{ }_{p}\)
    set $p s^{\prime} \vee$
$(a, b) \in_{p}$ set $(p s--s p s)-_{p}$ set $p s^{\prime}$ by (auto simp: rec2.hyps(7)
$\left.p s^{\prime}-d e f\right)$
assume $B$-sub: $f s t{ }^{\prime} B \subseteq d g r a d-p$-set $d m$ and $f s t a \neq 0$ and $f s t b \neq 0$
assume $*: \bigwedge x y . x \in$ set $g s \cup$ set $b s \cup$ set $h s \Longrightarrow y \in$ set $g s \cup$ set $b s \cup$ set hs
$\Longrightarrow$
$(x, y) \not \oiint_{p} D^{\prime} \Longrightarrow f s t x \neq 0 \Longrightarrow$ fst $y \neq 0 \Longrightarrow$
crit-pair-cbelow-on d $m\left(\right.$ fst $\left.^{\prime} B\right)($ sst $x)($ fst $y)$
from rec2.prems(2) have ps-sps-sub: set $(p s--s p s) \subseteq$ set bs $\times($ set gs $\cup$ set $b s)$
by (auto simp: set-diff-list)
from uid have uid': unique-idx (gs @ bs @ hs) data' by (simp add: unique-idx-def $a b-s p e c D 1[O F a b])$
have a: crit-pair-cbelow-on $d m(f s t$ ' $B)(f s t x)(f s t y)$
if $f s t x \neq 0$ and $f$ st $y \neq 0$ and $x y$-in: $(x, y) \in_{p}$ set $(p s--s p s)-{ }_{p}$ set $p s^{\prime}$ for $x y$
proof (cases $x=y$ )
case True
from $x y$-in rec2.prems(2) have $y \in$ set gs $\cup$ set bs
unfolding in-pair-minus-pairs unfolding True in-pair-iff set-diff-list by auto
hence $f s t y \in f s t$ ' set $g s \cup f s t$ ' set bs by fastforce
from this assms(4) rec2.prems(1) have fst $y \in d g r a d-p-s e t ~ d m$ by blast
with assms(3) show ?thesis unfolding True by (rule crit-pair-cbelow-same)
next
case False
from ap assms(3) B-sup B-sub ps-sps-sub disj1 uid' assms(5) False $\langle f s t ~ x \neq$ $0\rangle\langle f s t y \neq 0\rangle x y$-in
show ?thesis unfolding $p s^{\prime}$-def
proof (rule ap-specD3)
fix $a 1$ b1 :: ( $\left.' t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata
assume fst a1 $\neq 0$ and $f$ st $b 1 \neq 0$
assume $a 1 \in$ set $h s$ and b1-in: b1 $\in$ set gs $\cup$ set $b s \cup$ set $h s$
hence a1-in: a1 $\in$ set $g s \cup$ set $b s \cup$ set hs by fastforce
assume (a1, b1) $\in_{p}$ set (ap gs bs ( $p s--s p s$ ) hs data')
hence $(a 1, b 1) \in_{p}$ set $p s^{\prime}$ by (simp only: $p s^{\prime}$-def)
with disj2 have $(a 1, b 1) \not \oiint_{p} D^{\prime}$ unfolding $p s^{\prime}$-def
by (metis empty-iff in-pair-Int-pairs in-pair-alt)
with a1-in b1-in show crit-pair-cbelow-on d $m(f s t ' B)(f s t ~ a 1)(f s t ~ b 1)$ using $\langle f s t$ a1 $\neq 0\rangle\langle f s t$ b1 $\neq 0\rangle$ by (rule *)

```
        qed
    qed
    have b: crit-pair-cbelow-on d m (fst' B) (fst x) (fst y)
        if (x,y)\in\mp@subsup{\in}{p}{}D\mathrm{ and fst }x\not=0\mathrm{ and fst }y\not=0\mathrm{ for }xy
        using <set gs \cup set bs\subseteqB>B-sub that
    proof (rule rec2.prems(6))
    fix a1 b1 :: ('t, 'b, 'c) pdata
    assume a1 \in set gs \cup set bs and b1 \in set gs \cup set bs
    hence a1-in:a1 \in set gs U set bs U set hs and b1-in: b1 \in set gs U set bs U
set hs
            by fastforce+
    assume (a1, b1) \not\inp D and fst a1 }\not=0\mathrm{ and fst b1 =0
    show crit-pair-cbelow-on d m (fst'B) (fst a1) (fst b1)
    proof (cases (a1,b1) \in 的 ? 
            case True
            moreover from <a1 \in set gs \cup set bs\rangle\langleb1 \in set gs \cup set bs\rangle disj1
            have (a1,b1)\not\inp set hs \times (set gs \cup set bs \cup set hs)
            by (auto simp: in-pair-def)
            ultimately have (a1,b1) \inp set (ps-- sps) - p set ps' by auto
            with <fst a1 \not=0\rangle\langlefst b1 \not=0\rangle show ?thesis by (rule a)
    next
            case False
            with «(a1, b1) \not\inp D` have (a1, b1) \not\inp D' by (auto simp: rec2.hyps(7)
ps'-def)
            with a1-in b1-in show ?thesis using <fst a1 # 0\rangle\langlefst b1 \not=0\rangle by (rule *)
        qed
    qed
    have c: crit-pair-cbelow-on d m (fst' B) (fst x) (fst y)
        if x-in:x set gs U set bs U set hs and y-in:y set gs U set bs U set hs
        and xy:(x,y)\not\inp (?X - 
    proof (cases (x,y)\inp D)
        case True
        thus ?thesis using <fst x}\not=0\rangle\langlefst y\not=0\rangle by (rule b
    next
        case False
        with xy have ( }x,y)\not\inp \mp@subsup{D}{}{\prime}\mathrm{ unfolding rec2.hyps(7) ps'-def by auto
        with x-in y-in show ?thesis using <fst x\not=0\rangle\langlefst y}\not=0\rangle\mathrm{ by (rule *)
    qed
    from ab-cases show crit-pair-cbelow-on d m (fst ' B) (fst a) (fst b)
    proof (elim disjE)
        assume (a,b) \inp}
        thus ?thesis using <fst a\not=0\rangle\langlefst b\not=0\rangle by (rule b)
    next
        assume ab-in: (a,b)\in_ set hs }\times(\mathrm{ set gs U set bs U set hs) - 
        hence ab-in':(a,b) \inp set hs \times (set gs \cup set bs \cup set hs) and (a,b)\not\inp set
ps' by simp-all
```

```
    show ?thesis
    proof (cases a =b)
    case True
        from ab-in' rec2.prems(2) have b\in set hs unfolding True in-pair-iff
set-diff-list by auto
    hence fst b \infst' set hs by fastforce
    from this hs-sub have fst b\indgrad-p-set d m ..
    with assms(3) show ?thesis unfolding True by (rule crit-pair-cbelow-same)
    next
    case False
    from ap assms(3) B-sup B-sub ab-in' ps-sps-sub uid' assms(5) False <fst a
# 0〉<fst b = 0>
    show ?thesis
    proof (rule ap-specD2)
        fix x y :: ('t,'b, 'c) pdata
        assume (x,y)\inp set (ap gs bs (ps-- sps) hs data')
        also from ap-sub have .. \subseteq(set bs \cup set hs )}\times(\mathrm{ set gs U set bs }\cup\mathrm{ set hs)
            by (simp only: ab-specD1[OF ab] Un-assoc)
        also have ...\subseteq(set gs U set bs U set hs )}\times(\mathrm{ set gs U set bs U set hs) by
fastforce
    finally have }(x,y)\in(\mathrm{ set gs U set bs U set hs )}\times(\mathrm{ set gs U set bs U set hs )
        unfolding in-pair-same.
        hence }x\in\mathrm{ set gs U set bs U set hs and y set gs U set bs U set hs by
simp-all
    moreover from <(x,y)\in 的 set (ap gs bs (ps-- sps) hs data')> have (x,
y) }\not\inp\mp@code{?X - 
            by (simp add: ps'-def)
            moreover assume fst x\not=0 and fst y}\not=
            ultimately show crit-pair-cbelow-on d m (fst' B) (fst x) (fst y) by (rule
c)
    next
        fix x y :: ('t, 'b,'c) pdata
        assume fst x\not=0 and fst y}\not=
        assume 1:x\in set gs \cup set bs and 2:y set gs \cup set bs
        hence x-in: x set gs U set bs U set hs and y-in: y f set gs U set bs U
set hs by simp-all
    show crit-pair-cbelow-on d m (fst` B) (fst x) (fst y)
    proof (cases (x,y) \inp set (ps-- sps) - 
            case True
            with <fst x\not=0\rangle\langlefst y}\not=0\mathrm{ \ show ?thesis by (rule a)
        next
        case False
        have }(x,y)\not\inp\mathrm{ set (ps-- sps) U set hs }\times(\mathrm{ set gs U set bs U set hs ) -p
set ps'
        proof
        assume (x,y)\inp set (ps-- sps) \cup set hs \times (set gs U set bs U set hs)
-p set ps'
        hence (x,y)\inp set hs }\times(\mathrm{ set gs }\cup\mathrm{ set bs }\cup\mathrm{ set hs) using False
            by simp
```

```
                hence x\in set hs \veey\in set hs by (auto simp: in-pair-iff)
                    with 12 disj1 show False by blast
                    qed
                with x-in y-in show ?thesis using <fst x}\not=0\rangle\langlefst y\not=0\rangle by (rule c
                qed
            qed
        qed
    next
        assume (a,b) \in 的 set (ps-- sps) - p set ps'
        with <fst a\not=0\rangle\langlefst b\not=0\rangle show ?thesis by (rule a)
    qed
    next
    fix }xy::('t,'b,'c) pdat
    let ?res = gb-schema-dummy (rc, data') D' (ab gs bs hs data') (ap gs bs (ps
-- sps) hs data')
    assume x\in set (fst?res) and y set (fst ?res) and (x,y)\not\inp snd ?res and
fst }x\not=0\mathrm{ and fst y}=
            thus crit-pair-cbelow-on d m (fst' set (fst ?res)) (fst x) (fst y) by (rule
rec2.prems(7))
    qed
qed
corollary gb-schema-dummy-connectible:
    assumes struct-spec sel ap ab compl and compl-conn compl and dickson-grading
d
    and fst' set gs \subseteqdgrad-p-set d m and is-Groebner-basis (fst' set gs)
    and fst'set bs \subseteqdgrad-p-set d m
    and set ps\subseteq set bs \times (set gs U set bs) and D\subseteq(set gs U set bs ) }\times(\mathrm{ set gs U
set bs)
    and set ps }\mp@subsup{\cap}{p}{}D={}\mathrm{ and unique-idx (gs @ bs) (snd data)
    and }\bigwedgepq. processed (p,q)(gs@ bs) ps\Longrightarrow(p,q)\not\inpD\Longrightarrowfst p\not=0\Longrightarrowfs
q\not=0\Longrightarrow
                            crit-pair-cbelow-on d m (fst ' (set gs U set bs)) (fst p) (fst q)
    and }\bigwedgeBab. set gs \cup set bs\subseteqB\Longrightarrowfst' B\subseteqdgrad-p-set d m\Longrightarrow (a,b)\in
D\Longrightarrow
                    fst }a\not=0\Longrightarrow\mathrm{ fst }b\not=0
```



```
                        fst }x\not=0\Longrightarrow\mathrm{ fst }y\not=0\Longrightarrow\mathrm{ crit-pair-cbelow-on d m (fst ' B) (fst x)
(fst y)) \Longrightarrow
    crit-pair-cbelow-on d m (fst' B) (fst a) (fst b)
    assumes f}\in\operatorname{set}(fst (gb-schema-dummy data D bs ps))
    and g\in set (fst (gb-schema-dummy data D bs ps))
    and fst f}=0\mathrm{ and fst g}\not=
    shows crit-pair-cbelow-on d m (fst'set (fst (gb-schema-dummy data D bs ps)))
(fst f) (fst g)
proof (cases \existsxs.fst (gb-schema-dummy data D bs ps)= full-gb xs)
    case True
    then obtain xs where xs: fst (gb-schema-dummy data D bs ps)=full-gb xs ..
    note assms(3)
```

```
    moreover have fst ' set (full-gb xs)\subseteq dgrad-p-set d m
    proof (rule dgrad-p-set-le-dgrad-p-set)
    have dgrad-p-set-le d (fst'set (full-gb xs)) (args-to-set (gs, bs,ps))
    unfolding xs[symmetric] using assms(3, 1) by (rule fst-gb-schema-dummy-dgrad-p-set-le)
    also from assms(7) have ... = fst'set gs \cupfst'set bs by (rule args-to-set-subset-Times)
    finally show dgrad-p-set-le d (fst' set (full-gb xs)) (fst' set gs }\cupfst' set bs)
next
    from assms(4,6) show fst'set gs Ufst'set bs \subseteqdgrad-p-set d m by blast
    qed
    moreover note full-gb-isGB
    moreover from assms(13) have fst f\infst'set (full-gb xs) by (simp add: xs)
    moreover from assms(14) have fst g\infst'set (full-gb xs) by (simp add: xs)
    ultimately show ?thesis using assms(15,16) unfolding xs
    by (rule GB-imp-crit-pair-cbelow-dgrad-p-set)
next
    case not-full: False
    show ?thesis
    proof (cases (f,g)\inf snd (gb-schema-dummy data D bs ps))
    case True
    from assms(1-10,12) - not-full True assms(15,16) show ?thesis
    proof (rule gb-schema-dummy-connectible2)
        fix }x
        assume x \in set (fst (gb-schema-dummy data D bs ps))
            and}y\in\operatorname{set}(fst(gb-schema-dummy data D bs ps))
            and (x,y)\not\inp snd (gb-schema-dummy data D bs ps)
            and fst }x\not=0\mathrm{ and fst y}\not=
        with assms(1-7,10,11) not-full
        show crit-pair-cbelow-on d m (fst ' set (fst (gb-schema-dummy data D bs ps)))
(fst x)(fst y)
            by (rule gb-schema-dummy-connectible1)
        qed
    next
        case False
    from assms(1-7,10,11) not-full assms(13,14) False assms(15,16) show ?thesis
        by (rule gb-schema-dummy-connectible1)
    qed
qed
lemma fst-gb-schema-dummy-dgrad-p-set-le-init:
    assumes dickson-grading d and struct-spec sel ap ab compl
    shows dgrad-p-set-le d (fst'set (fst (gb-schema-dummy data D (ab gs [] bs (snd
data)) (ap gs [] [] bs (snd data)))))
    (fst '(set gs U set bs))
proof -
    let ?bs=ab gs [] bs (snd data)
    from assms(2) have ap: ap-spec ap and ab: ab-spec ab by (rule struct-specD)+
    from ap-specD1[OF ap, of gs [] [] bs]
    have *: set (ap gs [] [] bs (snd data))\subseteq set ?bs \times (set gs \cup set ?bs)
        by (simp add: ab-specD1[OF ab])
```

from assms have dgrad-p-set-le d (fst' set (fst (gb-schema-dummy data D ?bs (ap gs [] [] bs (snd data)) )) )
(args-to-set (gs, ?bs, (ap gs [] [] bs (snd data))))
by (rule fst-gb-schema-dummy-dgrad-p-set-le)
also have $\ldots=f s t$ ' (set $g s \cup$ set $b s)$
by (simp add: args-to-set-subset-Times[OF *] image-Un ab-specD1 [OF ab])
finally show ?thesis .
qed
corollary fst-gb-schema-dummy-dgrad-p-set-init:
assumes dickson-grading $d$ and struct-spec sel ap ab compl
and $f s t$ ' $($ set $g s \cup$ set $b s) \subseteq$ dgrad- $p$-set $d m$
shows $f$ st' set (fst (gb-schema-dummy (rc, data) D (ab gs [] bs data) (ap gs [] []
bs data))) $\subseteq$ dgrad-p-set $d m$
proof (rule dgrad-p-set-le-dgrad-p-set)
let ?data $=(r c$, data $)$
from $\operatorname{assms}(1,2)$
have dgrad-p-set-le d (fst'set (fst (gb-schema-dummy ?data D (ab gs [] bs (snd ?data)) (ap gs [] [] bs (snd ?data)))))
$(f s t$ ' $($ set $g s \cup$ set $b s))$
by (rule fst-gb-schema-dummy-dgrad-p-set-le-init)
thus dgrad-p-set-le d (fst'set (fst (gb-schema-dummy ?data D (ab gs [] bs data) (ap gs [] [] bs data))))
$(f s t$ ' $($ set $g s \cup$ set $b s))$
by (simp only: snd-conv)
qed fact
lemma fst-gb-schema-dummy-components-init:
fixes $b s$ data
defines $b s 0 \equiv a b g s[] b s$ data
defines $p s 0 \equiv a p g s[][] b s$ data
assumes struct-spec sel ap ab compl
shows component-of-term ' Keys ( $f$ st' set (fst (gb-schema-dummy (rc, data) D bs( ps0))) $=$
component-of-term 'Keys (fst'set (gs @ bs)) (is ?l = ?r)
proof -
from $a s s m s(3)$ have $a p: a p$-spec $a p$ and $a b: a b$-spec $a b$ by (rule struct-spec $D$ ) +
from ap-specD1[OF ap, of gs [] [] bs]
have $*:$ set ps $0 \subseteq$ set bs0 $\times($ set gs $\cup$ set bs0) by (simp add: ps0-def bs0-def $a b-$ spec $D 1[$ OF $a b]$ )
with $\operatorname{assms}(3)$ have ?l = component-of-term'Keys (args-to-set (gs,bs0, ps0))
by (rule fst-gb-schema-dummy-components)
also have ... $=$ ? $r$
by (simp only: args-to-set-subset-Times[OF *], simp add: ab-specD1[OF ab] bs0-def image-Un)
finally show ?thesis .
qed
lemma fst-gb-schema-dummy-pmdl-init:
fixes bs data
defines $b s 0 \equiv a b g s[] b s$ data
defines $p s 0 \equiv a p$ gs [] [] bs data
assumes struct-spec sel ap ab compl and compl-pmdl compl and is-Groebner-basis (fst'set gs)
and unique-idx (gs @ bs0) data and rem-comps-spec (gs @ bs0) (rc, data)
shows pmdl (fst'set (fst (gb-schema-dummy (rc, data) D bs0 ps0))) = $p m d l(f s t$ ' $(\operatorname{set}(g s @ b s)))($ is ?l $=? r)$

## proof -

from $a s s m s(3)$ have $a b: a b-s p e c ~ a b$ by (rule struct-specD3)
let ? data $=(r c$, data $)$
from $\operatorname{assms}(6)$ have unique-idx ( $g s$ @ bs0) (snd?data) by (simp only: snd-conv)
from $\operatorname{assms}(3,4,5)-$ this assms (7) have ?l $=p m d l(f s t '(s e t(g s @ b s 0)))$
proof (rule fst-gb-schema-dummy-pmdl)
from $\operatorname{assms}(3)$ have ap-spec ap by (rule struct-specD2)
from ap-specD1[OF this, of gs [] [] bs]
show set ps $0 \subseteq$ set bs0 $\times($ set gs $\cup$ set bs0) by (simp add: ps0-def bs0-def $a b-s p e c D 1\left[\begin{array}{ll}O F & a b]\end{array}\right)$
qed
also have $\ldots=$ ?r by (simp add: bs0-def ab-specD $1[O F a b]$ )
finally show ?thesis.
qed
lemma fst-gb-schema-dummy-isGB-init:
fixes bs data
defines $b s 0 \equiv a b$ gs [] bs data
defines $p s 0 \equiv a p$ gs [] [] bs data
defines $D 0 \equiv$ set $b s \times($ set $g s \cup$ set $b s)-{ }_{p}$ set $p s 0$
assumes struct-spec sel ap ab compl and compl-conn compl and is-Groebner-basis (fst' set gs)
and unique-idx (gs @ bs0) data and rem-comps-spec (gs @ bs0) (rc, data)
shows is-Groebner-basis (fst'set (fst (gb-schema-dummy (rc, data) D0 bs0 ps0)))
proof -
let ? data $=(r c$, data $)$
let ?res $=$ gb-schema-dummy ? data D0 bs0 ps0
from $a s s m s(4)$ have $a p: a p-s p e c ~ a p$ and $a b: a b-s p e c ~ a b$ by (rule struct-specD2, rule struct-specD3)
have set-bs0: set bs0 = set bs by (simp add: bs0-def ab-specD1[OF ab])
from ap-specD1[OF ap, of gs [] [] bs] have ps0-sub: set ps0 $\subseteq$ set bs0 $\times$ (set gs $\cup$ set bso)
by (simp add: ps0-def set-bs0)
from ex-dgrad obtain $d::^{\prime} a \Rightarrow$ nat where dg: dickson-grading $d$..
have finite (fst' (set gs $\cup$ set bs)) by (rule, rule finite-UnI, fact finite-set, fact finite-set)
then obtain $m$ where $g s$-bs-sub: fst' (set gs $\cup$ set bs $) \subseteq$ dgrad- $p$-set $d m$ by (rule dgrad-p-set-exhaust)
with $d g \operatorname{assms}(4)$ have $f s t$ ' set $(f s t$ ?res $) \subseteq d g r a d-p$-set $d m$ unfolding bs0-def ps0-def
by (rule fst-gb-schema-dummy-dgrad-p-set-init)
with $d g$ show ?thesis
proof (rule crit-pair-cbelow-imp-GB-dgrad-p-set)
fix $p 0 q 0$
assume $p 0$-in: $p 0 \in f s t$ 'set (fst ?res) and $q 0$-in: $q 0 \in f s t$ 'set (fst ?res)
assume $p 0 \neq 0$ and $q 0 \neq 0$
from $\langle f s t$ ' (set gs $\cup$ set $b s) \subseteq$ dgrad-p-set $d m\rangle$
have $f_{s t}$ ' set $g s \subseteq d g r a d-p$-set $d m$ and $f s t$ 'set $b s \subseteq d g r a d-p$-set $d m$ by (simp-all add: image-Un)
from $p 0$-in obtain $p$ where $p$-in: $p \in$ set (fst ? res) and $p 0: p 0=$ fst $p .$.
from $q 0$-in obtain $q$ where $q$-in: $q \in$ set (fst ?res) and $q 0: q 0=f s t q .$.
from $\operatorname{assms}(7)$ have unique-idx (gs @ bs0) (snd ?data) by (simp only: snd-conv)
from $\operatorname{assms}(4,5) d g\langle f s t ‘$ set $g s \subseteq$ dgrad- $p$-set $d$ m〉assms(6)-ps0-sub -this - p-in $q$-in $\langle p 0 \neq 0\rangle\langle q 0 \neq 0\rangle$
show crit-pair-cbelow-on d $m$ (fst'set (fst ?res)) p0 q0 unfolding p0 q0 proof (rule gb-schema-dummy-connectible)
from 〈fst'set bs $\subseteq$ dgrad-p-set d m> show fst' set bs0 $\subseteq$ dgrad- $p$-set d m by (simp only: set-bs0)

## next

have $D 0 \subseteq$ set bs $\times($ set gs $\cup$ set bs) by (auto simp: assms(3) minus-pairs-def)
also have $\ldots \subseteq($ set $g s \cup$ set $b s) \times($ set $g s \cup$ set bs $)$ by fastforce
finally show $D 0 \subseteq($ set gs $\cup$ set bs0 $) \times($ set gs $\cup$ set bs0) by (simp only:
set-bs0)
next
show set ps $0 \cap_{p} D 0=\{ \}$
proof
show set ps $0 \cap_{p} D 0 \subseteq\{ \}$
proof
fix $x$
assume $x \in$ set ps $0 \cap_{p} D 0$
hence $x \in_{p}$ set ps0 $\cap_{p}$ DO by (simp add: in-pair-alt)
thus $x \in\}$ by (auto simp: assms(3))
qed
qed $\operatorname{simp}$
next
fix $p^{\prime} q^{\prime}$
assume processed $\left(p^{\prime}, q^{\prime}\right)(g s$ @ bs0) ps0
hence proc: processed ( $p^{\prime}, q^{\prime}$ ) (gs @ bs) ps0
by (simp add: set-bs0 processed-alt)
hence $p^{\prime} \in$ set $g s \cup$ set bs and $q^{\prime} \in$ set $g s \cup$ set bs and $\left(p^{\prime}, q^{\prime}\right) \not{ }_{p}$ set ps0
by (auto dest: processedD1 processedD2 processedD3)
assume $\left(p^{\prime}, q^{\prime}\right) \not \not_{p} D 0$ and $f s t p^{\prime} \neq 0$ and $f s t q^{\prime} \neq 0$
have crit-pair-cbelow-on d $m$ (fst ' (set gs $\cup$ set bs)) $($ fst $p$ ') $(f s t q$ ')
proof (cases $p^{\prime}=q^{\prime}$ )
case True
from $d g$ show ?thesis unfolding True
proof (rule crit-pair-cbelow-same)
from $\left\langle q^{\prime} \in\right.$ set $g s \cup$ set $\left.b s\right\rangle$ have $f s t q^{\prime} \in f s t$ ' (set $g s \cup$ set bs) by simp from this $\langle f s t$ ' (set gs $\cup$ set $b s) \subseteq$ dgrad-p-set $d$ m> show fst $q^{\prime} \in$ dgrad-p-set d m ..

```
        qed
    next
        case False
        show ?thesis
        proof (cases component-of-term (lt (fst p
q}\mp@subsup{}{}{\prime})
        case True
        show ?thesis
        proof (cases p' fet gs ^ q' ( set gs)
            case True
            note dg<fst` set gs \subseteqdgrad-p-set d m> assms(6)
            moreover from True have fst p'\infst' set gs and fst q' \infst' set gs
by simp-all
            ultimately have crit-pair-cbelow-on d m (fst' set gs) (fst p') (fst q')
            using <fst p}\mp@subsup{p}{}{\prime}\not=0\rangle\langlefst \mp@subsup{q}{}{\prime}\not=0\rangle\mathrm{ by (rule GB-imp-crit-pair-cbelow-dgrad-p-set)
            moreover have fst ' set gs \subseteqfst ' (set gs U set bs) by blast
            ultimately show ?thesis by (rule crit-pair-cbelow-mono)
        next
            case False
            with < p' \in set gs \cup set bs\rangle\langleq'\in set gs \cup set bs>
            have ( }\mp@subsup{p}{}{\prime},\mp@subsup{q}{}{\prime})\mp@subsup{\in}{p}{}\mathrm{ set bs }\times(\mathrm{ set gs }\cup\mathrm{ set bs) by (auto simp: in-pair-iff)
            with }\langle(\mp@subsup{p}{}{\prime},\mp@subsup{q}{}{\prime})\not\inp D0> have ( p', q') \inp set ps0 by (simp add: assms(3)
            with 〈( }\mp@subsup{p}{}{\prime},\mp@subsup{q}{}{\prime})\not\not\inp\mathrm{ set ps0> show ?thesis ..
        qed
    next
            case False
            thus ?thesis by (rule crit-pair-cbelow-distinct-component)
        qed
    qed
    thus crit-pair-cbelow-on d m (fst '(set gs U set bs0)) (fst p') (fst q')
    by (simp only: set-bs0)
    next
    fix Bab
    assume set gs \cup set bs0\subseteqB
    hence B-sup: set gs \cup set bs\subseteqB by (simp only: set-bs0)
    assume B-sub: fst' }B\subseteqdgrad-p-set d m
    assume (a,b) \inp D0
    hence ab-in: (a,b) \inp set bs \times (set gs U set bs) and (a,b)\not\inp set ps0
        by (simp-all add: assms(3))
    assume fst a\not=0 and fst b}=
    assume *: \bigwedgex y.x\in set gs U set bs0 \Longrightarrowy\in set gs \cup set bs0 \Longrightarrow(x,y)\not\inp
D0 \Longrightarrow
                    fst }x\not=0\Longrightarrow\mathrm{ fst }y\not=0\Longrightarrow\mathrm{ crit-pair-cbelow-on d m (fst' B) (fst
x) (fst y)
    show crit-pair-cbelow-on d m(fst' B) (fst a) (fst b)
    proof (cases a=b)
        case True
        from ab-in have b\in set gs U set bs unfolding True in-pair-iff set-diff-list
by auto
```

```
    hence fst b f fst'(set gs U set bs) by fastforce
    from this gs-bs-sub have fst b\indgrad-p-set d m ..
    with dg show ?thesis unfolding True by (rule crit-pair-cbelow-same)
    next
    case False
    note ap dg
    moreover from B-sup have B-sup': set gs \cup set [] \cup set bs\subseteqB by simp
    moreover note B-sub
    moreover from ab-in have (a,b)\inf set bs \times(set gs \cup set [] \cup set bs) by
simp
    moreover have set [] \subseteq set [] \times (set gs U set []) by simp
    moreover from assms(7) have unique-idx (gs @ [] @ bs) data by (simp
add: unique-idx-def set-bs0)
    ultimately show ?thesis using assms(6) False〈fst a\not=0\rangle\langlefst b\not=0\rangle
    proof (rule ap-specD2)
        fix x y :: ('t,'b,'c) pdata
        assume (x,y) \inp set (ap gs [] [] bs data)
        hence (x,y)\inp set ps0 by (simp only: ps0-def)
        also have ...\subseteq set bs0 }\times(\mathrm{ set gs }\cup\mathrm{ set bsO) by (fact ps0-sub)
        also have ...\subseteq(set gs \cup set bs0) }\times(\mathrm{ set gs }\cup\mathrm{ set bs0) by fastforce
            finally have (x,y)\in(set gs U set bs0) }\times(\mathrm{ set gs U set bs0) by (simp
only: in-pair-same)
            hence }x\in\mathrm{ set gs U set bs0 and y set gs U set bs0 by simp-all
            moreover from <(x,y) \inp set ps0\rangle have (x,y)\not\inp D0 by (simp add:
D0-def)
            moreover assume fst x\not=0 and fst y}\not=
            ultimately show crit-pair-cbelow-on d m (fst ' B) (fst x) (fst y) by (rule
*)
            next
            fix x y :: ('t, 'b, 'c) pdata
            assume x e set gs \cup set [] and y\in set gs \cup set []
            hence fst }x\infst'set gs and fst y\infst' set gs by simp-all
            assume fst x\not=0 and fst y}\not=
            with dg <fst' set gs \subseteqdgrad-p-set d m> assms(6)<fst x f fst' set gs><fst
y\infst' set gs>
            have crit-pair-cbelow-on d m (fst 'set gs) (fst x) (fst y)
                by (rule GB-imp-crit-pair-cbelow-dgrad-p-set)
            moreover from B-sup have fst'set gs \subseteqfst'B by fastforce
            ultimately show crit-pair-cbelow-on d m (fst ' B) (fst x) (fst y)
                by (rule crit-pair-cbelow-mono)
            qed
        qed
    qed
    qed
qed
```

6.2.10 Function $g b$-schema-aux
function (domintros) gb-schema-aux :: nat $\times$ nat $\times{ }^{\prime} d \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$

$$
\left({ }^{\prime} t,,^{\prime} b,{ }^{\prime} c\right) \text { pdata-pair list } \Rightarrow\left({ }^{\prime} t,,^{\prime} b,{ }^{\prime} c\right) \text { pdata list }
$$

```
where
    gb-schema-aux data bs ps=
        (if ps=[] then
        gs @ bs
        else
            (let sps = sel gs bs ps (snd data); ps0 = ps -- sps; aux = compl gs bs
ps0 sps (snd data);
            remcomps = fst (data) - count-const-lt-components (fst aux) in
            (if remcomps = 0 then
            full-gb (gs @ bs)
            else
                    let (hs,data') = add-indices aux (snd data) in
                        gb-schema-aux (remcomps, data') (ab gs bs hs data') (ap gs bs ps0 hs
data')
            )
        )
        )
    by pat-completeness auto
```

The data parameter of $g b$-schema-aux is a triple $(c, i, d)$, where $c$ is the number of components $c m p$ of the input list for which the current basis $g s$ @ $b s$ does not yet contain an element whose leading power-product is 0 and has component $c m p$. As soon as $c$ gets 0 , the function can return a trivial Gröbner basis, since then the submodule generated by the input list is just the full module. This idea generalizes the well-known fact that if a set of scalar polynomials contains a non-zero constant, the ideal generated by that set is the whole ring. $i$ is the total number of polynomials generated during the execution of the function so far; it is used to attach unique indices to the polynomials for fast equality tests. $d$, finally, is some arbitrary datafield that may be used by concrete instances of gb-schema-aux for storing information.

```
lemma gb-schema-aux-domI1:gb-schema-aux-dom (data, bs, [])
```

by (rule gb-schema-aux.domintros, simp)
lemma gb-schema-aux-domI2:
assumes struct-spec sel ap ab compl
shows gb-schema-aux-dom (data, args)
proof -
from assms have sel: sel-spec sel and $a p$ : $a p$-spec $a p$ and $a b: a b$-spec $a b$ by (rule struct-spec $D$ ) +
from ex-dgrad obtain $d::^{\prime} a \Rightarrow$ nat where $d g$ : dickson-grading $d$..
let $? R=$ gb-schema-aux-term $d$ gs
from $d g$ have $w f$ ? $R$ by (rule gb-schema-aux-term-wf)
thus ?thesis
proof (induct args arbitrary: data rule: wf-induct-rule)
fix $x$ data
assume $I H: \bigwedge y d a t a{ }^{\prime} .(y, x) \in ? R \Longrightarrow g b$-schema-aux-dom $\left(d a t a^{\prime}, y\right)$
obtain bs ps where $x: x=(b s, p s)$ by (meson case-prodE case-prodI2)
show gb-schema-aux-dom (data, x) unfolding $x$
proof (rule gb-schema-aux.domintros)
fix rc0 n0 data0 hs n1 data1
assume $p s \neq[]$
and $h s$-data ${ }^{\prime}:(h s, n 1$, data1 $)=a d d$-indices (compl gs bs $(p s--$ sel gs bs $p s(n 0$, data0 $))$

```
                                    (sel gs bs ps (n0, data0)) (n0, data0)) (n0,
```

data0)
and data: data $=(r c 0$, n0, data0 $)$
define $s p s$ where $s p s=$ sel gs bs ps (n0, data0)
define $d a t a^{\prime}$ where $d a t a^{\prime}=(n 1$, data1 $)$
define $r c$ where $r c=r c 0$ - count-const-lt-components (fst (compl gs bs (ps -- sel gs bs ps (n0, data0))
(sel gs bs ps (n0, data0)) (n0, data0)))
from $h s$-data' have $h s: h s=f s t$ (add-indices (compl gs bs (ps -- sps) sps (snd data)) (snd data))
unfolding sps-def data snd-conv by (metis fstI)
show gb-schema-aux-dom ((rc, data'), ab gs bs hs data', ap gs bs (ps -- sps) hs data')
proof (rule IH, simp add: x gb-schema-aux-term-def gb-schema-aux-term1-def gb-schema-aux-term2-def, intro conjI)
show $f s t$ ' set (ab gs bs hs data') $\sqsupset p$ fst ' set bs $\vee$
ab gs bs hs data' $=b s \wedge$ card $\left(\right.$ set (ap gs bs $(p s--$ sps $\left.\left.) h s d a t a^{\prime}\right)\right)<$ card (set ps)
proof (cases hs = [])
case True
have $a b$ gs bs hs data' $=b s \wedge$ card (set (ap gs bs (ps -- sps) hs data')) $<\operatorname{card}$ (set ps)
proof (simp only: True, rule)
from $a b$ show $a b$ gs $b s[] d a t a^{\prime}=b s$ by (rule $\left.a b-s p e c D 2\right)$
next
from sel $\langle p s \neq[]\rangle$ have $s p s \neq[]$ and set sps $\subseteq$ set $p s$
unfolding sps-def by (rule sel-specD1, rule sel-specD2)
moreover from sel-specD1[OF sel $\langle p s \neq[]\rangle]$ have set sps $\neq\{ \}$ by (simp add: sps-def)
ultimately have set $p s \cap$ set $s p s \neq\{ \}$ by (simp add: inf.absorb-iff2)
hence set ( $p s--s p s$ ) $\subset$ set ps unfolding set-diff-list by fastforce
hence card (set (ps -- sps)) < card (set ps) by (simp add: psub-set-card-mono)
moreover have card (set (ap gs bs (ps-- sps) [] data')) $\leq$ card (set ( $p s--s p s)$ )
by (rule card-mono, fact finite-set, rule ap-spec-Nil-subset, fact ap)
ultimately show card (set (ap gs bs (ps -- sps) [] data')) < card (set
$p s)$ by $\operatorname{simp}$
qed
thus ?thesis..
next

```
            case False
            with assms «ps\not=[]> sps-def hs have fst 'set (ab gs bs hs data') }\existspfst
set bs
            unfolding data snd-conv by (rule struct-spec-red-supset)
            thus ?thesis ..
        qed
    next
        from dg assms <ps \not=[]> sps-def hs
        show dgrad-p-set-le d (args-to-set (gs, ab gs bs hs data', ap gs bs (ps --
sps) hs data')) (args-to-set (gs, bs, ps))
            unfolding data snd-conv by (rule dgrad-p-set-le-args-to-set-struct)
        next
        from assms <ps \not= []> sps-def hs
        show component-of-term' Keys (args-to-set (gs, ab gs bs hs data', ap gs bs
(ps -- sps) hs data'))\subseteq
            component-of-term` Keys (args-to-set (gs, bs,ps))
            unfolding data snd-conv by (rule components-subset-struct)
        qed
    qed
    qed
qed
lemma gb-schema-aux-Nil [simp, code]: gb-schema-aux data bs [] = gs @ bs by (simp add: gb-schema-aux.psimps[OF gb-schema-aux-domI1])
lemmas gb-schema-aux-simps \(=\) gb-schema-aux.psimps[OF gb-schema-aux-domI2]
lemma gb-schema-aux-induct [consumes 1, case-names base rec1 rec2]:
assumes struct-spec sel ap ab compl
assumes base: \(\bigwedge\) bs data. P data bs [] (gs @ bs)
and rec1: \(\bigwedge\) bs ps sps data. ps \(\neq[] \Longrightarrow\) sps \(=\) sel gs bs ps (snd data) \(\Longrightarrow\)
fst \((\) data \() \leq\) count-const-lt-components (fst (compl gs bs (ps -- sps)
sps (snd data) )) \(\Longrightarrow\)
P data bs ps (full-gb (gs @ bs))
and rec2: \(\bigwedge\) bs ps sps aux hs rc data data' \({ }^{\prime}\) ps \(\neq[] \Longrightarrow\) sps \(=\) sel gs bs ps (snd data) \(\Longrightarrow\)
aux \(=\) compl gs bs \((p s--\) sps \()\) sps \((\) snd data \() \Longrightarrow(h s, d a t a)=\) add-indices aux (snd data) \(\Longrightarrow\)
\(r c=f s t\) data - count-const-lt-components \((f s t a u x) \Longrightarrow 0<r c \Longrightarrow\)
\(\left.P(r c, d a t a)^{\prime}\right)(a b\) gs bs hs data') (ap gs bs (ps -- sps) hs data')
(gb-schema-aux (rc, data') (ab gs bs hs data') (ap gs bs (ps -- sps)
hs data' \()\) ) \(\Longrightarrow\)
\(P\) data bs ps (gb-schema-aux (rc, data') (ab gs bs hs data') (ap gs bs ( \(p s--s p s\) ) hs data'))
shows \(P\) data bs ps (gb-schema-aux data bs ps)
proof -
from \(\operatorname{assms}(1)\) have \(g b\)-schema-aux-dom (data, bs, ps) by (rule gb-schema-aux-domI2)
thus ?thesis
proof (induct data bs ps rule: gb-schema-aux.pinduct)
```

```
    case (1 data bs ps)
    show ?case
    proof (cases ps=[])
    case True
    show ?thesis by (simp add: True, rule base)
    next
    case False
    show ?thesis
    proof (simp add: gb-schema-aux-simps[OF assms(1), of data bs ps] False
Let-def split: if-split,
            intro conjI impI)
            define sps where sps=sel gs bs ps (snd data)
            assume fst data \leq count-const-lt-components (fst (compl gs bs (ps -- sps)
sps (snd data)))
            with False sps-def show P data bs ps (full-gb (gs @ bs)) by (rule rec1)
    next
            define sps where sps=sel gs bs ps (snd data)
            define aux where aux = compl gs bs (ps-- sps) sps (snd data)
            define hs where hs = fst (add-indices aux (snd data))
            define data' where data' = snd (add-indices aux (snd data))
            define rc where rc = fst data - count-const-lt-components (fst aux)
            have eq: add-indices aux (snd data) = (hs, data') by (simp add: hs-def
data'-def)
            assume \negfst data \leqcount-const-lt-components (fst aux)
    hence 0<rc by (simp add: rc-def)
    hence rc\not=0 by simp
    show P data bs ps
                (case add-indices aux (snd data) of
            (hs,data') =>gb-schema-aux (rc, data') (ab gs bs hs data') (ap gs bs (ps
-- sps) hs data'))
            unfolding eq prod.case using False sps-def aux-def eq[symmetric] rc-def
<0<rc>
    proof (rule rec2)
            show P (rc, data') (ab gs bs hs data') (ap gs bs (ps -- sps)hs data')
                (gb-schema-aux (rc,data') (ab gs bs hs data') (ap gs bs (ps -- sps)
hs data'))
            using False sps-def refl aux-def rc-def 〈rc \not= 0〉 eq[symmetric] refl by
(rule 1)
            qed
            qed
    qed
    qed
qed
lemma gb-schema-dummy-eq-gb-schema-aux:
    assumes struct-spec sel ap ab compl
    shows fst (gb-schema-dummy data D bs ps)=gb-schema-aux data bs ps
    using assms
proof (induct data D bs ps rule: gb-schema-dummy-induct)
```

```
    case (base bs data D)
    show ?case by simp
next
    case (rec1 bs ps sps data D)
    thus ?case by (simp add: gb-schema-aux.psimps[OF gb-schema-aux-domI2, OF
assms])
next
    case (rec2 bs ps sps aux hs rc data data' D D')
    note rec2.hyps(8)
    also from rec2.hyps(1, 2, 3) rec2.hyps(4)[symmetric] rec2.hyps(5, 6, 7)
    have gb-schema-aux (rc, data') (ab gs bs hs data') (ap gs bs (ps -- sps) hs data')
=
gb-schema-aux data bs ps
    by (simp add: gb-schema-aux.psimps[OF gb-schema-aux-domI2, OF assms, of
data] Let-def)
    finally show ?case .
qed
corollary gb-schema-aux-dgrad-p-set-le:
    assumes dickson-grading d and struct-spec sel ap ab compl
    shows dgrad-p-set-le d (fst' set (gb-schema-aux data bs ps)) (args-to-set (gs, bs,
ps))
    using fst-gb-schema-dummy-dgrad-p-set-le[OF assms] unfolding gb-schema-dummy-eq-gb-schema-aux[OF
assms(2)].
corollary gb-schema-aux-components:
    assumes struct-spec sel ap ab compl and set ps\subseteq set bs \times (set gs U set bs)
    shows component-of-term 'Keys (fst' set (gb-schema-aux data bs ps)) =
                        component-of-term'Keys (args-to-set (gs, bs, ps))
    using fst-gb-schema-dummy-components[OF assms] unfolding gb-schema-dummy-eq-gb-schema-aux[OF
assms(1)].
lemma gb-schema-aux-pmdl:
    assumes struct-spec sel ap ab compl and compl-pmdl compl and is-Groebner-basis
(fst' set gs)
    and set ps\subseteq set bs }\times(\mathrm{ set gs U set bs) and unique-idx (gs @ bs) (snd data)
    and rem-comps-spec (gs @ bs)data
    shows pmdl (fst`set (gb-schema-aux data bs ps)) = pmdl (fst`set (gs @ bs))
    using fst-gb-schema-dummy-pmdl[OF assms] unfolding gb-schema-dummy-eq-gb-schema-aux[OF
assms(1)].
corollary gb-schema-aux-dgrad-p-set-le-init:
    assumes dickson-grading d and struct-spec sel ap ab compl
    shows dgrad-p-set-le d (fst'set (gb-schema-aux data (ab gs [] bs (snd data)) (ap
gs [] [] bs (snd data))))
                        (fst '(set gs U set bs))
```

    using fst-gb-schema-dummy-dgrad-p-set-le-init \([O F\) assms \(]\) unfolding \(g b\)-schema-dummy-eq-gb-schema-aux \([0\)
    $\operatorname{assms}(2)]$.
corollary gb-schema-aux-dgrad-p-set-init:
assumes dickson-grading $d$ and struct-spec sel ap ab compl
and $f s t$ ' $($ set $g s \cup$ set $b s) \subseteq$ dgrad- $p$-set $d m$
shows fst'set (gb-schema-aux (rc, data) (ab gs [] bs data) (ap gs [] [] bs data))
$\subseteq d g r a d-p-s e t d m$
using fst-gb-schema-dummy-dgrad-p-set-init[OF assms] unfolding gb-schema-dummy-eq-gb-schema-aux[OF $\operatorname{assms}(2)]$.
corollary gb-schema-aux-components-init:
assumes struct-spec sel ap ab compl
shows component-of-term 'Keys ( $f s t$ ' set (gb-schema-aux (rc, data) (ab gs [] bs
data) $($ ap gs [] [] bs data) $))=$
component-of-term'Keys (fst'set (gs @bs))
using fst-gb-schema-dummy-components-init [OF assms] unfolding gb-schema-dummy-eq-gb-schema-aux [OF assms] .
corollary gb-schema-aux-pmdl-init:
assumes struct-spec sel ap ab compl and compl-pmdl compl and is-Groebner-basis (fst' set gs)
and unique-idx (gs @ ab gs [] bs data) data and rem-comps-spec (gs @ ab gs [] bs data) (rc, data)
shows pmdl (fst'set (gb-schema-aux (rc, data) (ab gs [] bs data) (ap gs [] [] bs
data) $)$ ) $=$ $p m d l(f s t$ ' $(s e t(g s$ @ $b s)))$
using $f$ ft-gb-schema-dummy-pmdl-init[OF assms] unfolding gb-schema-dummy-eq-gb-schema-aux[OF $\operatorname{assms}(1)]$.
lemma gb-schema-aux-isGB-init:
assumes struct-spec sel ap ab compl and compl-conn compl and is-Groebner-basis (fst'set gs)
and unique-idx (gs @abgs [] bs data) data and rem-comps-spec (gs @ ab gs [] bs data) (rc, data)
shows is-Groebner-basis (fst'set (gb-schema-aux (rc, data) (ab gs [] bs data)
(ap gs [] [] bs data)))
using fst-gb-schema-dummy-isGB-init [OF assms] unfolding gb-schema-dummy-eq-gb-schema-aux [OF $\operatorname{assms}(1)]$.
end
6.2.11 Functions $g b$-schema-direct and term gb-schema-incr
definition $g b$-schema-direct :: ( $\left.{ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right) \operatorname{sel} T \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right) a p T \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b\right.$, $\left.{ }^{\prime} c,^{\prime} d\right) a b T \Rightarrow$

$$
\begin{aligned}
& \left({ }^{\prime} t,,^{\prime} b,,^{\prime} c,,^{\prime} d\right) \text { compl } \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right) \text { pdata' } \text { list } \Rightarrow^{\prime} d \Rightarrow \\
& (' t, \text { 'b::field, 'c::default) pdata' list }
\end{aligned}
$$

where gb-schema-direct sel ap ab compl bs0 data0 $=$
$($ let data $=($ length bs0, data0 $) ; b s 1=$ fst $($ add-indices $(b s 0, d a t a 0)(0$,
data0));

$$
b s=a b[][] \text { bs1 data in }
$$

```
map (\lambda(f,-,d). (f,d))
    (gb-schema-aux sel ap ab compl [] (count-rem-components bs, data)
bs (ap [] [] [] bs1 data))
```

primrec $g b$-schema-incr :: ('t, 'b, 'c, 'd) selT $\Rightarrow\left(' t,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right)$ ap $T \Rightarrow\left(' t,{ }^{\prime} b,{ }^{\prime} c\right.$, $\left.{ }^{\prime} d\right) a b T \Rightarrow$

> ('t, 'b, 'c, 'd) complT $\Rightarrow$
> $\left(\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)\right.$ pdata list $\left.\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right) p d a t a \Rightarrow{ }^{\prime} d \Rightarrow{ }^{\prime} d\right) \Rightarrow$ $\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right) p d a t a{ }^{\prime}$ list $\Rightarrow{ }^{\prime} d \Rightarrow\left(' t, ' b::\right.$ field, ${ }^{\prime} c::$ default $)$
pdata' list
where
gb-schema-incr --- - [] - = []|
gb-schema-incr sel ap ab compl upd (b0 \# bs) data $=$
(let $(g s, n, d a t a)=a d d-$ indices $(g b-s c h e m a-i n c r$ sel ap ab compl upd bs data, data) (0, data);
$b=\left(\right.$ fst b0, n, snd b0); data ${ }^{\prime \prime}=$ upd gs $b$ data $^{\prime}$ in
$\operatorname{map}(\lambda(f,-, d) .(f, d))$
(gb-schema-aux sel ap ab compl gs (count-rem-components (b\#gs), Suc $n$, data ${ }^{\prime \prime}$ )

$$
\left.(\text { ab gs }[][b](\text { Suc n, data'f }))\left(\text { ap gs }[][][b]\left(\text { Suc n, }{ }^{\prime} \text { data }^{\prime \prime}\right)\right)\right)
$$

)
lemma (in -) fst-set-drop-indices:
$f_{s t}{ }^{\prime}(\lambda(f,-, d) .(f, d))^{\prime} A=f s t$ ' $A$ for $A::\left({ }^{\prime} x \times{ }^{\prime} y \times{ }^{\prime} z\right)$ set by (simp add: image-image, rule image-cong, fact refl, simp add: prod.case-eq-if)
lemma fst-gb-schema-direct:
fst'set (gb-schema-direct sel ap ab compl bs0 data0) $=$
(let data $=($ length bs0, data0 $) ;$ bs $1=$ fst (add-indices $($ bs0 0 data 0$)(0$, data 0$))$; $b s=a b[][] b s 1$ data in
fst' set (gb-schema-aux sel ap ab compl [] (count-rem-components bs, data) bs (ap [] [] [] bs1 data))
)
by (simp add: gb-schema-direct-def Let-def fst-set-drop-indices)
lemma gb-schema-direct-dgrad-p-set:
assumes dickson-grading $d$ and struct-spec sel ap ab compl and fst'set bs $\subseteq$ dgrad-p-set d m
shows $f s t$ ' set ( $g b$-schema-direct sel ap ab compl bs data) $\subseteq$ dgrad- $p$-set $d m$
unfolding fst-gb-schema-direct Let-def using $\operatorname{assms}(1,2)$
proof (rule gb-schema-aux-dgrad-p-set-init)
show $f s t$ ' $($ set []$\cup$ set $(f s t(a d d-i n d i c e s ~(b s, d a t a)(0, d a t a)))) \subseteq$ dgrad-p-set d m
using assms(3) by (simp add: image-Un fst-set-add-indices)
qed
theorem gb-schema-direct-isGB:
assumes struct-spec sel ap ab compl and compl-conn compl

```
    shows is-Groebner-basis (fst' set (gb-schema-direct sel ap ab compl bs data))
    unfolding fst-gb-schema-direct Let-def using assms
proof (rule gb-schema-aux-isGB-init)
    from is-Groebner-basis-empty show is-Groebner-basis (fst' set []) by simp
next
    let ?data = (length bs, data)
    from assms(1) have ab-spec ab by (rule struct-specD)
    moreover have unique-idx ([] @ []) (0, data) by (simp add: unique-idx-Nil)
    ultimately show unique-idx ([] @ ab [] [] (fst (add-indices (bs,data) (0,data)))
    ?data) ?data
    proof (rule unique-idx-ab)
    show (fst (add-indices (bs,data) (0,data)), length bs,data)=add-indices (bs,
data) (0, data)
            by (simp add: add-indices-def)
    qed
qed (simp add: rem-comps-spec-count-rem-components)
theorem gb-schema-direct-pmdl:
    assumes struct-spec sel ap ab compl and compl-pmdl compl
    shows pmdl (fst'set (gb-schema-direct sel ap ab compl bs data)) = pmdl (fst`
set bs)
proof -
    have pmdl (fst'set (gb-schema-direct sel ap ab compl bs data)) =
                    pmdl (fst'set ([] @ (fst (add-indices (bs,data) (0,data)))))
    unfolding fst-gb-schema-direct Let-def using assms
    proof (rule gb-schema-aux-pmdl-init)
    from is-Groebner-basis-empty show is-Groebner-basis (fst' set []) by simp
    next
    let ?data = (length bs, data)
    from assms(1) have ab-spec ab by (rule struct-specD)
    moreover have unique-idx ([] @ []) (0, data) by (simp add: unique-idx-Nil)
    ultimately show unique-idx ([] @ ab [] [] (fst (add-indices (bs,data) (0,data)))
?data) ?data
    proof (rule unique-idx-ab)
            show (fst (add-indices (bs, data) (0, data)), length bs, data) = add-indices
(bs, data) (0, data)
            by (simp add: add-indices-def)
        qed
    qed (simp add: rem-comps-spec-count-rem-components)
    thus ?thesis by (simp add: fst-set-add-indices)
qed
lemma fst-gb-schema-incr:
    fst'set (gb-schema-incr sel ap ab compl upd (b0 # bs) data)=
        (let (gs, n, data') = add-indices (gb-schema-incr sel ap ab compl upd bs data,
data) (0, data);
            b=(fst b0, n, snd b0); data" = upd gs b data' in
    fst'set (gb-schema-aux sel ap ab compl gs (count-rem-components (b # gs),
Suc n, data'')
```

```
                                    (ab gs [] [b] (Suc n, data'')) (ap gs [] [] [b] (Suc n, data'')))
    )
    by (simp only:gb-schema-incr.simps Let-def prod.case-distrib[of set]
        prod.case-distrib[of image fst] set-map fst-set-drop-indices)
lemma gb-schema-incr-dgrad-p-set:
    assumes dickson-grading d and struct-spec sel ap ab compl
        and fst' set bs\subseteqdgrad-p-set d m
    shows fst' set (gb-schema-incr sel ap ab compl upd bs data)\subseteq dgrad-p-set d m
    using assms(3)
proof (induct bs)
    case Nil
    show ?case by simp
next
    case (Cons b0 bs)
    from Cons(2) have 1: fst b0 \in dgrad-p-set d m and 2: fst' set bs \subseteqdgrad-p-set
d m by simp-all
    show ?case
    proof (simp only: fst-gb-schema-incr Let-def split: prod.splits, simp, intro allI
impI)
    fix gs n data'
        assume add-indices (gb-schema-incr sel ap ab compl upd bs data, data) (0,
data) = (gs, n, data')
    hence gs: gs = fst (add-indices (gb-schema-incr sel ap ab compl upd bs data,
data) (0, data)) by simp
    define b where b}=(fst b0,n, snd b0
    define data" where data" = upd gs b data'
    from assms(1, 2)
    show fst'set (gb-schema-aux sel ap ab compl gs (count-rem-components (b #
gs), Suc n, data'")
                            (ab gs [] [b] (Suc n, data')) (ap gs [] [] [b] (Suc n,data'))) \subseteq dgrad-p-set
dm
    proof (rule gb-schema-aux-dgrad-p-set-init)
            from 1 Cons(1)[OF 2] show fst '(set gs \cup set [b])\subseteqdgrad-p-set d m
                by (simp add:gs fst-set-add-indices b-def)
    qed
    qed
qed
theorem gb-schema-incr-dgrad-p-set-isGB:
    assumes struct-spec sel ap ab compl and compl-conn compl
    shows is-Groebner-basis (fst'set (gb-schema-incr sel ap ab compl upd bs data))
proof (induct bs)
    case Nil
    from is-Groebner-basis-empty show ?case by simp
next
    case (Cons b0 bs)
    show ?case
    proof (simp only: fst-gb-schema-incr Let-def split: prod.splits, simp, intro allI
```

```
impI)
    fix gs n data'
    assume *: add-indices (gb-schema-incr sel ap ab compl upd bs data, data) (0,
data)}=(gs,n,data'
    hence gs: gs = fst (add-indices (gb-schema-incr sel ap ab compl upd bs data,
data) (0, data)) by simp
    define b where b=(fst b0, n, snd b0)
    define data"' where data" = upd gs b data'
    from assms(1) have ab: ab-spec ab by (rule struct-specD3)
    from Cons have is-Groebner-basis (fst' set gs) by (simp add: gs fst-set-add-indices)
    with assms
    show is-Groebner-basis (fst'set (gb-schema-aux sel ap ab compl gs (count-rem-components
(b # gs), Suc n, data')
                                    (ab gs [] [b] (Suc n, data'\prime)) (ap gs [] [] [b] (Suc n, data'))))
    proof (rule gb-schema-aux-isGB-init)
        from ab show unique-idx (gs @ ab gs [] [b] (Suc n,data'')) (Suc n, data')
            proof (rule unique-idx-ab)
                from unique-idx-Nil *[symmetric] have unique-idx ([]@ gs) (n,data')
                by (rule unique-idx-append)
                thus unique-idx (gs @ []) (n,data') by simp
            next
                show ([b], Suc n, data'') = add-indices ([b0], data'') (n, data')
                by (simp add: add-indices-def b-def)
            qed
    next
            have rem-comps-spec (b # gs) (count-rem-components (b # gs), Suc n,data')
                by (fact rem-comps-spec-count-rem-components)
            moreover have set (b#gs)=set (gs @ ab gs [] [b] (Suc n,data''))
                by (simp add: ab-specD1[OF ab])
            ultimately show rem-comps-spec (gs @ ab gs [] [b] (Suc n,data'`))
                                    (count-rem-components (b # gs), Suc n,data'')
                by (simp only: rem-comps-spec-def)
    qed
    qed
qed
theorem gb-schema-incr-pmdl:
    assumes struct-spec sel ap ab compl and compl-conn compl compl-pmdl compl
    shows pmdl (fst'set (gb-schema-incr sel ap ab compl upd bs data)) = pmdl (fst
' set bs)
proof (induct bs)
    case Nil
    show ?case by simp
next
    case (Cons b0 bs)
    show ?case
    proof (simp only: fst-gb-schema-incr Let-def split: prod.splits, simp, intro allI
impI)
    fix gs n data'
```

assume *: add-indices (gb-schema-incr sel ap ab compl upd bs data, data) (0, data $)=\left(g s, n, d a t a^{\prime}\right)$
hence $g s: g s=f s t$ (add-indices (gb-schema-incr sel ap ab compl upd bs data, data) ( 0 , data)) by simp
define $b$ where $b=(f s t b 0, n$, snd b0)
define data" where data ${ }^{\prime \prime}=u p d ~ g s ~ b d a t a^{\prime}$
from $\operatorname{assms}(1)$ have $a b: a b-$ spec $a b$ by (rule struct-specD3)
from $\operatorname{assms}(1,2)$ have $i s$-Groebner-basis (fst'set gs) unfolding gs fst-conv fst-set-add-indices
by (rule gb-schema-incr-dgrad-p-set-isGB)
with $\operatorname{assms}(1,3)$
have eq: pmdl (fst'set (gb-schema-aux sel ap ab compl gs (count-rem-components ( $b$ \# gs), Suc n, data' ${ }^{\prime \prime}$ )
$($ ab gs []$[b]($ Suc n, data'ر $))\left(\right.$ ap gs [] [] [b] $\left(\right.$ Suc n, data $\left.\left.\left.\left.{ }^{\prime \prime}\right)\right)\right)\right)=$ $p m d l(f s t$ 'set (gs @ [b]))
proof (rule gb-schema-aux-pmdl-init)
from $a b$ show unique-idx (gs @ ab gs [] [b] (Suc n, data $\left.{ }^{\prime \prime}\right)$ ) (Suc n,data ${ }^{\prime \prime}$ )
proof (rule unique-idx-ab)
from unique-idx-Nil *[symmetric] have unique-idx ([] @ gs) (n, data')
by (rule unique-idx-append)
thus unique-idx (gs @ []) (n, data') by simp
next
show $\left([b]\right.$, Suc $n$, data $\left.{ }^{\prime \prime}\right)=$ add-indices $\left([b 0]\right.$, data $\left.^{\prime \prime}\right)(n$, data $)$
by (simp add: add-indices-def b-def)
qed
next
have rem-comps-spec (b\#gs) (count-rem-components (b \# gs), Suc n, data') by (fact rem-comps-spec-count-rem-components)
moreover have set $(b \# g s)=\operatorname{set}\left(g s @ a b g s[][b]\right.$ (Suc n, data $\left.{ }^{\prime \prime}\right)$ ) by (simp add: ab-specD1 [OF ab])
ultimately show rem-comps-spec (gs @ ab gs [] [b] (Suc n, data'ر))
(count-rem-components ( $b$ \# gs), Suc n, data ${ }^{\prime \prime}$ )
by (simp only: rem-comps-spec-def)
qed
also have $\ldots=p m d l($ insert $(f s t b)(f s t ' s e t ~ g s))$ by simp
also from Cons have $\ldots=p m d l$ (insert (fst b) (fst' set bs))
unfolding gs fst-conv fst-set-add-indices by (rule pmdl.span-insert-cong)
finally show pmdl ( $f s t$ 'set (gb-schema-aux sel ap ab compl gs (count-rem-components ( $b$ \# gs), Suc n, data ${ }^{\prime \prime}$ )
(ab gs [] [b] (Suc n, data'ر $)$ ) (ap gs [] [] [b] (Suc n, data' $)$ ) ) $)$
$=$

$$
p m d l\left(\text { insert } ( f _ { s t } b 0 ) \left(f_{s t} \cdot\right.\right. \text { set bs)) by (simp add: b-def) }
$$

qed
qed

### 6.3 Suitable Instances of the add-pairs Parameter

### 6.3.1 Specification of the crit parameters

type-synonym (in -$)\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right)$ icrit $T=n a t \times{ }^{\prime} d \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$ $\left(^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
$\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right) p d a t a \Rightarrow\left(' t,{ }^{\prime} b\right.$,
'c) pdata $\Rightarrow$ bool
type-synonym (in -$)\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c,^{\prime} d\right) n c r i t T=n a t \times{ }^{\prime} d \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
('t, 'b, 'c) pdata list $\Rightarrow$ bool $\Rightarrow$
(bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair) list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$
pdata $\Rightarrow$

$$
\left(^{\prime} t,,^{\prime} b,{ }^{\prime} c\right) p d a t a \Rightarrow \text { bool }
$$

type-synonym (in -) $\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right)$ ocrit $T=n a t \times{ }^{\prime} d \Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$ (bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair) list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$
pdata $\Rightarrow$

$$
\left(' t,{ }^{\prime} b,{ }^{\prime} c\right) p d a t a \Rightarrow \text { bool }
$$

definition icrit-spec :: ('t, 'b::field, 'c, 'd) icrit $T \Rightarrow$ bool
where icrit-spec crit $\longleftrightarrow$
$(\forall d m$ data gs bs hs $p$. dickson-grading $d \longrightarrow$ $f s t$ ' $($ set $g s \cup$ set $b s \cup$ set $h s) \subseteq$ dgrad- $p$-set $d m \longrightarrow$ unique-idx $(g s @$ bs @ hs) data $\longrightarrow$
is-Groebner-basis (fst'set gs) $\longrightarrow p \in$ set $h s \longrightarrow q \in$ set $g s \cup$ set $b s \cup$
set $h s \longrightarrow$
fst $p \neq 0 \longrightarrow$ fst $q \neq 0 \longrightarrow$ crit data gs bs hs $p q \longrightarrow$
crit-pair-cbelow-on d $m(f s t$ ' $($ set $g s \cup$ set $b s \cup$ set $h s))(f s t p)(f s t q))$
Criteria satisfying icrit-spec can be used for discarding pairs instantly, without reference to any other pairs. The product criterion for scalar polynomials satisfies icrit-spec, and so does the component criterion (which checks whether the component-indices of the leading terms of two polynomials are identical).
definition ncrit-spec :: ('t, 'b::field, ' $\left.c,{ }^{\prime} d\right)$ ncrit $T \Rightarrow$ bool
where ncrit-spec crit $\longleftrightarrow$
( $\forall d m$ data gs bs hs ps B q-in-bs p q. dickson-grading $d \longrightarrow$ set $g s \cup$ set bs $\cup$ set $h s \subseteq B \longrightarrow$
fst ' $B \subseteq d g r a d-p-s e t ~ d m \longrightarrow s n d$ ' set $p s \subseteq$ set $h s \times($ set $g s \cup$ set bs
$\cup$ set $h s) \longrightarrow$
unique-idx (gs @ bs @ hs) data $\longrightarrow$ is-Groebner-basis (fst'set gs) $\longrightarrow$
$(q-i n-b s \longrightarrow(q \in$ set $g s \cup$ set $b s)) \longrightarrow$
$\left(\forall p^{\prime} q^{\prime} .\left(p^{\prime}, q^{\prime}\right) \in_{p}\right.$ snd ' set $p s \longrightarrow$ fst $p^{\prime} \neq 0 \longrightarrow$ fst $q^{\prime} \neq 0 \longrightarrow$ crit-pair-cbelow-on d $\left.m(f s t ' B)\left(f s t p^{\prime}\right)\left(f s t q^{\prime}\right)\right) \longrightarrow$
$\left(\forall p^{\prime} q^{\prime} \cdot p^{\prime} \in\right.$ set $g s \cup$ set bs $\longrightarrow q^{\prime} \in$ set $g s \cup$ set $b s \longrightarrow f s t p^{\prime} \neq 0 \longrightarrow$ fst $q^{\prime} \neq 0 \longrightarrow$
crit-pair-cbelow-on d $\left.m\left(f s t t^{\prime} B\right)\left(f s t p^{\prime}\right)\left(f s t q^{\prime}\right)\right) \longrightarrow$

$$
p \in \text { set } h s \longrightarrow q \in \text { set } g s \cup \text { set } b s \cup \text { set hs } \longrightarrow \text { fst } p \neq 0 \longrightarrow \text { fst } q \neq 0
$$

$\qquad$
crit data gs bs hs $q$-in-bs ps $p q \longrightarrow$ crit-pair-cbelow-on d $m(f s t$ ' $B)(f s t p)(f s t ~ q))$
definition ocrit-spec :: ('t, 'b::field, 'c, 'd) ocrit $T \Rightarrow$ bool
where ocrit-spec crit $\longleftrightarrow$
$\left(\forall d m\right.$ data hs ps $B$ p q. dickson-grading $d \longrightarrow$ set $h s \subseteq B \longrightarrow f s t{ }^{\prime} B \subseteq$ dgrad-p-set $d m \longrightarrow$
unique-idx $(p \# q \# h s @(\operatorname{map}(f s t \circ s n d) p s) @(m a p(s n d \circ s n d)$ ps)) data $\longrightarrow$

$$
\begin{aligned}
& \left(\forall p^{\prime} q^{\prime} \cdot\left(p^{\prime}, q^{\prime}\right) \in_{p} \text { snd 'set } p s \longrightarrow \text { fst } p^{\prime} \neq 0 \longrightarrow \text { fst } q^{\prime} \neq 0 \longrightarrow\right. \\
& \left.\quad \text { crit-pair-cbelow-ond } m(\text { fst ‘ } B)\left(\text { fst } p^{\prime}\right)\left(\text { fst } q^{\prime}\right)\right) \longrightarrow \\
& p \in B \longrightarrow q \in B \longrightarrow \text { fst } p \neq 0 \longrightarrow \text { fst } q \neq 0 \longrightarrow \\
& \text { crit data hs ps p } \left.q \longrightarrow \text { crit-pair-cbelow-on d } m\left(\text { fst }{ }^{\prime} B\right)(\text { fst } p)(\text { fst } q)\right)
\end{aligned}
$$

Criteria satisfying ncrit-spec can be used for discarding new pairs by reference to new and old elements, whereas criteria satisfying ocrit-spec can be used for discarding old pairs by reference to new elements only (no existing ones!). The chain criterion satisfies both ncrit-spec and ocrit-spec.

## lemma icrit-specI:

assumes $\bigwedge d m$ data $g s$ bs hs $p q$.
dickson-grading $d \Longrightarrow f s t$ ' (set gs $\cup$ set $b s \cup$ set $h s) \subseteq d g r a d-p-s e t d m$
$\Longrightarrow$
unique-idx (gs @ bs @ hs) data $\Longrightarrow$ is-Groebner-basis (fst'set gs) $\Longrightarrow$ $p \in$ set $h s \Longrightarrow q \in$ set $g s \cup$ set $b s \cup$ set $h s \Longrightarrow f s t p \neq 0 \Longrightarrow f s t q \neq 0$
$\Longrightarrow$
crit data gs bs hs p $q \Longrightarrow$
crit-pair-cbelow-on d $m(f s t$ ' $($ set gs $\cup$ set $b s \cup$ set hs $))(f s t p)(f s t q)$
shows icrit-spec crit
unfolding icrit-spec-def using assms by auto
lemma icrit-spec $D$ :
assumes icrit-spec crit and dickson-grading d
and $f s t$ ' $($ set gs $\cup$ set $b s \cup$ set $h s) \subseteq$ dgrad- $p$-set $d m$ and unique-idx (gs @bs @ hs) data
and is-Groebner-basis (fst'set gs) and $p \in$ set hs and $q \in$ set gs $\cup$ set bs $\cup$ set hs
and fst $p \neq 0$ and fst $q \neq 0$ and crit data gs bs hs $p q$
shows crit-pair-cbelow-on d $m(f s t$ ' (set gs $\cup$ set $b s \cup$ set hs) $)(f s t p)(f s t q)$
using assms unfolding icrit-spec-def by blast
lemma ncrit-specI:
assumes $\bigwedge d m$ data gs bs hs ps $B q$-in-bs $p q$.
dickson-grading $d \Longrightarrow$ set $g s \cup$ set bs $\cup$ set $h s \subseteq B \Longrightarrow$
fst ' $B \subseteq$ dgrad- $p$-set $d m \Longrightarrow$ snd' set $p s \subseteq$ set $h s \times($ set $g s \cup$ set bs
$\cup$ set $h s) \Longrightarrow$
unique-idx (gs @ bs @ hs) data $\Longrightarrow$ is-Groebner-basis (fst'set gs) $\Longrightarrow$ $(q$-in-bs $\longrightarrow q \in$ set $g s \cup$ set $b s) \Longrightarrow$
$\left(\bigwedge p^{\prime} q^{\prime} .\left(p^{\prime}, q^{\prime}\right) \in_{p}\right.$ snd ' set $p s \Longrightarrow f s t p^{\prime} \neq 0 \Longrightarrow$ fst $q^{\prime} \neq 0 \Longrightarrow$ crit-pair-cbelow-on dm $\left.(f s t ' B)\left(f s t p^{\prime}\right)\left(f s t q^{\prime}\right)\right) \Longrightarrow$
$\left(\bigwedge p^{\prime} q^{\prime} . p^{\prime} \in\right.$ set $g s \cup$ set $b s \Longrightarrow q^{\prime} \in$ set $g s \cup$ set $b s \Longrightarrow f s t p^{\prime} \neq 0 \Longrightarrow$ fst $q^{\prime} \neq 0 \Longrightarrow$
crit-pair-cbelow-on d m (fst ' B) (fst $\left.p^{\prime}\right)(f s t q$ ' $\left.)\right) \Longrightarrow$
$p \in$ set $h s \Longrightarrow q \in$ set $g s \cup$ set $b s \cup$ set $h s \Longrightarrow f s t p \neq 0 \Longrightarrow f s t q \neq 0$
$\Longrightarrow$
crit data gs bs hs $q$-in-bs ps $p q \Longrightarrow$
crit-pair-cbelow-on dm(fst ' B) (fst p) (fst q)
shows ncrit-spec crit
unfolding ncrit-spec-def by (intro allI impI, rule assms, assumption+, meson, meson, assumption+)
lemma ncrit-specD:
assumes ncrit-spec crit and dickson-grading $d$ and set gs $\cup$ set bs $\cup$ set hs $\subseteq B$ and $f s t$ ' $B \subseteq$ dgrad- $p$-set $d m$ and snd ' set $p s \subseteq$ set $h s \times($ set gs $\cup$ set bs $\cup$ set hs)
and unique-idx (gs @ bs @ hs) data and is-Groebner-basis (fst'set gs)
and $q$-in-bs $\Longrightarrow q \in$ set $g s \cup$ set $b s$
and $\bigwedge p^{\prime} q^{\prime} .\left(p^{\prime}, q^{\prime}\right) \in_{p}$ snd ' set $p s \Longrightarrow$ fst $p^{\prime} \neq 0 \Longrightarrow$ fst $q^{\prime} \neq 0 \Longrightarrow$ crit-pair-cbelow-on d $m$ ( $f s t$ ' $B)\left(f s t p^{\prime}\right)(f s t q$ ')
and $\wedge p^{\prime} q^{\prime} \cdot p^{\prime} \in$ set $g s \cup$ set bs $\Longrightarrow q^{\prime} \in$ set gs $\cup$ set bs $\Longrightarrow f s t p^{\prime} \neq 0 \Longrightarrow f s t$ $q^{\prime} \neq 0 \Longrightarrow$ crit-pair-cbelow-on dm(fst 'B) $\left(f s t p^{\prime}\right)(f s t q$ ')
and $p \in$ set hs and $q \in$ set $g s \cup$ set $b s \cup$ set hs and fst $p \neq 0$ and fst $q \neq 0$ and crit data gs bs hs q-in-bs ps $p q$
shows crit-pair-cbelow-on dm (fst ' B) (fst p) (fst q)
using assms unfolding ncrit-spec-def by blast
lemma ocrit-specI:
assumes $\wedge d m$ data hs $p s B p q$.
dickson-grading $d \Longrightarrow$ set $h s \subseteq B \Longrightarrow f_{s t}$ ' $B \subseteq$ dgrad- $p$-set $d m \Longrightarrow$
unique-idx ( $p$ \# q \# hs @ (map (fst $\circ$ snd) ps) @ (map (snd $\circ$ snd)
ps)) data $\Longrightarrow$

$$
\begin{aligned}
& \left(\bigwedge p^{\prime} q^{\prime} \cdot\left(p^{\prime}, q^{\prime}\right) \in_{p} \text { snd ' set } p s \Longrightarrow f s t p^{\prime} \neq 0 \Longrightarrow f s t q^{\prime} \neq 0 \Longrightarrow\right. \\
& \text { crit-pair-cbelow-on dm(fst 'B) (fst } \left.\left.p^{\prime}\right)\left(f s t q^{\prime}\right)\right) \Longrightarrow \\
& p \in B \Longrightarrow q \in B \Longrightarrow f s t p \neq 0 \Longrightarrow f s t q \neq 0 \Longrightarrow \\
& \text { crit data hs ps p } q \Longrightarrow \text { crit-pair-cbelow-on d } m \text { (fst ' } B \text { ) (fst } p \text { ) (fst } q \text { ) }
\end{aligned}
$$

shows ocrit-spec crit
unfolding ocrit-spec-def by (intro allI impI, rule assms, assumption+, meson, assumption+)
lemma ocrit-specD:
assumes ocrit-spec crit and dickson-grading $d$ and set $h s \subseteq B$ and $f s t$ ' $B \subseteq$ dgrad-p-set dm
and unique-idx $(p \# q \# h s @(m a p(f s t \circ s n d) p s) @(m a p(s n d \circ s n d) p s))$ data

$$
\text { and } \wedge p^{\prime} q^{\prime} \cdot\left(p^{\prime}, q^{\prime}\right) \in_{p} \text { snd ' set } p s \Longrightarrow \text { fst } p^{\prime} \neq 0 \Longrightarrow f_{s t} q^{\prime} \neq 0 \Longrightarrow
$$ crit-pair-cbelow-on dm( $\left.f_{s t}{ }^{\prime} B\right)\left(f_{s t} p^{\prime}\right)\left(f s t q{ }^{\prime}\right)$

and $p \in B$ and $q \in B$ and fst $p \neq 0$ and fst $q \neq 0$
and crit data hs ps $p q$
shows crit-pair-cbelow-on d m (fst'B) (fst p) (fst q)
using assms unfolding ocrit-spec-def by blast

### 6.3.2 Suitable instances of the crit parameters

definition component-crit :: ('t, 'b::zero, ' $c$, 'd) icritT
where component-crit data gs bs hs p $q \longleftrightarrow$ (component-of-term (lt (fst p)) $\neq$ component-of-term (lt (fst q)))
lemma icrit-spec-component-crit: icrit-spec (component-crit::('t, 'b::field, ' $c,{ }^{\prime} d$ )
icritT)
proof (rule icrit-specI)
fix $d m$ and data::nat $\times{ }^{\prime} d$ and $g s$ bs $h s$ and $p q::\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right) p d a t a$
assume component-crit data gs bs hs $p q$
hence component-of-term $(l t(f s t ~ p)) \neq$ component-of-term (lt (fst q))
by (simp add: component-crit-def)
thus crit-pair-cbelow-on dm(fst' (set gs $\cup$ set $b s \cup$ set hs $)$ ) $(f s t p)(f s t q)$
by (rule crit-pair-cbelow-distinct-component)
qed
The product criterion is only applicable to scalar polynomials.
definition product-crit :: ('a, 'b::zero, 'c, 'd) icritT
where product-crit data gs bs hs p $q \longleftrightarrow$ (gcs (punit.lt (fst p)) (punit.lt (fst q))
$=0$ )
lemma (in gd-term) icrit-spec-product-crit: punit.icrit-spec (product-crit::('a, 'b::field, ' $\left.c,{ }^{\prime} d\right)$ icrit $\left.T\right)$
proof (rule punit.icrit-specI)
fix $d m$ and data::nat $\times{ }^{\prime} d$ and $g s$ bs $h s$ and $p q::\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right) p d a t a$
assume product-crit data gs bs hs $p q$
hence *: gcs (punit.lt (fst p)) (punit.lt $(f s t ~ q))=0$ by (simp only: prod-
uct-crit-def)
assume $p \in$ set $h s$ and $q-i n: q \in$ set gs $\cup$ set $b s \cup$ set $h s($ is $-\in ? B)$
assume dickson-grading $d$ and sub: fst' ' set gs $\cup$ set bs $\cup$ set hs $) \subseteq$ punit.dgrad-p-set d m
moreover from $\langle p \in$ set $h s\rangle$ have $f s t p \in f s t$ '? $B$ by simp
moreover from $q$-in have fst $q \in f_{s t}$ '?B by simp
moreover assume fst $p \neq 0$ and fst $q \neq 0$
ultimately show punit.crit-pair-cbelow-on d $m$ (fst ‘?B) (fst p) (fst q) using * by (rule product-criterion)
qed
component-crit and product-crit ignore the data parameter.
fun (in -) pair-in-list :: (bool $\times\left({ }^{\prime} a,^{\prime} b,{ }^{\prime} c\right)$ pdata-pair) list $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool where
pair-in-list [] - = False
|pair-in-list ((-, (-, $\left.\left.\left.i^{\prime},-\right),\left(-, j^{\prime},-\right)\right) \# p s\right) i j=$

$$
\left(\left(i=i^{\prime} \wedge j=j^{\prime}\right) \vee\left(i=j^{\prime} \wedge j=i^{\prime}\right) \vee \text { pair-in-list ps } i j\right)
$$

lemma (in -) pair-in-listE:

## assumes pair-in-list ps ij

obtains $p q a b$ where $((p, i, a),(q, j, b)) \in_{p}$ snd'set $p s$
using assms
proof (induct ps ij arbitrary: thesis rule: pair-in-list.induct)
case ( $1 i j$ )
from 1 (2) show ? case by simp
next
case (2 c p $i^{\prime}$ aqj $j^{\prime} b$ ps $\left.i j\right)$
from 2(3) have $\left(i=i^{\prime} \wedge j=j^{\prime}\right) \vee\left(i=j^{\prime} \wedge j=i^{\prime}\right) \vee$ pair-in-list ps $i j$ by simp
thus? case
proof (elim disjE conjE)
assume $i=i^{\prime}$ and $j=j^{\prime}$
have $((p, i, a),(q, j, b)) \in_{p}$ snd' set $\left(\left(c,\left(p, i^{\prime}, a\right), q, j^{\prime}, b\right) \# p s\right)$
unfolding $\left\langle i=i^{\prime}\right\rangle\left\langle j=j^{\prime}\right\rangle$ in-pair-iff by fastforce
thus ?thesis by (rule 2(2))

## next

            assume \(i=j^{\prime}\) and \(j=i^{\prime}\)
            have \(((q, i, b),(p, j, a)) \in_{p}\) snd ' set \(\left(\left(c,\left(p, i^{\prime}, a\right), q, j^{\prime}, b\right) \# p s\right)\)
                unfolding \(\left\langle i=j^{\prime}\right\rangle\left\langle j=i^{\prime}\right\rangle\) in-pair-iff by fastforce
        thus ?thesis by (rule 2(2))
    next
        assume pair-in-list ps \(i j\)
        obtain \(p^{\prime} q^{\prime} a^{\prime} b^{\prime}\) where \(\left(\left(p^{\prime}, i, a^{\prime}\right),\left(q^{\prime}, j, b^{\prime}\right)\right) \in_{p}\) snd ' set \(p s\)
            by (rule 2(1), assumption, rule 〈pair-in-list ps ij〉)
        also have \(\ldots \subseteq\) snd ' set \(\left(\left(c,\left(p, i^{\prime}, a\right), q, j^{\prime}, b\right) \# p s\right)\) by auto
        finally show ?thesis by (rule 2(2))
    qed
    qed
definition chain-ncrit :: ('t, 'b::zero, 'c, 'd) ncritT
where chain-ncrit data gs bs hs q-in-bs ps p $q \longleftrightarrow$
(let $v=l t(f s t p) ; l=$ term-of-pair (lcs (pp-of-term $v)(l p(f s t q))$, component-of-term $v$ );
$i=f s t(s n d p) ; j=f s t(s n d q)$ in
( $\exists r \in$ set gs. let $k=$ fst $($ snd $r)$ in
$k \neq i \wedge k \neq j \wedge l t(f s t r) a d d s_{t} l \wedge$ pair-in-list ps $i k \wedge(q-i n-b s \vee$
pair-in-list ps $j k) \wedge$ fst $r \neq 0) \vee$
( $\exists r \in$ set bs. let $k=$ fst (snd $r$ ) in
$k \neq i \wedge k \neq j \wedge l t(f s t r) a d d s_{t} l \wedge$ pair-in-list ps $i k \wedge(q-i n-b s \vee$
pair-in-list ps $j k) \wedge f s t r \neq 0) \vee$
( $\exists$ heset hs. let $k=$ fst $($ snd $h)$ in
$k \neq i \wedge k \neq j \wedge l t(f s t h) a d d s_{t} l \wedge$ pair-in-list ps $i k \wedge$ pair-in-list ps $j k \wedge$ fst $h \neq 0)$ )
definition chain-ocrit :: ('t, 'b::zero, 'c, 'd) ocritT
where chain-ocrit data hs ps p $q \longleftrightarrow$
$(l e t v=l t(f s t p) ; l=$ term-of-pair $(l c s(p p-o f-t e r m v)(l p(f s t q))$, component-of-term $v$ );
$i=f s t(s n d p) ; j=f s t(s n d q)$ in
( $\exists$ heset hs. let $k=$ fst (snd $h$ ) in
$k \neq i \wedge k \neq j \wedge l t(f s t h) a d d s_{t} l \wedge$ pair-in-list ps $i k \wedge$ pair-in-list ps $j k \wedge f s t h \neq 0)$ )
chain-ncrit and chain-ocrit ignore the data parameter.
lemma chain-ncritE:
assumes chain-ncrit data gs bs hs q-in-bs ps p $q$ and snd' set $p s \subseteq$ set $h s \times$ (set gs $\cup$ set $b s \cup$ set $h s)$
and unique-idx (gs @bs @ hs)data and $p \in$ set hs and $q \in$ set gs $\cup$ set $b s \cup$ set hs
obtains $r$ where $r \in$ set $g s \cup$ set $b s \cup$ set hs and $f s t r \neq 0$ and $r \neq p$ and $r$ $\neq q$
and $l t(f s t r) a d d s_{t}$ term-of-pair (lcs (lp (fst p)) (lp (fst q)), component-of-term $(l t(f s t p)))$
and $(p, r) \in_{p}$ snd' set $p s$ and $(r \in$ set $g s \cup$ set $b s \wedge q$-in-bs $) \vee(q, r) \in_{p}$ snd ' set ps
proof -
let ?l $=$ term-of-pair $(l c s(l p(f s t p))(l p(f s t q))$, component-of-term $(l t(f s t p)))$
let $? i=f s t($ snd $p)$
let $? j=f s t(s n d q)$
let ? $x s=g s$ @ bs @ hs
have 3: $x \in$ set ? $x$ if $(x, y) \in_{p}$ snd' set $p s$ for $x y$
proof -
note that
also have $s n d$ ' set $p s \subseteq$ set $h s \times($ set $g s \cup$ set $b s \cup$ set hs) by (fact assms(2)) also have $\ldots \subseteq($ set $g s \cup$ set bs $\cup$ set $h s) \times($ set $g s \cup$ set $b s \cup$ set $h s)$ by fastforce
finally have $(x, y) \in($ set $g s \cup$ set $b s \cup$ set $h s) \times($ set $g s \cup$ set $b s \cup$ set $h s)$
by (simp only: in-pair-same)
thus ?thesis by simp
qed
have 4: $x \in$ set ? $x s$ if $(y, x) \in_{p}$ snd' set ps for $x y$
proof -
from that have $(x, y) \in_{p}$ snd' set ps by (simp add: in-pair-iff disj-commute) thus ?thesis by (rule 3)
qed
from $\operatorname{assms}(1)$ have
$\exists r \in$ set $g s \cup$ set bs $\cup$ set hs. let $k=f s t(s n d r)$ in $k \neq ? i \wedge k \neq ? j \wedge l t(f s t r)$ addst $? l \wedge$ pair-in-list ps ? $i k \wedge$ $((r \in$ set $g s \cup$ set $b s \wedge q$-in-bs $) \vee$ pair-in-list ps ?j $k) \wedge f s t r \neq 0$
by (smt UnI1 chain-ncrit-def sup-commute)
then obtain $r$ where $r$-in: $r \in$ set $g s \cup$ set $b s \cup$ set $h s$ and $f s t r \neq 0$ and $r p$ : fst (snd $r) \neq ? i$
and $r q: f_{s t}(s n d r) \neq ? j$ and $l t(f s t r) a d d s_{t} ? l$
and 1: pair-in-list ps ?i $(f s t(s n d r))$
and 2: $(r \in$ set gs $\cup$ set bs $\wedge q-i n-b s) \vee$ pair-in-list ps? $j(f s t(s n d r))$
unfolding Let-def by blast
let $? k=f s t(s n d r)$
note $r$-in $\langle f s t r \neq 0\rangle$
moreover from $r p$ have $r \neq p$ by auto
moreover from $r q$ have $r \neq q$ by auto
ultimately show ?thesis using 〈lt (fst r) addst ?l>
proof
from 1 obtain $p^{\prime} r^{\prime} a b$ where $*:\left(\left(p^{\prime}, ? i, a\right),\left(r^{\prime}, ? k, b\right)\right) \in_{p}$ snd' set $p s$ by (rule pair-in-listE)
note assms(3)
moreover from $*$ have $\left(p^{\prime}, ? i, a\right) \in$ set ? xs by (rule 3)
moreover from $\operatorname{assms}(4)$ have $p \in$ set ?xs by simp
moreover have $f s t\left(\right.$ snd $\left.\left(p^{\prime}, ? i, a\right)\right)=$ ? $i$ by simp
ultimately have $p^{\prime}:\left(p^{\prime}, ? i, a\right)=p$ by (rule unique-idxD1)
note assms(3)
moreover from $*$ have $\left(r^{\prime}, ? k, b\right) \in$ set ?xs by (rule 4)
moreover from $r$-in have $r \in$ set ? xs by simp
moreover have $f s t\left(s n d\left(r^{\prime}, ? k, b\right)\right)=? k$ by $\operatorname{simp}$
ultimately have $r^{\prime}:\left(r^{\prime}, ? k, b\right)=r$ by (rule unique-idxD1)
from $*$ show $(p, r) \in_{p}$ snd ' set $p s$ by (simp only: $p^{\prime} r^{\prime}$ )
next
from 2 show $(r \in$ set $g s \cup$ set $b s \wedge q-i n-b s) \vee(q, r) \in_{p}$ snd' set $p s$
proof
assume $r \in$ set $g s \cup$ set bs $\wedge q$-in-bs
thus ?thesis..
next
assume pair-in-list ps? ? ?k
then obtain $q^{\prime} r^{\prime} a b$ where $*:\left(\left(q^{\prime}, ? j, a\right),\left(r^{\prime}, ? k, b\right)\right) \in_{p}$ snd' set $p s$ by (rule pair-in-listE)
note assms(3)
moreover from $*$ have $\left(q^{\prime}, ? j, a\right) \in$ set ?xs by (rule 3)
moreover from $\operatorname{assms}(5)$ have $q \in$ set ?xs by simp
moreover have $f s t\left(\right.$ snd $\left.\left(q^{\prime}, ? j, a\right)\right)=? j$ by simp
ultimately have $q^{\prime}:\left(q^{\prime}, ? j, a\right)=q$ by (rule unique-idxD1)
note assms(3)
moreover from * have ( $\left.r^{\prime}, ? k, b\right) \in$ set ? xs by (rule 4)
moreover from $r$-in have $r \in$ set ? xs by simp
moreover have $f$ st (snd $\left.\left(r^{\prime}, ? k, b\right)\right)=? k$ by simp
ultimately have $r^{\prime}:\left(r^{\prime}, ? k, b\right)=r$ by (rule unique-idxD1)
from * have $(q, r) \in_{p}$ snd ' set $p$ s by (simp only: $q^{\prime} r$ ')
thus ?thesis ..

```
qed qed qed
```

lemma chain-ocritE:
assumes chain-ocrit data hs ps p q
and unique-idx $(p \# q \#$ hs @ (map $(f s t \circ s n d) p s) @(m a p(s n d \circ s n d) p s))$
data (is unique-idx ? $x s$-)
obtains $h$ where $h \in$ set $h s$ and $f s t h \neq 0$ and $h \neq p$ and $h \neq q$
and $l t(f s t h)$ addst term-of-pair (lcs (lp (fst p)) (lp (fst q)), component-of-term
(lt $\left.\left(f_{s t} p\right)\right)$ )
and $(p, h) \in_{p}$ snd' set $p s$ and $(q, h) \in_{p}$ snd'set $p s$
proof -
let ?l $=$ term-of-pair $(l c s(l p(f s t p))(l p(f s t q))$, component-of-term $(l t(f s t ~ p)))$
have 3: $x \in$ set ? $x$ s if $(x, y) \in_{p}$ snd' set $p s$ for $x y$
proof -
from that have $(x, y) \in$ snd' set $p s \vee(y, x) \in$ snd' set ps by (simp only: in-pair-iff)
thus ?thesis
proof
assume $(x, y) \in$ snd' set $p s$
hence $f s t(x, y) \in f_{s t}$ ' snd ' set ps by fastforce
thus ?thesis by (simp add: image-comp)
next
assume $(y, x) \in$ snd ' set ps
hence snd $(y, x) \in$ snd ' snd ' set ps by fastforce
thus ?thesis by (simp add: image-comp)
qed
qed
have 4: $x \in$ set ? $x s$ if $(y, x) \in_{p}$ snd' set $p s$ for $x y$
proof -
from that have $(x, y) \in_{p}$ snd' set ps by (simp add: in-pair-iff disj-commute)
thus ?thesis by (rule 3)
qed
from $\operatorname{assms}(1)$ obtain $h$ where $h \in$ set $h s$ and fst $h \neq 0$ and $h p: f s t(s n d h)$ $\neq$ fst (snd $p$ )
and $h q: f s t(s n d h) \neq f s t(s n d q)$ and $l t(f s t h) a d d s_{t} ? l$
and 1: pair-in-list ps $(f s t($ snd $p))(f s t(s n d h))$ and 2: pair-in-list ps (fst (snd
q)) (fst (snd h))
unfolding chain-ocrit-def Let-def by blast
let $? i=f s t(s n d p)$
let $? j=f s t(s n d q)$
let $? k=f s t(s n d h)$
note $\langle h \in$ set $h s\rangle\langle f s t h \neq 0\rangle$
moreover from $h p$ have $h \neq p$ by auto
moreover from $h q$ have $h \neq q$ by auto
ultimately show ?thesis using <lt (fst h) addst ?l>
proof

```
    from 1 obtain \(p^{\prime} h^{\prime} a b\) where \(*:\left(\left(p^{\prime}, ? i, a\right),\left(h^{\prime}, ? k, b\right)\right) \in_{p}\) snd 'set \(p s\)
    by (rule pair-in-listE)
    note assms(2)
    moreover from \(*\) have \(\left(p^{\prime}\right.\), ? \(\left.i, a\right) \in\) set ?xs by (rule 3)
    moreover have \(p \in\) set ?xs by simp
    moreover have \(f s t\left(\right.\) snd \(\left.\left(p^{\prime}, ?, i, a\right)\right)=? i\) by simp
    ultimately have \(p^{\prime}:\left(p^{\prime}, ? i, a\right)=p\) by (rule unique-idxD1)
    note assms(2)
    moreover from \(*\) have \(\left(h^{\prime}, ? k, b\right) \in\) set ?xs by (rule 4)
    moreover from \(\langle h \in\) set \(h s\rangle\) have \(h \in\) set ?xs by simp
    moreover have \(f s t\left(s n d\left(h^{\prime}, ? k, b\right)\right)=? k\) by simp
    ultimately have \(h^{\prime}:\left(h^{\prime}, ? k, b\right)=h\) by (rule unique-idxD1)
    from \(*\) show \((p, h) \in_{p}\) snd ' set ps by (simp only: \(\left.p^{\prime} h^{\prime}\right)\)
    next
    from 2 obtain \(q^{\prime} h^{\prime} a b\) where \(*:\left(\left(q^{\prime}, ? j, a\right),\left(h^{\prime}, ? k, b\right)\right) \in_{p}\) snd'set ps
        by (rule pair-in-listE)
    note assms(2)
    moreover from \(*\) have \(\left(q^{\prime}, ? j, a\right) \in\) set ? \(x s\) by (rule 3)
    moreover have \(q \in\) set ?xs by simp
    moreover have \(f s t\left(\right.\) snd \(\left.\left(q^{\prime}, ? j, a\right)\right)=? j\) by simp
    ultimately have \(q^{\prime}:\left(q^{\prime}, ? j, a\right)=q\) by (rule unique-idxD1)
    note assms(2)
    moreover from \(*\) have \(\left(h^{\prime}, ? k, b\right) \in\) set ?xs by (rule 4)
    moreover from \(\langle h \in\) set \(h s\rangle\) have \(h \in\) set ?xs by simp
    moreover have \(f s t\left(s n d\left(h^{\prime}, ? k, b\right)\right)=? k\) by simp
    ultimately have \(h^{\prime}:\left(h^{\prime}, ? k, b\right)=h\) by (rule unique-idxD1)
    from \(*\) show \((q, h) \in_{p}\) snd ' set ps by (simp only: \(\left.q^{\prime} h^{\prime}\right)\)
    qed
qed
lemma ncrit-spec-chain-ncrit: ncrit-spec (chain-ncrit::('t, 'b::field, 'c, 'd) ncritT)
proof (rule ncrit-specI)
    fix \(d m\) and data::nat \(\times{ }^{\prime} d\) and \(g s\) bs \(h s\) and \(p s::\left(b o o l \times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)\right.\) pdata-pair \()\)
list
    and \(B q\)-in-bs and \(p q::\left(' t,{ }^{\prime} b,{ }^{\prime} c\right)\) pdata
    assume dg: dickson-grading \(d\) and \(B\)-sup: set \(g s \cup\) set \(b s \cup\) set \(h s \subseteq B\)
        and \(B\)-sub: \(f s t\) ' \(B \subseteq d g r a d-p\)-set \(d m\) and \(q\)-in-bs: \(q\)-in-bs \(\longrightarrow q \in\) set \(g s \cup\) set
bs
    and \(1: \bigwedge p^{\prime} q^{\prime} .\left(p^{\prime}, q^{\prime}\right) \in_{p}\) snd ' set \(p s \Longrightarrow f s t p^{\prime} \neq 0 \Longrightarrow f s t q^{\prime} \neq 0 \Longrightarrow\)
                crit-pair-cbelow-on d \(m\) ( \(f\) st ' \(B)\left(f s t p^{\prime}\right)\left(f s t q^{\prime}\right)\)
    and 2: \(\bigwedge p^{\prime} q^{\prime} \cdot p^{\prime} \in\) set \(g s \cup\) set \(b s \Longrightarrow q^{\prime} \in\) set \(g s \cup\) set \(b s \Longrightarrow f s t p^{\prime} \neq 0 \Longrightarrow\)
fst \(q^{\prime} \neq 0 \Longrightarrow\)
        crit-pair-cbelow-on d \(m\left(f s t{ }^{\prime} B\right)\left(f s t p^{\prime}\right)\left(f s t q^{\prime}\right)\)
```

and fst $p \neq 0$ and $f s t q \neq 0$
let $? l=$ term－of－pair $(l c s(l p(f s t p))(l p(f s t q))$ ，component－of－term $(l t(f s t p)))$
assume chain－ncrit data gs bs hs $q$－in－bs ps $p q$ and snd＇set ps $\subseteq$ set hs $\times$（set $g s \cup$ set $b s \cup$ set $h s)$ and
unique－idx（ $g s$＠bs＠hs）data and $p \in$ set $h s$ and $q \in$ set $g s \cup$ set $b s \cup$ set hs then obtain $r$ where $r \in$ set $g s \cup$ set bs $\cup$ set hs and fst $r \neq 0$ and $r \neq p$ and $r \neq q$
and adds：lt（fst r）addst ？l and $(p, r) \in_{p}$ snd＇set ps
and disj：$(r \in$ set $g s \cup$ set $b s \wedge q$－in－bs $) \vee(q, r) \in_{p}$ snd＇set ps by（rule chain－ncritE）
note $d g B$－sub
moreover from $\langle p \in$ set $h s\rangle\langle q \in$ set $g s \cup$ set $b s \cup$ set $h s\rangle B$－sup
have $f s t p \in f_{s t}$＇$B$ and $f s t ~ q \in f_{s t}$＇$B$
by auto
moreover note $\langle f s t p \neq 0\rangle\langle f s t q \neq 0\rangle$
moreover from adds have lp（fst r）adds lcs（lp（fst p））（lp（fst q））
by（simp add：adds－term－def term－simps）
moreover from $a d d s$ have component－of－term $(l t(f s t r))=$ component－of－term （lt（fst p））
by（simp add：adds－term－def term－simps）
ultimately show crit－pair－cbelow－on d $m$（fst＇B）（fst p）（fst q）
proof（rule chain－criterion）
from $\left\langle(p, r) \in_{p}\right.$ snd＇set ps〉〈fst $\left.p \neq 0\right\rangle\langle f s t r \neq 0\rangle$
show crit－pair－cbelow－on $d m(f s t ' B)(f s t p)(f s t r)$ by（rule 1）
next
from disj show crit－pair－cbelow－on d $m(f s t ' B)(f s t r)(f s t ~ q)$
proof
assume $r \in$ set $g s \cup$ set bs $\wedge q$－in－bs
hence $r \in$ set $g s \cup$ set bs and $q$－in－bs by simp－all
from $q$－in－bs this（2）have $q \in$ set $g s \cup$ set bs．．
with $\langle r \in$ set $g s \cup$ set bs〉show ？thesis using $\langle f s t r \neq 0\rangle\langle f s t q \neq 0\rangle$ by
（rule 2）
next
assume $(q, r) \in_{p}$ snd＇set $p s$
hence $(r, q) \in_{p}$ snd＇set ps by（simp only：in－pair－iff disj－commute）
thus ？thesis using $\langle f$ st $r \neq 0\rangle\langle$ fst $q \neq 0\rangle$ by（rule 1 ）
qed
qed
qed
lemma ocrit－spec－chain－ocrit：ocrit－spec（chain－ocrit：：（＇t，＇b：：field，＇$c,{ }^{\prime} d$ ）ocritT） proof（rule ocrit－specI）
fix $d m$ and data：：nat $\times{ }^{\prime} d$ and $h s::\left(' t,{ }^{\prime} b, ' c\right)$ pdata list and $p s::\left(b o o l \times\left(' t,{ }^{\prime} b\right.\right.$ ，
＇c）pdata－pair）list
and $B$ and $p q::\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata
assume $d g$ ：dickson－grading $d$ and $B$－sup：set $h s \subseteq B$
and $B$－sub：$f_{s t}{ }^{\prime} B \subseteq d g r a d-p$－set $d m$
and 1：$\bigwedge p^{\prime} q^{\prime} .\left(p^{\prime}, q^{\prime}\right) \in_{p}$ snd＇set $p s \Longrightarrow f s t p^{\prime} \neq 0 \Longrightarrow f s t q^{\prime} \neq 0 \Longrightarrow$
crit－pair－cbelow－on d m（fst＇B）（fst $\left.p^{\prime}\right)\left(f_{s t} q^{\prime}\right)$
and fst $p \neq 0$ and fst $q \neq 0$ and $p \in B$ and $q \in B$
let ?l $=$ term-of-pair $(l c s(l p(f s t ~ p))(l p(f s t q))$, component-of-term $(l t(f s t p)))$
assume chain-ocrit data hs ps p $q$ and unique-idx ( $p \# q \#$ hs @ map (fst $\circ$ snd) ps @ map (snd $\circ$ snd) ps) data
then obtain $h$ where $h \in$ set $h s$ and fst $h \neq 0$ and $h \neq p$ and $h \neq q$
and adds: lt (fst $h$ ) addst $? l$ and $(p, h) \in_{p}$ snd' set ps and $(q, h) \in_{p}$ snd' set ps
by (rule chain-ocritE)
note $d g B$-sub
moreover from $\langle p \in B\rangle\langle q \in B\rangle B$-sup
have $f s t p \in f s t$ ' $B$ and $f s t ~ q \in f s t$ ' $B$ by auto
moreover note $\langle f s t p \neq 0\rangle\langle f s t q \neq 0\rangle$
moreover from adds have lp (fst h) adds lcs (lp (fst p)) (lp (fst q))
by (simp add: adds-term-def term-simps)
moreover from adds have component-of-term (lt (fst h)) = component-of-term (lt (fst p))
by (simp add: adds-term-def term-simps)
ultimately show crit-pair-cbelow-on $d m\left(f_{s t}{ }^{\prime} B\right)(f s t p)(f s t q)$
proof (rule chain-criterion)
from $\left\langle(p, h) \in_{p}\right.$ snd ' set ps〉〈fst $\left.p \neq 0\right\rangle\langle f s t h \neq 0\rangle$
show crit-pair-cbelow-on d m (fst'B) (fst p) (fst h) by (rule 1)
next
from $\left\langle(q, h) \in_{p}\right.$ snd ' set $\left.p s\right\rangle$ have $(h, q) \in_{p}$ snd' set ps by (simp only: in-pair-iff disj-commute)
thus crit-pair-cbelow-on dm $\left(f_{s t}{ }^{\prime} B\right)(f$ st $h)(f s t q)$ using $\langle f s t h \neq 0\rangle\langle f s t q \neq$ 0 ) by (rule 1 ) qed
qed
lemma icrit-spec-no-crit: icrit-spec (( $\lambda----$. False) $::\left({ }^{\prime} t\right.$, 'b::field, ' $\left.c,{ }^{\prime} d\right)$ icritT $)$ by (rule icrit-specI, simp)
lemma ncrit-spec-no-crit: ncrit-spec (( $\lambda-\cdots-\operatorname{col}^{-}$. False)::('t, 'b::field, 'c, 'd) ncritT)
by (rule ncrit-specI, simp)
lemma ocrit-spec-no-crit: ocrit-spec (( $\lambda-\cdots-$-. False)::('t, 'b::field, ' $c,{ }^{\prime} d$ ) ocrit $\left.T\right)$ by (rule ocrit-specI, simp)

### 6.3.3 Creating Initial List of New Pairs

type-synonym (in -$)\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ aps $T=$ bool $\Rightarrow\left({ }^{\prime} t, ' b,{ }^{\prime} c\right)$ pdata list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b\right.$, 'c) pdata list $\Rightarrow$
$\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata $\Rightarrow\left(\right.$ bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair $)$ list
$\Rightarrow$
(bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair) list
type-synonym (in -$)\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right) n p T=\left(' t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
$\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$ nat $\times{ }^{\prime} d \Rightarrow$
$\left(\right.$ bool $\times\left({ }^{\prime} t,{ }^{\prime} b,,^{\prime} c\right)$ pdata-pair $)$ list
definition $n p$-spec :: $\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right) n p T \Rightarrow$ bool
where $n p$-spec $n p \longleftrightarrow(\forall$ gs bs hs data.
snd ' set (np gs bs hs data) $\subseteq$ set $h s \times($ set gs $\cup$ set bs $\cup$ set $h s) \wedge$
set hs $\times($ set gs $\cup$ set bs $) \subseteq$ snd' set (np gs bs hs data) $) \wedge$
$\left(\forall a b . a \in \operatorname{set} h s \longrightarrow b \in \operatorname{set} h s \longrightarrow a \neq b \longrightarrow(a, b) \in_{p}\right.$ snd ' set (np gs bs hs data)) $\wedge$
$(\forall p q .(\operatorname{True}, p, q) \in \operatorname{set}(n p$ gs bs hs data) $\longrightarrow q \in$ set gs
$\cup$ set $b s)$ )
lemma np-spec $I$ :
assumes $\bigwedge g s$ bs hs data.
snd ' set $(n p$ gs bs hs data $) \subseteq$ set hs $\times($ set gs $\cup$ set bs $\cup$ set hs $) \wedge$
set hs $\times($ set gs $\cup$ set $b s) \subseteq$ snd' set $(n p$ gs bs hs data $) \wedge$
$\left(\forall a b . a \in\right.$ set $h s \longrightarrow b \in$ set $h s \longrightarrow a \neq b \longrightarrow(a, b) \in_{p}$ snd'set ( $n p$ gs bs hs data)) ^
$(\forall p q .($ True $, p, q) \in \operatorname{set}(n p$ gs bs hs data $) \longrightarrow q \in$ set $g s \cup$ set bs $)$
shows $n p$-spec $n p$
unfolding $n p$-spec-def using assms by meson
lemma np-specD1:
assumes $n p$-spec $n p$
shows snd'set (np gs bs hs data) $\subseteq$ set hs $\times($ set gs $\cup$ set $b s \cup$ set hs $)$
using assms[unfolded np-spec-def, rule-format, of gs bs hs data] ..
lemma np-specD2:
assumes $n p$-spec $n p$
shows set $h s \times($ set gs $\cup$ set $b s) \subseteq$ snd' set ( $n p$ gs bs hs data)
using assms[unfolded np-spec-def, rule-format, of gs bs hs data] by auto
lemma $n p-s p e c D 3$ :
assumes $n p$-spec $n p$ and $a \in$ set $h s$ and $b \in$ set hs and $a \neq b$
shows $(a, b) \in_{p}$ snd' set ( $n p$ gs bs hs data)
using assms(1)[unfolded np-spec-def, rule-format, of gs bs hs data] assms(2,3,4) by blast

## lemma np-specD4:

assumes $n p$-spec $n p$ and (True, $p, q) \in \operatorname{set}(n p$ gs bs hs data)
shows $q \in$ set $g s \cup$ set bs
using assms(1)[unfolded np-spec-def, rule-format, of gs bs hs data] assms(2) by blast
lemma np-specE:
assumes np-spec $n p$ and $p \in$ set $h s$ and $q \in$ set gs $\cup$ set $b s \cup$ set hs and $p \neq q$
assumes 1: $\bigwedge q$-in-bs. $(q$-in-bs, $p, q) \in$ set $(n p$ gs bs hs data) $\Longrightarrow$ thesis
assumes 2: $\bigwedge p$-in-bs. $(p-i n-b s, q, p) \in$ set $(n p$ gs bs hs data) $\Longrightarrow$ thesis

```
    shows thesis
proof (cases q set gs U set bs)
    case True
    with assms(2) have ( p,q)\in set hs }\times(\mathrm{ set gs U set bs) by simp
    also from assms(1) have ...\subseteq snd'set (np gs bs hs data) by (rule np-specD2)
    finally obtain q-in-bs where ( q-in-bs, p,q)\in set (np gs bs hs data) by fastforce
    thus ?thesis by (rule 1)
next
    case False
    with assms(3) have q\in set hs by simp
    from assms(1,2) this assms(4) have (p,q) \inp snd' set (np gs bs hs data) by
(rule np-specD3)
    hence (p,q)\in snd 'set (np gs bs hs data) \vee ( }q,p)\in\mathrm{ snd ' set (np gs bs hs data)
    by (simp only: in-pair-iff)
    thus ?thesis
    proof
        assume (p,q)\in snd ' set (np gs bs hs data)
        then obtain q-in-bs where (q-in-bs,p,q)\in set (np gs bs hs data) by fastforce
        thus ?thesis by (rule 1)
    next
        assume (q, p)\in snd' set (np gs bs hs data)
    then obtain p-in-bs where (p-in-bs,q,p)\in set (np gs bs hs data) by fastforce
    thus ?thesis by (rule 2)
    qed
qed
definition add-pairs-single-naive :: 'd }=>('t,'b::zero,'c) aps
    where add-pairs-single-naive data flag gs bs h ps = ps @ (map (\lambdag.(flag,h,g))
gs)@ (map (\lambdab. (flag, h, b)) bs)
lemma set-add-pairs-single-naive:
    set (add-pairs-single-naive data flag gs bs h ps)= set ps \cupPair flag'({h} }\times(\mathrm{ set
gs U set bs))
    by (auto simp add: add-pairs-single-naive-def Let-def)
fun add-pairs-single-sorted :: ((bool }\times(\mp@subsup{(}{}{\prime}t,'b,'c) pdata-pair ) =>(bool \times ('t, 'b, 'c) 
pdata-pair) = bool) =
                            ('t, 'b::zero, 'c) apsT where
    add-pairs-single-sorted - - [] [] - ps=ps
    add-pairs-single-sorted rel flag [] (b # bs) h ps=
    add-pairs-single-sorted rel flag [] bs h (insort-wrt rel (flag, h, b) ps)|
    add-pairs-single-sorted rel flag (g # gs) bs h ps=
    add-pairs-single-sorted rel flag gs bs h (insort-wrt rel (flag, h, g) ps)
lemma set-add-pairs-single-sorted:
    set (add-pairs-single-sorted rel flag gs bs h ps)= set ps \cupPair flag'({h} }\times(\mathrm{ set
gs U set bs))
proof (induct gs arbitrary: ps)
    case Nil
```

```
    show ?case
    proof (induct bs arbitrary: ps)
        case Nil
        show ?case by simp
    next
        case (Cons b bs)
        show ?case by (simp add: Cons)
    qed
next
    case (Cons g gs)
    show ?case by (simp add: Cons)
qed
primrec (in -) pairs :: ('t, 'b, 'c) apsT => bool => ('t, 'b, 'c) pdata list }=>\mathrm{ (bool
* ('t, 'b,'c) pdata-pair) list
    where
    pairs - [] = []|
    pairs aps flag (x# xs) = aps flag [] xs x (pairs aps flag xs)
lemma pairs-subset:
    assumes \gs bs h ps. set (aps flag gs bs h ps)= set ps UPair flag'({h} }\times(\mathrm{ set
gs U set bs))
    shows set (pairs aps flag xs)\subseteq Pair flag'(set xs }\times\mathrm{ set xs)
proof (induct xs)
    case Nil
    show ?case by simp
next
    case (Cons x xs)
    from Cons have set (pairs aps flag xs)\subseteqPair flag'(set (x # xs) < set (x #
xs)) by fastforce
    moreover have {x} < set xs \subseteq set (x# xs) \times set (x# xs) by fastforce
    ultimately show ?case by (auto simp add: assms)
qed
lemma in-pairsI:
    assumes \gs bs h ps. set (aps flaggs bs h ps)= set ps \cupPair flag'({h} \times (set
gs U set bs))
    and }a\not=b\mathrm{ and }a\in\mathrm{ set xs and b}\mathrm{ b set xs
    shows (flag, a, b)\in set (pairs aps flag xs) \vee (flag, b,a) \in set (pairs aps flag xs)
    using assms(3, 4)
proof (induct xs)
    case Nil
    thus ?case by simp
next
    case (Cons x xs)
    from Cons(3) have d: b=x\vee b\in set xs by simp
    from Cons(2) have }a=x\veea\in\mathrm{ set xs by simp
    thus ?case
    proof
```

```
    assume }a=
    with assms(2) have b}=x\mathrm{ by simp
    with d have b\in set xs by simp
    hence (flag, a,b) \in set (pairs aps flag (x # xs)) by (simp add: <a = x>
assms(1))
    thus ?thesis by simp
    next
    assume a\in set xs
    from d show ?thesis
    proof
            assume b = x
            from <a\in set xs〉 have (flag, b,a) \in set (pairs aps flag (x # xs)) by (simp
add: <b = x`assms(1))
            thus ?thesis by simp
    next
            assume b \in set xs
            with }<a\in\mathrm{ set xs` have (flag, a, b) f set (pairs aps flag xs) }\vee(flag,b,a)
set (pairs aps flag xs)
            by (rule Cons(1))
            thus ?thesis by (auto simp: assms(1))
    qed
    qed
qed
corollary in-pairsI':
    assumes \gs bs h ps. set (aps flag gs bs h ps)= set ps \cupPair flag'({h} > (set
gs U set bs))
    and }a\in\mathrm{ set xs and b}\mathrm{ fet xs and }a\not=
    shows (a,b)\inp snd'set (pairs aps flag xs)
proof -
    from assms(1,4,2,3) have (flag, a,b)\in set (pairs aps flag xs) \vee (flag,b,a)\in
set (pairs aps flag xs)
            by (rule in-pairsI)
    thus ?thesis
    proof
            assume (flag, a,b)\in set (pairs aps flag xs)
            hence snd (flag, a, b) \in snd'set (pairs aps flag xs) by fastforce
            thus ?thesis by (simp add: in-pair-iff)
    next
            assume (flag, b, a)\in set (pairs aps flag xs)
            hence snd (flag, b,a) \in snd'set (pairs aps flag xs) by fastforce
            thus ?thesis by (simp add: in-pair-iff)
    qed
qed
definition new-pairs-naive :: ('t, 'b::zero, 'c, 'd) npT
    where new-pairs-naive gs bs hs data=
        fold (add-pairs-single-naive data True gs bs) hs (pairs (add-pairs-single-naive
data) False hs)
```

definition new-pairs-sorted :: (nat $\times{ }^{\prime} d \Rightarrow\left(\right.$ bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair $) \Rightarrow($ bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair $) \Rightarrow$ bool $) \Rightarrow$ ('t, 'b::zero, 'c, 'd) npT
where new-pairs-sorted rel gs bs hs data $=$
fold (add-pairs-single-sorted (rel data) True gs bs) hs (pairs (add-pairs-single-sorted (rel data)) False hs)
lemma set-fold-aps:
assumes $\bigwedge$ gs bs h ps. set (aps flag gs bs h ps) = set ps $\cup$ Pair flag' $(\{h\} \times($ set $g s \cup$ set $b s))$
shows set (fold (aps flag gs bs) hs ps) $=$ Pair flag' $($ set $h s \times($ set gs $\cup$ set bs $)$ )
$\cup$ set $p s$
proof (induct hs arbitrary: ps)
case Nil
show ? case by simp
next
case (Cons h hs)
show ?case by (auto simp add: Cons assms)
qed
lemma set-new-pairs-naive:
set (new-pairs-naive gs bs hs data) $=$
Pair True' (set hs $\times($ set gs $\cup$ set bs $)) \cup$ set (pairs (add-pairs-single-naive
data) False hs)
proof -
have set (new-pairs-naive gs bs hs data) =
Pair True' (set hs $\times($ set gs $\cup$ set bs $)) \cup$ set (pairs (add-pairs-single-naive data) False hs)
unfolding new-pairs-naive-def by (rule set-fold-aps, fact set-add-pairs-single-naive) thus ?thesis by (simp add: ac-simps)
qed
lemma set-new-pairs-sorted: set (new-pairs-sorted rel gs bs hs data) $=$

Pair True' $($ set hs $\times($ set gs $\cup$ set bs $)) \cup$ set (pairs (add-pairs-single-sorted (rel data)) False hs)
proof -
have set (new-pairs-sorted rel gs bs hs data) $=$
Pair True ' $($ set hs $\times($ set gs $\cup$ set bs $)) \cup$ set (pairs (add-pairs-single-sorted (rel data)) False hs)
unfolding new-pairs-sorted-def by (rule set-fold-aps, fact set-add-pairs-single-sorted)
thus ?thesis by (simp add: set-merge-wrt ac-simps)
qed
lemma (in -) fst-snd-Pair [simp]:
shows $f$ st $\circ$ Pair $x=(\lambda-. x)$ and snd $\circ$ Pair $x=i d$
by auto

```
lemma np-spec-new-pairs-naive: np-spec new-pairs-naive
proof (rule np-specI)
    fix gs bs hs :: ('t, 'b, 'c) pdata list and data::nat }\times\mp@subsup{}{}{\prime}
    have 1: set hs \times (set gs U set bs)\subseteq set hs \times (set gs U set bs U set hs) by
fastforce
    have set (pairs (add-pairs-single-naive data) False hs)\subseteq Pair False ' (set hs }
set hs)
    by (rule pairs-subset, simp add: set-add-pairs-single-naive)
    hence snd'set (pairs (add-pairs-single-naive data) False hs)\subseteq snd ' Pair False
    '(set hs \times set hs)
        by (rule image-mono)
    also have ... = set hs }\times\mathrm{ set hs by (simp add: image-comp)
    finally have 2: snd'set (pairs (add-pairs-single-naive data) False hs) \subseteq set hs
* (set gs U set bs U set hs)
    by fastforce
    show snd'set (new-pairs-naive gs bs hs data)\subseteq set hs }\times(\mathrm{ set gs U set bs U set
hs)^
    set hs }\times(\mathrm{ set gs U set bs)}\subseteq\mathrm{ snd'set (new-pairs-naive gs bs hs data)}
                            (\forallab.a \in set hs \longrightarrow b fet hs \longrightarrowa\not=b\longrightarrow(a,b)\inp snd'set
(new-pairs-naive gs bs hs data)) ^
            (\forallp q. (True, p, q)\in set (new-pairs-naive gs bs hs data) \longrightarrowq\in set gs U
set bs)
    proof (intro conjI allI impI)
            show snd' set (new-pairs-naive gs bs hs data)\subseteq set hs }\times(\mathrm{ set gs U set bs U
set hs)
            by (simp add: set-new-pairs-naive image-Un image-comp 1 2)
    next
        show set hs }\times(\mathrm{ set gs U set bs) }\subseteq\mathrm{ snd'set (new-pairs-naive gs bs hs data)
            by (simp add: set-new-pairs-naive image-Un image-comp)
    next
        fix ab
        assume }a\in\mathrm{ set hs and b}\mathrm{ set hs and }a\not=
        with set-add-pairs-single-naive
        have (a,b) \inp snd'set (pairs (add-pairs-single-naive data) False hs)
            by (rule in-pairsI')
        thus (a,b)\inf snd'set (new-pairs-naive gs bs hs data)
            by (simp add: set-new-pairs-naive image-Un)
    next
    fix pq
    assume (True, p,q)\in set (new-pairs-naive gs bs hs data)
    hence q\in set gs \cup set bs \vee (True, p,q)\in set (pairs (add-pairs-single-naive
data) False hs)
            by (auto simp: set-new-pairs-naive)
            thus q}\in\mathrm{ set gs U set bs
            proof
            assume (True, p,q)\in set (pairs (add-pairs-single-naive data) False hs)
            also from set-add-pairs-single-naive have ... \subseteqPair False '(set hs }\times\mathrm{ set hs)
            by (rule pairs-subset)
```

```
        finally show ?thesis by auto
        qed
    qed
qed
lemma np-spec-new-pairs-sorted: np-spec (new-pairs-sorted rel)
proof (rule np-specI)
    fix \(g s\) bs hs :: ('t, 'b, 'c) pdata list and data::nat \(\times\) 'd
    have 1: set \(h s \times(\) set \(g s \cup\) set \(b s) \subseteq\) set \(h s \times(\) set \(g s \cup\) set \(b s \cup\) set hs \()\) by
fastforce
    have set (pairs (add-pairs-single-sorted (rel data)) False hs) \(\subseteq\) Pair False ' (set
\(h s \times\) set \(h s)\)
    by (rule pairs-subset, simp add: set-add-pairs-single-sorted)
    hence snd'set (pairs (add-pairs-single-sorted (rel data)) False hs) \(\subseteq\) snd ' Pair
False ' (set hs \(\times\) set hs)
    by (rule image-mono)
    also have \(\ldots=\) set \(h s \times\) set hs by (simp add: image-comp)
    finally have 2: snd' set (pairs (add-pairs-single-sorted (rel data)) False hs) \(\subseteq\)
set \(h s \times(\) set \(g s \cup\) set \(b s \cup\) set \(h s)\)
    by fastforce
    show snd'set (new-pairs-sorted rel gs bs hs data) \(\subseteq\) set \(h s \times(\) set gs \(\cup\) set bs \(\cup\)
set hs) \(\wedge\)
            set \(h s \times(\) set \(g s \cup\) set \(b s) \subseteq\) snd'set (new-pairs-sorted rel gs bs hs data) \(\wedge\)
                    \(\left(\forall a b, a \in\right.\) set \(h s \longrightarrow b \in\) set \(h s \longrightarrow a \neq b \longrightarrow(a, b) \in_{p}\) snd'set
(new-pairs-sorted rel gs bs hs data)) \(\wedge\)
            \((\forall p q .(\) True \(, p, q) \in\) set (new-pairs-sorted rel gs bs hs data) \(\longrightarrow q \in\) set gs
\(\cup\) set \(b s\) )
    proof (intro conjI allI impI)
    show snd'set (new-pairs-sorted rel gs bs hs data) \(\subseteq\) set \(h s \times(\) set gs \(\cup\) set bs
\(\cup\) set \(h s\) )
            by (simp add: set-new-pairs-sorted image-Un image-comp 1 2)
    next
        show set \(h s \times(\) set \(g s \cup\) set bs \() \subseteq\) snd' set (new-pairs-sorted rel gs bs hs data)
            by (simp add: set-new-pairs-sorted image-Un image-comp)
    next
        fix \(a b\)
        assume \(a \in\) set \(h s\) and \(b \in\) set \(h s\) and \(a \neq b\)
        with set-add-pairs-single-sorted
        have \((a, b) \in_{p}\) snd'set (pairs (add-pairs-single-sorted (rel data)) False hs)
            by (rule in-pairsI')
        thus \((a, b) \in_{p}\) snd' set (new-pairs-sorted rel gs bs hs data)
            by (simp add: set-new-pairs-sorted image-Un)
    next
        fix \(p q\)
        assume (True, \(p, q) \in\) set (new-pairs-sorted rel gs bs hs data)
        hence \(q \in\) set \(g s \cup\) set bs \(\vee(T r u e, p, q) \in\) set (pairs (add-pairs-single-sorted
(rel data)) False hs)
            by (auto simp: set-new-pairs-sorted)
```

```
        thus q\in set gs U set bs
        proof
            assume (True, p,q)\in set (pairs (add-pairs-single-sorted (rel data)) False hs)
            also from set-add-pairs-single-sorted have ...\subseteqPair False'(set hs }\times\mathrm{ set hs)
            by (rule pairs-subset)
            finally show ?thesis by auto
        qed
    qed
qed
```

new-pairs-naive gs bs hs data and new-pairs-sorted rel gs bs hs data return lists of triples $(q-i n-b s, p, q)$, where $q$-in-bs indicates whether $q$ is contained in the list $g s$ @ bs or in the list $h s . p$ is always contained in $h s$.
definition canon-pair-order-aux :: ('t, 'b::zero, 'c) pdata-pair $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair $\Rightarrow$ bool
where canon-pair-order-aux p $q \longleftrightarrow$
$(l c s(l p(f s t(f s t p)))(l p(f s t(s n d p))) \preceq l c s(l p(f s t(f s t ~ q)))(l p(f s t(s n d$ $q))$ )
abbreviation canon-pair-order data $p q \equiv$ canon-pair-order-aux (snd p) (snd q)
abbreviation canon-pair-comb $\equiv$ merge-wrt canon-pair-order-aux

### 6.3.4 Applying Criteria to New Pairs

definition apply-icrit $::\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right)$ icrit $T \Rightarrow\left(n a t \times{ }^{\prime} d\right) \Rightarrow\left(' t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
$\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$ (bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair) list $\Rightarrow$ (bool $\times$ bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair) list
where apply-icrit crit data gs bs hs ps $=$ (let $c=$ crit data gs bs hs in map $(\lambda(q-i n-b s, p, q) .(c p q, q-i n-b s, p, q)) p s)$
lemma fst-apply-icrit:
assumes icrit-spec crit and dickson-grading d
and fst' (set gs $\cup$ set bs $\cup$ set hs $) \subseteq$ dgrad- $p$-set $d m$ and unique-idx (gs @bs @ hs) data
and is-Groebner-basis (fst'set gs) and $p \in$ set hs and $q \in$ set $g s \cup$ set bs $\cup$ set hs
and fst $p \neq 0$ and $f$ st $q \neq 0$ and (True, $q$-in-bs, $p, q) \in$ set (apply-icrit crit data gs bs hs ps)
shows crit-pair-cbelow-on d $m(f s t$ ' (set $g s \cup$ set $b s \cup$ set $h s))(f s t p)(f s t q)$
proof -
from assms(10) have crit data gs bs hs $p q$ by (auto simp: apply-icrit-def) with $\operatorname{assms}(1-9)$ show ?thesis by (rule icrit-specD)
qed
lemma snd-apply-icrit [simp]: map snd (apply-icrit crit data gs bs hs ps) $=$ ps by (auto simp add: apply-icrit-def case-prod-beta' intro: nth-equalityI)
lemma set-snd-apply-icrit [simp]: snd' set (apply-icrit crit data gs bs hs ps) $=$ set ps
proof -
have snd' set (apply-icrit crit data gs bs hs ps) $=$ set (map snd (apply-icrit crit data gs bs hs ps))
by (simp del: snd-apply-icrit)
also have $\ldots=$ set ps by (simp only: snd-apply-icrit)
finally show? thesis.
qed
definition apply-ncrit :: ('t, 'b, 'c, 'd) ncritT $\Rightarrow\left(n a t \times{ }^{\prime} d\right) \Rightarrow\left(' t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
$\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow\left(' t,{ }^{\prime} b,^{\prime} c\right)$ pdata list $\Rightarrow$
(bool $\times$ bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair) list $\Rightarrow$
(bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair) list
where apply-ncrit crit data gs bs hs ps=
(let $c=$ crit data gs bs hs in
rev (fold $\left(\lambda(i c, q-i n-b s, p, q) . \lambda p s^{\prime}\right.$. if $\neg i c \wedge c q-i n-b s p s^{\prime} p q$ then $p s^{\prime}$ else $\left.\left.\left.(i c, p, q) \# p s^{\prime}\right) p s[]\right)\right)$
lemma apply-ncrit-append:
apply-ncrit crit data gs bs hs (xs @ ys) =
rev (fold $\left(\lambda(i c, q-i n-b s, p, q) . \lambda p s^{\prime}\right.$. if $\neg i c \wedge$ crit data gs bs hs $q$-in-bs $p s^{\prime} p q$ then $p s^{\prime}$ else ( $\left.\left.i c, p, q\right) \# p s^{\prime}\right)$ ys
(rev (apply-ncrit crit data gs bs hs xs)))
by (simp add: apply-ncrit-def Let-def)

## lemma fold-superset:

set acc $\subseteq$
set (fold $\left(\lambda(i c, q-i n-b s, p, q) . \lambda p s^{\prime}\right.$. if $\neg i c \wedge c q-i n-b s p s^{\prime} p q$ then $p s^{\prime}$ else (ic, $\left.\left.p, q) \# p s^{\prime}\right) p s a c c\right)$
proof (induct ps arbitrary: acc)
case Nil
show? case by simp
next
case (Cons x ps)
obtain $i c^{\prime} q$-in-bs' $p^{\prime} q^{\prime}$ where $x: x=\left(i c^{\prime}, q\right.$-in-bs', $\left.p^{\prime}, q^{\prime}\right)$ using prod-cases 4
by blast
have 1: set acc0 $\subseteq$ set $\left(\right.$ fold $\left(\lambda(i c, q-i n-b s, p, q) p s^{\prime}\right.$. if $\neg i c \wedge c q-i n-b s p s^{\prime} p q$ then $p s^{\prime}$ else (ic, $\left.\left.p, q\right) \# p s^{\prime}\right) p s$ acc 0$)$
for acc0 by (rule Cons)
have set acc $\subseteq$ set $\left(\left(i c^{\prime}, p^{\prime}, q^{\prime}\right) \# a c c\right)$ by fastforce
also have $\ldots \subseteq$ set (fold $\left(\lambda(i c, q-i n-b s, p, q) p s^{\prime}\right.$. if $\neg i c \wedge c q-i n-b s p s^{\prime} p q$ then $p s^{\prime}$ else (ic, $\left.\left.p, q\right) \# p s^{\prime}\right) p s$
$\left.\left(\left(i c^{\prime}, p^{\prime}, q^{\prime}\right) \# a c c\right)\right)$ by (fact 1)
finally have 2: set acc $\subseteq$ set (fold $\left(\lambda(i c, q-i n-b s, p, q) p s^{\prime}\right.$. if $\neg i c \wedge c q-i n-b s$ $p s^{\prime} p q$ then $p s^{\prime}$ else $\left.(i c, p, q) \# p s^{\prime}\right) p s$

$$
\left.\left(\left(i c^{\prime}, p^{\prime}, q^{\prime}\right) \# a c c\right)\right) \cdot
$$

```
    show ?case by (simp add: x 1 2)
qed
lemma apply-ncrit-superset:
    set (apply-ncrit crit data gs bs hs ps)\subseteq set (apply-ncrit crit data gs bs hs (ps @
qs))(is ?l \subseteq?r)
proof -
    have ?l = set (rev (apply-ncrit crit data gs bs hs ps)) by simp
    also have ... \subseteqset (fold ( }\lambda(ic,q-in-bs,p,q)ps'
                                    if \neg ic ^ crit data gs bs hs q-in-bs ps' p q then ps' else (ic,p,
q) # ps')
                qs (rev (apply-ncrit crit data gs bs hs ps))) by (fact fold-superset)
    also have \ldots.=?r by (simp add: apply-ncrit-append)
    finally show ?thesis.
qed
lemma apply-ncrit-subset-aux:
    assumes (ic, p,q)\in set (fold
                            (\lambda(ic,q-in-bs,p,q). \lambdaps'. if ᄀic ^c q-in-bs ps' p q then ps' else (ic, p,
q) # ps') ps acc)
    shows (ic, p,q)\in set acc \vee (\existsq-in-bs. (ic, q-in-bs, p,q)\in set ps)
    using assms
proof (induct ps arbitrary: acc)
    case Nil
    thus ?case by simp
next
    case (Cons x ps)
    obtain ic' q-in-bs' p' q' where x: x = (ic', q-in-bs', p', q') using prod-cases4
by blast
    from Cons(2) have (ic, p,q)\in
        set (fold ( }\lambda(ic,q-in-bs,p,q)p\mp@subsup{s}{}{\prime}. if \negic\wedgecq-in-bs ps' p q then ps' else (ic
p,q) # ps') ps
            (if ᄀic
add: x)
    hence (ic,p,q)\in\operatorname{set}(if\negi\mp@subsup{c}{}{\prime}\wedgecq-in-b\mp@subsup{s}{}{\prime} acc \mp@subsup{p}{}{\prime}}\mp@subsup{q}{}{\prime}\mathrm{ then acc else (ic', p
acc) \vee
            (\existsq-in-bs. (ic, q-in-bs, p,q) \in set ps) by (rule Cons(1))
    hence (ic,p,q)\in set acc \vee (ic,p,q)=(ic', p', q')\vee(\existsq-in-bs. (ic, q-in-bs,p,
q) \in set ps)
    by (auto split: if-splits)
thus?case
proof (elim disjE)
    assume (ic,p,q)\in set acc
    thus ?thesis ..
next
    assume (ic, p,q)=(ic', p', q')
    hence }x=(ic,q-in-b\mp@subsup{s}{}{\prime},p,q) by (simp add: x
    thus ?thesis by auto
next
```

```
    assume }\existsq-in-bs.(ic,q-in-bs,p,q)\in set p
    then obtain q-in-bs where (ic,q-in-bs,p,q)\in set ps ..
    thus?thesis by auto
    qed
qed
corollary apply-ncrit-subset:
    assumes (ic, p,q)\in set (apply-ncrit crit data gs bs hs ps)
    obtains q-in-bs where (ic, q-in-bs,p,q)\in set ps
proof -
    from assms
    have (ic, p,q) \in set (fold
                            (\lambda(ic,q-in-bs,p,q). \lambdaps'. if \neg ic ^crit data gs bs hs q-in-bs ps'' p q then
ps' else (ic, p,q) # p\mp@subsup{s}{}{\prime}) ps [])
            by (simp add: apply-ncrit-def)
    hence (ic,p,q) \in set [] \vee (\existsq-in-bs. (ic, q-in-bs,p,q) \in set ps)
            by (rule apply-ncrit-subset-aux)
    hence }\existsq\mathrm{ -in-bs. (ic, q-in-bs, p,q) є set ps by simp
    then obtain q-in-bs where (ic,q-in-bs, p,q)\in set ps..
    thus ?thesis..
qed
corollary apply-ncrit-subset': snd ' set (apply-ncrit crit data gs bs hs ps)\subseteq snd '
snd' set ps
proof
    fix pq
    assume (p,q)\in snd'set (apply-ncrit crit data gs bs hs ps)
    then obtain ic where (ic, p,q)\in set (apply-ncrit crit data gs bs hs ps) by
fastforce
    then obtain q-in-bs where (ic, q-in-bs, p,q)\in set ps by (rule apply-ncrit-subset)
    thus (p,q)\in snd'snd' set ps by force
qed
lemma not-in-apply-ncrit:
    assumes (ic,p,q)\not\in set (apply-ncrit crit data gs bs hs (xs @ ((ic, q-in-bs, p,q)
# ys)))
    shows crit data gs bs hs q-in-bs (rev (apply-ncrit crit data gs bs hs xs)) p q
    using assms
proof (simp add: apply-ncrit-append split: if-splits)
    assume (ic, p,q)\not\in
                    set (fold ( }\lambda(ic,q-in-bs,p,q)ps'. if \neg ic ^ crit data gs bs hs q-in-bs ps'
p q then ps' else (ic, p,q) # ps')
                        ys ((ic, p,q) # rev (apply-ncrit crit data gs bs hs xs))) (is - & ?A)
    have (ic, p,q)\in set ((ic,p,q) # rev (apply-ncrit crit data gs bs hs xs)) by simp
    also have ... }\subseteq\mathrm{ ?A by (rule fold-superset)
    finally have (ic, p,q) \in?A.
    with «(ic, p,q)\not\in?A〉 show ?thesis ..
qed
```

```
lemma (in -) setE:
    assumes }x\in\mathrm{ set xs
    obtains ys zs where xs = ys @ (x#zs)
    using assms
proof (induct xs arbitrary: thesis)
    case Nil
    from Nil(2) show ?case by simp
next
    case (Cons a xs)
    from Cons(3) have x=a\veex\in set xs by simp
    thus ?case
    proof
        assume x =a
        show ?thesis by (rule Cons(2)[of [] xs], simp add: <x = a>)
    next
        assume x f set xs
        then obtain ys zs where xs=ys @ (x#zs) by (meson Cons(1))
        show ?thesis by (rule Cons(2)[of a # ys zs], simp add: <xs = ys @ (x # zs)〉)
    qed
qed
lemma apply-ncrit-connectible:
assumes ncrit-spec crit and dickson-grading d
and set \(g s \cup\) set \(b s \cup\) set \(h s \subseteq B\) and \(f s t\) ' \(B \subseteq\) dgrad- \(p\)-set d m
and snd'snd' set ps set hs \(\times\) (set gs \(\cup\) set \(b s \cup\) set hs) and unique-idx (gs
@ bs @ hs)data
and is-Groebner-basis (fst' set gs)
and \(\Lambda p^{\prime} q^{\prime} .\left(p^{\prime}, q^{\prime}\right) \in\) snd ' set (apply-ncrit crit data gs bs hs ps) \(\Longrightarrow\)
fst \(p^{\prime} \neq 0 \Longrightarrow\) fst \(q^{\prime} \neq 0 \Longrightarrow\) crit-pair-cbelow-on d \(m\left(f_{s t}\right.\) ' \(\left.B\right)(f s t\)
\(\left.p^{\prime}\right)\left(f s t q^{\prime}\right)\)
and \(\bigwedge p^{\prime} q^{\prime} \cdot p^{\prime} \in\) set \(g s \cup\) set \(b s \Longrightarrow q^{\prime} \in\) set \(g s \cup\) set \(b s \Longrightarrow f s t p^{\prime} \neq 0 \Longrightarrow f s t\) \(q^{\prime} \neq 0 \Longrightarrow\)
crit-pair-cbelow-on d \(m\) (fst ' \(B)\left(f s t p^{\prime}\right)(f s t ~ q ')\)
assumes \((i c, q-i n-b s, p, q) \in\) set \(p s\) and fst \(p \neq 0\) and fst \(q \neq 0\)
and \(q\)-in-bs \(\Longrightarrow(q \in\) set \(g s \cup\) set \(b s)\)
shows crit-pair-cbelow-on d m (fst' \(B\) ) (fst p) (fst q)
proof (cases \((p, q) \in\) snd'set (apply-ncrit crit data gs bs hs ps))
case True
thus ?thesis using assms \((11,12)\) by (rule assms(8))
next
case False
from \(\operatorname{assms}(10)\) have \((p, q) \in\) snd' \(s n d\) ' set \(p s\) by force
also have \(\ldots \subseteq\) set \(h s \times(\) set gs \(\cup\) set \(b s \cup\) set hs) by (fact assms(5))
finally have \(p \in\) set \(h s\) and \(q \in\) set \(g s \cup\) set \(b s \cup\) set hs by simp-all
from \(\langle(i c, q\)-in-bs, \(p, q) \in\) set \(p s\rangle\) obtain \(x s y s\) where \(p s: p s=x s @((i c, q-i n-b s\), \(p, q) \# y s)\)
by (rule setE)
let ?ps \(=\) rev (apply-ncrit crit data gs bs hs xs)
```

```
    have snd'set ?ps \subseteqsnd ' snd ' set xs by (simp add: apply-ncrit-subset')
    also have ...\subseteq snd 'snd'set ps unfolding ps by fastforce
    finally have sub: snd'set ?ps \subseteq set hs }\times(\mathrm{ set gs U set bs U set hs)
    using assms(5) by (rule subset-trans)
    from False have (p,q)\not\in snd'set (apply-ncrit crit data gs bs hs ps) by (simp
add: in-pair-iff)
    hence (ic, p,q)\not\in set (apply-ncrit crit data gs bs hs (xs @ ((ic, q-in-bs, p,q) #
ys)))
    unfolding ps by force
    hence crit data gs bs hs q-in-bs ?ps p q by (rule not-in-apply-ncrit)
    with assms(1-4) sub assms(6,7,13) --\langlep\in set hs><q\in set gs U set bs U set
hs>assms(11,12)
    show ?thesis
    proof (rule ncrit-specD)
    fix p' q
    assume ( }\mp@subsup{p}{}{\prime},\mp@subsup{q}{}{\prime})\mp@subsup{\in}{p}{}\mathrm{ snd' set ?ps
    also have ...\subseteq snd'set (apply-ncrit crit data gs bs hs ps)
        by (rule image-mono, simp add: ps apply-ncrit-superset)
    finally have disj: ( }\mp@subsup{p}{}{\prime},\mp@subsup{q}{}{\prime})\in\mathrm{ snd 'set (apply-ncrit crit data gs bs hs ps) 
        ( }\mp@subsup{q}{}{\prime},\mp@subsup{p}{}{\prime})\in\mathrm{ snd' set (apply-ncrit crit data gs bs hs ps) by (simp
only: in-pair-iff)
    assume fst p'}=0\mathrm{ and fst q}\mp@subsup{q}{}{\prime}\not=
    from disj show crit-pair-cbelow-on d m (fst` B) (fst p') (fst q')
    proof
            assume ( }\mp@subsup{p}{}{\prime},\mp@subsup{q}{}{\prime})\in\mathrm{ snd ' set (apply-ncrit crit data gs bs hs ps)
            thus ?thesis using <fst p' }=0\rangle\langlefst q' = 0 > by (rule assms(8)
    next
            assume ( }\mp@subsup{q}{}{\prime},\mp@subsup{p}{}{\prime})\in\mathrm{ snd ' set (apply-ncrit crit data gs bs hs ps)
            hence crit-pair-cbelow-on d m (fst ' B) (fst q') (fst p')
                using〈fst q'}=0\rangle\langlefst \mp@subsup{p}{}{\prime}\not=0\rangle\mathrm{ by (rule assms(8))
            thus ?thesis by (rule crit-pair-cbelow-sym)
    qed
    qed (assumption, fact assms(9))
qed
```


### 6.3.5 Applying Criteria to Old Pairs

definition apply-ocrit :: ('t, 'b, 'c, 'd) ocrit $T \Rightarrow\left(n a t \times{ }^{\prime} d\right) \Rightarrow\left(' t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
(bool $\times\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair) list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair
list $\Rightarrow$

$$
\left(^{\prime} t,,^{\prime} b,{ }^{\prime} c\right) \text { pdata-pair list }
$$

where apply-ocrit crit data hs ps' $p s=$ (let $c=$ crit data hs ps' in $[(p, q) \leftarrow p s$. $\neg c p q])$
lemma set-apply-ocrit:
set (apply-ocrit crit data hs ps $\left.{ }^{\prime} p s\right)=\{(p, q) \mid p q .(p, q) \in$ set $p s \wedge \neg$ crit data $\left.h s p s^{\prime} p q\right\}$
by (auto simp: apply-ocrit-def)

```
corollary set-apply-ocrit-iff:
    (p,q)\in set (apply-ocrit crit data hs ps' ps)\longleftrightarrow((p,q) \in set ps }\wedge\neg\mathrm{ crit data
hs ps' p q)
    by (auto simp: apply-ocrit-def)
lemma apply-ocrit-connectible:
    assumes ocrit-spec crit and dickson-grading d and set hs \subseteqB and fst'B\subseteq
dgrad-p-set d m
    and unique-idx (p#q# hs @ (map (fst \circ snd) ps') @ (map (snd \circ snd) ps'))
data
    and }\bigwedge\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}.(\mp@subsup{p}{}{\prime},\mp@subsup{q}{}{\prime})\in\mathrm{ snd'set ps' }\Longrightarrow\mathrm{ fst p' 
                    crit-pair-cbelow-on d m (fst ' B) (fst p') (fst q')
    assumes p\inB and q\inB and fst p\not=0 and fst q\not=0
        and (p,q)\in set ps and (p,q)\not\in set (apply-ocrit crit data hs ps' ps)
    shows crit-pair-cbelow-on d m (fst' B) (fst p) (fst q)
proof -
    from assms(11,12) have crit data hs ps' p q by (simp add: set-apply-ocrit-iff)
    with assms(1-5)-assms(7-10) show ?thesis
    proof (rule ocrit-specD)
        fix p}\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime
        assume ( }\mp@subsup{p}{}{\prime},\mp@subsup{q}{}{\prime})\in\mp@subsup{\in}{p}{}\mathrm{ snd ' set ps'
            hence disj: ( p', q') \in snd' set ps'\vee ( }\mp@subsup{q}{}{\prime},\mp@subsup{p}{}{\prime})\in\mathrm{ snd' set ps' by (simp only:
in-pair-iff)
            assume fst p}\mp@subsup{p}{}{\prime}\not=0\mathrm{ and fst q}\mp@subsup{q}{}{\prime}\not=
            from disj show crit-pair-cbelow-on d m (fst' B) (fst p')(fst q')
            proof
                    assume ( }\mp@subsup{p}{}{\prime},\mp@subsup{q}{}{\prime})\in\mathrm{ snd ' set ps'
                    thus ?thesis using <fst p'}=0\rangle\langlefst \mp@subsup{q}{}{\prime}\not=0\rangle\mathrm{ by (rule assms(6))
        next
            assume ( }\mp@subsup{q}{}{\prime},\mp@subsup{p}{}{\prime})\in\mathrm{ snd ' set ps'
            hence crit-pair-cbelow-on d m (fst ' B) (fst q') (fst p') using <fst q' = 0〉\langlefst
p
            by (rule assms(6))
            thus ?thesis by (rule crit-pair-cbelow-sym)
        qed
    qed
qed
```


### 6.3.6 Creating Final List of Pairs

## context

fixes $n p::\left(' t,{ }^{\prime} b::\right.$ field, $\left.{ }^{\prime} c,{ }^{\prime} d\right) n p T$
and icrit::('t, 'b, ' $\left.c,{ }^{\prime} d\right)$ icritT
and ncrit:: ('t, 'b, 'c, 'd) ncritT
and ocrit::('t, 'b, 'c, 'd) ocritT
and comb::('t, 'b, 'c) pdata-pair list $\Rightarrow\left(' t, ' b,{ }^{\prime} c\right)$ pdata-pair list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair list
begin
definition add-pairs :: ('t, 'b, 'c, 'd) apT
where add-pairs gs bs ps hs data $=$
(let ps1 = apply-ncrit ncrit data gs bs hs (apply-icrit icrit data gs bs hs ( $n \mathrm{p}$ gs bs hs data));
$p s 2=$ apply-ocrit ocrit data hs ps1 ps in comb (map snd $[x \leftarrow p s 1 . \neg$
fst $x]$ ) ps 2)
lemma set-add-pairs:
assumes $\bigwedge x s$ ys. set $(\operatorname{comb} x s y s)=$ set $x s \cup$ set ys
assumes $p s 1=$ apply-ncrit ncrit data gs bs hs (apply-icrit icrit data gs bs hs (np gs bs hs data))
shows set (add-pairs gs bs ps hs data) $=$

$$
\{(p, q) \mid p q .(\text { False }, p, q) \in \text { set } p s 1 \vee((p, q) \in \text { set } p s \wedge \neg \text { ocrit data }
$$

hs ps1 p q) \}
proof -
have eq: snd ' $\{x \in$ set ps 1. $\neg$ fst $x\}=\{(p, q) \mid p q .($ False, $p, q) \in$ set $p s 1\}$ by force
thus ?thesis by (auto simp: add-pairs-def Let-def assms(1) assms(2)[symmetric] set-apply-ocrit)
qed
lemma set-add-pairs-iff:
assumes $\bigwedge x s$ ys. set $($ comb xs ys) $=$ set $x s \cup$ set ys
assumes ps1 = apply-ncrit ncrit data gs bs hs (apply-icrit icrit data gs bs hs (np gs bs hs data))
shows $((p, q) \in$ set (add-pairs gs bs ps hs data $)) \longleftrightarrow$
$(($ False $, p, q) \in$ set $p s 1 \vee((p, q) \in$ set $p s \wedge \neg$ ocrit data hs ps1 $p q))$

## proof -

from assms have eq: set (add-pairs gs bs ps hs data) $=$

$$
\{(p, q) \mid p q .(\text { False }, p, q) \in \text { set ps } 1 \vee((p, q) \in \text { set } p s \wedge \neg \text { ocrit data }
$$ $h s p s 1 p q)\}$

by (rule set-add-pairs)
obtain $a$ aa $b$ where $p: p=(a, a a, b)$ using prod-cases3 by blast
obtain $a b a c b a$ where $q: q=(a b, a c, b a)$ using prod-cases3 by blast show ?thesis by (simp add: eq $p$ q)
qed
lemma ap-spec-add-pairs:
assumes np-spec $n p$ and icrit-spec icrit and ncrit-spec ncrit and ocrit-spec ocrit and $\bigwedge x s$ ys. set $(\operatorname{comb} x s y s)=$ set $x s \cup$ set ys
shows ap-spec add-pairs
proof (rule ap-specI)
fix $g s$ bs $::\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list and $p s h s$ and data::nat $\times$ 'd
define $p s 1$ where $p s 1=$ apply-ncrit ncrit data gs bs hs (apply-icrit icrit data gs bs hs (np gs bs hs data))
show set (add-pairs gs bs ps hs data) $\subseteq$ set $p s \cup$ set $h s \times($ set $g s \cup$ set bs $\cup$ set $h s$ )
proof
fix $p q$
assume $(p, q) \in$ set (add-pairs gs bs ps hs data)
with assms(5) ps1-def have (False, p, q) $\in$ set ps1 $\vee((p, q) \in$ set $p s \wedge \neg$ ocrit data hs ps1 p q)
by (simp add: set-add-pairs-iff)
thus $(p, q) \in$ set $p s \cup$ set $h s \times($ set gs $\cup$ set $b s \cup$ set $h s)$
proof
assume (False, p,q) $\in$ set ps 1
hence snd (False, $p, q) \in$ snd' set ps1 by fastforce
hence $(p, q) \in$ snd ' set ps1 by simp
also have $\ldots \subseteq$ snd' snd' set (apply-icrit icrit data gs bs hs (np gs bs hs data))
unfolding ps1-def by (fact apply-ncrit-subset')
also have $\ldots=$ snd' set ( $n p$ gs bs hs data) by simp
also from $\operatorname{assms}(1)$ have $\ldots \subseteq$ set hs $\times($ set $g s \cup$ set $b s \cup$ set hs) by (rule $n p-s p e c D 1)$
finally show ?thesis ..

## next

assume $(p, q) \in$ set $p s \wedge \neg$ ocrit data hs ps1 $p q$
thus ?thesis by simp
qed
qed
next
fix $g s$ bs :: $\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list and $p s h s$ and data::nat $\times{ }^{\prime} d$ and $B$ and $d::^{\prime} a$ $\Rightarrow$ nat and $m h g$
assume dg: dickson-grading $d$ and $B$-sup: set $g s \cup$ set $b s \cup$ set $h s \subseteq B$
and $B$-sub: fst ' $B \subseteq$ dgrad-p-set $d m$ and $h$-in: $h \in$ set $h s$ and $g$-in: $g \in$ set $g s \cup$ set bs $\cup$ set hs
and $p s$-sub: set $p s \subseteq$ set $b s \times($ set $g s \cup$ set bs $)$
and uid: unique-idx (gs @ bs @ hs) data and gb: is-Groebner-basis (fst'set $g s)$ and $h \neq g$
and $f$ st $h \neq 0$ and $f s t g \neq 0$
assume $a: \bigwedge a b .(a, b) \in_{p}$ set (add-pairs gs bs ps hs data) $\Longrightarrow$

$$
\text { fst } a \neq 0 \Longrightarrow \text { fst } b \neq 0 \Longrightarrow \text { crit-pair-cbelow-on d } m\left(f_{s t} \cdot B\right)\left(f_{s t} a\right)
$$

(fst b)
assume $b: \bigwedge a b . a \in$ set $g s \cup$ set $b s \Longrightarrow$

$$
\begin{aligned}
& b \in \text { set } g s \cup \text { set } b s \Longrightarrow \\
& \text { fst } a \neq 0 \Longrightarrow \text { fst } b \neq 0 \Longrightarrow \text { crit-pair-cbelow-on d } m\left(f_{\text {st }} \text { ‘ } B\right)(\text { fst } a)
\end{aligned}
$$

(fst b)
define $p s 0$ where $p s 0=$ apply-icrit icrit data gs bs hs (np gs bs hs data)
define ps1 where ps1 = apply-ncrit ncrit data gs bs hs ps0
have snd' snd' set ps0 $=$ snd' set (np gs bs hs data) by (simp add: ps0-def)
also from $\operatorname{assms}(1)$ have $\ldots \subseteq$ set $h s \times($ set gs $\cup$ set $b s \cup$ set hs) by (rule $n p-s p e c D 1$ )
finally have $p s 0$-sub: snd'snd' set ps0 $\subseteq$ set $h s \times($ set $g s \cup$ set bs $\cup$ set hs).
have crit-pair-cbelow-on $d m(f s t$ ' $B)(f s t ~ p)(f s t q)$
if $(p, q) \in$ snd'set $p s 1$ and $f$ st $p \neq 0$ and fst $q \neq 0$ for $p q$

```
proof -
    from \(\langle(p, q) \in\) snd ' set \(p s 1\) > obtain \(i c\) where \((i c, p, q) \in\) set \(p s 1\) by fastforce
    show ?thesis
    proof (cases ic)
        case True
        from \(\langle(i c, p, q) \in\) set \(p s 1\rangle\) obtain \(q\)-in-bs where \((i c, q-i n-b s, p, q) \in\) set \(p s 0\)
            unfolding ps1-def by (rule apply-ncrit-subset)
        with True have (True, \(q\)-in-bs, p,q) \(\in\) set ps0 by simp
        hence snd (snd (True, \(q\)-in-bs, p,q)) \(\in\) snd' snd' set ps0 by fastforce
        hence \((p, q) \in\) snd ' snd ' set ps0 by simp
        also have \(\ldots \subseteq\) set \(h s \times(\) set gs \(\cup\) set \(b s \cup\) set \(h s)\) by (fact ps0-sub)
        finally have \(p \in\) set \(h s\) and \(q \in\) set \(g s \cup\) set bs \(\cup\) set hs by simp-all
        from \(B\)-sup have \(B\)-sup': \(f s t\) ' (set gs \(\cup\) set \(b s \cup\) set \(h s) \subseteq f s t\) ' \(B\) by (rule
image-mono)
    hence \(f s t\) ' \((\) set \(g s \cup\) set \(b s \cup\) set \(h s) \subseteq\) dgrad- \(p\)-set \(d m\) using \(B\)-sub by (rule
subset-trans)
            from assms(2) dg this uid \(g b\langle p \in\) set \(h s\rangle\langle q \in\) set \(g s \cup\) set \(b s \cup\) set \(h s\rangle\langle f s t\)
\(p \neq 0\rangle\langle f s t q \neq 0\rangle\)
            〈(True, \(q\)-in-bs, \(p, q) \in\) set \(p s 0\rangle\)
        have crit-pair-cbelow-on d \(m(f s t\) ' \((\) set gs \(\cup\) set \(b s \cup\) set hs \())(f s t p)(f s t q)\)
                unfolding ps0-def by (rule fst-apply-icrit)
            thus ?thesis using \(B\)-sup' by (rule crit-pair-cbelow-mono)
    next
            case False
            with \(\langle(i c, p, q) \in\) set \(p s 1\rangle\) have (False, \(p, q) \in\) set \(p s 1\) by simp
            with \(\operatorname{assms}(5)\) ps1-def have \((p, q) \in\) set (add-pairs gs bs ps hs data)
                by (simp add: set-add-pairs-iff ps0-def)
            hence \((p, q) \in_{p}\) set (add-pairs gs bs ps hs data) by (simp add: in-pair-iff)
            thus ?thesis using \(\langle\) fst \(p \neq 0\rangle\langle f s t q \neq 0\rangle\) by (rule \(a\) )
    qed
qed
with assms(3) dg B-sup B-sub ps0-sub uid gb
have \(*:(i c, q-i n-b s, p, q) \in \operatorname{set} p s 0 \Longrightarrow f s t p \neq 0 \Longrightarrow f s t ~ q \neq 0 \Longrightarrow\)
                \((q-i n-b s \Longrightarrow q \in\) set \(g s \cup\) set \(b s) \Longrightarrow\) crit-pair-cbelow-on d \(m(f s t ‘ B)\)
(fst p) (fst q)
    for ic \(q\)-in-bs \(p q\) using \(b\) unfolding ps1-def by (rule apply-ncrit-connectible)
show crit-pair-cbelow-on d \(m(f s t\) ' \(B)(f s t h)(f s t g)\)
proof (cases \(h=g\) )
    case True
    from \(g\)-in \(B\)-sup have \(g \in B\)..
    hence \(f s t g \in f s t\) ' \(B\) by simp
    hence \(f\) st \(g \in d g r a d-p\)-set \(d m\) using \(B\)-sub ..
    with \(d g\) show ?thesis unfolding True by (rule crit-pair-cbelow-same)
next
    case False
    with assms(1) \(h\)-in \(g\)-in show ?thesis
    proof (rule np-specE)
            fix \(g\) - \(i n-b s\)
```

```
    assume (g-in-bs,h,g)\in set (np gs bs hs data)
    also have ... = snd'set ps0 by (simp add: ps0-def)
    finally obtain ic where (ic,g-in-bs,h,g)\in set ps0 by fastforce
    moreover note <fst h\not=0\rangle\langlefst g\not=0\rangle
    moreover from assms(1) have g}\in\mathrm{ set gs }\cup\mathrm{ set bs if g-in-bs
    proof (rule np-specD4)
            from <(g-in-bs,h,g)\in set (np gs bs hs data)\rangle that show (True, h,g) \in set
(np gs bs hs data)
            by simp
        qed
        ultimately show ?thesis by (rule *)
    next
        fix h-in-bs
    assume (h-in-bs,g,h)\in set (np gs bs hs data)
    also have ... = snd ' set ps0 by (simp add: ps0-def)
    finally obtain ic where (ic,h-in-bs,g,h) \in set ps0 by fastforce
    moreover note <fst g\not=0\rangle\langlefst h\not=0\rangle
    moreover from assms(1) have h\in set gs Uset bs if h-in-bs
    proof (rule np-specD4)
    from}\langle(h-in-bs,g,h)\in set (np gs bs hs data)> that show (True, g,h)\in se
(np gs bs hs data)
                by simp
            qed
            ultimately have crit-pair-cbelow-on d m (fst' B) (fst g) (fst h) by (rule *)
            thus ?thesis by (rule crit-pair-cbelow-sym)
        qed
    qed
next
    fix gs bs :: ('t, 'b, 'c) pdata list and ps hs and data::nat }\times\mp@subsup{}{}{\prime}d\mathrm{ and }B\mathrm{ and }d::''
=> nat and mhg
    define ps1 where ps1 = apply-ncrit ncrit data gs bs hs (apply-icrit icrit data gs
bs hs (np gs bs hs data))
    assume (h,g) \in set ps - 
    hence (h,g)\in set ps and (h,g)\not\inp set (add-pairs gs bs ps hs data) by simp-all
    from this(2) have (h,g)\not\in set (add-pairs gs bs ps hs data) by (simp add:
in-pair-iff)
    assume dg:dickson-grading d and B-sup: set gs U set bs \cup set hs\subseteqB and
B-sub: fst ' B\subseteqdgrad-p-set d m
    and ps-sub: set ps\subseteq set bs\times(set gs \cup set bs)
    and (set gs \cup set bs) \cap set hs = {} - unused
    and uid:unique-idx (gs@ @s @ hs) data and gb: is-Groebner-basis (fst'set gs)
    and h\not=g and fst h\not=0 and fst g\not=0
    assume *: \bigwedgeab. (a,b) \inp set (add-pairs gs bs ps hs data) \Longrightarrow
        (a,b) \inp set hs }\times(\mathrm{ set gs U set bs U set hs )}
        fst }a\not=0\Longrightarrowfst b\not=0\Longrightarrowcrit-pair-cbelow-on dm(fst'B) (fst a
(fst b)
```

have snd' set ps1 $\subseteq$ snd'snd' set (apply-icrit icrit data gs bs hs (np gs bs hs data))
unfolding ps1－def by（rule apply－ncrit－subset＇）
also have $\ldots=$ snd＇set（ $n p$ gs bs hs data）by simp
also from $\operatorname{assms}(1)$ have $\ldots \subseteq$ set hs $\times($ set $g s \cup$ set $b s \cup$ set hs）by（rule $n p-s p e c D 1$ ）
finally have ps1－sub：snd＇set ps1 $\subseteq$ set $h s \times($ set $g s \cup$ set $b s \cup$ set $h s)$.
from $\langle(h, g) \in$ set $p s\rangle p s$－sub have $h$－in：$h \in$ set $g s \cup$ set bs and $g$－in：$g \in$ set $g s \cup$ set bs

## by fastforce＋

with $B$－sup have $h \in B$ and $g \in B$ by auto
with $\operatorname{assms}(4) d g-B$－sub－－show crit－pair－cbelow－on d m $(f s t$＇$B)(f s t h)(f s t$
g）
using $\langle f s t h \neq 0\rangle\langle f s t g \neq 0\rangle\langle(h, g) \in$ set $p s\rangle$ proof（rule apply－ocrit－connectible）
from $B$－sup show set $h s \subseteq B$ by simp
next
from ps1－sub h－in g－in
have set $(h \# g \# h s @ \operatorname{map}(f s t \circ s n d) p s 1 @ \operatorname{map}($ snd $\circ$ snd $) p s 1) \subseteq$ set （ $g s$＠bs＠hs） by fastforce
with uid show unique－idx $(h \# g \# h s @ m a p(f s t \circ s n d) p s 1 @ m a p(s n d \circ$ snd）ps1）data
by（rule unique－idx－subset）
next
fix $p q$
assume $(p, q) \in$ snd＇set ps1
hence pq－in：$(p, q) \in$ set hs $\times($ set gs $\cup$ set $b s \cup$ set hs $)$ using $p s 1-s u b$ ．．
hence $p$－in：$p \in$ set $h s$ and $q$－in：$q \in$ set $g s \cup$ set $b s \cup$ set $h s$ by simp－all
assume fst $p \neq 0$ and fst $q \neq 0$
from $\langle(p, q) \in$ snd＇set $p s 1\rangle$ obtain $i c$ where $(i c, p, q) \in$ set $p s 1$ by fastforce
show crit－pair－cbelow－on d $m(f s t$＇$B)(f s t p)(f s t q)$
proof（cases ic）
case True
hence $i c=$ True by simp
from $B$－sup have $B-s u p$＇：$f s t$＇（set gs $\cup$ set bs $\cup$ set $h s) \subseteq f s t$＇$B$ by（rule image－mono）
note $\operatorname{assms}(2) d g$
moreover from $B-s u p$＇$B$－sub have $f s t$＇（set gs $\cup$ set bs $\cup$ set $h s) \subseteq d g r a d-p$－set $d m$
by（rule subset－trans）
moreover note uid gb p－in $q$－in 〈fst $p \neq 0\rangle\langle f s t q \neq 0\rangle$
moreover from $\langle(i c, p, q) \in$ set $p s 1\rangle$ obtain $q$－in－bs
where（True，$q$－in－bs，$p, q$ ）$\in$ set（apply－icrit icrit data gs bs hs（np gs bs hs data））
unfolding ps1－def〈ic＝True〉 by（rule apply－ncrit－subset）
ultimately have crit－pair－cbelow－on $d m(f s t$＇（set gs $\cup$ set $b s \cup$ set hs）$)(f s t$ p）$(f s t q)$

> by (rule fst-apply-icrit)
thus ？thesis using $B$－sup＇by（rule crit－pair－cbelow－mono）

```
    next
            case False
            with «(ic, p,q)\in set ps1> have (False, p,q)\in set ps1 by simp
            with assms(5) ps1-def have (p,q)\in set (add-pairs gs bs ps hs data)
            by (simp add: set-add-pairs-iff)
            hence (p,q)\inp}\mp@subsup{\in}{p}{\mathrm{ set (add-pairs gs bs ps hs data) by (simp add: in-pair-iff)}
            moreover from pq-in have (p,q)\inp set hs \times( set gs \cup set bs U set hs)
                by (simp add: in-pair-iff)
            ultimately show ?thesis using <fst p}\not=0\rangle\langlefst q\not=0\rangle\mathrm{ by (rule *)
    qed
next
        show (h,g)\not\inset (apply-ocrit ocrit data hs ps1 ps)
        proof
            assume (h,g)\in set (apply-ocrit ocrit data hs ps1 ps)
            hence (h,g)\in set (add-pairs gs bs ps hs data)
                by (simp add: add-pairs-def assms(5) Let-def ps1-def)
            with}«(h,g)\not\in set (add-pairs gs bs ps hs data)〉 show False ..
        qed
qed
qed
end
```

abbreviation add-pairs-canon $\equiv$
add-pairs (new-pairs-sorted canon-pair-order) component-crit chain-ncrit chain-ocrit canon-pair-comb
lemma ap-spec-add-pairs-canon: ap-spec add-pairs-canon
using np-spec-new-pairs-sorted icrit-spec-component-crit ncrit-spec-chain-ncrit ocrit-spec-chain-ocrit set-merge-wrt
by (rule ap-spec-add-pairs)

### 6.4 Suitable Instances of the completion Parameter

definition rcp-spec :: ('t, 'b::field, ' $\left.c,{ }^{\prime} d\right)$ compl $T \Rightarrow$ bool
where rcp-spec rcp $\longleftrightarrow$
( $\forall$ gs bs ps sps data.
$0 \notin f s t$ ' set $(f s t(r c p$ gs bs ps sps data $)) \wedge$
$(\forall h b . h \in \operatorname{set}(f s t(r c p$ gs bs ps sps data $)) \longrightarrow b \in$ set $g s \cup$ set $b s \longrightarrow$
fst $b \neq 0 \longrightarrow$
$\left.\neg l t\left(f_{s t} b\right) a d d s_{t} l t(f s t h)\right) \wedge$
$(\forall d$. dickson-grading $d \longrightarrow$ dgrad-p-set-le d (fst'set (fst (rcp gs bs ps sps data))) (args-to-set
$(g s, b s, s p s))) \wedge$
component-of-term'Keys $(f s t$ ' $($ set $(f s t(r c p ~ g s ~ b s ~ p s ~ s p s ~ d a t a ~) ~) ~) ~) ~ \subseteq ~$ component-of-term ' Keys (args-to-set (gs, bs, sps)) ^
(is-Groebner-basis (fst'set gs) $\longrightarrow$ unique-idx (gs @ bs) data $\longrightarrow$ $(f s t$ ' set $(f s t(r c p ~ g s ~ b s ~ p s ~ s p s ~ d a t a)) \subseteq p m d l(\operatorname{args-to-set~(gs,bs,~sps))})$

```
    (\forall(p,q)\inset sps. set sps\subseteq set bs \times (set gs \cup set bs) \longrightarrow
    (red (fst'( set gs U set bs) \cupfst' set (fst (rcp gs bs ps sps data))))**
(spoly (fst p)(fst q)) 0))))
```

Informally, rcp-spec rcp expresses that, for suitable $g s, b s$ and $s p s$, the value of rcp gs bs ps sps

- is a list consisting exclusively of non-zero polynomials contained in the module generated by set $b s \cup$ set $g s$, whose leading terms are not divisible by the leading term of any non-zero $b \in$ set $b s$, and
- contains sufficiently many new polynomials such that all S-polynomials originating from sps can be reduced to 0 modulo the enlarged list of polynomials.

```
lemma rcp-specI:
    assumes \gs bs ps sps data. 0 #fst' set (fst (rcp gs bs ps sps data))
    assumes \gs bs ps spsh b data. h\in set (fst (rcp gs bs ps sps data)) \Longrightarrowb\in set
gs \cup set bs \Longrightarrow fst b}=0
                        \neg l t ( f s t ~ b ) ~ a d d s t ~ l t ~ ( f s t ~ h )
    assumes \gs bs ps sps d data. dickson-grading d \Longrightarrow
        dgrad-p-set-le d (fst' set (fst (rcp gs bs ps sps data))) (args-to-set
(gs,bs, sps))
    assumes \gs bs ps sps data. component-of-term'Keys (fst' (set (fst (rcp gs bs
ps sps data))))}
                            component-of-term'Keys (args-to-set (gs, bs, sps))
    assumes \bigwedgegs bs ps sps data. is-Groebner-basis (fst'set gs)\Longrightarrowunique-idx (gs
@ bs) data \Longrightarrow
    (fst ' set (fst (rcp gs bs ps sps data)) \subseteqpmdl (args-to-set (gs, bs, sps))
^
    (\forall(p,q)\inset sps. set sps\subseteq set bs \times (set gs \cup set bs) \longrightarrow
    (red (fst'( set gs \cup set bs)\cupfst'set (fst (rcp gs bs ps sps data))))**
(spoly (fst p) (fst q)) 0))
    shows rcp-spec rcp
    unfolding rcp-spec-def using assms by auto
lemma rcp-specD1:
    assumes rcp-spec rcp
    shows 0 &fst' set (fst (rcp gs bs ps sps data))
    using assms unfolding rcp-spec-def by (elim allE conjE)
lemma rcp-specD2:
    assumes rср-spec rср
        and h\in set (fst (rcp gs bs ps sps data)) and b\in set gs \cup set bs and fst b}=
    shows \neglt (fst b) adds\mp@subsup{s}{t}{}lt (fst h)
    using assms unfolding rcp-spec-def by (elim allE conjE, blast)
lemma rcp-specD3:
    assumes rcp-spec rср and dickson-grading d
```

shows dgrad-p-set-le d (fst'set (fst (rcp gs bs ps sps data))) (args-to-set (gs, bs, sps))
using assms unfolding rcp-spec-def by (elim allE conjE, blast)
lemma rcp-specD4:
assumes rcp-spec rcp
shows component-of-term' Keys $(f s t$ ' $($ set $(f s t(r c p ~ g s ~ b s ~ p s ~ s p s ~ d a t a) ~))) ~ \subseteq$ component-of-term 'Keys (args-to-set (gs, bs, sps))
using assms unfolding rcp-spec-def by (elim allE conjE)
lemma rcp-specD5:
assumes rcp-spec rcp and is-Groebner-basis (fst'set gs) and unique-idx (gs @ bs) data
shows $f s t$ ' set (fst (rcp gs bs ps sps data)) $\subseteq p m d l$ (args-to-set (gs, bs, sps))
using assms unfolding rep-spec-def by blast
lemma rcp-specD6:
assumes rcp-spec rcp and is-Groebner-basis (fst'set gs) and unique-idx (gs @ bs) data
and set sps $\subseteq$ set $b s \times($ set $g s \cup$ set $b s)$
and $(p, q) \in$ set sps
shows $($ red $(f s t$ ' $($ set gs $\cup$ set bs $) \cup f s t$ 'set $(f s t(r c p ~ g s ~ b s ~ p s ~ s p s ~ d a t a)))))^{* *}$ (spoly (fst p) (fst q)) 0
using assms unfolding rcp-spec-def by blast
lemma compl-struct-rcp:
assumes rcp-spec rcp
shows compl-struct rcp
proof (rule compl-structI)
fix $d::^{\prime} a \Rightarrow$ nat and $g s$ bs ps and sps::('t, 'b, 'c) pdata-pair list and data::nat $\times$ 'd
assume dickson-grading $d$ and set sps $\subseteq$ set ps
from assms this(1) have dgrad-p-set-le d (fst'set (fst (rcp gs bs (ps -- sps) sps data)))

$$
(\text { args-to-set }(g s, b s, s p s))
$$

by (rule rcp-specD3)
also have dgrad-p-set-le $d \ldots$ (args-to-set ( $g s, b s, p s)$ )
by (rule dgrad-p-set-le-subset, rule args-to-set-subset3, fact «set sps $\subseteq$ set ps〉)
finally show dgrad-p-set-le d (fst'set (fst (rcp gs bs (ps -- sps) sps data))) (args-to-set $(g s, b s, p s))$.
next
fix $g s$ bs ps and sps::('t, 'b, 'c) pdata-pair list and data::nat $\times{ }^{\prime} d$
from assms show $0 \notin f s t '$ set (fst (rcp gs bs (ps -- sps) sps data))
by (rule rcp-specD1)
next
fix gs bs ps sps h b data
assume $h \in \operatorname{set}(f s t$ (rcp gs bs (ps -- sps) sps data))
and $b \in$ set $g s \cup$ set $b s$ and $f s t b \neq 0$
with assms show $\neg l t(f s t b) a d d s_{t} l t(f s t h)$ by (rule rcp-specD2)

```
next
    fix gs bs ps and sps::('t, 'b,'c) pdata-pair list and data::nat }\times\mp@subsup{}{}{\prime}
    assume set sps \subseteq set ps
    from assms
    have component-of-term 'Keys (fst'set (fst (rcp gs bs (ps -- sps) sps data)))
\subseteq
            component-of-term'Keys (args-to-set (gs, bs, sps))
        by (rule rcp-specD4)
    also have ...\subseteq component-of-term 'Keys (args-to-set (gs, bs,ps))
    by (rule image-mono, rule Keys-mono, rule args-to-set-subset3, fact «set sps }
set ps>)
    finally show component-of-term'Keys (fst' set (fst (rcp gs bs (ps -- sps) sps
data)))}
                component-of-term'Keys (args-to-set (gs,bs, ps)).
qed
lemma compl-pmdl-rcp:
    assumes rcp-spec rср
    shows compl-pmdl rcp
proof (rule compl-pmdlI)
    fix gs bs ::('t, 'b,'c) pdata list and ps sps :: ('t,'b,'c) pdata-pair list and
data::nat > 'd
    assume gb: is-Groebner-basis (fst' set gs) and set sps \subseteq set ps
        and un:unique-idx (gs@ @s)data
    let ?res = fst (rcp gs bs (ps-- sps) sps data)
    from assms gb un have fst' set ?res }\subseteqpmdl (args-to-set (gs,bs, sps)
        by (rule rcp-specD5)
    also have .. \subseteqpmdl (args-to-set (gs,bs,ps))
        by (rule pmdl.span-mono, rule args-to-set-subset3, fact <set sps \subseteq set ps〉)
    finally show fst'set ?res \subseteqpmdl (args-to-set (gs,bs,ps)).
qed
lemma compl-conn-rcp:
    assumes rср-spec rср
    shows compl-conn rcp
proof (rule compl-connI)
    fix d::'a m nat and m gs bs ps sps p and q::('t, 'b, 'c) pdata and data::nat }\times\mp@subsup{}{}{\prime}
    assume dg: dickson-grading d and gs-sub: fst'set gs \subseteqdgrad-p-set d m
        and gb: is-Groebner-basis (fst' set gs) and bs-sub: fst' set bs\subseteq dgrad-p-set d
m
    and ps-sub: set ps\subseteq set bs \times (set gs \cup set bs) and set sps\subseteq set ps
    and uid:unique-idx (gs@ bs)data
    and (p,q)\in set sps and fst p\not=0 and fst q}\not=
    from <set sps\subseteq set ps`ps-sub have sps-sub: set sps\subseteq set bs \times (set gs U set bs)
        by (rule subset-trans)
    let ?res = fst (rcp gs bs (ps-- sps) sps data)
    have fst' set ?res \subseteqdgrad-p-set d m
```

```
    proof (rule dgrad-p-set-le-dgrad-p-set, rule rcp-specD3, fact+)
    show args-to-set (gs, bs, sps)\subseteqdgrad-p-set d m
        by (simp add: args-to-set-subset-Times[OF sps-sub], rule, fact+)
    qed
```

    moreover have \(g s\)-bs-sub: fst ' (set gs \(\cup\) set \(b s) \subseteq\) dgrad-p-set d \(m\) by (simp
    add: image-Un, rule, fact+)
ultimately have res-sub: $f s t$ ' (set gs $\cup$ set bs $) \cup f$ st' set ?res $\subseteq$ dgrad-p-set d
$m$ by $\operatorname{simp}$
from $\langle(p, q) \in$ set sps $\langle$ set sps $\subseteq$ set $p s\rangle p s$-sub
have $f s t p \in f s t$ ' set bs and fst $q \in f s t$ ' (set gs $\cup$ set bs) by auto
with $\langle f s t$ ' set $b s \subseteq d g r a d-p$-set $d$ m> gs-bs-sub
have fst $p \in d g r a d$ - $p$-set $d m$ and fst $q \in d g r a d-p$-set $d m$ by auto
with $d g$ res-sub show crit-pair-cbelow-on $d m(f s t$ ' (set gs $\cup$ set bs $) \cup f s t$ 'set
?res) (fst p) (fst q)
using 〈fst $p \neq 0\rangle\langle f s t q \neq 0\rangle$
proof (rule spoly-red-zero-imp-crit-pair-cbelow-on)
from assms gb uid sps-sub $\langle(p, q) \in$ set sps〉
show (red (fst' (set gs $\cup$ set bs) $\cup$ fst' set (fst (rcp gs bs (ps -- sps) sps
data))) )**
(spoly (fst p) (fst q)) 0
by (rule rcp-specD6)
qed
qed
end

### 6.5 Suitable Instances of the add-basis Parameter

definition add-basis-naive :: ('a, 'b, 'c, 'd) abT
where add-basis-naive gs bs ns data $=b s$ @ $n s$
lemma ab-spec-add-basis-naive: ab-spec add-basis-naive
by (rule ab-specI, simp-all add: add-basis-naive-def)
definition add-basis-sorted $::\left(n a t \times{ }^{\prime} d \Rightarrow\left({ }^{\prime} a,^{\prime} b,{ }^{\prime} c\right)\right.$ pdata $\Rightarrow\left({ }^{\prime} a,^{\prime} b,{ }^{\prime} c\right)$ pdata $\Rightarrow$ bool $) \Rightarrow\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c,{ }^{\prime} d\right) a b T$
where add-basis-sorted rel gs bs ns data $=$ merge-wrt (rel data) bs ns
lemma ab-spec-add-basis-sorted: ab-spec (add-basis-sorted rel)
by (rule ab-specI, simp-all add: add-basis-sorted-def set-merge-wrt)
definition card-keys :: $\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b::\right.$ zero $) \Rightarrow$ nat
where card-keys $=$ card $\circ$ keys
definition (in ordered-term) canon-basis-order :: 'd $\Rightarrow\left(' t,{ }^{\prime} b:: z e r o,{ }^{\prime} c\right)$ pdata $\Rightarrow$ ('t, 'b, 'c) pdata $\Rightarrow$ bool
where canon-basis-order data p $q \longleftrightarrow$

```
(let cp = card-keys (fst p); cq = card-keys (fst q) in
    cp<cq\vee (cp=cq\wedgelt (fst p)}\mp@subsup{\prec}{t}{}lt(fst q))
```

abbreviation (in ordered-term) add-basis-canon $\equiv$ add-basis-sorted canon-basis-order

```
6.6 Special Case: Scalar Polynomials
context gd-powerprod
begin
lemma remdups-map-component-of-term-punit:
    remdups (map (\lambda-. ()) (punit.Keys-to-list (map fst bs))) =
        (if (\forallb\inset bs. fst b=0) then [] else [()])
proof (split if-split, intro conjI impI)
    assume }\forallb\inset bs. fst b=
    hence fst' set bs\subseteq{0} by blast
    hence Keys (fst'set bs)={} by (metis Keys-empty Keys-zero subset-singleton-iff)
    hence punit.Keys-to-list (map fst bs)= []
            by (simp add: set-empty[symmetric] punit.set-Keys-to-list del: set-empty)
    thus remdups (map (\lambda-. ()) (punit.Keys-to-list (map fst bs))) = [] by simp
next
    assume }\neg(\forallb\in\mathrm{ set bs. fst b=0)
    hence }\existsb\in\mathrm{ set bs. fst }b\not=0\mathrm{ by simp
    then obtain b where b\in set bs and fst b\not=0 ..
    hence Keys (fst'set bs)}={}\mathrm{ by (meson Keys-not-empty <fst b}\not=0`\mathrm{ imageI)
    hence set (punit.Keys-to-list (map fst bs)) \not={} by (simp add: punit.set-Keys-to-list)
    hence punit.Keys-to-list (map fst bs) \not= [] by simp
    thus remdups (map (\lambda-. ()) (punit.Keys-to-list (map fst bs))) = [()]
    by (metis (full-types) remdups-adj.cases old.unit.exhaust Nil-is-map-conv <punit.Keys-to-list
(map fst bs)\not=[]> distinct-length-2-or-more distinct-remdups remdups-eq-nil-right-iff)
qed
lemma count-const-lt-components-punit [code]:
    punit.count-const-lt-components hs =
    (if (\existsh\inset hs. punit.const-lt-component (fst h)=Some ()) then 1 else 0)
proof (simp add: punit.count-const-lt-components-def cong del: image-cong-simp,
    simp add: card-set [symmetric] cong del: image-cong-simp, rule)
    assume \existsh\inset hs. punit.const-lt-component (fst h)= Some ()
    then obtain h}\mathrm{ where }h\in\mathrm{ set hs and punit.const-lt-component (fst h)=Some
() ..
    from this(2) have (punit.const-lt-component \circ fst) h=Some () by simp
    with }\langleh\in\mathrm{ set hs` have Some () ( punit.const-lt-component ○ fst)' set hs
    by (metis rev-image-eqI)
    hence {x.x=Some () ^x\in(punit.const-lt-component \circfst)'set hs} ={Some
()} by auto
    thus card {x. x = Some () ^ x ( punit.const-lt-component \circfst)'set hs} =
Suc 0 by simp
qed
```

```
lemma count-rem-components-punit [code]:
    punit.count-rem-components bs =
    (if ( }\forallb\in\mathrm{ set bs. fst b=0) then 0
    else
        if (\existsb\inset bs. fst b}=0\wedge punit.const-lt-component (fst b)=Some ()) then
0 else 1)
proof (cases }\forallb\in\mathrm{ set bs.fst b=0)
    case True
    thus ?thesis by (simp add: punit.count-rem-components-def remdups-map-component-of-term-punit)
next
    case False
    have eq: (\existsb\inset [b\leftarrowbs.fst b\not=0]. punit.const-lt-component (fst b)=Some ())
=
(\existsb\inset bs. fst b\not=0^ punit.const-lt-component (fst b)=Some ())
    by (metis (mono-tags, lifting) filter-set member-filter)
    show ?thesis
    by (simp only: False punit.count-rem-components-def eq if-False
        remdups-map-component-of-term-punit count-const-lt-components-punit punit-component-of-term,
simp)
qed
lemma full-gb-punit [code]:
    punit.full-gb bs =( if (\forallb\inset bs.fst b=0) then [] else [(1,0, default )])
    by (simp add: punit.full-gb-def remdups-map-component-of-term-punit)
abbreviation add-pairs-punit-canon \equiv
    punit.add-pairs (punit.new-pairs-sorted punit.canon-pair-order) punit.product-crit
punit.chain-ncrit
    punit.chain-ocrit punit.canon-pair-comb
lemma ap-spec-add-pairs-punit-canon: punit.ap-spec add-pairs-punit-canon
    using punit.np-spec-new-pairs-sorted punit.icrit-spec-product-crit punit.ncrit-spec-chain-ncrit
    punit.ocrit-spec-chain-ocrit set-merge-wrt
    by (rule punit.ap-spec-add-pairs)
end
end
```


## 7 Buchberger's Algorithm

```
theory Buchberger
    imports Algorithm-Schema
begin
context gd-term
begin
```


### 7.1 Reduction

definition trdsp::('t $\left.\Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata-pair $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.$ field $)$ where trdsp bs $p \equiv \operatorname{trd}$ bs $(\operatorname{spoly}(f s t(f s t ~ p))(f s t(s n d p)))$
lemma trdsp-alt: trdsp bs $(p, q)=$ trd bs $($ spoly $(f s t p)(f s t ~ q))$
by (simp add: trdsp-def)
lemma trdsp-in-pmdl: trdsp bs $(p, q) \in p m d l$ (insert (fst $p$ ) (insert (fst q) (set bs)))
unfolding trdsp-alt
proof (rule pmdl-closed-trd)
have spoly $(f s t p)(f s t ~ q) \in p m d l\{f s t p, f s t q\}$
proof (rule pmdl-closed-spoly)
show fst $p \in p m d l\{f s t p$, fst $q\}$ by (rule pmdl.span-base, simp)
next
show fst $q \in p m d l\{f s t p, f s t ~ q\}$ by (rule pmdl.span-base, simp)
qed
also have $\ldots \subseteq p m d l($ insert $(f s t p)($ insert $(f s t q)($ set $b s)))$
by (rule pmdl.span-mono, simp)
finally show spoly $(f s t p)(f s t q) \in \operatorname{pmdl}($ insert $(f s t p)(i n s e r t(f s t q)($ set bs $)))$
.
next
have set $b s \subseteq$ insert $(f s t p)($ insert $(f s t q)($ set bs)) by blast
also have $\ldots \subseteq p m d l$ ( insert $(f s t p)($ insert $(f s t q)($ set $b s)))$
by (fact pmdl.span-superset)
finally show set $b s \subseteq p m d l($ insert $($ fst $p)($ insert $(f s t q)($ set bs))).
qed
lemma dgrad-p-set-le-trdsp:
assumes dickson-grading $d$
shows dgrad-p-set-le $d\{$ trdsp bs $(p, q)\}$ (insert (fst p) (insert (fst q) (set bs)))
proof -
let $? h=t r d s p b s(p, q)$
 trd-red-rtrancl)
with assms have dgrad-p-set-le d \{?h\} (insert (spoly (fst p) (fst q)) (set bs))
by (rule dgrad-p-set-le-red-rtrancl)
also have dgrad-p-set-le d ... $(\{f s t p, f s t ~ q\} \cup$ set $b s)$
proof (rule dgrad-p-set-leI-insert)
show dgrad-p-set-le $d($ set bs) $(\{f$ st $p, f s t q\} \cup$ set bs) by (rule dgrad-p-set-le-subset, blast)
next
from assms have dgrad-p-set-le d \{spoly (fst p) (fst q)\} \{fst p, fst q\}
by (rule dgrad-p-set-le-spoly)
also have dgrad-p-set-le $d \ldots(\{f s t p, f s t ~ q\} \cup$ set $b s)$
by (rule dgrad-p-set-le-subset, blast)
finally show dgrad-p-set-le $d\{$ spoly $(f s t p)(f s t q)\}(\{f s t p, f s t q\} \cup$ set $b s)$.
qed
finally show ?thesis by simp

## qed

lemma components-trdsp-subset:

```
    component-of-term' keys (trdsp bs \((p, q)) \subseteq\) component-of-term'Keys (insert
(fst p) (insert (fst q) ( set bs)))
proof -
    let \(? h=\operatorname{trdsp} b s(p, q)\)
```



```
trd-red-rtrancl)
    hence component-of-term'keys ? \(h \subseteq\)
                component-of-term ‘keys (spoly \((f s t p)(f s t ~ q)) \cup\) component-of-term ‘
Keys (set bs)
    by (rule components-red-rtrancl-subset)
    also have \(\ldots \subseteq\) component-of-term'Keys \(\{\) fst \(p\), fst \(q\} \cup\) component-of-term'
Keys (set bs)
    using components-spoly-subset by force
    also have \(\ldots=\) component-of-term ' Keys \((\operatorname{insert}(f s t p)(\operatorname{insert}(f s t q)(\) set bs))\()\)
    by (simp add: Keys-insert image-Un Un-assoc)
    finally show ?thesis .
qed
definition gb-red-aux :: ('t, 'b:: field, 'c) pdata list \(\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)\) pdata-pair list \(\Rightarrow\)
                    ( \(' t \Rightarrow{ }_{0}{ }^{\prime} b\) ) list
    where \(g b\)-red-aux bs ps =
                (let bs \({ }^{\prime}=\) map fst bs in
                        filter \((\lambda h . h \neq 0)(\operatorname{map}(t r d s p b s) p s)\)
        )
```

Actually, $g b$-red-aux is only called on singleton lists.
lemma set-gb-red-aux: set (gb-red-aux bs ps)=(trdsp (map fst bs))' set ps $-\{0\}$
by (simp add: gb-red-aux-def, blast)
lemma in-set-gb-red-auxI:
assumes $(p, q) \in$ set $p s$ and $h=$ trdsp (map fst $b s)(p, q)$ and $h \neq 0$
shows $h \in$ set (gb-red-aux bs ps)
using assms (1, 3) unfolding set-gb-red-aux assms(2) by force
lemma in-set-gb-red-auxE:
assumes $h \in$ set (gb-red-aux bs ps)
obtains $p q$ where $(p, q) \in$ set $p s$ and $h=$ trdsp (map fst bs) $(p, q)$
using assms unfolding set-gb-red-aux by force
lemma gb-red-aux-not-zero: $0 \notin$ set (gb-red-aux bs ps)
by (simp add: set-gb-red-aux)
lemma gb-red-aux-irredudible:
assumes $h \in$ set (gb-red-aux bs ps) and $b \in$ set bs and fst $b \neq 0$
shows $\neg l t(f s t b) a d d s_{t} l t h$
proof

```
    assume lt (fst b) addst (lt h)
    from assms(1) obtain p q :: ('t, 'b,'c) pdata where h: h= trdsp (map fst bs)
(p,q)
    by (rule in-set-gb-red-auxE)
    have \negis-red (set (map fst bs)) h unfolding h trdsp-def by (rule trd-irred)
    moreover have is-red (set (map fst bs)) h
    proof (rule is-red-addsI)
        from assms(2) show fst b\in set (map fst bs) by (simp)
    next
        from assms(1) have h\not=0 by (simp add: set-gb-red-aux)
        thus lt h}\in\mathrm{ keys }h\mathrm{ by (rule lt-in-keys)
    qed fact+
    ultimately show False ..
qed
lemma gb-red-aux-dgrad-p-set-le:
    assumes dickson-grading d
    shows dgrad-p-set-le d (set (gb-red-aux bs ps)) (args-to-set ([],bs, ps))
proof (rule dgrad-p-set-leI)
    fix }
    assume h\in set (gb-red-aux bs ps)
    then obtain pq where (p,q)\in set ps and h:h=trdsp (map fst bs) (p,q)
        by (rule in-set-gb-red-auxE)
    from assms have dgrad-p-set-le d {h} (insert (fst p) (insert (fst q) (set (map fst
bs))))
        unfolding }h\mathrm{ by (rule dgrad-p-set-le-trdsp)
    also have dgrad-p-set-le d ... (args-to-set ([],bs, ps))
    proof (rule dgrad-p-set-le-subset, intro insert-subsetI)
        from «(p,q)\in set ps` have fst p\infst'fst' set ps by force
        thus fst p \in args-to-set ([],bs, ps) by (auto simp add: args-to-set-alt)
    next
        from «(p,q) \in set ps> have fst q \in fst' snd ' set ps by force
        thus fst q \in args-to-set ([], bs, ps) by (auto simp add: args-to-set-alt)
    next
        show set (map fst bs)\subseteqargs-to-set ([],bs, ps) by (auto simp add: args-to-set-alt)
    qed
    finally show dgrad-p-set-le d {h} (args-to-set ([], bs, ps)).
qed
lemma components-gb-red-aux-subset:
    component-of-term'Keys (set (gb-red-aux bs ps)) \subseteqcomponent-of-term'Keys
(args-to-set ([],bs,ps))
proof
    fix }
    assume k\in component-of-term 'Keys (set (gb-red-aux bs ps))
    then obtain v where v\inKeys (set (gb-red-aux bs ps)) and k: k=compo-
nent-of-term v ..
    from this(1) obtain h where h\inset (gb-red-aux bs ps) and v\inkeys h by
(rule in-KeysE)
```

from this(1) obtain $p q$ where $(p, q) \in$ set $p s$ and $h: h=t r d s p(m a p f s t b s)$ ( $p, q$ )
by (rule in-set-gb-red-auxE)
from $\langle v \in$ keys $h\rangle$ have $k \in$ component-of-term ' keys $h$ by (simp add: $k$ )
have component-of-term ' keys $h \subseteq$ component-of-term' Keys (insert (fst p)
(insert (fst q) (set (map fst bs))))
unfolding $h$ by (rule components-trdsp-subset)
also have $\ldots \subseteq$ component-of-term' Keys (args-to-set ([],bs, ps))
proof (rule image-mono, rule Keys-mono, intro insert-subsetI)
from $\langle(p, q) \in$ set $p s\rangle$ have $f s t p \in f s t$ ' fst' set $p s$ by force
thus $f$ st $p \in \operatorname{args-to-set~([],bs,~ps)~by~(auto~simp~add:~args-to-set-alt)~}$
next
from $\langle(p, q) \in$ set $p s\rangle$ have $f s t q \in f s t$ 'snd' set $p s$ by force
thus $f$ st $q \in$ args-to-set ([], bs, ps) by (auto simp add: args-to-set-alt)
next
show set $($ map fst bs) $\subseteq$ args-to-set $([], b s, p s)$ by (auto simp add: args-to-set-alt)
qed
finally have component-of-term'keys $h \subseteq$ component-of-term' Keys (args-to-set ([], bs, ps)).
with $\langle k \in$ component-of-term ' keys $h\rangle$ show $k \in$ component-of-term 'Keys (args-to-set ([], bs, ps)) ..
qed
lemma $p m d l-g b-r e d-a u x: ~ s e t(g b-r e d-a u x ~ b s ~ p s) \subseteq p m d l(a r g s-t o-s e t([], b s, p s))$
proof proof
fix $h$
assume $h \in$ set (gb-red-aux bs ps)
then obtain $p q$ where $(p, q) \in$ set $p s$ and $h: h=\operatorname{trdsp}(\operatorname{map} f s t b s)(p, q)$
by (rule in-set-gb-red-auxE)
have $h \in \operatorname{pmdl}($ insert $(f s t p)($ insert $(f s t q)($ set $($ map fst bs)))) unfolding $h$ by (fact trdsp-in-pmdl)
also have $\ldots \subseteq p m d l($ args-to-set $([], b s, p s))$
proof (rule pmdl.span-mono, intro insert-subsetI)
from $\langle(p, q) \in$ set $p s\rangle$ have fst $p \in f s t$ 'fst' set ps by force
thus $f$ st $p \in$ args-to-set ([], bs, ps) by (auto simp add: args-to-set-alt)
next
from $\langle(p, q) \in$ set $p s\rangle$ have fst $q \in f s t$ ' snd' set $p s$ by force
thus $f$ st $q \in$ args-to-set ([],bs, ps) by (auto simp add: args-to-set-alt)
next
show set $($ map fst bs) $\subseteq$ args-to-set $([], b s, p s)$ by (auto simp add: args-to-set-alt)
qed
finally show $h \in \operatorname{pmdl}(\operatorname{args-to-set}([], b s, p s))$.
qed
lemma gb-red-aux-spoly-reducible:
assumes $(p, q) \in$ set $p s$
shows $(\text { red }(f s t ' \text { set } b s \cup \text { set }(g b-r e d-a u x ~ b s ~ p s)))^{* *}($ spoly $(f s t ~ p)(f s t q)) 0$ proof -
define $h$ where $h=$ trdsp (map fst $b s)(p, q)$

```
    from trd-red-rtrancl[of map fst bs spoly (fst p) (fst q)]
    have (red (set (map fst bs)))** (spoly (fst p) (fst q)) h
    by (simp only: h-def trdsp-alt)
    hence (red (fst'set bs \cup set (gb-red-aux bs ps)))** (spoly (fst p) (fst q)) h
    proof (rule red-rtrancl-subset)
    show set (map fst bs)\subseteqfst'set bs U set (gb-red-aux bs ps) by simp
qed
moreover have (red (fst` set bs \cup set (gb-red-aux bs ps)))** h 0
proof (cases h=0)
    case True
    show ?thesis unfolding True ..
    next
        case False
    hence red {h} h0 by (rule red-self)
    hence red (fst' set bs \cup set (gb-red-aux bs ps)) h 0
    proof (rule red-subset)
    from assms h-def False have h\in set (gb-red-aux bs ps) by (rule in-set-gb-red-auxI)
            thus {h}\subseteqfst' set bs \cup set (gb-red-aux bs ps) by simp
    qed
    thus ?thesis ..
    qed
    ultimately show ?thesis by simp
qed
definition gb-red :: ('t, 'b::field, 'c::default, 'd) complT
    where gb-red gs bs ps sps data = (map ( }\lambda\textrm{h}.(h,default)) (gb-red-aux (gs @ bs)
sps), snd data)
lemma fst-set-fst-gb-red: fst'set (fst (gb-red gs bs ps sps data)) = set (gb-red-aux
(gs @ bs) sps)
    by (simp add: gb-red-def, force)
lemma rcp-spec-gb-red: rcp-spec gb-red
proof (rule rcp-specI)
    fix gs bs::('t, 'b,'c) pdata list and ps sps and data::nat }\times'
    from gb-red-aux-not-zero show 0}\not\infst'set (fst (gb-red gs bs ps sps data)), 
    unfolding fst-set-fst-gb-red .
next
    fix gs bs::('t, 'b, 'c) pdata list and ps sps h b and data::nat }\times\mp@subsup{}{}{\prime}
    assume h\in set (fst (gb-red gs bs ps sps data)) and b\in set gs \cup set bs
    from this(1) have fst h\infst'set (fst (gb-red gs bs ps sps data)) by simp
    hence fst h \in set (gb-red-aux (gs @ bs) sps) by (simp only: fst-set-fst-gb-red)
    moreover from <b\in set gs U set bs> have b\in set (gs @ bs) by simp
    moreover assume fst b}=
    ultimately show \neglt (fst b) addst lt (fst h) by (rule gb-red-aux-irredudible)
next
    fix gs bs::('t, 'b, 'c) pdata list and ps sps and d::'a a nat and data::nat }\times '
    assume dickson-grading d
    hence dgrad-p-set-le d (set (gb-red-aux (gs @ bs) sps)) (args-to-set ([], gs @ bs,
```

```
sps))
    by (rule gb-red-aux-dgrad-p-set-le)
    also have ... = args-to-set (gs, bs, sps) by (simp add: args-to-set-alt image-Un)
    finally show dgrad-p-set-le d (fst'set (fst (gb-red gs bs ps sps data))) (args-to-set
(gs,bs, sps))
    by (simp only: fst-set-fst-gb-red)
next
    fix gs bs::('t, 'b, 'c) pdata list and ps sps and data::nat }\times\mathrm{ 'd
    have component-of-term'Keys (set (gb-red-aux (gs @ bs) sps)) \subseteq
                component-of-term 'Keys (args-to-set ([], gs @ bs, sps))
    by (rule components-gb-red-aux-subset)
    also have ... = component-of-term'Keys (args-to-set (gs, bs, sps))
        by (simp add: args-to-set-alt image-Un)
    finally show component-of-term'Keys(fst'set (fst (gb-red gs bs ps sps data)))
\subseteq
                                    component-of-term'Keys (args-to-set (gs, bs, sps)) by (simp only:
fst-set-fst-gb-red)
next
    fix gs bs::('t, 'b, 'c) pdata list and ps sps and data::nat }\times\mp@subsup{}{}{\prime}
    have set (gb-red-aux (gs @ bs) sps)\subseteqpmdl (args-to-set ([], gs @ bs,sps))
        by (fact pmdl-gb-red-aux)
    also have ... = pmdl (args-to-set (gs, bs, sps)) by (simp add: args-to-set-alt
image-Un)
    finally have fst'set (fst (gb-red gs bs ps sps data))\subseteqpmdl (args-to-set (gs,bs,
sps))
    by (simp only: fst-set-fst-gb-red)
    moreover {
        fix p q :: ('t, 'b, 'c) pdata
        assume (p,q)\in set sps
        hence (red (fst'set (gs @ bs)\cup set (gb-red-aux (gs @ bs) sps)))** (spoly (fst
p) (fst q)) 0
            by (rule gb-red-aux-spoly-reducible)
    }
    ultimately show
    fst ' set (fst (gb-red gs bs ps sps data))\subseteqpmdl (args-to-set (gs,bs, sps))}
        (}\forall(p,q)\inset sps
            set sps \subseteq set bs \times (set gs \cup set bs) \longrightarrow
                (red (fst' (set gs U set bs) \cupfst' set (fst (gb-red gs bs ps sps data))))**
(spoly (fst p) (fst q)) 0)
    by (auto simp add: image-Un fst-set-fst-gb-red)
qed
lemmas compl-struct-gb-red = compl-struct-rcp[OF rcp-spec-gb-red]
lemmas compl-pmdl-gb-red = compl-pmdl-rcp[OF rcp-spec-gb-red]
lemmas compl-conn-gb-red = compl-conn-rcp[OF rcp-spec-gb-red]
```


### 7.2 Pair Selection

```
primrec \(g b\)-sel :: ('t, 'b::zero, ' \(\left.c,{ }^{\prime} d\right)\) sel \(T\) where
```

$$
\begin{aligned}
& \text { gb-sel gs bs }[] \text { data }=[] \mid \\
& \text { gb-sel gs bs }(p \# p s) \text { data }=[p]
\end{aligned}
$$

lemma sel-spec-gb-sel: sel-spec gb-sel
proof (rule sel-specI)
fix $g s$ bs :: ('t, 'b, 'c) pdata list and ps::('t, 'b, 'c) pdata-pair list and data::nat $\times$ 'd
assume $p s \neq[]$
then obtain $p p^{\prime}$ where $p s: p s=p \# p s^{\prime}$ by (meson list.exhaust)
show gb-sel gs bs ps data $\neq[] \wedge$ set (gb-sel gs bs ps data) $\subseteq$ set ps by (simp add:
$p s)$
qed

### 7.3 Buchberger's Algorithm

lemma struct-spec-gb: struct-spec gb-sel add-pairs-canon add-basis-canon gb-red using sel-spec-gb-sel ap-spec-add-pairs-canon ab-spec-add-basis-sorted compl-struct-gb-red by (rule struct-specI)
definition gb-aux $::\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$ nat $\times n a t \times{ }^{\prime} d \Rightarrow\left(' t,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$
$\left(' t, ' b,{ }^{\prime} c\right)$ pdata-pair list $\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b::\right.$ field, ' $c::$ default $)$ pdata list where $g b$-aux $=$ gb-schema-aux gb-sel add-pairs-canon add-basis-canon gb-red
lemmas gb-aux-simps $[$ code $]=g b$-schema-aux-simps $[O F$ struct-spec-gb, folded gb-aux-def]
definition $g b::\left(' t,{ }^{\prime} b,{ }^{\prime} c\right) p d a t a^{\prime}$ list $\Rightarrow{ }^{\prime} d \Rightarrow\left(' t,{ }^{\prime} b:: f i e l d,{ }^{\prime} c:: d e f a u l t\right) p d a t a{ }^{\prime}$ list where $g b=g b$-schema-direct $g b$-sel add-pairs-canon add-basis-canon gb-red
lemmas gb-simps $[$ code $]=g b$-schema-direct-def[of gb-sel add-pairs-canon add-basis-canon gb-red, folded gb-def gb-aux-def]
lemmas gb-is $G B=$ gb-schema-direct-isGB[OF struct-spec-gb compl-conn-gb-red, folded $g b-d e f]$
lemmas $g b$-pmdl $=g b$-schema-direct-pmdl[ $[O F$ struct-spec-gb compl-pmdl-gb-red, folded gb-def]

### 7.3.1 Special Case: punit

lemma (in gd-term) struct-spec-gb-punit: punit.struct-spec punit.gb-sel add-pairs-punit-canon
punit.add-basis-canon punit.gb-red
using punit.sel-spec-gb-sel ap-spec-add-pairs-punit-canon ab-spec-add-basis-sorted punit.compl-struct-gb-red
by (rule punit.struct-specI)
definition gb-aux-punit :: ('a, 'b, 'c) pdata list $\Rightarrow$ nat $\times$ nat $\times{ }^{\prime} d \Rightarrow\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)$ pdata list $\Rightarrow$

$$
\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right) \text { pdata-pair list } \Rightarrow\left({ }^{\prime} a,{ }^{\prime} b:: \text { field, ' } c:: \text { default }\right) \text { pdata list }
$$

where gb-aux-punit $=$ punit.gb-schema-aux punit.gb-sel add-pairs-punit-canon punit.add-basis-canon punit.gb-red
lemmas gb-aux-punit-simps $[$ code $]=$ punit.gb-schema-aux-simps $[$ OF struct-spec-gb-punit , folded gb-aux-punit-def]
definition gb-punit $::\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right) p d a t a^{\prime}$ list $\Rightarrow{ }^{\prime} d \Rightarrow\left({ }^{\prime} a\right.$, 'b::field, ${ }^{\prime} c::$ default $) p d a t a{ }^{\prime}$ list
where gb-punit $=$ punit.gb-schema-direct punit.gb-sel add-pairs-punit-canon punit.add-basis-canon punit.gb-red
lemmas gb-punit-simps $[$ code $]=$ punit.gb-schema-direct-def[of punit.gb-sel add-pairs-punit-canon punit.add-basis-canon punit.gb-red, folded gb-punit-def
gb-aux-punit-def]
lemmas gb-punit-isGB = punit.gb-schema-direct-isGB[OF struct-spec-gb-punit punit.compl-conn-gb-red, folded gb-punit-def]
lemmas $g b$-punit-pmdl $=$ punit.gb-schema-direct-pmdl $[$ OF struct-spec-gb-punit punit.compl-pmdl-gb-red, folded gb-punit-def]
end
end

## 8 Benchmark Problems for Computing Gröbner Bases

theory Benchmarks<br>imports Polynomials.MPoly-Type-Class-OAlist<br>begin

This theory defines various well-known benchmark problems for computing Gröbner bases. The actual tests of the different algorithms on these problems are contained in the theories whose names end with -Examples.

### 8.1 Cyclic

definition cycl-pp :: nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ (nat, nat) pp
where cycl-pp ndi$=\operatorname{sparse}_{0}(\operatorname{map}(\lambda k .(\operatorname{modulo}(k+i) n, 1))[0 . .<d])$
definition cyclic :: (nat, nat) pp nat-term-order $\Rightarrow$ nat $\Rightarrow$ ((nat, nat) $p p \Rightarrow_{0}$
' $a::\{$ zero,one,uminus $\}$ ) list
where cyclic to $n=$

```
(let xs = [0..<n] in
    (map (\lambdad. distro to (map (\lambdai. (cycl-pp nd i, 1)) xs)) [1..<n])@
    [distro to [(cycl-pp n n 0, 1), (0, -1)]]
    )
```

cyclic $n$ is a system of $n$ polynomials in $n$ indeterminates, with maximum degree $n$.

### 8.2 Katsura

definition katsura-poly :: (nat, nat) pp nat-term-order $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ ((nat, nat) $p p \Rightarrow_{0}$ 'a::comm-ring-1)
where katsura-poly to $n i=$
change-ord to $\left(\left(\sum j::\right.\right.$ int $=-$ int $n . .<n+1$. if abs $(i-j) \leq n$ then $V_{0}$ $($ nat $(a b s j)) * V_{0}(n a t(a b s(i-j)))$ else 0$\left.)-V_{0} i\right)$
definition katsura :: (nat, nat) pp nat-term-order $\Rightarrow$ nat $\Rightarrow$ ((nat, nat) $p p \Rightarrow_{0}$ 'a::comm-ring-1) list
where katsura to $n=$
$($ let $x s=[0 . .<n]$ in
( $\operatorname{distr}_{0}$ to $\left(\left(\operatorname{sparse}_{0}[(0,1)], 1\right) \#\left(\operatorname{map}\left(\lambda i .\left(\right.\right.\right.\right.$ sparse $_{0}[($ Suc $\left.\left.i, 1)], 2\right)\right)$
$x s) @[(0,-1)])) \#$
(map (katsura-poly to $n$ ) xs)
)
For $\left(1::^{\prime} a\right) \leq n$, katsura $n$ is a system of $n+1$ polynomials in $n+1$ indeterminates, with maximum degree 2.

### 8.3 Eco

definition eco-poly :: (nat, nat) pp nat-term-order $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ ((nat, nat) $p p \Rightarrow{ }_{0}{ }^{\prime} a::$ comm-ring-1)
where eco-poly to $m i=$ distr $_{0}$ to $\left(\left(\right.\right.$ sparse $\left._{0}[(i, 1),(m, 1)], 1\right) \# \operatorname{map}\left(\lambda j .\left(\right.\right.$ sparse $_{0}[(j, 1),(j+$ $i+1,1),(m, 1)], 1))[0 . .<m-i-1])$

```
definition eco :: (nat, nat) pp nat-term-order \(\Rightarrow\) nat \(\Rightarrow\left((n a t, n a t) p p \Rightarrow_{0}{ }^{\prime} a::\right.\) comm-ring-1 \()\)
list
    where eco to \(n=\)
        (let \(m=n-1 \mathrm{in}\)
                            \(\left(\operatorname{distr}_{0}\right.\) to \(\left.\left(\left(\operatorname{map}\left(\lambda j .\left(\operatorname{sparse}_{0}[(j, 1)], 1\right)\right)[0 . .<m]\right) @[(0,1)]\right)\right) \#\)
    ( distr \(_{0}\) to \(\left[\left(\right.\right.\) sparse \(\left._{0}[(m-1,1),(m, 1)], 1\right),(0,-\) of-nat \(\left.\left.m)\right]\right) \#\)
    (rev (map (eco-poly to \(m\) ) \([0 . .<m-1]))\)
    )
```

For $\left(2:^{\prime}: a\right) \leq n$, eco $n$ is a system of $n$ polynomials in $n$ indeterminates, with maximum degree 3.

### 8.4 Noon

definition noon-poly :: (nat, nat) pp nat-term-order $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ ((nat, nat) $p p \Rightarrow_{0}{ }^{\prime} a::$ comm-ring-1)
where noon-poly to $n i=$

```
(let ten \(=\) of-nat 10; eleven \(=-\) of-nat 11 in
    distr \(_{0}\) to \(\left(\left(\operatorname{map}\left(\lambda j\right.\right.\right.\). if \(j=i\) then sparse \(_{0}[(i, 1)]\), eleven \()\) else (sparse \({ }_{0}\)
\([(j, 2),(i, 1)]\), ten \())[0 . .<n]) @\)
    \([(0\), ten \()]))\)
```

definition noon $::$ (nat, nat) pp nat-term-order $\Rightarrow$ nat $\Rightarrow$ ((nat, nat) $p p \Rightarrow_{0}$ 'a::comm-ring-1) list
where noon to $n=($ noon-poly to $n 1) \#($ noon-poly to $n 0) \#($ map (noon-poly to $n$ ) $[2 . .<n])$

For (2::' $a$ ) $\leq n$, noon $n$ is a system of $n$ polynomials in $n$ indeterminates, with maximum degree 3 .
end

## 9 Code Equations Related to the Computation of Gröbner Bases

```
theory Algorithm-Schema-Impl
    imports Algorithm-Schema Benchmarks
begin
lemma card-keys-MP-oalist [code]: card-keys (MP-oalist xs) = length (fst (list-of-oalist-ntm
xs))
proof -
    let ?rel = ko.lt (key-order-of-nat-term-order-inv (snd (list-of-oalist-ntm xs)))
    have irreflp ?rel by (simp add: irreflp-def)
    moreover have transp ?rel by (simp add: lt-of-nat-term-order-alt)
    ultimately have *: distinct (map fst (fst (list-of-oalist-ntm xs))) using oa-ntm.list-of-oalist-sorted
        by (rule distinct-sorted-wrt-irrefl)
    have card-keys (MP-oalist xs) = length (map fst (fst (list-of-oalist-ntm xs)))
    by (simp only: card-keys-def keys-MP-oalist image-set o-def oa-ntm.sorted-domain-def [symmetric],
                rule distinct-card, fact *)
    also have ... = length (fst (list-of-oalist-ntm xs)) by simp
    finally show ?thesis.
qed
end
```

theory Code-Target-Rat
imports Complex-Main HOL-Library.Code-Target-Numeral
begin

Mapping type rat to type "Rat.rat" in Isabelle/ML. Serialization for other target languages will be provided in the future.
context includes integer.lifting begin
lift-definition rat-of-integer :: integer $\Rightarrow$ rat is Rat.of-int .

```
lift-definition quotient-of \({ }^{\prime}::\) rat \(\Rightarrow\) integer \(\times\) integer is quotient-of.
lemma [code]: Rat.of-int (int-of-integer \(x)=\) rat-of-integer \(x\)
    by transfer simp
x))
    by transfer simp
end
code-printing
    type-constructor rat -
        (SML) Rat.rat |
    constant plus :: rat \(\Rightarrow-\Rightarrow-\rightharpoonup\)
        (SML) Rat.add |
    constant minus :: rat \(\Rightarrow-\Rightarrow-\rightharpoonup\)
    (SML) Rat.add ((-)) (Rat.neg ((-))) |
    constant times :: rat \(\Rightarrow-\Rightarrow--\)
        (SML) Rat.mult
    constant inverse :: rat \(\Rightarrow\) - -
        (SML) Rat.inv |
    constant divide :: rat \(\Rightarrow-\Rightarrow-\rightharpoonup\)
        (SML) Rat.mult ((-)) (Rat.inv ((-))) |
    constant rat-of-integer :: integer \(\Rightarrow\) rat \(\rightharpoonup\)
        (SML) Rat.of \({ }^{\prime}\)-int
    constant abs :: rat \(\Rightarrow\) - -
        (SML) Rat.abs
    constant \(0::\) rat \(\rightarrow\)
        (SML)! (Rat.make (0, 1))|
    constant 1 :: rat \(\rightarrow\)
        (SML) ! (Rat.make (1, 1)) |
    constant uminus :: rat \(\Rightarrow\) rat -
        (SML) Rat.neg
    constant HOL.equal :: rat \(\Rightarrow\) - -
        \((S M L)!((-\) Rat.rat \()=-) \mid\)
    constant quotient-of \({ }^{\prime} \rightarrow\)
        (SML) Rat.dest
```

lemma [code-unfold]: quotient-of $=\left(\lambda x\right.$. map-prod int-of-integer int-of-integer (quotient-of ${ }^{\prime}$
end

# 10 Sample Computations with Buchberger's Algorithm 

```
theory Buchberger-Examples
    imports Buchberger Algorithm-Schema-Impl Code-Target-Rat
begin
lemma (in gd-term) compute-trd-aux [code]:
    trd-aux fs p r =
        (if is-zero p then
        r
    else
        case find-adds fs (lt p) of
            None }=>\mathrm{ trd-aux fs (tail p) (plus-monomial-less r (lc p) (lt p))
            | Some f = trd-aux fs (tail p - monom-mult (lc p / lc f) (lp p - lp f) (tail
f))}
    )
    by (simp only: trd-aux.simps[of fs p r] plus-monomial-less-def is-zero-def)
```


### 10.1 Scalar Polynomials

global-interpretation punit $^{\prime}$ : gd-powerprod ord-pp-punit cmp-term ord-pp-strict-punit cmp-term
rewrites punit.adds-term $=($ adds $)$
and punit.pp-of-term $=(\lambda x . x)$
and punit.component-of-term $=(\lambda-.())$
and punit.monom-mult $=$ monom-mult-punit
and punit.mult-scalar $=$ mult-scalar-punit
and punit'.punit.min-term $=$ min-term-punit
and punit'.punit.lt $=l t$-punit cmp-term
and punit'.punit.lc $=l c-$ punit cmp-term
and punit'.punit.tail $=$ tail-punit cmp-term
and punit'.punit.ord- $p=o r d-p-p u n i t ~ c m p-t e r m ~$
and punit'.punit.ord-strict- $p=$ ord-strict-p-punit cmp-term
for cmp-term :: ('a::nat, 'b::\{nat,add-wellorder\}) pp nat-term-order
defines find-adds-punit $=$ punit' ${ }^{\prime}$.punit.find-adds
and trd-aux-punit $=$ punit ${ }^{\prime} \cdot$ punit.trd-aux
and trd-punit $=$ punit' ${ }^{\text {.punit.trd }}$
and spoly-punit $=$ punit' ${ }^{\prime}$.punit.spoly
and count-const-lt-components-punit $=$ punit'.punit.count-const-lt-components
and count-rem-components-punit $=$ punit' ${ }^{\prime}$.punit.count-rem-components
and const-lt-component-punit $=$ punit' ${ }^{\prime}$.punit.const-lt-component
and full-gb-punit $=$ punit' ${ }^{\prime}$.punit.full-gb
and add-pairs-single-sorted-punit $=$ punit' ${ }^{\prime}$.punit.add-pairs-single-sorted
and add-pairs-punit $=$ punit' ${ }^{\prime}$.punit.add-pairs
and canon-pair-order-aux-punit $=$ punit'.punit.canon-pair-order-aux
and canon-basis-order-punit $=$ punit' ${ }^{\prime}$.punit.canon-basis-order
and new-pairs-sorted-punit $=$ punit' ${ }^{\prime}$.punit.new-pairs-sorted
and product-crit-punit $=$ punit' $\cdot$.punit.product-crit
and chain-ncrit-punit $=$ punit'. punit.chain-ncrit
and chain-ocrit-punit $=$ punit'.punit.chain-ocrit
and apply-icrit-punit $=$ punit' ${ }^{\prime}$.punit.apply-icrit
and apply-ncrit-punit $=$ punit' ${ }^{\prime}$.punit.apply-ncrit
and apply-ocrit-punit $=$ punit' ${ }^{\prime}$.punit.apply-ocrit
and trdsp-punit $=$ punit' ${ }^{\prime}$.punit.trdsp
and gb-sel-punit $=$ punit' ${ }^{\prime}$ punit.gb-sel
and gb-red-aux-punit $=$ punit' ${ }^{\prime}$.punit.gb-red-aux
and $g b$-red-punit $=$ punit' $\cdot$ punit.gb-red
and gb-aux-punit $=$ punit'.punit.gb-aux-punit
and gb-punit $=$ punit' .punit.gb-punit - Faster, because incorporates product criterion.
subgoal by (fact gd-powerprod-ord-pp-punit)
subgoal by (fact punit-adds-term)
subgoal by (simp add: id-def)
subgoal by (fact punit-component-of-term)
subgoal by (simp only: monom-mult-punit-def)
subgoal by (simp only: mult-scalar-punit-def)
subgoal using min-term-punit-def by fastforce
subgoal by (simp only: lt-punit-def ord-pp-punit-alt)
subgoal by (simp only: lc-punit-def ord-pp-punit-alt)
subgoal by (simp only: tail-punit-def ord-pp-punit-alt)
subgoal by (simp only: ord-p-punit-def ord-pp-strict-punit-alt)
subgoal by (simp only: ord-strict-p-punit-def ord-pp-strict-punit-alt)
done
lemma compute-spoly-punit [code]:
spoly-punit to $p q=($ let $t 1=l t$-punit to $p ; t 2=l t$-punit to $q ; l=l$ cs 11 t2 in
(monom-mult-punit $(1 / l c$-punit to $p)(l-t 1) p)-($ monom-mult-punit
(1 / lc-punit to q) $(l-t 2) q))$
by (simp add: punit'.punit.spoly-def Let-def punit'.punit.lc-def)
lemma compute-trd-punit [code]: trd-punit to fs $p=$ trd-aux-punit to fs $p$ (change-ord to 0 )
by (simp only: punit'.punit.trd-def change-ord-def)
experiment begin interpretation trivariate ${ }_{0}-r a t$.

## lemma

lt-punit DRLEX $\left(X^{2} * Z^{\wedge} 3+3 * X^{2} * Y\right)=\operatorname{sparse}_{0}[(0,2),(2,3)]$
by eval
lemma
lc-punit DRLEX $\left(X^{2} * Z^{\wedge} 3+3 * X^{2} * Y\right)=1$
by eval

## lemma

tail-punit DRLEX $\left(X^{2} * Z^{\wedge} 3+3 * X^{2} * Y\right)=3 * X^{2} * Y$
by eval

## lemma

ord-strict-p-punit $\operatorname{DRLEX}\left(X^{2} * Z^{\wedge} 4-2 * Y^{\wedge} 3 * Z^{2}\right)\left(X^{2} * Z^{\wedge} 7+2 *\right.$ $Y^{\wedge} 3 * Z^{2}$ )
by eval

## lemma

trd-punit DRLEX $\left[Y^{2} * Z+2 * Y * Z \wedge 3\right]\left(X^{2} * Z^{\wedge} 4-2 * Y \wedge 3 * Z \wedge\right.$
3) $=$
$X^{2} * Z^{\wedge} 4+Y^{\wedge} 4 * Z$
by eval

## lemma

```
spoly-punit DRLEX \(\left(X^{2} * Z^{\wedge} 4-2 * Y\right.\) ^3 \(\left.* Z^{2}\right)\left(Y^{2} * Z+2 * Z \wedge 3\right)=\)
```

    \(-2 * Y^{\wedge} 3 * Z^{2}-\left(C_{0}(1 / 2)\right) * X^{2} * Y^{2} * Z^{2}\)
    by eval
    
## lemma

gb-punit DRLEX

```
\(\left(X^{2} * Z^{\wedge} 4-2 * Y^{\wedge} 3 * Z^{2},()\right)\),
    \(\left(Y^{2} * Z+2 * Z^{\wedge} 3,()\right)\)
    ] ()\(=\)
    \(\left(-2 * Y^{\wedge} 3 * Z^{2}-\left(C_{0}(1 / 2)\right) * X^{2} * Y^{2} * Z^{2},()\right)\),
    \(\left(X^{2} * Z \wedge 4-2 * Y へ 3 * Z^{2},()\right)\),
    \(\left(Y^{2} * Z+2 * Z^{\wedge} 3,()\right)\),
    \(\left(-\left(C_{0}(1 / 2)\right) * X^{2} * Y^{\wedge} 4 * Z-2 * Y^{\wedge} 5 * Z,()\right)\)
    ]
    by eval
```


## lemma

```
gb-punit DRLEX
        \(\left(X^{2} * Z^{2}-Y,()\right)\),
        \(\left(Y^{2} * Z-1,()\right)\)
    ] ()\(=\)
        \(\left(-\left(Y^{\wedge} 3\right)+X^{2} * Z,()\right)\),
        \(\left(X^{2} * Z^{2}-Y,()\right)\),
        \(\left(Y^{2} * Z-1,()\right)\)
    ]
```

    by eval
    
## lemma

gb-punit DRLEX

$$
\left(X \wedge 3-X * Y * Z^{2},()\right)
$$

```
        \(\left(Y^{2} * Z-1,()\right)\)
        ] ()\(=\)
    [
    \(\left(-\left(X^{\wedge} 3 * Y\right)+X * Z,()\right)\),
    \(\left(X^{\wedge} 3-X * Y * Z^{2},()\right)\),
    \(\left(Y^{2} * Z-1,()\right)\),
    \(\left(-\left(X * Z^{\wedge} 3\right)+X^{\wedge} 5,()\right)\)
    ]
    by eval
lemma
    gb-punit DRLEX
    [
        \(\left(X^{2}+Y^{2}+Z^{2}-1,()\right)\),
        \((X * Y-Z-1,())\),
        \(\left(Y^{2}+X,()\right)\),
    \(\left(Z^{2}+X,()\right)\)
    ] ()\(=\)
    [
    (1, ())
    ]
    by eval
end
value [code] length (gb-punit DRLEX (map ( \(\lambda\). \((p,())\) ) ((katsura DRLEX 2)::(\(\Rightarrow_{0}\) rat) list)) ())
value [code] length (gb-punit DRLEX (map ( \(\lambda p .(p,()))((c y c l i c ~ D R L E X ~ 5)::(-\) \(\Rightarrow_{0}\) rat) list)) ())
```


### 10.2 Vector Polynomials

We must define the following four 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 monom-mult-pprod :: 'c::semiring-0 \(\Rightarrow\) ('a::nat, 'b::nat) \(p p \Rightarrow\) -
    where monom-mult-pprod \(=\) pprod.monom-mult
definition mult-scalar-pprod :: (('a::nat, 'b::nat) \(\left.p p \Rightarrow \Rightarrow_{0}{ }^{\prime} c:: s e m i r i n g-0\right) \Rightarrow\) -
    where mult-scalar-pprod \(=\) pprod.mult-scalar
definition adds-term-pprod :: (('a::nat, 'b::nat) pp \(\times-) \Rightarrow-\)
    where adds-term-pprod \(=\) pprod.adds-term
global-interpretation pprod': gd-nat-term \(\lambda x::(' a, ~ ' b) ~ p p \times ' c . x \lambda x\). x cmp-term
    rewrites pprod.pp-of-term \(=f s t\)
```

and pprod.component-of-term $=$ snd
and pprod.splus $=$ splus-pprod
and pprod.monom-mult $=$ monom-mult-pprod
and pprod.mult-scalar $=$ mult-scalar-pprod
and pprod.adds-term $=$ adds-term-pprod
for cmp-term :: (('a::nat, 'b::nat) pp $\times$ ' $c::\{n a t$, the-min $\})$ nat-term-order
defines shift-map-keys-pprod $=$ pprod'.shift-map-keys
and min-term-pprod $=$ pprod'.min-term
and $l t-$ pprod $=$ pprod ${ }^{\prime} . l t$
and $l c$-pprod $=$ pprod $^{\prime} . l c$
and tail-pprod $=$ pprod'.tail
and comp-opt-p-pprod $=$ pprod $^{\prime}$. comp-opt-p
and ord-p-pprod $=$ pprod $^{\prime}$.ord-p
and ord-strict- $p$-pprod $=$ pprod $^{\prime}$. ord-strict- $p$
and find-adds-pprod $=$ pprod $^{\prime}$.find-adds
and trd-aux-pprod $=$ pprod'.trd-aux
and trd-pprod $=$ pprod $^{\prime} . t r d$
and spoly-pprod $=$ pprod $^{\prime}$.spoly
and count-const-lt-components-pprod $=$ pprod' .count-const-lt-components
and count-rem-components-pprod $=$ pprod'.count-rem-components
and const-lt-component-pprod $=$ pprod'.const-lt-component
and full-gb-pprod $=$ pprod $^{\prime} . f u l l-g b$
and keys-to-list-pprod $=$ pprod'.keys-to-list
and Keys-to-list-pprod $=$ pprod $^{\prime}$. Keys-to-list
and add-pairs-single-sorted-pprod $=$ pprod'.add-pairs-single-sorted
and add-pairs-pprod $=$ pprod $^{\prime}$. add-pairs
and canon-pair-order-aux-pprod $=$ pprod'.canon-pair-order-aux
and canon-basis-order-pprod $=$ pprod'.canon-basis-order
and new-pairs-sorted-pprod $=$ pprod'.new-pairs-sorted
and component-crit-pprod $=$ pprod'. .component-crit
and chain-ncrit-pprod $=$ pprod'. .chain-ncrit
and chain-ocrit-pprod $=$ pprod $^{\prime}$.chain-ocrit
and apply-icrit-pprod $=$ pprod $^{\prime}$. apply-icrit
and apply-ncrit-pprod $=$ pprod $^{\prime}$. apply-ncrit
and apply-ocrit-pprod $=$ pprod $^{\prime}$. apply-ocrit
and trdsp-pprod $=$ pprod $^{\prime} . t r d s p$
and gb-sel-pprod $=$ pprod' $^{\prime} . g b-s e l$
and $g b$-red-aux-pprod $=$ pprod $^{\prime} \cdot g b$-red-aux
and $g b$-red-pprod $=$ pprod'.gb-red
and $g b$-aux-pprod $=$ pprod $^{\prime} . g b-a u x$
and $g b-$ pprod $=$ pprod $^{\prime} . g b$
subgoal by (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: monom-mult-pprod-def)
subgoal by (simp only: mult-scalar-pprod-def)
subgoal by (simp only: adds-term-pprod-def)
done

```
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-shift-map-keys-pprod [code abstract]:
    list-of-oalist-ntm (shift-map-keys-pprod \(t f x s)=\) map-raw \((\lambda(k, v)\). (splus-pprod
\(t k, f v)\) ) (list-of-oalist-ntm xs)
    by (simp add: pprod'.list-of-oalist-shift-keys case-prod-beta')
lemma compute-trd-pprod [code]: trd-pprod to fs \(p=\) trd-aux-pprod to fs \(p\) (change-ord
to 0)
    by (simp only: pprod'.trd-def change-ord-def)
lemmas \([\) code \(]=\) conversep-iff
definition \(V_{e c}::\) nat \(\Rightarrow\left(\left({ }^{\prime} a\right.\right.\), nat \(\left.) p p \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left(\left({ }^{\prime} a:: n a t\right.\right.\), nat \(\left.) p p \times n a t\right) \Rightarrow_{0}\)
'b::semiring-1 where
    Vec \({ }_{0}\) i \(p=\) mult-scalar-pprod \(p\) (Poly-Mapping.single ( \(0, i\) ) 1)
experiment begin interpretation trivariate \({ }_{0}\) rat .
lemma
    ord-p-pprod (POT DRLEX) \(\left(\operatorname{Vec}_{0} 1\left(X^{2} * Z\right)+\operatorname{Vec}_{0} 0\left(2 * Y^{\wedge} 3 * Z^{2}\right)\right)\left(\operatorname{Vec}_{0}\right.\)
\(\left.1\left(X^{2} * Z^{2}+2 * Y^{\wedge} 3 * Z^{2}\right)\right)\)
    by eval
```


## lemma

```
    tail-pprod \((\) POT DRLEX \()\left(\operatorname{Vec}_{0} 1\left(X^{2} * Z\right)+\operatorname{Vec}_{0} 0\left(2 * Y^{\wedge} 3 * Z^{2}\right)\right)=\operatorname{Vec}_{0}\)
\(0\left(2 * Y^{\wedge} 3 * Z^{2}\right)\)
    by eval
```


## lemma

```
lt-pprod (POT DRLEX) \(\left(\operatorname{Vec}_{0} 1\left(X^{2} * Z\right)+\operatorname{Vec}_{0} 0\left(2 * Y^{\wedge} 3 * Z^{2}\right)\right)=\left(\right.\) sparse \(_{0}\) \([(0,2),(2,1)], 1)\)
by eval
```


## lemma

```
keys \(\left(\operatorname{Vec}_{0} 0\left(X^{2} * Z^{\wedge} 3\right)+\operatorname{Vec}_{0} 1\left(2 * Y^{\wedge} 3 * Z^{2}\right)\right)=\) \(\left\{\left(\operatorname{sparse}_{0}[(0,2),(2,3)], 0\right),\left(\right.\right.\) sparse \(\left.\left._{0}[(1,3),(2,2)], 1\right)\right\}\)
by eval
```


## lemma

```
keys \(\left(\operatorname{Vec}_{0} 0\left(X^{2} * Z^{\wedge} 3\right)+\operatorname{Vec}_{0} 2\left(2 * Y^{\wedge} 3 * Z^{2}\right)\right)=\) \(\left\{\left(\operatorname{sparse}_{0}[(0,2),(2,3)], 0\right),\left(\right.\right.\) sparse \(\left.\left._{0}[(1,3),(2,2)], 2\right)\right\}\)
by eval
```


## lemma

$\operatorname{Vec}_{0} 1\left(X^{2} * Z^{\wedge} 7+2 * Y^{\wedge} 3 * Z^{2}\right)+\operatorname{Vec}_{0} 3\left(X^{2} * Z^{\wedge} 4\right)+\operatorname{Vec}_{0} 1(-2$

* $\left.Y^{\wedge} 3 * Z^{2}\right)=$
$\operatorname{Vec}_{0} 1\left(X^{2} * Z^{\wedge} 7\right)+\operatorname{Vec}_{0} 3\left(X^{2} * Z^{\wedge} 4\right)$
by eval


## lemma

lookup $\left(\operatorname{Vec}_{0} 0\left(X^{2} * Z^{\wedge} 7\right)+\operatorname{Vec}_{0} 1\left(2 * Y^{\wedge} 3 * Z^{2}+2\right)\right)\left(\right.$ sparse $_{0}[(0,2)$, $(2,7)], 0)=1$
by eval

## lemma

lookup $\left(\operatorname{Vec}_{0} 0\left(X^{2} * Z^{\wedge} 7\right)+\operatorname{Vec}_{0} 1\left(2 * Y^{\wedge} 3 * Z^{2}+2\right)\right)\left(\right.$ sparse $_{0}[(0,2)$, $(2,7)], 1)=0$
by eval

## lemma

$\operatorname{Vec}_{0} 0\left(0 * X^{\wedge} 2 * Z^{\wedge} 7\right)+\operatorname{Vec}_{0} 1\left(0 * Y^{\wedge} 3 * Z^{2}\right)=0$ by eval

## lemma

monom-mult-pprod 3 ( sparse $_{0}[(1,2::$ nat $\left.)]\right)\left(\operatorname{Vec}_{0} 0\left(X^{2} * Z\right)+\operatorname{Vec}_{0} 1(2 * Y\right.$ ^ $\left.3 * Z^{2}\right)$ ) $=$
$\operatorname{Vec}_{0} 0\left(3 * Y^{2} * Z * X^{2}\right)+\operatorname{Vec}_{0} 1\left(6 * Y^{\wedge} 5 * Z^{2}\right)$
by eval

## lemma

trd-pprod DRLEX $\left[\operatorname{Vec}_{0} 0\left(Y^{2} * Z+2 * Y * Z^{\wedge} 3\right)\right]\left(\operatorname{Vec}_{0} 0\left(X^{2} * Z^{\wedge} 4-\right.\right.$ $\left.\left.2 * Y^{\wedge} 3 * Z^{\wedge} 3\right)\right)=$
$\operatorname{Vec}_{0} 0\left(X^{2} * Z^{\wedge} 4+Y^{\wedge} 4 * Z\right)$
by eval

## lemma

```
length (gb-pprod (POT DRLEX)
\(\left(\operatorname{Vec}_{0} 0\left(X^{2} * Z^{\wedge} 4-2 * Y へ 3 * Z^{2}\right),()\right)\),
    ( \(\left.\operatorname{Vec}_{0} 0\left(Y^{2} * Z+2 * Z^{\wedge} 3\right),()\right)\)
    ] ()\()=4\)
```

    by eval
    end
end

# 11 Further Properties of Multivariate Polynomials 

theory More-MPoly-Type-Class<br>imports Polynomials.MPoly-Type-Class-Ordered General<br>begin

Some further general properties of (ordered) multivariate polynomials needed for Gröbner bases. This theory is an extension of Polynomials.MPoly-Type-Class-Ordered.

### 11.1 Modules and Linear Hulls

## context module

begin
lemma span-listE:
assumes $p \in \operatorname{span}($ set bs)
obtains $q s$ where length $q s=$ length $b s$ and $p=\operatorname{sum-list~(map2~(*s)~qs~bs)~}$
proof -
have finite (set bs)...
from this assms obtain $q$ where $p: p=\left(\sum b \in\right.$ set $b s .(q b) * s$ ) by (rule
span-finiteE)
let ?qs = map-dup $q(\lambda-.0) b s$
show ?thesis
proof show length ? $q s=$ length bs by simp
next let $? z s=z i p(m a p q($ remdups bs $))($ remdups bs $)$ have $*$ : distinct ?zs by (rule distinct-zipI2, rule distinct-remdups) have inj: inj-on $(\lambda b .(q b, b))($ set $b s)$ by (rule, simp)
have $p=\left(\sum(q, b) \leftarrow\right.$ ? zs. $\left.q * s b\right)$
by (simp add: sum-list-distinct-conv-sum-set[OF *] set-zip-map1 p comm-monoid-add-class.sum.reindex [O1 inj])
also have $\ldots=\left(\sum(q, b) \leftarrow(\right.$ filter $(\lambda(q, b) . q \neq 0)$ ?zs $\left.) . q * s b\right)$
by (rule monoid-add-class.sum-list-map-filter[symmetric], auto)
also have $\ldots=\left(\sum(q, b) \leftarrow(\right.$ filter $(\lambda(q, b) . q \neq 0)(z i p$ ? $q s$ bs $\left.)) . q * s b\right)$
by (simp only: filter-zip-map-dup-const)
also have $\ldots=\left(\sum(q, b) \leftarrow z i p\right.$ ? $q s$ bs. $\left.q * s b\right)$
by (rule monoid-add-class.sum-list-map-filter, auto)
finally show $p=\left(\sum(q, b) \leftarrow z i p\right.$ ? $q s$ bs. $\left.q * s b\right)$.
qed
qed
lemma span-listI: sum-list (map2 $(* s)$ qs bs) $\in \operatorname{span}($ set bs)
proof (induct qs arbitrary: bs)
case Nil
show ?case by (simp add: span-zero)
next
case step: (Cons q qs)

```
    show ?case
    proof (simp add: zip-Cons1 span-zero split: list.split, intro allI impI)
    fix a as
    have sum-list (map2 (*s) qs as) \in span (insert a (set as)) (is ?x \in?A)
        by (rule, fact step, rule span-mono, auto)
    moreover have a\in?A by (rule span-base) simp
    ultimately show q*sa+?x\in?A by (intro span-add span-scale)
    qed
qed
end
lemma (in term-powerprod) monomial-1-in-pmdlI:
    assumes (f::- =\mp@subsup{#}{0}{\prime}}\mp@subsup{}{\prime}{\prime}::field) \inpmdl F and keys f={t
    shows monomial 1t pmdl F
proof -
    define c where c \equivlookup ft
    from assms(2) have f-eq: f = monomial c t unfolding c-def
    by (metis (mono-tags, lifting) Diff-insert-absorb cancel-comm-monoid-add-class.add-cancel-right-right
        plus-except insert-absorb insert-not-empty keys-eq-empty keys-except)
    from assms(2) have c\not=0
    unfolding c-def by auto
    hence monomial 1t=monom-mult (1 / c) 0f by (simp add: f-eq monom-mult-monomial
term-simps)
    also from assms(1) have ... \in pmdl F by (rule pmdl-closed-monom-mult)
    finally show ?thesis.
qed
```


### 11.2 Ordered Polynomials

## context ordered-term

begin

### 11.2.1 Sets of Leading Terms and -Coefficients

definition lt-set :: ('t, 'b::zero) poly-mapping set $\Rightarrow$ 't set where
$l t-s e t F=l t \times(F-\{0\})$
definition lc-set $::(' t$, ' $b::$ zero $)$ poly-mapping set $\Rightarrow$ ' $b$ set where $l c$-set $F=l c \prime(F-\{0\})$
lemma lt-setI:
assumes $f \in F$ and $f \neq 0$
shows $l t f \in l t$-set $F$
unfolding lt-set-def using assms by simp
lemma lt-setE:
assumes $t \in l t$-set $F$
obtains $f$ where $f \in F$ and $f \neq 0$ and $l t f=t$
using assms unfolding lt-set-def by auto

```
lemma lt-set-iff:
    shows t\inlt-set F}\longleftrightarrow(\existsf\inF.f\not=0\wedgeltf=t
    unfolding lt-set-def by auto
lemma lc-setI:
    assumes f\inF and f\not=0
    shows lc f\inlc-set F
    unfolding lc-set-def using assms by simp
lemma lc-setE:
    assumes c\inlc-set F
    obtains f}\mathrm{ where f}\inF\mathrm{ and }f\not=0\mathrm{ and lc f=c
    using assms unfolding lc-set-def by auto
lemma lc-set-iff:
    shows c\inlc-set F}\longleftrightarrow(\existsf\inF.f\not=0\wedgelcf=c
    unfolding lc-set-def by auto
lemma lc-set-nonzero:
    shows 0\not\inlc-set F
proof
    assume 0 \inlc-set F
    then obtain f}\mathrm{ where }f\inF\mathrm{ and }f\not=0\mathrm{ and lc f=0 by (rule lc-setE)
    from }\langlef\not=0\rangle\mathrm{ have lc f}\not=0\mathrm{ by (rule lc-not-0)
    from this «lc f=0` show False ..
qed
lemma lt-sum-distinct-eq-Max:
    assumes finite I and sum p I\not=0
        and \bigwedgei1 i2. i1 }\inI\Longrightarrowi2\inI\Longrightarrowpi1\not=0\Longrightarrowpi2\not=0\Longrightarrowlt (pi1)=l
(p i2) \Longrightarrow i1 = i2
    shows lt (sum p I) = ord-term-lin.Max (lt-set (p'I))
proof -
    have }\neg\mp@subsup{p}{}{\prime}I\subseteq{0
    proof
        assume p'I\subseteq{0}
        hence sum pI=0 by (rule sum-poly-mapping-eq-zeroI)
        with assms(2) show False ..
    qed
    from assms(1) this assms(3) show ?thesis
    proof (induct I)
        case empty
        from empty(1) show ?case by simp
    next
        case (insert x I)
        show ?case
        proof (cases p'I\subseteq{0})
            case True
```

hence $p$ ' $I-\{0\}=\{ \}$ by $\operatorname{simp}$
have $p x \neq 0$
proof
assume $p x=0$
with True have $p^{\prime}$ insert $x I \subseteq\{0\}$ by simp
with insert(4) show False ..
qed
hence $\operatorname{insert}(p x)(p \prime I)-\{0\}=\operatorname{insert}(p x)\left(p^{\prime} I-\{0\}\right)$ by auto
hence lt-set $\left(p^{\prime}\right.$ insert $\left.x I\right)=\{l t(p x)\}$ by $(\operatorname{simp}$ add: lt-set-def<p'I-
$\{0\}=\{ \}>)$
hence eq1: ord-term-lin.Max (lt-set $\left(p^{\prime}\right.$ insert $\left.\left.x I\right)\right)=l t(p x)$ by $\operatorname{simp}$
have eq2: sum $p I=0$
proof (rule ccontr)
assume sum $p I \neq 0$
then obtain $y$ where $y \in I$ and $p y \neq 0$ by (rule sum.not-neutral-contains-not-neutral)
with True show False by auto
qed
show ?thesis by (simp only: eq1 sum.insert[OF insert(1) insert(2)], simp
add: eq2)
next
case False
hence $I H: l t(\operatorname{sum} p I)=$ ord-term-lin.Max $\left(l t-s e t\left(p^{\prime} I\right)\right)$
proof (rule insert(3))
fix i1 i2
assume $i 1 \in I$ and $i 2 \in I$
hence $i 1 \in$ insert $x I$ and $i 2 \in$ insert $x I$ by simp-all
moreover assume $p i 1 \neq 0$ and $p i 2 \neq 0$ and $l t(p i 1)=l t(p i 2)$
ultimately show $i 1=i 2$ by (rule insert(5))
qed
show ?thesis
proof (cases p $x=0$ )
case True
hence eq: lt-set ( $p^{\prime}$ insert $\left.x I\right)=l t$-set $\left(p^{\prime} I\right)$ by (simp add: lt-set-def)
show ?thesis by (simp only: eq, simp add: sum.insert[OF insert(1) insert(2)]
True, fact IH)
next
case False
hence eq1: lt-set ( $p$ 'insert $x I)=\operatorname{insert}(l t(p x))(l t-s e t(p ' I))$
by (auto simp add: lt-set-def)
from insert(1) have finite (lt-set ( $\left.p^{\prime} I\right)$ ) by (simp add: lt-set-def)
moreover from $\left\langle\neg p^{\prime} I \subseteq\{0\}\right.$ 〉 have lt-set $\left(p^{\prime} I\right) \neq\{ \}$ by (simp add: lt-set-def)
ultimately have eq2: ord-term-lin.Max (insert $(l t(p x))(l t-s e t(p \prime I)))=$
ord-term-lin.max $(l t \quad(p x))$ (ord-term-lin.Max (lt-set ( $p$ ' $I)$ ))
by (rule ord-term-lin.Max-insert)
show ?thesis
proof (simp only: eq1, simp add: sum.insert[OF insert(1) insert(2)] eq2 IH [symmetric],

```
                rule lt-plus-distinct-eq-max, rule)
```

```
            assume *:lt (px)=lt (sum p I)
                            have lt (px)\inlt-set ( p'I) by (simp only:*IH, rule ord-term-lin.Max-in,
fact+)
            then obtain f}\mathrm{ where f}\in\mp@subsup{p}{}{\prime}I\mathrm{ and }f\not=0\mathrm{ and ltf:lt f=lt (px) by
(rule lt-setE)
            from this(1) obtain y where }y\inI\mathrm{ and }f=py.
            from this(2)}\langlef\not=0\rangleltf have py\not=0 and lt-eq:lt (py)=lt (px) by
simp-all
            from - - this(1)<px\not=0> this(2) have y=x
            proof (rule insert(5))
                    from }\langley\inI\rangle\mathrm{ show }y\in\mathrm{ insert x I by simp
                    next
                    show }x\in\mathrm{ insert x I by simp
                    qed
                    with }\langley\inI\rangle\mathrm{ have }x\inI\mathrm{ by simp
            with }\langlex\not\inI\rangle\mathrm{ show False ..
            qed
            qed
        qed
    qed
qed
lemma lt-sum-distinct-in-lt-set:
    assumes finite I and sum p I\not=0
        and \bigwedgei1 i2. i1 }\inI\Longrightarrowi2\inI\Longrightarrowpi1\not=0\Longrightarrowpi2\not=0\Longrightarrowlt (pi1)=l
(pi2) \Longrightarrowi1 = i2
    shows lt (sum p I) \inlt-set (p'I)
proof -
    have }\neg\mp@subsup{p}{}{\prime}I\subseteq{0
    proof
        assume p'}I\subseteq{0
        hence sum pI=0 by (rule sum-poly-mapping-eq-zeroI)
        with assms(2) show False ..
    qed
    have lt (sum p I) = ord-term-lin.Max (lt-set (p'I))
    by (rule lt-sum-distinct-eq-Max, fact+)
    also have ... \inlt-set ( }p\mathrm{ ' I)
    proof (rule ord-term-lin.Max-in)
    from assms(1) show finite (lt-set ( }\mp@subsup{p}{}{\prime}I))\mathrm{ by (simp add:lt-set-def)
    next
            from «\neg p'I\subseteq{0}`\mathrm{ show lt-set ( p'I)}\not={} by (simp add:lt-set-def)
    qed
    finally show ?thesis .
qed
```


### 11.2.2 Monicity

```
definition monic :: \(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) field \()\) where
monic \(p=\) monom-mult \((1 / l c p) 0 p\)
```

```
definition is-monic-set :: ('t }\mp@subsup{=>}{0}{\prime}\mp@subsup{}{}{\prime}b::{ield) set => bool where
    is-monic-set }B\equiv(\forallb\inB.b\not=0\longrightarrowlcb=1
lemma lookup-monic:lookup (monic p) v=(lookup p v)/ lc p
proof -
    have lookup (monic p) (0\oplusv)=(1/lc p)*(lookup p v) unfolding monic-def
        by (rule lookup-monom-mult-plus)
    thus ?thesis by (simp add: term-simps)
qed
lemma lookup-monic-lt:
    assumes p\not=0
    shows lookup (monic p) (lt p)=1
    unfolding monic-def
proof -
    from assms have lc p\not=0 by (rule lc-not-0)
    hence 1 / lc p\not=0 by simp
    let ?q = monom-mult (1/lc p) 0p
    have lookup ?q (0\opluslt p)=(1 / lc p)*(lookup p (lt p)) by (rule lookup-monom-mult-plus)
    also have ... = (1 / lc p)* lc p unfolding lc-def ..
    also have ... = 1 using <lc p\not=0> by simp
    finally have lookup ? q (0 \opluslt p)=1.
    thus lookup ? q (lt p)=1 by (simp add: term-simps)
qed
lemma monic-0 [simp]: monic 0 = 0
    unfolding monic-def by (rule monom-mult-zero-right)
lemma monic-0-iff:(monic p=0)\longleftrightarrow(p=0)
proof
    assume monic p=0
    show p=0
    proof (rule ccontr)
        assume p\not=0
        hence lookup (monic p) (lt p)=1 by (rule lookup-monic-lt)
        with <monic p=0` have lookup 0 (lt p) = (1::'b) by simp
        thus False by simp
    qed
next
    assume p0: p=0
    show monic p = 0 unfolding p0 by (fact monic-0)
qed
lemma keys-monic [simp]: keys (monic p) = keys p
proof (cases p=0)
    case True
    show ?thesis unfolding True monic-0 ..
next
```

```
    case False
    hence lc p\not=0 by (rule lc-not-0)
    show ?thesis by (rule set-eqI, simp add: in-keys-iff lookup-monic «lc p\not=0>)
qed
lemma lt-monic [simp]:lt (monic p)=lt p
proof (cases p=0)
    case True
    show ?thesis unfolding True monic-0 ..
next
    case False
    have lt (monom-mult (1 / lc p) 0 p)=0 \oplus lt p
    proof (rule lt-monom-mult)
        from False have lc p\not=0 by (rule lc-not-0)
    thus 1 / lc p\not=0 by simp
    qed fact
    thus ?thesis by (simp add: monic-def term-simps)
qed
lemma lc-monic:
    assumes p\not=0
    shows lc (monic p) = 1
    using assms by (simp add:lc-def lookup-monic-lt)
lemma mult-lc-monic:
    assumes p\not=0
    shows monom-mult (lc p) 0 (monic p)=p(is ?q = p)
proof (rule poly-mapping-eqI)
    fix }
    from assms have lc p\not=0 by (rule lc-not-0)
    have lookup ?q (0 \oplusv)=(lc p)*(lookup (monic p) v) by (rule lookup-monom-mult-plus)
    also have ... = (lc p)*((lookup p v) / lc p) by (simp add: lookup-monic)
    also have ... = lookup p v using <lc p F 0 > by simp
    finally show lookup ?q v = lookup p v by (simp add: term-simps)
qed
lemma is-monic-setI:
    assumes \b.b\inB\Longrightarrowb\not=0\Longrightarrowlcb=1
    shows is-monic-set B
    unfolding is-monic-set-def using assms by auto
lemma is-monic-setD
    assumes is-monic-set B and b\inB and b\not=0
    shows lc b = 1
    using assms unfolding is-monic-set-def by auto
lemma Keys-image-monic [simp]:Keys (monic'A) = Keys A
    by (simp add: Keys-def)
```

```
lemma image-monic-is-monic-set: is-monic-set (monic ' A)
proof (rule is-monic-setI)
    fix p
    assume pin: p}\in\mathrm{ monic ' }A\mathrm{ and p}\not=
    from pin obtain }\mp@subsup{p}{}{\prime}\mathrm{ where p-def:p = monic p' and p' }\inA.
    from }\langlep\not=0\rangle\mathrm{ have }\mp@subsup{p}{}{\prime}\not=0\mathrm{ unfolding p-def monic-0-iff .
    thus lc p=1 unfolding p-def by (rule lc-monic)
qed
lemma pmdl-image-monic [simp]:pmdl (monic'B) = pmdl B
proof
    show pmdl (monic ' B)\subseteqpmdl B
    proof
        fix p
        assume p f pmdl (monic ` B)
        thus p f pmdl B
        proof (induct p rule: pmdl-induct)
            case base: module-0
            show ?case by (fact pmdl.span-zero)
    next
        case ind: (module-plus a b c t)
            from ind(3) obtain }\mp@subsup{b}{}{\prime}\mathrm{ where b-def:b= monic b}\mp@subsup{b}{}{\prime}\mathrm{ and }\mp@subsup{b}{}{\prime}\inB.
            have eq: b = monom-mult (1 / lc b}\mp@subsup{b}{}{\prime})0\mp@subsup{b}{}{\prime}\mathrm{ by (simp only: b-def monic-def)
        show ?case unfolding eq monom-mult-assoc
            by (rule pmdl.span-add, fact, rule monom-mult-in-pmdl, fact)
        qed
    qed
next
    show pmdl B\subseteqpmdl (monic ' B)
    proof
        fix p
        assume p f pmdl B
        thus p f pmdl (monic ' B)
        proof (induct p rule: pmdl-induct)
            case base: module-0
            show ?case by (fact pmdl.span-zero)
    next
                case ind: (module-plus a b c t)
                show ?case
        proof (cases b=0)
            case True
                from ind(2) show ?thesis by (simp add: True)
            next
                case False
                    let ?b = monic b
                from ind(3) have ?b }\in\mathrm{ monic ' }B\mathrm{ by (rule imageI)
                have a + monom-mult c t (monom-mult (lc b) 0 ?b) \in pmdl (monic 'B)
                unfolding monom-mult-assoc
                by (rule pmdl.span-add, fact, rule monom-mult-in-pmdl, fact)
```

```
                thus ?thesis unfolding mult-lc-monic[OF False].
            qed
        qed
    qed
qed
end
end
```


## 12 Auto-reducing Lists of Polynomials

theory Auto-Reduction<br>imports Reduction More-MPoly-Type-Class<br>begin

### 12.1 Reduction and Monic Sets

context ordered-term
begin
lemma is-red-monic: is-red $B($ monic $p) \longleftrightarrow$ is-red $B p$ unfolding is-red-adds-iff keys-monic ..
lemma red-image-monic $[$ simp $]$ : red (monic ' $B$ ) $=$ red $B$
proof (rule, rule)
fix $p q$
show red (monic 'B) $p q \longleftrightarrow$ red $B p q$
proof
assume red (monic ' $B$ ) $p q$
then obtain $f t$ where $f \in$ monic ' $B$ and $*$ : red-single $p q f t$ by (rule
red-setE)
from this(1) obtain $g$ where $g \in B$ and $f=$ monic $g$..
from $*$ have $f \neq 0$ by (simp add: red-single-def)
hence $g \neq 0$ by (simp add: monic- 0 -iff $\langle f=$ monic $g\rangle$ )
hence $l c g \neq 0$ by (rule lc-not-0)
have eq: monom-mult (lc g) $0 f=g$ by (simp add: $\langle f=$ monic $g\rangle$ mult-lc-monic $[O F$
$\langle g \neq 0\rangle])$
from $\langle g \in B\rangle$ show red $B p q$
proof (rule red-setI)
from * 〈lc $g \neq 0\rangle$ have red-single $p q$ (monom-mult (lc g) $0 f$ ) $t$ by (rule red-single-mult-const)
thus red-single p qg t by (simp only: eq)
qed

## next

assume red $B p q$
then obtain $f t$ where $f \in B$ and $*$ : red-single $p q f t$ by (rule red-setE)
from $*$ have $f \neq 0$ by (simp add: red-single-def)
hence $l c f \neq 0$ by (rule lc-not-0)

```
    hence 1 / lc f\not=0 by simp
    from}\langlef\inB\rangle\mathrm{ have monic }f\in\mathrm{ monic ' }B\mathrm{ by (rule imageI)
    thus red (monic ' B) pq
    proof (rule red-setI)
        from * <1 /lc f\not=0\rangle show red-single p q(monic f)t unfolding monic-def
            by (rule red-single-mult-const)
    qed
    qed
qed
lemma is-red-image-monic [simp]: is-red (monic ' B) p\longleftrightarrow is-red B p
    by (simp add: is-red-def)
```


### 12.2 Minimal Bases and Auto-reduced Bases

definition is-auto-reduced :: ( $' ~ t \Rightarrow_{0}{ }^{\prime} b::$ field $)$ set $\Rightarrow$ bool where
is-auto-reduced $B \equiv(\forall b \in B . \neg$ is-red $(B-\{b\}) b)$
definition is-minimal-basis :: ( ${ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::$ zero $)$ set $\Rightarrow$ bool where
is-minimal-basis $B \longleftrightarrow(0 \notin B \wedge(\forall p q . p \in B \longrightarrow q \in B \longrightarrow p \neq q \longrightarrow \neg l t$
paddst $\left.{ }^{l t} q\right)$ )
lemma is-auto-reducedD:
assumes is-auto-reduced $B$ and $b \in B$
shows $\neg$ is-red $(B-\{b\}) b$
using assms unfolding is-auto-reduced-def by auto
The converse of the following lemma is only true if $B$ is minimal!
lemma image-monic-is-auto-reduced:
assumes is-auto-reduced $B$
shows is-auto-reduced (monic ' $B$ )
unfolding is-auto-reduced-def
proof
fix $b$
assume $b \in$ monic ' $B$
then obtain $b^{\prime}$ where $b$-def: $b=$ monic $b^{\prime}$ and $b^{\prime} \in B .$.
from assms $\left\langle b^{\prime} \in B\right\rangle$ have nred: $\neg i s$-red $\left(B-\left\{b^{\prime}\right\}\right) b^{\prime}$ by (rule is-auto-reducedD)
show $\neg$ is-red $(($ monic $' B)-\{b\}) b$
proof
assume red: is-red $(($ monic ' $B)-\{b\}) b$
have (monic ' $B)-\{b\} \subseteq$ monic ' $\left(B-\left\{b^{\prime}\right\}\right)$ unfolding $b$-def by auto with red have is-red (monic ' $\left.\left(B-\left\{b^{\prime}\right\}\right)\right)$ b by (rule is-red-subset)
hence is-red $\left(B-\left\{b^{\prime}\right\}\right) b^{\prime}$ unfolding $b$-def is-red-monic is-red-image-monic. with nred show False ..
qed
qed
lemma is-minimal-basisI:
assumes $\bigwedge p . p \in B \Longrightarrow p \neq 0$ and $\bigwedge p q . p \in B \Longrightarrow q \in B \Longrightarrow p \neq q \Longrightarrow \neg$

```
lt p addst lt q
    shows is-minimal-basis B
    unfolding is-minimal-basis-def using assms by auto
lemma is-minimal-basisD1:
    assumes is-minimal-basis B and p\inB
    shows p}\not=
    using assms unfolding is-minimal-basis-def by auto
lemma is-minimal-basisD2:
    assumes is-minimal-basis B and p\inB and q\inB and p\not=q
    shows \neglt p addst lt q
    using assms unfolding is-minimal-basis-def by auto
lemma is-minimal-basisD3:
    assumes is-minimal-basis B and p\inB and q\inB and p\not=q
    shows \neg lt q addst lt p
    using assms unfolding is-minimal-basis-def by auto
lemma is-minimal-basis-subset:
    assumes is-minimal-basis B and }A\subseteq
    shows is-minimal-basis A
proof (intro is-minimal-basisI)
    fix p
    assume p\inA
    with }\langleA\subseteqB\rangle\mathrm{ have }p\inB.
    with «is-minimal-basis B` show p}=0\mathrm{ by (rule is-minimal-basisD1)
next
    fix pq
    assume p\inA and q\inA and p\not=q
    from }\langlep\inA\rangle\mathrm{ and }\langleq\inA\rangle\mathrm{ have }p\inB\mathrm{ and q}B\mathrm{ using }\langleA\subseteqB\rangle\mathrm{ by auto
    from 〈is-minimal-basis B\rangle this }\langlep\not=q\rangle\mathrm{ show }\neglt p addst lt q by (rule
is-minimal-basisD2)
qed
lemma nadds-red:
    assumes nadds: \bigwedgeq. q\inB\Longrightarrow\neglt qaddst lt p and red: red B p r
    shows r\not=0\wedgelt r=lt p
proof -
    from red obtain qt where q}\inB\mathrm{ and rs: red-single p r q t by (rule red-setE)
    from rs have q}=0\mathrm{ and lookup p (t }\oplusltq)\not=
        and r-def:r=p - monom-mult (lookup p (t\opluslt q) / lc q) t q unfolding
red-single-def by simp-all
    have t\opluslt q \preceq_ lt p by (rule lt-max, fact)
    moreover have t\opluslt q\not=lt p
    proof
        assume t\opluslt q=lt p
        hence lt q addst lt p by (metis adds-term-triv)
        with nadds[OF}\langleq\inB`] show False ..
```

```
    qed
    ultimately have t\opluslt q\mp@subsup{\prec}{t}{}lt p by simp
    let ?m = monom-mult (lookup p (t\opluslt q) / lc q) tq
    from «lookup p (t\opluslt q)\not=0` lc-not-0[OF <q\not= 0`] have c0: lookup p (t\opluslt
q) / lc q\not=0 by simp
    from }\langleq\not=0\ranglec0\mathrm{ have ?m}\not=0\mathrm{ by (simp add: monom-mult-eq-zero-iff)
    have lt (-?m) = lt ?m by (fact lt-uminus)
    also have lt1:lt ?m =t\opluslt q by (rule lt-monom-mult, fact+)
    finally have lt2:lt (-?m)=t\opluslt q.
    show ?thesis
    proof
        show r}=
        proof
            assume r=0
            hence p=?m unfolding r-def by simp
            with lt1 <t \oplus lt q}\not=lt p> show False by sim
    qed
    next
        have lt (-?m + p)=lt p
        proof (rule lt-plus-eqI)
            show lt (-?m) < lt lt unfolding lt2 by fact
        qed
        thus lt r =lt p unfolding r-def by simp
    qed
qed
lemma nadds-red-nonzero:
    assumes nadds: \q. q\inB\Longrightarrow\neglt q addst lt p and red B pr
    shows r\not=0
    using nadds-red[OF assms] by simp
lemma nadds-red-lt:
    assumes nadds: }\q.q\inB\Longrightarrow\neglt q addst lt p and red B pr
    shows lt r = lt p
    using nadds-red[OF assms] by simp
lemma nadds-red-rtrancl-lt:
    assumes nadds: \q. q\inB\Longrightarrow\neglt q addst lt p and rtrancl: (red B)** pr
    shows lt r = lt p
    using rtrancl
proof (induct rule: rtranclp-induct)
    case base
    show ?case ..
next
    case (step y z)
    have lt z= lt y
    proof (rule nadds-red-lt)
    fix q
```

```
        assume q\inB
        thus \neglt q adds st lt y unfolding <lt y = lt p> by (rule nadds)
    qed fact
    with «lt y = lt p\rangle show ?case by simp
qed
lemma nadds-red-rtrancl-nonzero:
    assumes nadds: \bigwedgeq. q\inB\Longrightarrow\neglt qaddst lt p and p\not=0 and rtrancl:(red
B)** pr
    shows r\not=0
    using rtrancl
proof (induct rule: rtranclp-induct)
    case base
    show ?case by fact
next
    case (step y z)
    from nadds «(red B)** p y> have lt y = lt p by (rule nadds-red-rtrancl-lt)
    show }z\not=
    proof (rule nadds-red-nonzero)
        fix q
        assume q\inB
        thus \neglt q addst lt y unfolding <lt y = lt p> by (rule nadds)
    qed fact
qed
lemma minimal-basis-red-rtrancl-nonzero:
    assumes is-minimal-basis B and p\inB and (red (B-{p}))** pr
    shows r\not=0
proof (rule nadds-red-rtrancl-nonzero)
    fix q
    assume q\in(B-{p})
    hence q}\inB\mathrm{ and q}=p\mathrm{ by auto
    show \neglt q addst lt p by (rule is-minimal-basisD2, fact+)
next
    show }p\not=0\mathrm{ by (rule is-minimal-basisD1, fact+)
qed fact
lemma minimal-basis-red-rtrancl-lt:
    assumes is-minimal-basis B and p\inB and (red (B-{p}))** pr
    shows lt r = lt p
proof (rule nadds-red-rtrancl-lt)
    fix q
    assume q\in(B-{p})
    hence q\inB and q}\not=p\mathrm{ by auto
    show \neglt q addst lt p by (rule is-minimal-basisD2, fact+)
qed fact
lemma is-minimal-basis-replace:
assumes major: is-minimal-basis \(B\) and \(p \in B\) and red: \((\operatorname{red}(B-\{p\}))^{* *} p r\)
```

```
    shows is-minimal-basis (insert r (B-{p}))
proof (rule is-minimal-basisI)
    fix q
    assume q}\in\mathrm{ insert r (B-{p})
    hence q=r\veeq\inB\wedgeq\not=p by simp
    thus q\not=0
    proof
        assume q=r
    from assms show ?thesis unfolding }\langleq=r\rangle\mathrm{ by (rule minimal-basis-red-rtrancl-nonzero)
    next
        assume q}\inB\wedgeq\not=
        hence q}q\inB.
        with major show ?thesis by (rule is-minimal-basisD1)
    qed
next
    fix ab
    assume }a\in\mathrm{ insert r (B-{p}) and b insert r (B-{p}) and a}
    from assms have ltr:lt r=lt p by (rule minimal-basis-red-rtrancl-lt)
    from }\langleb\in\mathrm{ insert r (B-{p})> have b: b=r`b 隹 位^b}=p\mathrm{ by simp
    from <a\in insert r (B-{p})> have }a=r\veea\inB\wedgea\not=p\mathrm{ by simp
    thus \neglt a addst lt b
    proof
    assume a=r
    hence lta: lt a=lt p using ltr by simp
    from b show ?thesis
    proof
        assume b=r
        with }\langlea\not=b\rangle\mathrm{ show ?thesis unfolding <a=r〉 by simp
    next
        assume b\inB\wedgeb\not=p
        hence }b\inB\mathrm{ and }p\not=b\mathrm{ by auto
        with major }\langlep\inB\rangle\mathrm{ have }\neglt paddst lt b by (rule is-minimal-basisD2)
        thus ?thesis unfolding lta.
    qed
    next
    assume }a\inB\wedgea\not=
    hence }a\inB\mathrm{ and }a\not=p\mathrm{ by simp-all
    from b show ?thesis
    proof
        assume b=r
            from major }\langlea\inB\rangle\langlep\inB\rangle\langlea\not=p\rangle\mathrm{ have ᄀlt a addst lt p by (rule
is-minimal-basisD2)
            thus ?thesis unfolding <b=r\rangle ltr by simp
    next
        assume b\inB^b\not=p
        hence b\in B ..
        from major }\langlea\inB\rangle\langleb\inB\rangle\langlea\not=b\rangle\mathrm{ show ?thesis by (rule is-minimal-basisD2)
    qed
qed
```


## qed

### 12.3 Computing Minimal Bases

definition comp-min-basis :: ( $\left.' t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.$ zero $)$ list where comp-min-basis $x s=$ filter-min $(\lambda x y$. lt $x$ addst lt $y)($ filter $(\lambda x . x \neq 0) x s)$
lemma comp-min-basis-subset': set (comp-min-basis $x s) \subseteq\{x \in$ set $x s . x \neq 0\}$ proof -
have set $($ comp-min-basis xs $) \subseteq \operatorname{set}($ filter $(\lambda x . x \neq 0) x s)$
unfolding comp-min-basis-def by (rule filter-min-subset)
also have $\ldots=\{x \in$ set $x s . x \neq 0\}$ by simp
finally show? ?hesis.
qed
lemma comp-min-basis-subset: set (comp-min-basis $x s) \subseteq$ set xs proof -
have set (comp-min-basis $x s) \subseteq\{x \in$ set $x s . x \neq 0\}$ by (rule comp-min-basis-subset')
also have $\ldots \subseteq$ set $x s$ by simp
finally show? thesis.
qed
lemma comp-min-basis-nonzero: $p \in \operatorname{set}$ (comp-min-basis $x s) \Longrightarrow p \neq 0$
using comp-min-basis-subset' by blast
lemma comp-min-basis-adds:
assumes $p \in$ set $x s$ and $p \neq 0$
obtains $q$ where $q \in$ set (comp-min-basis $x s$ ) and $l t q$ adds $s_{t}$ lt $p$
proof -
let ? $r e l=(\lambda x y$. lt $x$ addst lt $y)$
have transp ?rel by (auto intro!: transpI dest: adds-term-trans)
moreover have reflp ?rel by (simp add: reflp-def adds-term-refl)
moreover from assms have $p \in \operatorname{set}($ filter $(\lambda x . x \neq 0)$ xs) by simp
ultimately obtain $q$ where $q \in \operatorname{set}$ (comp-min-basis xs) and lt $q$ addst lt $p$ unfolding comp-min-basis-def by (rule filter-min-relE)
thus ?thesis ..
qed
lemma comp-min-basis-is-red:
assumes is-red (set xs) f
shows is-red (set (comp-min-basis xs)) $f$
proof -
from assms obtain $x t$ where $x \in$ set $x s$ and $t \in \operatorname{keys} f$ and $x \neq 0$ and $l t x$ $a d d s_{t} t$
by (rule is-red-addsE)
from $\langle x \in$ set $x s\rangle\langle x \neq 0\rangle$ obtain $y$ where yin: $y \in \operatorname{set}$ (comp-min-basis $x s$ )
and $l t y$ addst ${ }_{t}$ lt $x$
by (rule comp-min-basis-adds)
show ?thesis

```
    proof (rule is-red-addsI)
    from <lt y addst lt x\rangle<lt x addst t> show lt y adds }\mp@subsup{s}{t}{}t\mathrm{ by (rule adds-term-trans)
    next
    from yin show y\not=0 by (rule comp-min-basis-nonzero)
    qed fact+
qed
lemma comp-min-basis-nadds:
    assumes p\in set (comp-min-basis xs) and q\in set (comp-min-basis xs) and p\not=
q
    shows ᄀlt q addst lt p
proof
    have transp ( }\lambdaxy.lt x addst lt y) by (auto intro!: transpI dest: adds-term-trans
    moreover note assms(2, 1)
    moreover assume lt q addst lt p
    ultimately have q=p unfolding comp-min-basis-def by (rule filter-min-minimal)
    with assms(3) show False by simp
qed
lemma comp-min-basis-is-minimal-basis: is-minimal-basis (set (comp-min-basis xs))
    by (rule is-minimal-basisI, rule comp-min-basis-nonzero, assumption, rule comp-min-basis-nadds,
            assumption+, simp)
lemma comp-min-basis-distinct: distinct (comp-min-basis xs)
    unfolding comp-min-basis-def by (rule filter-min-distinct) (simp add: reflp-def
adds-term-refl)
end
```


### 12.4 Auto-Reduction

context $g d$-term
begin
lemma is-minimal-basis-trd-is-minimal-basis:
assumes is-minimal-basis (set $(x \# x s))$ and $x \notin$ set $x s$
shows is-minimal-basis (set ((trd xs x) \# xs ))
proof -
from $\operatorname{assms}(1)$ have is-minimal-basis (insert (trd xs $x)($ set $(x \# x s)-\{x\}))$
proof (rule is-minimal-basis-replace, simp)
from $\operatorname{assms}(2)$ have eq: set $(x \# x s)-\{x\}=$ set $x s$ by simp
show (red $(\operatorname{set}(x \# x s)-\{x\}))^{* *} x($ trd $x s x)$ unfolding eq by (rule
trd-red-rtrancl)
qed
also from $\operatorname{assms}(2)$ have $\ldots=$ set $((\operatorname{trd} x s x) \# x s)$ by auto
finally show ?thesis.
qed
lemma is-minimal-basis-trd-distinct:

```
    assumes min: is-minimal-basis (set (x # xs)) and dist: distinct (x # xs)
    shows distinct ((trd xs x) # xs)
proof -
    let ?y = trd xs x
    from min have lty:lt ?y = lt x
    proof (rule minimal-basis-red-rtrancl-lt, simp)
        from dist have x\not\in set xs by simp
        hence eq: set (x#xs)-{x} = set xs by simp
            show (red (set (x # xs) - {x}))** x (trd xs x) unfolding eq by (rule
trd-red-rtrancl)
    qed
    have ?y & set xs
    proof
        assume ?y f set xs
        hence ?y \in set (x # xs) by simp
        with min have ᄀ lt ?y addst lt x
        proof (rule is-minimal-basisD2, simp)
            show ?}\boldsymbol{y}\not=
            proof
                    assume ? }y=
                            from dist have x\not\in set xs by simp
                    with 〈?y f set xs\rangle show False unfolding <?y = x〉 by simp
                    qed
        qed
        thus False unfolding lty by (simp add: adds-term-refl)
    qed
    moreover from dist have distinct xs by simp
    ultimately show ?thesis by simp
qed
```



```
list where
    comp-red-basis-aux-base: comp-red-basis-aux Nil ys = ys|
    comp-red-basis-aux-rec: comp-red-basis-aux (x # xs) ys = comp-red-basis-aux xs
((trd (xs@ @s) x) # ys)
lemma subset-comp-red-basis-aux: set ys \subseteq set (comp-red-basis-aux xs ys)
proof (induct xs arbitrary: ys)
    case Nil
    show ?case unfolding comp-red-basis-aux-base ..
next
    case (Cons a xs)
    have set ys \subseteq set ((trd (xs@ ys) a) # ys) by auto
    also have ... \subseteq set (comp-red-basis-aux xs ((trd (xs @ ys) a) # ys)) by (rule
Cons.hyps)
    finally show ?case unfolding comp-red-basis-aux-rec .
qed
lemma comp-red-basis-aux-nonzero:
```

```
    assumes is-minimal-basis (set (xs @ ys)) and distinct (xs @ ys) and p\in set
(comp-red-basis-aux xs ys)
    shows p\not=0
    using assms
proof (induct xs arbitrary: ys)
    case Nil
    show ?case
    proof (rule is-minimal-basisD1)
    from Nil(1) show is-minimal-basis (set ys) by simp
    next
        from Nil(3) show p\in set ys unfolding comp-red-basis-aux-base .
    qed
next
    case (Cons a xs)
    have eq:(a#xs)@ys=a # (xs @ ys) by simp
    have }a\in\operatorname{set ( a # xs @ ys) by simp
    from Cons(3) have a& set (xs @ ys) unfolding eq by simp
    let ?ys = trd (xs @ ys) a # ys
    show ?case
    proof (rule Cons.hyps)
        from Cons(3) have a\not\in set (xs @ ys) unfolding eq by simp
        with Cons(2) show is-minimal-basis (set (xs @ ?ys)) unfolding set-reorder
eq
            by (rule is-minimal-basis-trd-is-minimal-basis)
    next
        from Cons(2) Cons(3) show distinct (xs @ ?ys) unfolding distinct-reorder eq
            by (rule is-minimal-basis-trd-distinct)
    next
    from Cons(4) show p\in set (comp-red-basis-aux xs ?ys) unfolding comp-red-basis-aux-rec
    qed
qed
lemma comp-red-basis-aux-lt:
    assumes is-minimal-basis (set (xs @ ys)) and distinct (xs @ ys)
    shows lt'set (xs @ ys) =lt'set (comp-red-basis-aux xs ys)
    using assms
proof (induct xs arbitrary: ys)
    case Nil
    show ?case unfolding comp-red-basis-aux-base by simp
next
    case (Cons a xs)
    have eq: (a # xs)@ ys = a # (xs @ ys) by simp
    from Cons(3) have a: a & set (xs @ ys) unfolding eq by simp
    let ?b = trd (xs@ @s) a
    let ?ys = ?b # ys
    from Cons(2) have lt ?b = lt a unfolding eq
    proof (rule minimal-basis-red-rtrancl-lt, simp)
    from a have eq2: set (a# xs @ ys) - {a} = set (xs@ ys) by simp
```

```
    show (red (set (a # xs @ ys) - {a}))** a ?b unfolding eq2 by (rule
trd-red-rtrancl)
    qed
    hence lt 'set ((a# xs) @ ys) = lt ' set ((?b # xs)@ ys) by simp
    also have ... = lt'set (xs @ (?b # ys)) by simp
    finally have eq2:lt'set ((a# xs)@ys) = lt'set (xs @ (?b # ys)).
    show ?case unfolding comp-red-basis-aux-rec eq2
    proof (rule Cons.hyps)
        from Cons(3) have a\not\inset (xs@ys) unfolding eq by simp
        with Cons(2) show is-minimal-basis (set (xs @ ?ys)) unfolding set-reorder
eq
            by (rule is-minimal-basis-trd-is-minimal-basis)
    next
        from Cons(2) Cons(3) show distinct (xs @ ?ys) unfolding distinct-reorder eq
            by (rule is-minimal-basis-trd-distinct)
    qed
qed
lemma comp-red-basis-aux-pmdl:
    assumes is-minimal-basis (set (xs @ ys)) and distinct (xs @ ys)
    shows pmdl (set (comp-red-basis-aux xs ys)) \subseteqpmdl (set (xs @ ys))
    using assms
proof (induct xs arbitrary: ys)
    case Nil
    show ?case unfolding comp-red-basis-aux-base by simp
next
    case (Cons a xs)
    have eq: (a# xs)@ ys=a# (xs @ ys) by simp
    from Cons(3) have a: a & set (xs @ ys) unfolding eq by simp
    let ?b = trd (xs @ ys)a
    let ?ys = ?b # ys
    have pmdl (set (comp-red-basis-aux xs ?ys)) \subseteqpmdl (set (xs @ ?ys))
    proof (rule Cons.hyps)
        from Cons(3) have a\not\in set (xs @ ys) unfolding eq by simp
        with Cons(2) show is-minimal-basis (set (xs @ ?ys)) unfolding set-reorder
eq
            by (rule is-minimal-basis-trd-is-minimal-basis)
    next
    from Cons(2) Cons(3) show distinct (xs @ ?ys) unfolding distinct-reorder eq
            by (rule is-minimal-basis-trd-distinct)
    qed
    also have ... = pmdl (set (?b # xs @ ys)) by simp
    also from a have ... = pmdl (insert ?b (set (a # xs @ ys) - {a})) by auto
    also have ...\subseteqpmdl (set (a # xs @ ys))
    proof (rule pmdl.replace-span)
        have a-(trd (xs@ys)a) \inpmdl (set (xs @ ys)) by (rule trd-in-pmdl)
        have }a-(trd (xs@ys)a)\inpmdl (set (a#xs@ ys)
        proof
        show pmdl (set (xs @ ys))\subseteqpmdl (set (a#xs @ys)) by (rule pmdl.span-mono)
```

```
auto
    qed fact
        hence - (a-(trd (xs @ ys) a)) \inpmdl (set (a# xs @ ys)) by (rule
pmdl.span-neg)
    hence (trd (xs @ ys)a) - a \in pmdl (set (a # xs @ ys)) by simp
    hence ((trd (xs@ys)a)-a)+a\inpmdl (set (a # xs @ ys))
    proof (rule pmdl.span-add)
            show a pmdl (set (a#xs@ys))
            proof
                show a cset (a#xs@ @s) by simp
            qed (rule pmdl.span-superset)
    qed
    thus trd (xs @ ys) a \in pmdl (set (a# xs @ ys)) by simp
    qed
    also have ... = pmdl (set ((a# xs) @ ys)) by simp
    finally show ?case unfolding comp-red-basis-aux-rec .
qed
lemma comp-red-basis-aux-irred:
    assumes is-minimal-basis (set (xs @ ys)) and distinct (xs @ ys)
        and }\bigwedgey.y\in set ys \Longrightarrow\negis-red (set (xs@ @s) - {y}) y
        and p\in set (comp-red-basis-aux xs ys)
    shows }\neg\mathrm{ is-red (set (comp-red-basis-aux xs ys) - {p})p
    using assms
proof (induct xs arbitrary: ys)
    case Nil
    have ᄀ is-red (set ([] @ ys) - {p})p
    proof (rule Nil(3))
        from Nil(4) show p\in set ys unfolding comp-red-basis-aux-base .
    qed
    thus ?case unfolding comp-red-basis-aux-base by simp
next
    case (Cons a xs)
    have eq:(a#xs)@ys=a# (xs @ ys) by simp
    from Cons(3) have a-notin: a & set (xs @ ys) unfolding eq by simp
    from Cons(2) have is-min: is-minimal-basis (set (a# xs @ ys)) unfolding eq
    let ?b = trd (xs @ ys)a
    let ?ys = ?b # ys
    have dist:distinct (?b # (xs @ ys))
    proof (rule is-minimal-basis-trd-distinct, fact is-min)
    from Cons(3) show distinct (a # xs @ ys) unfolding eq.
    qed
    show ?case unfolding comp-red-basis-aux-rec
    proof (rule Cons.hyps)
        from Cons(2) a-notin show is-minimal-basis (set (xs @ ?ys)) unfolding
set-reorder eq
    by (rule is-minimal-basis-trd-is-minimal-basis)
```

```
    next
    from dist show distinct (xs @ ?ys) unfolding distinct-reorder.
    next
    fix }
    assume y set ?ys
    hence }y=?b\veey\in\mathrm{ set ys by simp
    thus ᄀis-red (set (xs @ ?ys) - {y})y
    proof
        assume y =?b
        from dist have ?b & set (xs @ ys) by simp
        hence eq3: set (xs @ ?ys) - {?b} = set (xs @ ys) unfolding set-reorder by
simp
        have }\negis-red (set (xs @ ys)) ?b by (rule trd-irred)
        thus ?thesis unfolding < y = ?b> eq3 .
    next
        assume y \in set ys
        hence irred: ᄀ is-red (set ((a# xs)@ ys) - {y}) y by (rule Cons(4))
        from }<y\in\mathrm{ set ys> a-notin have }y\not=a\mathrm{ by auto
    hence eq3: set ((a#xs)@ys)-{y}={a}\cup(set (xs@ ys) - {y}) by auto
    from irred have i1: ᄀis-red {a} y and i2: ᄀis-red (set (xs@ ys) - {y}) y
        unfolding eq3 is-red-union by simp-all
    show ?thesis unfolding set-reorder
    proof (cases y=?b)
            case True
        from i2 show \neg is-red (set (?b # xs @ ys) - {y}) y by (simp add:True)
    next
        case False
        hence eq4: set (?b # xs@ @ys) - {y} = {?b}\cup(set (xs@ @s) - {y}) by
auto
        show ᄀ is-red (set (?b # xs @ ys) - {y}) y unfolding eq4
        proof
            assume is-red ({?b} \cup (set (xs@ @s) - {y})) y
            thus False unfolding is-red-union
            proof
                have ltb:lt ?b = lt a
                proof (rule minimal-basis-red-rtrancl-lt, fact is-min)
                        show a\in set (a# xs @ ys) by simp
                next
                        from a-notin have eq: set (a# xs @ ys) - {a} = set (xs @ ys) by
simp
                    show (red (set (a#xs@ys) - {a}))** a ?b unfolding eq by (rule
trd-red-rtrancl)
                qed
                assume is-red {?b} y
        then obtain t where t\inkeys y and lt ?b addst t unfolding is-red-adds-iff
by auto
        with ltb have lt a addst t by simp
        have is-red {a} y
            by (rule is-red-addsI, rule, rule is-minimal-basisD1, fact is-min, simp,
```

```
fact+)
                with i1 show False ..
                next
                        assume is-red (set (xs@ys)-{y})y
                with i2 show False ..
                qed
                qed
            qed
        qed
    next
    from Cons(5) show p\in set (comp-red-basis-aux xs ?ys) unfolding comp-red-basis-aux-rec
    qed
qed
lemma comp-red-basis-aux-dgrad-p-set-le:
    assumes dickson-grading d
    shows dgrad-p-set-le d (set (comp-red-basis-aux xs ys)) (set xs U set ys)
proof (induct xs arbitrary: ys)
    case Nil
    show ?case by (simp, rule dgrad-p-set-le-subset, fact subset-refl)
next
    case (Cons x xs)
    let ?h = trd (xs @ ys) x
    have dgrad-p-set-le d (set (comp-red-basis-aux xs (?h # ys))) (set xs \cup set (?h
# ys))
    by (fact Cons)
    also have ... = insert ?h (set xs U set ys) by simp
    also have dgrad-p-set-le d ... (insert x (set xs \cup set ys))
    proof (rule dgrad-p-set-leI-insert)
    show dgrad-p-set-le d (set xs U set ys) (insert x (set xs \cup set ys))
        by (rule dgrad-p-set-le-subset, blast)
    next
    have (red (set (xs @ ys)))** x ?h by (rule trd-red-rtrancl)
    with assms have dgrad-p-set-le d {?h} (insert x (set (xs @ ys)))
            by (rule dgrad-p-set-le-red-rtrancl)
    thus dgrad-p-set-le d {?h} (insert x (set xs \cup set ys)) by simp
    qed
    finally show ?case by simp
qed
definition comp-red-basis :: ('t =00 'b) list => ('t }\mp@subsup{=>}{0}{\prime}\mp@subsup{}{}{\prime}b:::field) lis
    where comp-red-basis xs = comp-red-basis-aux (comp-min-basis xs) []
lemma comp-red-basis-nonzero:
    assumes p\in set (comp-red-basis xs)
    shows p}\not=
proof -
    have is-minimal-basis (set ((comp-min-basis xs) @ [])) by (simp add: comp-min-basis-is-minimal-basis)
```

```
moreover have distinct ((comp-min-basis xs) @ []) by (simp add: comp-min-basis-distinct)
    moreover from assms have p\in set (comp-red-basis-aux (comp-min-basis xs)
[]) unfolding comp-red-basis-def .
    ultimately show ?thesis by (rule comp-red-basis-aux-nonzero)
qed
lemma pmdl-comp-red-basis-subset: pmdl (set (comp-red-basis xs)) \subseteq pmdl (set
xs)
proof
    fix f
    assume fin: f\inpmdl (set (comp-red-basis xs))
    have f}\in\mathrm{ pmdl (set (comp-min-basis xs))
    proof
            from fin show f\inpmdl (set (comp-red-basis-aux (comp-min-basis xs) []))
            unfolding comp-red-basis-def .
    next
            have pmdl (set (comp-red-basis-aux (comp-min-basis xs) [])) \subseteq pmdl (set
((comp-min-basis xs)@ []))
                            by (rule comp-red-basis-aux-pmdl, simp-all, rule comp-min-basis-is-minimal-basis,
rule comp-min-basis-distinct)
            thus pmdl (set (comp-red-basis-aux (comp-min-basis xs)[])) \subseteq pmdl (set
(comp-min-basis xs))
            by simp
        qed
    also from comp-min-basis-subset have ...\subseteqpmdl (set xs) by (rule pmdl.span-mono)
    finally show f
qed
lemma comp-red-basis-adds:
    assumes p\in set xs and p\not=0
    obtains q where q\in set (comp-red-basis xs) and lt q addst lt p
proof -
    from assms obtain q1 where q1 \in set (comp-min-basis xs) and lt q1 addst lt p
        by (rule comp-min-basis-adds)
    from <q1 \in set (comp-min-basis xs)> have lt q1 \inlt'set (comp-min-basis xs)
by simp
    also have ... = lt'set ((comp-min-basis xs) @ []) by simp
    also have ... = lt'set (comp-red-basis-aux (comp-min-basis xs) [])
        by (rule comp-red-basis-aux-lt, simp-all, rule comp-min-basis-is-minimal-basis,
rule comp-min-basis-distinct)
    finally obtain q}\mathrm{ where qG set (comp-red-basis-aux (comp-min-basis xs) []) and
lt q=lt q1
    by auto
    show ?thesis
    proof
        show q}\in\mathrm{ set (comp-red-basis xs) unfolding comp-red-basis-def by fact
    next
    from <lt q1 addst lt p> show lt q addst lt p unfolding <lt q = lt q1>.
    qed
```


## qed

lemma comp-red-basis-lt:
assumes $p \in$ set (comp-red-basis xs)
obtains $q$ where $q \in$ set $x s$ and $q \neq 0$ and $l t q=l t p$
proof -
have eq: lt'set ((comp-min-basis xs) @ []) =lt'set (comp-red-basis-aux (comp-min-basis xs) [])
by (rule comp-red-basis-aux-lt, simp-all, rule comp-min-basis-is-minimal-basis, rule comp-min-basis-distinct)
from assms have $l t p \in l t$ ' set (comp-red-basis xs) by simp
also have $\ldots=l t$ 'set (comp-red-basis-aux (comp-min-basis xs) []) unfolding comp-red-basis-def ..
also have $\ldots=l t$ ' set (comp-min-basis $x s$ ) unfolding eq[symmetric] by simp
finally obtain $q$ where $q \in$ set (comp-min-basis xs) and $l t q=l t p$ by auto
show ?thesis
proof
show $q \in$ set $x s$ by (rule, fact, rule comp-min-basis-subset)
next
show $q \neq 0$ by (rule comp-min-basis-nonzero, fact)
qed fact
qed
lemma comp-red-basis-is-red: is-red (set (comp-red-basis xs)) $f \longleftrightarrow$ is-red (set xs)
$f$
proof
assume is-red (set (comp-red-basis xs)) f
then obtain $x t$ where $x \in$ set (comp-red-basis $x s$ ) and $t \in$ keys $f$ and $x \neq 0$
and $l t x a d d s_{t} t$
by (rule is-red-addsE)
from $\langle x \in$ set (comp-red-basis xs) $\rangle$ obtain $y$ where yin: $y \in$ set $x s$ and $y \neq 0$
and $l t y=l t x$
by (rule comp-red-basis-lt)
show is-red (set xs) f
proof (rule is-red-addsI)
from $\left\langle l t x a d d s_{t} t\right\rangle$ show $l t y a d d s_{t} t$ unfolding $\langle l t y=l t x\rangle$.
qed fact+
next
assume is-red (set xs) f
then obtain $x t$ where $x \in$ set $x s$ and $t \in$ keys $f$ and $x \neq 0$ and lt $x a d d s_{t} t$
by (rule is-red-addsE)
from $\langle x \in$ set $x s\rangle\langle x \neq 0\rangle$ obtain $y$ where yin: $y \in$ set (comp-red-basis $x s$ ) and
lt $y$ addst lt $x$
by (rule comp-red-basis-adds)
show is-red (set (comp-red-basis xs)) f
proof (rule is-red-addsI)
from $\left\langle l t y\right.$ $\left.y d d s_{t} l t x\right\rangle\left\langle l t x\right.$ adds $\left.s_{t} t\right\rangle$ show lt $y$ adds $s_{t} t$ by (rule adds-term-trans)
next
from yin show $y \neq 0$ by (rule comp-red-basis-nonzero)

```
    qed fact+
qed
lemma comp-red-basis-is-auto-reduced: is-auto-reduced (set (comp-red-basis xs))
    unfolding is-auto-reduced-def remove-def
proof (intro ballI)
    fix }
    assume xin: }x\in\mathrm{ set (comp-red-basis xs)
    show \neg is-red (set (comp-red-basis xs) - {x}) x unfolding comp-red-basis-def
    proof (rule comp-red-basis-aux-irred, simp-all, rule comp-min-basis-is-minimal-basis,
rule comp-min-basis-distinct)
    from xin show }x\in\mathrm{ set (comp-red-basis-aux (comp-min-basis xs) []) unfolding
comp-red-basis-def .
    qed
qed
lemma comp-red-basis-dgrad-p-set-le:
    assumes dickson-grading d
    shows dgrad-p-set-le d (set (comp-red-basis xs)) (set xs)
proof -
    have dgrad-p-set-le d (set (comp-red-basis xs)) (set (comp-min-basis xs) \cup set [])
    unfolding comp-red-basis-def using assms by (rule comp-red-basis-aux-dgrad-p-set-le)
    also have ... = set (comp-min-basis xs) by simp
    also from comp-min-basis-subset have dgrad-p-set-le d ... (set xs)
    by (rule dgrad-p-set-le-subset)
    finally show ?thesis.
qed
```


### 12.5 Auto-Reduction and Monicity

definition comp-red-monic-basis :: (' $\left.t \Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.$ field $)$ list where comp-red-monic-basis $x s=$ map monic (comp-red-basis xs)
lemma set-comp-red-monic-basis: set (comp-red-monic-basis xs) $=$ monic ' (set (comp-red-basis xs))
by (simp add: comp-red-monic-basis-def)
lemma comp-red-monic-basis-nonzero:
assumes $p \in$ set (comp-red-monic-basis xs)
shows $p \neq 0$
proof -
from assms obtain $p^{\prime}$ where $p$-def: $p=$ monic $p^{\prime}$ and $p^{\prime}: p^{\prime} \in$ set (comp-red-basis $x s)$
unfolding set-comp-red-monic-basis ..
from $p^{\prime}$ have $p^{\prime} \neq 0$ by (rule comp-red-basis-nonzero)
thus ?thesis unfolding $p$-def monic- 0 -iff .
qed
lemma comp-red-monic-basis-is-monic-set: is-monic-set (set (comp-red-monic-basis
xs))
unfolding set-comp-red-monic-basis by (rule image-monic-is-monic-set)
lemma pmdl-comp-red-monic-basis-subset: pmdl (set (comp-red-monic-basis xs)) $\subseteq p m d l($ set xs)
unfolding set-comp-red-monic-basis pmdl-image-monic by (fact pmdl-comp-red-basis-subset)
lemma comp-red-monic-basis-is-auto-reduced: is-auto-reduced (set (comp-red-monic-basis xs))
unfolding set-comp-red-monic-basis by (rule image-monic-is-auto-reduced, rule comp-red-basis-is-auto-reduced)
lemma comp-red-monic-basis-dgrad-p-set-le:
assumes dickson-grading d
shows dgrad-p-set-le d (set (comp-red-monic-basis xs)) (set xs)
proof -
have dgrad-p-set-le d (monic '(set (comp-red-basis xs))) (set (comp-red-basis xs))
by (simp add: dgrad-p-set-le-def, fact dgrad-set-le-refl)
also from assms have dgrad-p-set-le $d \ldots$. set $x s$ ) by (rule comp-red-basis-dgrad-p-set-le)
finally show ?thesis by (simp add: set-comp-red-monic-basis)
qed
end
end

## 13 Reduced Gröbner Bases

theory Reduced-GB
imports Groebner-Bases Auto-Reduction
begin
lemma (in gd-term) GB-image-monic: is-Groebner-basis (monic ' $G$ ) $\longleftrightarrow$ is-Groebner-basis G
by (simp add: GB-alt-1)

### 13.1 Definition and Uniqueness of Reduced Gröbner Bases

context ordered-term
begin
definition is-reduced- $G B::\left(' t \Rightarrow{ }_{0}{ }^{\prime} b::\right.$ field $)$ set $\Rightarrow$ bool where
is-reduced-GB $B \equiv$ is-Groebner-basis $B \wedge$ is-auto-reduced $B \wedge$ is-monic-set $B \wedge$
$0 \notin B$
lemma reduced-GB-D1:
assumes is-reduced-GB $G$
shows is-Groebner-basis $G$
using assms unfolding is-reduced-GB-def by simp

```
lemma reduced-GB-D2
    assumes is-reduced-GB G
    shows is-auto-reduced G
    using assms unfolding is-reduced-GB-def by simp
    lemma reduced-GB-D3:
    assumes is-reduced-GB G
    shows is-monic-set G
    using assms unfolding is-reduced-GB-def by simp
lemma reduced-GB-D4:
    assumes is-reduced-GB G and g\inG
    shows g\not=0
    using assms unfolding is-reduced-GB-def by auto
lemma reduced-GB-lc:
    assumes major: is-reduced-GB G and g\inG
    shows lc g=1
    by (rule is-monic-setD, rule reduced-GB-D3, fact major, fact }\langleg\inG\rangle\mathrm{ , rule re-
duced-GB-D4, fact major, fact }<g\inG`
end
context gd-term
begin
lemma is-reduced-GB-subsetI:
    assumes Ared: is-reduced-GB A and BGB:is-Groebner-basis B and Bmon:
is-monic-set B
    and *: \bigwedgeab. a \in A\Longrightarrowb\inB\Longrightarrowa\not=0\Longrightarrowb\not=0\Longrightarrowa-b\not=0\Longrightarrowlt (a
-b)\in keys b\Longrightarrowlt (a-b) \prec}\mp@subsup{}{t}{}lt b\Longrightarrow Fals
    and id-eq: pmdl }A=pmdl 
    shows A\subseteqB
proof
    fix }
    assume a\inA
    have }a\not=0\mathrm{ by (rule reduced-GB-D4, fact Ared, fact }\langlea\inA\rangle
    have lca: lc a=1 by (rule reduced-GB-lc, fact Ared, fact <a \inA〉)
    have AGB: is-Groebner-basis A by (rule reduced-GB-D1, fact Ared)
    from }\langlea\inA\rangle\mathrm{ have }a\inpmdl A by (rule pmdl.span-base)
    also have ... = pmdl B using id-eq by simp
    finally have }a\inpmdl B
    from BGB this }\langlea\not=0\rangle\mathrm{ obtain }b\mathrm{ where }b\inB\mathrm{ and }b\not=0\mathrm{ and baddsa:lt b
addst lt a
    by (rule GB-adds-lt)
```

from Bmon this(1) this(2) have $l c b: l c b=1$ by (rule is-monic-setD)
from $\langle b \in B\rangle$ have $b \in p m d l B$ by (rule pmdl.span-base)
also have $\ldots=p m d l A$ using $i d-e q$ by $\operatorname{simp}$
finally have $b \in p m d l A$.
have $l t$-eq: lt $b=l t a$
proof (rule ccontr)
assume lt $b \neq l t a$
from $A G B\langle b \in p m d l A\rangle\langle b \neq 0\rangle$ obtain $a^{\prime}$
where $a^{\prime} \in A$ and $a^{\prime} \neq 0$ and $a^{\prime} a d d s b$ : lt $a^{\prime} a d d s_{t}$ lt $b$ by (rule GB-adds-lt)
have $a^{\prime} a d d s a$ : lt $a^{\prime} a d d s_{t} l t a$ by (rule adds-term-trans, fact $a^{\prime}$ addsb, fact baddsa)
have $l t a^{\prime} \neq l t a$
proof
assume lt $a^{\prime}=l t a$
hence aaddsa': lt a addst lt a' by (simp add: adds-term-refl)
have lt a addst lt b by (rule adds-term-trans, fact aaddsa', fact a'addsb)
have $l t a=l t b$ by (rule adds-term-antisym, fact+)
with $\langle l t b \neq l t$ a〉 show False by simp
qed
hence $a^{\prime} \neq a$ by auto
with $\left\langle a^{\prime} \in A\right\rangle$ have $a^{\prime} \in A-\{a\}$ by blast
have is-red: is-red $(A-\{a\})$ a by (intro is-red-addsI, fact, fact, rule lt-in-keys, fact+)
have $\neg i s$-red $(A-\{a\})$ a by (rule is-auto-reduced $D$, rule reduced-GB-D2, fact Ared, fact+)
from this is-red show False ..
qed
have $a-b=0$
proof (rule ccontr)
let ${ }^{2} c=a-b$
assume ? $c \neq 0$
have ?c $\in$ pmdl $A$ by (rule pmdl.span-diff, fact+)
also have $\ldots=p m d l B$ using $i d-e q$ by simp
finally have $? c \in p m d l B$.
from $\langle b \neq 0\rangle$ have $-b \neq 0$ by simp
have $l t(-b)=l t$ a unfolding $l t$-uminus by fact
have $l c(-b)=-l c$ a unfolding $l c$-uminus lca lcb ..
from $\langle ? c \neq 0\rangle$ have $a+(-b) \neq 0$ by simp
have $l t ? c \in$ keys ?c by (rule lt-in-keys, fact)
have keys ?c $\subseteq($ keys $a \cup$ keys $b)$ by (fact keys-minus)
with $\langle l t ? c \in$ keys $? c\rangle$ have $l t ? c \in$ keys $a \vee l t ? c \in k e y s ~ b$ by auto
thus False
proof
assume $l t ? c \in$ keys a
from $A G B\langle ? c \in p m d l A\rangle\langle ? c \neq 0\rangle$ obtain $a^{\prime}$

```
            where }\mp@subsup{a}{}{\prime}\inA\mathrm{ and }\mp@subsup{a}{}{\prime}\not=0\mathrm{ and }\mp@subsup{a}{}{\prime}addsc:lt \mp@subsup{a}{}{\prime}addststlt ?c by (rule GB-adds-lt
            from a'addsc have lt a' }\mp@subsup{\preceq}{t}{}lt\mathrm{ ? c by (rule ord-adds-term)
            also have ... = lt (a+(-b)) by simp
            also have ... \prec}\mp@subsup{t}{l}{lt a by (rule lt-plus-lessI, fact+)
            finally have lt a' }\mp@subsup{⿱}{t}{}lta
            hence lt a'}=ll l a by sim
            hence }\mp@subsup{a}{}{\prime}\not=a\mathrm{ by auto
            with }\langle\mp@subsup{a}{}{\prime}\inA\rangle\mathrm{ have }\mp@subsup{a}{}{\prime}\inA-{a}\mathrm{ by blast
            have is-red: is-red (A - {a}) a by (intro is-red-addsI, fact, fact, fact+)
            have ᄀis-red (A - {a}) a by (rule is-auto-reducedD, rule reduced-GB-D2,
fact Ared, fact+)
            from this is-red show False ..
    next
        assume lt ?c c\in keys b
        with }\langlea\inA\rangle\langleb\inB\rangle\langlea\not=0\rangle\langleb\not=0\rangle\langle?c\not=0\rangle\mathrm{ show False
        proof (rule *)
            have lt ?c = lt ((-b)+a) by simp
            also have ... }\mp@subsup{\prec}{t}{lt}(-b
            proof (rule lt-plus-lessI)
                from <?c\not=0\rangle show -b+a\not=0 by simp
            next
                from}\langlelt(-b)=lt a> show lt a=lt (-b) by sim
            next
                from <lc (-b) = - lc a> show lc a = - lc (-b) by simp
            qed
            finally show lt ?c }\mp@subsup{\prec}{t}{}ltb\mathrm{ unfolding lt-uminus .
        qed
    qed
    qed
    hence }a=b\mathrm{ by simp
    with }\langleb\inB\rangle\mathrm{ show }a\inB\mathrm{ by simp
qed
lemma is-reduced-GB-unique':
    assumes Ared: is-reduced-GB A and Bred:is-reduced-GB B and id-eq: pmdl A
= pmdl B
    shows A\subseteqB
proof -
    from Bred have BGB: is-Groebner-basis B by (rule reduced-GB-D1)
    with assms(1) show ?thesis
    proof (rule is-reduced-GB-subsetI)
        from Bred show is-monic-set B by (rule reduced-GB-D3)
    next
        fix a b :: 't }\mp@subsup{=>}{0}{\prime}\mp@subsup{}{}{\prime}
        let ?c = a-b
```

```
    assume \(a \in A\) and \(b \in B\) and \(a \neq 0\) and \(b \neq 0\) and \(? c \neq 0\) and \(l t ? c \in\)
keys \(b\) and \(l t ? c \prec_{t} l t b\)
    from \(\langle a \in A\rangle\) have \(a \in p m d l B\) by (simp only: id-eq[symmetric], rule pmdl.span-base)
    moreover from \(\langle b \in B\rangle\) have \(b \in p m d l B\) by (rule pmdl.span-base)
    ultimately have ? \(c \in p m d l B\) by (rule pmdl.span-diff)
    from \(B G B\) this \(\langle ? c \neq 0\rangle\) obtain \(b^{\prime}\)
        where \(b^{\prime} \in B\) and \(b^{\prime} \neq 0\) and \(b^{\prime} a d d s c\) : lt \(b^{\prime} a d d s_{t}\) lt \(? c\) by (rule GB-adds-lt)
    from \(b^{\prime} a d d s c\) have \(l t b^{\prime} \preceq_{t} l t\) ?c by (rule ord-adds-term)
    also have \(\ldots \prec_{t} l t b\) by fact
    finally have \(l t b^{\prime} \prec_{t} l t b\) unfolding lt-uminus .
    hence \(l t b^{\prime} \neq l t b\) by simp
    hence \(b^{\prime} \neq b\) by auto
    with \(\left\langle b^{\prime} \in B\right\rangle\) have \(b^{\prime} \in B-\{b\}\) by blast
    have is-red: is-red \((B-\{b\}) b\) by (intro is-red-addsI, fact, fact, fact+)
    have \(\neg i s\)-red \((B-\{b\}) b\) by (rule is-auto-reducedD, rule reduced-GB-D2, fact
Bred, fact+)
    from this is-red show False ..
    qed fact
qed
theorem is-reduced-GB-unique:
    assumes Ared: is-reduced-GB \(A\) and Bred: is-reduced-GB B and id-eq: pmdl \(A\)
= pmdl \(B\)
    shows \(A=B\)
proof
    from assms show \(A \subseteq B\) by (rule is-reduced-GB-unique')
next
    from Bred Ared id-eq[symmetric] show \(B \subseteq A\) by (rule is-reduced-GB-unique')
qed
```


### 13.2 Computing Reduced Gröbner Bases by Auto-Reduction

### 13.2.1 Minimal Bases

lemma minimal-basis-is-reduced-GB:
assumes is-minimal-basis $B$ and is-monic-set $B$ and is-reduced- $G B G$ and $G$ $\subseteq B$
and $p m d l B=p m d l ~ G$
shows $B=G$
using - assms(3) assms(5)
proof (rule is-reduced-GB-unique)
from $\operatorname{assms}(3)$ have is-Groebner-basis $G$ by (rule reduced-GB-D1)
show is-reduced-GB $B$ unfolding is-reduced-GB-def
proof (intro conjI)
show $0 \notin B$
proof
assume $0 \in B$
with assms（1）have $0 \neq\left(0::^{\prime} t \Rightarrow_{0}^{\prime} b\right)$ by（rule is－minimal－basisD1）
thus False by simp
qed
next
from 〈is－Groebner－basis $G\rangle \operatorname{assms}(4)$ assms（5）show is－Groebner－basis B by （rule GB－subset）
next
show is－auto－reduced $B$ unfolding is－auto－reduced－def
proof（intro ballI notI）
fix $b$
assume $b \in B$
with $\operatorname{assms}(1)$ have $b \neq 0$ by（rule is－minimal－basisD1）
assume $i s$－red $(B-\{b\}) b$
then obtain $f$ where $f \in B-\{b\}$ and is－red $\{f\} b$ by（rule is－red－singletonI）
from this（1）have $f \in B$ and $f \neq b$ by simp－all
from $\operatorname{assms}(1)\langle f \in B\rangle$ have $f \neq 0$ by（rule is－minimal－basisD1）
from $\langle f \in B\rangle$ have $f \in p m d l$ B by（rule pmdl．span－base）
hence $f \in$ pmdl $G$ by（simp only：assms（5））
from $\langle i s-G r o e b n e r-b a s i s ~ G\rangle$ this $\langle f \neq 0\rangle$ obtain $g$ where $g \in G$ and $g \neq 0$
and $l t g$ addst $l t f$
by（rule GB－adds－lt）
from $\langle g \in G\rangle\langle G \subseteq B\rangle$ have $g \in B$ ．．
have $g=f$
proof（rule ccontr）
assume $g \neq f$
with $\operatorname{assms}(1)\langle g \in B\rangle\langle f \in B\rangle$ have $\neg l t g$ addst $l t f$ by（rule is－minimal－basisD2） from this $\left\langle l t ~ g a d d s_{t} l t f\right\rangle$ show False ．．
qed
with $\langle g \in G\rangle$ have $f \in G$ by simp
with $\langle f \in B-\{b\}\rangle\langle i s$－red $\{f\}$ b〉 have red：is－red $(G-\{b\}) b$
by（meson Diff－iff is－red－singletonD）
from $\langle b \in B\rangle$ have $b \in p m d l B$ by（rule pmdl．span－base）
hence $b \in p m d l G$ by（simp only：assms（5））
from $\langle i s-G r o e b n e r-b a s i s ~ G\rangle$ this $\langle b \neq 0\rangle$ obtain $g^{\prime}$ where $g^{\prime} \in G$ and $g^{\prime} \neq$ 0 and $l t g^{\prime}$ adds $s_{t} l t b$ by（rule GB－adds－lt）
from $\left\langle g^{\prime} \in G\right\rangle\langle G \subseteq B\rangle$ have $g^{\prime} \in B$ ．．
have $g^{\prime}=b$
proof（rule ccontr）
assume $g^{\prime} \neq b$
with $\operatorname{assms}(1)\left\langle g^{\prime} \in B\right\rangle\langle b \in B\rangle$ have $\neg l t g^{\prime} a d d s_{t} l t b$ by（rule
is－minimal－basisD2）
from this 〈lt $g^{\prime}$ adds $\left.s_{t} l t b\right\rangle$ show False ．．
qed
with $\left\langle g^{\prime} \in G\right\rangle$ have $b \in G$ by simp
from assms（3）have is－auto－reduced $G$ by（rule reduced－GB－D2）

```
        from this }\langleb\inG\rangle\mathrm{ have }\neg\mathrm{ is-red (G-{b}) b by (rule is-auto-reducedD)
        from this red show False ..
        qed
    qed fact
qed
```


### 13.2.2 Computing Minimal Bases

lemma comp-min-basis-pmdl:
assumes is-Groebner-basis (set xs)
shows $p m d l($ set $($ comp-min-basis $x s))=p m d l($ set $x s)($ is $p m d l($ set ?ys $)=-)$
using finite-set
proof (rule pmdl-eqI-adds-lt-finite)
from comp-min-basis-subset show $*: p m d l($ set ?ys) $\subseteq p m d l($ set $x s)$ by (rule pmdl.span-mono)
next
fix $f$
assume $f \in p m d l($ set $x s)$ and $f \neq 0$
with assms obtain $g$ where $g \in$ set xs and $g \neq 0$ and 1 : lt $g$ addst $l t f$ by
(rule GB-adds-lt)
from this(1, 2) obtain $g^{\prime}$ where $g^{\prime} \in$ set ?ys and 2: lt $g^{\prime}$ addst $l t g$
by (rule comp-min-basis-adds)
note this(1)
moreover from this have $g^{\prime} \neq 0$ by (rule comp-min-basis-nonzero)
moreover from 21 have $l t g^{\prime}$ adds $s_{t}$ lt $f$ by (rule adds-term-trans)
ultimately show $\exists g \in$ set ? ys. $g \neq 0 \wedge l t g$ addst $l t f$ by blast
qed
lemma comp-min-basis-GB:
assumes is-Groebner-basis (set xs)
shows is-Groebner-basis (set (comp-min-basis xs)) (is is-Groebner-basis (set ?ys))
unfolding GB-alt-2-finite[OF finite-set]
proof (intro ballI impI)
fix $f$
assume $f \in p m d l$ (set ?ys)
also from assms have $\ldots=p m d l$ (set xs) by (rule comp-min-basis-pmdl)
finally have $f \in p m d l$ (set xs).
moreover assume $f \neq 0$
ultimately have is-red (set xs) $f$ using assms unfolding GB-alt-2-finite[OF
finite-set] by blast
thus is-red (set ?ys) $f$ by (rule comp-min-basis-is-red)
qed

### 13.2.3 Computing Reduced Bases

lemma comp-red-basis-pmdl:
assumes is-Groebner-basis (set xs)
shows $p m d l($ set $($ comp-red-basis xs $))=p m d l($ set $x s)$
proof (rule, fact pmdl-comp-red-basis-subset, rule)
fix $f$

```
    assume f}\inpmdl (set xs
    show f\inpmdl (set (comp-red-basis xs))
    proof (cases f=0)
        case True
        show ?thesis unfolding True by (rule pmdl.span-zero)
    next
        case False
        let ?xs = comp-red-basis xs
        have (red (set ?xs))** f 0
        proof (rule is-red-implies-0-red-finite, fact finite-set, fact pmdl-comp-red-basis-subset)
        fix q
        assume q}\not=0\mathrm{ and }q\inpmdl (set xs
        with assms have is-red (set xs) q by (rule GB-imp-reducibility)
        thus is-red (set (comp-red-basis xs)) q unfolding comp-red-basis-is-red .
    qed fact
    thus ?thesis by (rule red-rtranclp-0-in-pmdl)
    qed
qed
lemma comp-red-basis-GB:
    assumes is-Groebner-basis (set xs)
    shows is-Groebner-basis (set (comp-red-basis xs))
    unfolding GB-alt-2-finite[OF finite-set]
proof (intro ballI impI)
    fix f
    assume fin:f\inpmdl (set (comp-red-basis xs))
    hence f\inpmdl (set xs) unfolding comp-red-basis-pmdl[OF assms].
    assume f}\not=
    from assms <f \not=0\rangle\langlef\in pmdl (set xs)\rangle show is-red (set (comp-red-basis xs)) f
        by (simp add: comp-red-basis-is-red GB-alt-2-finite)
qed
```


### 13.2.4 Computing Reduced Gröbner Bases

lemma comp-red-monic-basis-pmdl:
assumes is-Groebner-basis (set xs)
shows pmdl (set (comp-red-monic-basis xs)) = pmdl (set xs)
unfolding set-comp-red-monic-basis pmdl-image-monic comp-red-basis-pmdl[OF
assms] ..
lemma comp-red-monic-basis-GB:
assumes is-Groebner-basis (set xs)
shows is-Groebner-basis (set (comp-red-monic-basis xs))
unfolding set-comp-red-monic-basis GB-image-monic using assms by (rule comp-red-basis-GB)
lemma comp-red-monic-basis-is-reduced-GB:
assumes is-Groebner-basis (set xs)
shows is-reduced-GB (set (comp-red-monic-basis xs))
unfolding is-reduced-GB-def
proof (intro conjI, rule comp-red-monic-basis-GB, fact assms,
rule comp-red-monic-basis-is-auto-reduced, rule comp-red-monic-basis-is-monic-set, intro notI)
assume $0 \in$ set (comp-red-monic-basis $x s$ )
hence $0 \neq\left(0:: ' t \Rightarrow_{0}{ }^{\prime} b\right)$ by (rule comp-red-monic-basis-nonzero)
thus False by simp
qed
lemma ex-finite-reduced-GB-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set d m
obtains $G$ where $G \subseteq$ dgrad-p-set $d m$ and finite $G$ and is-reduced- $G B G$ and
$p m d l G=p m d l F$
proof -
from assms obtain $G 0$ where G0-sub: GO $\subseteq$ dgrad-p-set $d m$ and fin: finite G0
and $g b$ : is-Groebner-basis $G 0$ and pid: pmdl G0 $=p m d l F$
by (rule ex-finite-GB-dgrad-p-set)
from fin obtain $x s$ where set: $G 0=$ set xs using finite-list by blast
let ? $G=$ set (comp-red-monic-basis xs)
show ?thesis
proof
from $\operatorname{assms}(1)$ have dgrad-p-set-le d (set (comp-red-monic-basis xs)) G0 unfolding set
by (rule comp-red-monic-basis-dgrad-p-set-le)
from this G0-sub show set (comp-red-monic-basis xs) $\subseteq$ dgrad-p-set d m
by (rule dgrad-p-set-le-dgrad-p-set)
next
from $g b$ show $r g b$ : is-reduced- $G B$ ? $G$ unfolding set
by (rule comp-red-monic-basis-is-reduced-GB)
next
from $g b$ show $p m d l ? G=p m d l F$ unfolding set pid[symmetric]
by (rule comp-red-monic-basis-pmdl)
qed (fact finite-set)
qed
theorem ex-unique-reduced-GB-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set $d m$
shows $\exists!G . G \subseteq$ dgrad- $p$-set $d m \wedge$ finite $G \wedge$ is-reduced- $G B G \wedge p m d l G=$ pmdl F
proof -
from assms obtain $G$ where $G \subseteq d g r a d-p$-set $d m$ and finite $G$
and $i s$-reduced- $G B G$ and $G: p m d l ~ G=p m d l F$ by (rule ex-finite-reduced-GB-dgrad-p-set)
hence $G \subseteq d g r a d-p$-set $d m \wedge$ finite $G \wedge i s$-reduced- $G B G \wedge p m d l G=p m d l F$
by $\operatorname{simp}$
thus ?thesis
proof (rule ex1I)
fix $G^{\prime}$

```
            assume G'\subseteqdgrad-p-set d m ^ finite G' }\mp@subsup{G}{}{\prime}\wedge\mathrm{ is-reduced-GB G'^ pmdl G' =
pmdl F
    hence is-reduced-GB G' and G':pmdl G' = pmdl F by simp-all
    note this(1) <is-reduced-GB G〉
    moreover have pmdl G' = pmdl G by (simp only: G G')
    ultimately show }\mp@subsup{G}{}{\prime}=G\mathrm{ by (rule is-reduced-GB-unique)
    qed
qed
corollary ex-unique-reduced-GB-dgrad-p-set':
    assumes dickson-grading d and finite (component-of-term' Keys F) and F}
dgrad-p-set d m
    shows \exists!G. finite G}\wedge is-reduced-GB G\wedge pmdl G=pmdl 
proof -
    from assms obtain G where G\subseteqdgrad-p-set d m and finite G
    and is-reduced-GB G and G:pmdl G = pmdl F by (rule ex-finite-reduced-GB-dgrad-p-set)
    hence finite G}\wedge is-reduced-GB G\wedge pmdl G=pmdl F by sim
    thus ?thesis
    proof (rule ex1I)
        fix G}\mp@subsup{G}{}{\prime
        assume finite G'^ is-reduced-GB G'^ pmdl G' = pmdl F
        hence is-reduced-GB G' and G': pmdl G' = pmdl F by simp-all
        note this(1)<is-reduced-GB G`
        moreover have pmdl G' = pmdl G by (simp only: G G')
        ultimately show }\mp@subsup{G}{}{\prime}=G\mathrm{ by (rule is-reduced-GB-unique)
    qed
qed
definition reduced-GB :: ('t }\mp@subsup{=>}{0}{\prime}'b) set => (' ' >>0 ' b::field) se
    where reduced-GB B = (THE G. finite G \is-reduced-GB G^pmdl G=pmdl
B)
reduced- \(G B\) returns the unique reduced Gröbner basis of the given set, provided its Dickson grading is bounded. Combining comp-red-monic-basis with any function for computing Gröbner bases, e.g. \(g b\) from theory "Buchberger", makes reduced-GB computable.
lemma finite-reduced-GB-dgrad-p-set:
assumes dickson-grading \(d\) and finite (component-of-term'Keys \(F\) ) and \(F \subseteq\) dgrad-p-set d m
shows finite (reduced-GB F)
unfolding reduced-GB-def
by (rule the112, rule ex-unique-reduced-GB-dgrad-p-set', fact, fact, fact, elim conjE)
lemma reduced-GB-is-reduced-GB-dgrad-p-set:
assumes dickson-grading \(d\) and finite (component-of-term' Keys \(F\) ) and \(F \subseteq\) dgrad-p-set d m
shows is-reduced-GB (reduced-GB F)
unfolding reduced-GB-def
```

by (rule the1I2, rule ex-unique-reduced-GB-dgrad-p-set', fact, fact, fact, elim conjE)
lemma reduced-GB-is-GB-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set d m
shows is-Groebner-basis (reduced-GB F)
proof -
from assms have is-reduced-GB (reduced-GB F) by (rule reduced-GB-is-reduced-GB-dgrad-p-set) thus ?thesis unfolding is-reduced-GB-def ..
qed
lemma reduced-GB-is-auto-reduced-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set d m
shows is-auto-reduced (reduced-GB F)
proof -
from assms have is-reduced-GB (reduced-GB F) by (rule reduced-GB-is-reduced-GB-dgrad-p-set) thus ?thesis unfolding is-reduced-GB-def by simp
qed
lemma reduced-GB-is-monic-set-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$
dgrad-p-set $d m$
shows is-monic-set (reduced-GB F)
proof -
from assms have is-reduced-GB (reduced-GB F) by (rule reduced-GB-is-reduced-GB-dgrad-p-set)
thus ?thesis unfolding is-reduced-GB-def by simp
qed
lemma reduced-GB-nonzero-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$
dgrad-p-set d m
shows $0 \notin$ reduced-GB F
proof -
from assms have is-reduced-GB (reduced-GB $F$ ) by (rule reduced-GB-is-reduced-GB-dgrad-p-set)
thus? ?thesis unfolding is-reduced-GB-def by simp
qed
lemma reduced-GB-pmdl-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set d m
shows $p m d l($ reduced-GB $F)=p m d l F$
unfolding reduced-GB-def
by (rule the1I2, rule ex-unique-reduced-GB-dgrad-p-set', fact, fact, fact, elim conjE)
lemma reduced-GB-unique-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term' Keys $F$ ) and $F \subseteq$
dgrad-p-set d m
and $i s$-reduced- $G B G$ and $p m d l G=p m d l F$
shows reduced-GB $F=G$
by (rule is-reduced-GB-unique, rule reduced-GB-is-reduced-GB-dgrad-p-set, fact+, simp only: reduced-GB-pmdl-dgrad-p-set[OF $\operatorname{assms}(1,2,3)] \operatorname{assms}(5))$
lemma reduced-GB-dgrad-p-set:
assumes dickson-grading $d$ and finite (component-of-term'Keys $F$ ) and $F \subseteq$ dgrad-p-set d m
shows reduced-GB $F \subseteq$ dgrad- $p$-set $d m$
proof -
from assms obtain $G$ where $G: G \subseteq$ dgrad-p-set $d m$ and is-reduced- $G B G$ and $p m d l ~ G=p m d l ~ F$
by (rule ex-finite-reduced-GB-dgrad-p-set)
from assms this (2, 3) have reduced-GB F $=G$ by (rule reduced-GB-unique-dgrad-p-set)
with $G$ show ?thesis by simp
qed
lemma reduced-GB-unique:
assumes finite $G$ and $i s$-reduced- $G B G$ and $p m d l ~ G=p m d l ~ F$
shows reduced-GB $F=G$
proof -
from assms have finite $G \wedge i s$-reduced- $G B G \wedge$ pmdl $G=p m d l$ $F$ by simp
thus ?thesis unfolding reduced-GB-def
proof (rule the-equality)
fix $G^{\prime}$
assume finite $G^{\prime} \wedge i s$-reduced-GB $G^{\prime} \wedge p m d l G^{\prime}=p m d l F$
hence is-reduced- $G B G^{\prime}$ and eq: pmdl $G^{\prime}=p m d l ~ F$ by simp-all
note this(1)
moreover note assms(2)
moreover have $p m d l G^{\prime}=p m d l G$ by (simp only: assms(3)eq)
ultimately show $G^{\prime}=G$ by (rule is-reduced-GB-unique)
qed
qed
lemma is-reduced-GB-empty: is-reduced-GB \{\}
by (simp add: is-reduced-GB-def is-Groebner-basis-empty is-monic-set-def is-auto-reduced-def)
lemma is-reduced-GB-singleton: is-reduced-GB $\{f\} \longleftrightarrow l c f=1$
proof
assume is-reduced-GB $\{f\}$
hence is-monic-set $\{f\}$ and $f \neq 0$ by (rule reduced-GB-D3, rule reduced-GB-D4)
simp
from this(1)-this(2) show lc $f=1$ by (rule is-monic-setD) simp
next
assume $l c f=1$
moreover from this have $f \neq 0$ by auto
ultimately show is-reduced-GB $\{f\}$
by (simp add: is-reduced-GB-def is-Groebner-basis-singleton is-monic-set-def

```
is-auto-reduced-def
    not-is-red-empty)
qed
lemma reduced-GB-empty: reduced-GB {} = {}
    using finite.emptyI is-reduced-GB-empty refl by (rule reduced-GB-unique)
lemma reduced-GB-singleton: reduced-GB {f}=(if f=0 then {} else {monic f})
proof (cases f=0)
    case True
    from finite.emptyI is-reduced-GB-empty have reduced-GB {f} ={}
        by (rule reduced-GB-unique) (simp add: True flip: pmdl.span-Diff-zero[of {0}])
    with True show ?thesis by simp
next
    case False
    have reduced-GB {f}={monic f}
    proof (rule reduced-GB-unique)
        from False have lc f\not=0 by (rule lc-not-0)
        thus is-reduced-GB {monic f} by (simp add: is-reduced-GB-singleton monic-def)
    next
        have pmdl {monic f} = pmdl (monic'{f}) by simp
        also have ...=pmdl {f} by (fact pmdl-image-monic)
        finally show pmdl {monic f} = pmdl {f} .
    qed simp
    with False show ?thesis by simp
qed
```

lemma ex-unique-reduced-GB-finite: finite $F \Longrightarrow(\exists!G$. finite $G \wedge$ is-reduced- $G B$ $G \wedge p m d l G=p m d l F)$
by (rule ex-unique-reduced-GB-dgrad-p-set', rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma finite-reduced-GB-finite: finite $F \Longrightarrow$ finite (reduced-GB F) by (rule finite-reduced-GB-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma reduced-GB-is-reduced-GB-finite: finite $F \Longrightarrow$ is-reduced-GB (reduced-GB F)
by (rule reduced-GB-is-reduced-GB-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma reduced-GB-is-GB-finite: finite $F \Longrightarrow$ is-Groebner-basis (reduced-GB F) by (rule reduced-GB-is-GB-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma reduced-GB-is-auto-reduced-finite: finite $F \Longrightarrow$ is-auto-reduced (reduced-GB F) by (rule reduced-GB-is-auto-reduced-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma reduced-GB-is-monic-set-finite: finite $F \Longrightarrow$ is-monic-set (reduced-GB F) by (rule reduced-GB-is-monic-set-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma reduced-GB-nonzero-finite: finite $F \Longrightarrow 0 \notin$ reduced- $G B F$
by (rule reduced-GB-nonzero-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma reduced-GB-pmdl-finite: finite $F \Longrightarrow p m d l(r e d u c e d-G B F)=p m d l F$
by (rule reduced-GB-pmdl-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
lemma reduced-GB-unique-finite: finite $F \Longrightarrow$ is-reduced- $G B G \Longrightarrow p m d l G=$ pmdl $F \Longrightarrow$ reduced-GB $F=G$
by (rule reduced-GB-unique-dgrad-p-set, rule dickson-grading-dgrad-dummy, erule finite-imp-finite-component-Keys, erule dgrad-p-set-exhaust-expl)
end

### 13.2.5 Properties of the Reduced Gröbner Basis of an Ideal

## context gd-powerprod

begin
lemma ideal-eq-UNIV-iff-reduced-GB-eq-one-dgrad-p-set:
assumes dickson-grading $d$ and $F \subseteq$ punit.dgrad-p-set $d m$
shows ideal $F=U N I V \longleftrightarrow$ punit.reduced-GB $F=\{1\}$
proof -
have fin: finite (local.punit.component-of-term'Keys F) by simp
show ?thesis
proof
assume ideal $F=U N I V$
from $\operatorname{assms}(1)$ fin $\operatorname{assms}(2)$ show punit.reduced-GB $F=\{1\}$
proof (rule punit.reduced-GB-unique-dgrad-p-set)
show punit.is-reduced-GB \{1\} unfolding punit.is-reduced-GB-def
proof (intro conjI, fact punit.is-Groebner-basis-singleton)
show punit.is-auto-reduced $\{1\}$ unfolding punit.is-auto-reduced-def by (rule ballI, simp add: remove-def punit.not-is-red-empty) next
show punit.is-monic-set \{1\} by (rule punit.is-monic-setI, simp del: single-one add: single-one[symmetric]) qed $\operatorname{simp}$
next
have punit.pmdl $\{1\}=$ ideal $\{1\}$ by simp
also have...$=$ ideal $F$
proof (simp only: <ideal $F=$ UNIV $\rangle$ ideal-eq-UNIV-iff-contains-one)
have $1 \in\{1\}$..
with module-times show $1 \in$ ideal $\{1\}$ by (rule module.span-base)

```
        qed
        also have ... = punit.pmdl F by simp
        finally show punit.pmdl {1} = punit.pmdl F .
        qed
    next
        assume punit.reduced-GB F={1}
        hence 1 }\in\mathrm{ punit.reduced-GB F by simp
        hence 1 f punit.pmdl (punit.reduced-GB F) by (rule punit.pmdl.span-base)
    also from assms(1) fin assms(2) have ... = punit.pmdl F by (rule punit.reduced-GB-pmdl-dgrad-p-set)
    finally show ideal F = UNIV by (simp add: ideal-eq-UNIV-iff-contains-one)
    qed
qed
lemmas ideal-eq-UNIV-iff-reduced-GB-eq-one-finite =
    ideal-eq-UNIV-iff-reduced-GB-eq-one-dgrad-p-set[OF dickson-grading-dgrad-dummy
punit.dgrad-p-set-exhaust-expl]
end
```


### 13.2.6 Context od-term

context od-term
begin
lemmas ex-unique-reduced-GB=
ex-unique-reduced-GB-dgrad-p-set' $[$ OF dickson-grading-zero - subset-dgrad-p-set-zero]
lemmas finite-reduced-GB=
finite-reduced-GB-dgrad-p-set[OF dickson-grading-zero - subset-dgrad-p-set-zero]
lemmas reduced-GB-is-reduced-GB $=$
reduced-GB-is-reduced-GB-dgrad-p-set[OF dickson-grading-zero - subset-dgrad-p-set-zero]
lemmas reduced-GB-is-GB=
reduced-GB-is-GB-dgrad-p-set[OF dickson-grading-zero - subset-dgrad-p-set-zero]
lemmas reduced-GB-is-auto-reduced $=$
reduced-GB-is-auto-reduced-dgrad-p-set[OF dickson-grading-zero - subset-dgrad-p-set-zero]
lemmas reduced-GB-is-monic-set $=$
reduced-GB-is-monic-set-dgrad-p-set[OF dickson-grading-zero - subset-dgrad-p-set-zero]
lemmas reduced-GB-nonzero $=$
reduced-GB-nonzero-dgrad-p-set[OF dickson-grading-zero - subset-dgrad-p-set-zero]
lemmas reduced-GB-pmdl $=$
reduced-GB-pmdl-dgrad-p-set[OF dickson-grading-zero - subset-dgrad-p-set-zero]
lemmas reduced-GB-unique $=$
reduced-GB-unique-dgrad-p-set[OF dickson-grading-zero - subset-dgrad-p-set-zero]
end
end

## 14 Sample Computations of Reduced Gröbner Bases

```
theory Reduced-GB-Examples
    imports Buchberger Reduced-GB Polynomials.MPoly-Type-Class-OAlist Code-Target-Rat
begin
context gd-term
begin
definition rgb :: ('t =\mp@subsup{0}{0}{\prime}}\mp@subsup{}{}{\prime})\mathrm{ list }=>('t>\mp@subsup{=}{0}{\prime}'b::field) lis
    where rgb bs = comp-red-monic-basis (map fst (gb (map (\lambdab.(b, ())) bs)()))
definition rgb-punit :: (' }a=0\mp@subsup{|}{0}{\prime}b) list => (' ' a =00 'b::field) lis
    where rgb-punit bs = punit.comp-red-monic-basis (map fst (gb-punit (map ( }\lambdab\mathrm{ .
(b,()))bs)()))
lemma compute-trd-aux [code]:
    trd-aux fs p r =
        (if is-zero p then
            r
    else
            case find-adds fs (lt p) of
            None }=>\mathrm{ trd-aux fs (tail p) (plus-monomial-less r (lc p) (lt p))
            Some f = trd-aux fs (tail p - monom-mult (lc p / lc f) (lp p - lp f) (tail
f))}
    )
    by (simp only: trd-aux.simps[of fs p r] plus-monomial-less-def is-zero-def)
end
We only consider scalar polynomials here, but vector-polynomials could be handled, too.
global-interpretation punit \(^{\prime}\) : gd-powerprod ord-pp-punit cmp-term ord-pp-strict-punit cmp-term
rewrites punit.adds-term \(=(\) adds \()\)
and punit.pp-of-term \(=(\lambda x . x)\)
and punit.component-of-term \(=(\lambda-.())\)
and punit.monom-mult \(=\) monom-mult-punit
and punit.mult-scalar \(=\) mult-scalar-punit
and punit'.punit.min-term \(=\) min-term-punit
and punit'.punit.lt \(=l t-\) punit cmp-term
and punit'.punit.lc \(=l c\)-punit cmp-term
and punit'.punit.tail \(=\) tail-punit cmp-term
and punit' \({ }^{\text {.punit.ord }}\) - \(=\) ord- \(p\)-punit cmp-term
and punit'.punit.ord-strict- \(p=\) ord-strict-p-punit cmp-term
for cmp-term :: ('a::nat, 'b::\{nat,add-wellorder\}) pp nat-term-order
defines find-adds-punit \(=\) punit'. punit.find-adds
and trd-aux-punit \(=\) punit \({ }^{\prime} \cdot\) punit.trd-aux
```

and trd-punit $=$ punit ${ }^{\prime} \cdot$ punit.trd
and spoly-punit $=$ punit'. punit.spoly
and count-const-lt-components-punit $=$ punit ${ }^{\prime}$.punit.count-const-lt-components
and count-rem-components-punit $=$ punit' ${ }^{\prime}$.punit.count-rem-components
and const-lt-component-punit $=$ punit' ${ }^{\prime}$.punit.const-lt-component
and full-gb-punit $=$ punit' $\cdot$ punit.full-gb
and add-pairs-single-sorted-punit $=$ punit' ${ }^{\prime}$.punit.add-pairs-single-sorted
and add-pairs-punit $=$ punit' $\cdot$.punit.add-pairs
and canon-pair-order-aux-punit $=$ punit' ${ }^{\prime}$.punit.canon-pair-order-aux
and canon-basis-order-punit $=$ punit' ${ }^{\prime}$.punit.canon-basis-order
and new-pairs-sorted-punit $=$ punit' ${ }^{\prime}$.punit.new-pairs-sorted
and product-crit-punit $=$ punit' $\cdot$.punit.product-crit
and chain-ncrit-punit $=$ punit' ${ }^{\prime}$ punit.chain-ncrit
and chain-ocrit-punit $=$ punit'. .punit.chain-ocrit
and apply-icrit-punit $=$ punit' ${ }^{\prime}$.punit.apply-icrit
and apply-ncrit-punit $=$ punit ${ }^{\prime} \cdot$ punit.apply-ncrit
and apply-ocrit-punit $=$ punit ${ }^{\prime}$.punit.apply-ocrit
and $\operatorname{trdsp}$-punit $=$ punit' ${ }^{\prime}$.punit.trdsp
and gb-sel-punit $=$ punit ${ }^{\prime}$.punit.gb-sel
and gb-red-aux-punit $=$ punit' ${ }^{\prime}$.punit.gb-red-aux
and $g b$-red-punit $=$ punit' $\cdot$ punit.gb-red
and $g b$-aux-punit $=$ punit'.punit.gb-aux-punit

criterion.
and comp-min-basis-punit $=$ punit ${ }^{\prime} \cdot$ punit.comp-min-basis
and comp-red-basis-aux-punit $=$ punit' $\cdot$ punit.comp-red-basis-aux
and comp-red-basis-punit $=$ punit' $\cdot$.punit.comp-red-basis
and monic-punit $=$ punit' $\cdot$ punit.monic
and comp-red-monic-basis-punit $=$ punit'.punit.comp-red-monic-basis
and rgb-punit $=$ punit ${ }^{\prime}$.punit.rgb-punit
subgoal by (fact gd-powerprod-ord-pp-punit)
subgoal by (fact punit-adds-term)
subgoal by (simp add: id-def)
subgoal by (fact punit-component-of-term)
subgoal by (simp only: monom-mult-punit-def)
subgoal by (simp only: mult-scalar-punit-def)
subgoal using min-term-punit-def by fastforce
subgoal by (simp only: lt-punit-def ord-pp-punit-alt)
subgoal by (simp only: lc-punit-def ord-pp-punit-alt)
subgoal by (simp only: tail-punit-def ord-pp-punit-alt)
subgoal by (simp only: ord-p-punit-def ord-pp-strict-punit-alt)
subgoal by (simp only: ord-strict-p-punit-def ord-pp-strict-punit-alt)
done
lemma compute-spoly-punit [code]:
spoly-punit to $p q=($ let $t 1=l t$-punit to $p ; t 2=l t$-punit to $q ; l=l$ cs t1 t2 in (monom-mult-punit $(1 / l c-p u n i t ~ t o ~ p)(l-t 1) p)-($ monom-mult-punit $(1 / l c$-punit to $q)(l-t 2) q))$
by (simp add: punit'.punit.spoly-def Let-def punit'.punit.lc-def)
lemma compute-trd-punit [code]: trd-punit to fs $p=$ trd-aux-punit to fs $p$ (change-ord to 0)
by (simp only: punit'.punit.trd-def change-ord-def)
experiment begin interpretation trivariate ${ }_{0}$-rat .

## lemma

```
rgb-punit DRLEX
        [
        \(X^{\wedge} 3-X * Y * Z^{2}\),
        \(Y^{2} * Z-1\)
        \(]=\)
        [
        \(X^{\wedge} 3 * Y-X * Z\),
        \(-\left(X^{\wedge} 3\right)+X * Y * Z^{2}\),
        \(Y^{2} * Z-1\),
        \(-\left(X * Z^{\wedge} 3\right)+X^{\wedge} 5\)
        ]
    by eval
```

lemma

```
rgb-punit DRLEX
        \(X^{2}+Y^{2}+Z^{2}-1\),
        \(X * Y-Z-1\),
        \(Y^{2}+X\),
        \(Z^{2}+X\)
        ] \(=\)
    [
    ]
    by eval
```

Note: The above computations have been cross-checked with Mathematica 11.1.
end
end

## 15 Macaulay Matrices

```
theory Macaulay-Matrix
    imports More-MPoly-Type-Class Jordan-Normal-Form.Gauss-Jordan-Elimination
begin
```

We build upon vectors and matrices represented by dimension and characteristic function, because later on we need to quantify the dimensions of certain
matrices existentially. This is not possible (at least not easily possible) with a type-based approach, as in HOL-Multivariate Analysis.

### 15.1 More about Vectors

```
lemma vec-of-list-alt: vec-of-list xs = vec (length xs) (nth xs)
    by (transfer, rule refl)
lemma vec-cong:
    assumes n=m and \i. i<m\Longrightarrowfi=gi
    shows vec nf = vec mg
    using assms by auto
lemma scalar-prod-comm:
    assumes dim-vec v = dim-vec w
    shows v•w=w (v::'a::comm-semiring-0 vec)
    by (simp add: scalar-prod-def assms, rule sum.cong, rule refl, simp only: ac-simps)
lemma vec-scalar-mult-fun: vec n (\lambdax.c*fx)=c\cdotv vec n f
    by (simp add: smult-vec-def, rule vec-cong, rule refl, simp)
definition mult-vec-mat :: 'a vec = ' ' :: semiring-0 mat }=>\mp@subsup{}{}{\prime}\mp@subsup{}{}{\prime}a\mathrm{ vec (infixl }\mp@subsup{v}{*}{*}70
    where v v\mp@subsup{v}{}{*}}A\equiv\operatorname{vec}(dim-col A) (\lambdaj.v col A j)
definition resize-vec :: nat }=>\mathrm{ 'a vec }=>\mathrm{ 'a vec
    where resize-vec n v = vec n (vec-index v)
lemma dim-resize-vec[simp]: dim-vec (resize-vec n v) = n
    by (simp add: resize-vec-def)
lemma resize-vec-carrier: resize-vec n v\incarrier-vec n
    by (simp add: carrier-dim-vec)
lemma resize-vec-dim[simp]: resize-vec (dim-vec v) v=v
    by (simp add: resize-vec-def eq-vecI)
lemma resize-vec-index:
    assumes i<n
    shows resize-vec n v$i=v$i
    using assms by (simp add: resize-vec-def)
lemma mult-mat-vec-resize:
    v}\mp@subsup{v}{}{*}A=(\mathrm{ resize-vec (dim-row A) v) }\mp@subsup{v}{v}{*}
    by (simp add: mult-vec-mat-def scalar-prod-def, rule arg-cong2[of - . - vec],
rule, rule,
        rule sum.cong, rule, simp add: resize-vec-index)
lemma assoc-mult-vec-mat:
    assumes v\incarrier-vec n1 and A\incarrier-mat n1 n2 and B\incarrier-mat
```

```
n2 n3
    shows v}\mp@subsup{v}{v}{*}(A*B)=(\mp@subsup{v}{v*}{*}A)\mp@subsup{v}{v*}{*}
    using assms by (intro eq-vecI, auto simp add: mult-vec-mat-def mult-mat-vec-def
assoc-scalar-prod)
lemma mult-vec-mat-transpose:
    assumes dim-vec v = dim-row A
    shows v}\mp@subsup{v}{v}{*}A=(\mathrm{ transpose-mat A) *v (v::'a::comm-semiring-0 vec)
proof (simp add: mult-vec-mat-def mult-mat-vec-def, rule vec-cong, rule refl, simp)
    fix j
    show v•\operatorname{col A j = col A j •v by (rule scalar-prod-comm, simp add: assms)}
qed
```


### 15.2 More about Matrices

```
definition nzrows :: 'a::zero mat \(\Rightarrow\) 'a vec list
    where nzrows \(A=\) filter \(\left(\lambda r . r \neq O_{v}(\operatorname{dim}-c o l A)\right)(\) rows \(A)\)
definition row-space :: 'a mat \(\Rightarrow\) 'a::semiring-0 vec set
    where row-space \(A=(\lambda v\). mult-vec-mat \(v A)\) ' \((\) carrier-vec \((\operatorname{dim}-r o w ~ A))\)
definition row-echelon :: 'a mat \(\Rightarrow\) ' \(a:\) :field mat
    where row-echelon \(A=f s t\left(\right.\) gauss-jordan \(A\left(1_{m}(\right.\) dim-row \(\left.\left.A)\right)\right)\)
```


### 15.2.1 nzrows

lemma length-nzrows: length (nzrows $A$ ) $\leq$ dim-row $A$
by (simp add: nzrows-def length-rows[symmetric] del: length-rows)
lemma set-nzrows: set $($ nzrows $A)=$ set $($ rows $A)-\left\{0_{v}(\right.$ dim-col $\left.A)\right\}$
by (auto simp add: nzrows-def)
lemma nzrows-nth-not-zero:
assumes $i<$ length (nzrows A)
shows nzrows $A!i \neq 0_{v}(\operatorname{dim}-c o l ~ A)$
using assms unfolding nzrows-def using nth-mem by force

### 15.2.2 row-space

lemma row-spaceI:
assumes $x=v_{v^{*}} A$
shows $x \in$ row-space $A$
unfolding row-space-def assms by (rule, fact mult-mat-vec-resize, fact resize-vec-carrier)
lemma row-spaceE:
assumes $x \in$ row-space $A$
obtains $v$ where $v \in$ carrier-vec (dim-row $A$ ) and $x=v_{v^{*}} A$
using assms unfolding row-space-def by auto
lemma row-space-alt: row-space $A=$ range ( $\lambda v$. mult-vec-mat $v A$ )

```
proof
    show row-space A\subseteqrange (\lambdav. v \mp@subsup{v}{*}{*}A)\mathrm{ unfolding row-space-def by auto}
next
    show range (\lambdav. v }\mp@subsup{v}{*}{*}A)\subseteq\mathrm{ row-space }
    proof
        fix }
        assume x f range (\lambdav. v}\mp@subsup{v}{*}{*}A
        then obtain v}\mathrm{ where }x=\mp@subsup{v}{v}{*}A\mathrm{ ..
        thus }x\in\mathrm{ row-space A by (rule row-spaceI)
    qed
qed
lemma row-space-mult:
    assumes }A\in\mathrm{ carrier-mat nr nc and B}\in\mathrm{ carrier-mat nr nr
    shows row-space (B*A)\subseteq row-space }
proof
    from assms(2) assms(1) have B*A\incarrier-mat nr nc by (rule mult-carrier-mat)
    hence nr = dim-row (B*A) by blast
    fix }
    assume x frow-space ( }B*A
    then obtain v}\mathrm{ where v
        unfolding <nr = dim-row ( }B*A)\rangle\mathrm{ by (rule row-spaceE)
    from this(1) assms(2) assms(1) have }x=(\mp@subsup{v}{v*}{*}B)\mp@subsup{v}{v}{*}A\mathrm{ unfolding }x\mathrm{ by (rule
assoc-mult-vec-mat)
    thus }x\in\mathrm{ row-space A by (rule row-spaceI)
qed
lemma row-space-mult-unit:
    assumes P\inUnits (ring-mat TYPE('a::semiring-1) (dim-row A) b)
    shows row-space ( }P*A)=\mathrm{ row-space }
proof -
    have A:A\in carrier-mat (dim-row A) (dim-col A) by simp
    from assms have P:P\incarrier (ring-mat TYPE('a) (dim-row A) b) and
        *: \existsQ\in(carrier (ring-mat TYPE('a) (dim-row A) b)).
            Q \otimes ring-mat TYPE('a) (dim-row A) b P = 1 ring-mat TYPE('a) (dim-row A) b
        unfolding Units-def by auto
    from P have P-in: P carrier-mat (dim-row A) (dim-row A) by (simp add:
ring-mat-def)
    from * obtain Q where Q carrier (ring-mat TYPE('a) (dim-row A) b)
    and Q * ring-mat TYPE('a) (dim-row A)b}P=\mp@subsup{\mathbf{1}}{\mathrm{ ring-mat TYPE('a) (dim-row A) b}}{\mathrm{ ( }
    hence Q-in: Q\in carrier-mat (dim-row A) (dim-row A) and QP:Q*P=1m
(dim-row A)
    by (simp-all add: ring-mat-def)
    show ?thesis
    proof
    from A P-in show row-space ( }P*A)\subseteq\mathrm{ row-space A by (rule row-space-mult)
    next
            from A P-in Q-in have Q*(P*A)=(Q*P)*A by (simp only: as-
```

soc-mult-mat)
also from $A$ have $\ldots=A$ by $(\operatorname{simp}$ add: $Q P)$
finally have eq: row-space $A=$ row-space $(Q *(P * A))$ by simp
show row-space $A \subseteq$ row-space $(P * A)$ unfolding eq by (rule row-space-mult, rule mult-carrier-mat, fact+)
qed
qed

### 15.2.3 row-echelon

lemma row-eq-zero-iff-pivot-fun:
assumes pivot-fun $A f($ dim-col $A)$ and $i<$ dim-row ( $A::^{\prime}$ a::zero-neq-one mat)
shows $\left(\right.$ row $\left.A i=O_{v}(\operatorname{dim}-\operatorname{col} A)\right) \longleftrightarrow(f i=\operatorname{dim}-\operatorname{col} A)$
proof -
have $*$ : dim-row $A=\operatorname{dim}$-row $A .$.
show ?thesis
proof
assume $a$ : row $A i=O_{v}(\operatorname{dim}-\operatorname{col} A)$
show $f i=\operatorname{dim}-\operatorname{col} A$
proof (rule ccontr)
assume $f i \neq \operatorname{dim}$-col $A$
with pivot-funD $(1)[O F *$ assms $]$ have $* *: f i<\operatorname{dim}-c o l ~ A$ by simp
with $*$ assms have $A \$ \$(i, f i)=1$ by (rule pivot-funD)
with $* * \operatorname{assms}(2)$ have row $A i \$(f i)=1$ by simp
hence $\left(1::^{\prime} a\right)=\left(0_{v}(\operatorname{dim}-c o l A)\right) \$(f i)$ by (simp only: a)
also have $\ldots=\left(0::^{\prime} a\right)$ using $* *$ by simp
finally show False by simp
qed
next
assume $a$ : $f i=\operatorname{dim}-\operatorname{col} A$
show row $A i=0_{v}(\operatorname{dim}-\operatorname{col} A)$
proof (rule, simp-all add: assms(2))
fix $j$
assume $j<d i m$-col $A$
hence $j<f i$ by (simp only: a)
with $*$ assms show $A \$ \$(i, j)=0$ by (rule pivot-funD)
qed
qed
qed
lemma row-not-zero-iff-pivot-fun:
assumes pivot-fun $A f(\operatorname{dim-col} A)$ and $i<d i m-r o w\left(A::^{\prime} a:: z e r o-n e q-o n e ~ m a t\right)$
shows $\left(\right.$ row $\left.A i \neq O_{v}(\operatorname{dim}-\operatorname{col} A)\right) \longleftrightarrow(f i<\operatorname{dim}-\operatorname{col} A)$
proof (simp only: row-eq-zero-iff-pivot-fun[OF assms])
have $f i \leq \operatorname{dim}$-col $A$ by (rule pivot-fun $D[$ where ? $f=f]$, rule refl, fact + )
thus $(f i \neq \operatorname{dim}-\operatorname{col} A)=(f i<\operatorname{dim}-\operatorname{col} A)$ by auto
qed
lemma pivot-fun-stabilizes:
assumes pivot-fun $A f n c$ and $i 1 \leq i 2$ and $i 2<d i m-r o w ~ A$ and $n c \leq f i 1$ shows $f i 2=n c$
proof -
from $\operatorname{assms}(2)$ have $i 2=i 1+(i 2-i 1)$ by $\operatorname{simp}$
then obtain $k$ where $i 2=i 1+k$..
from $\operatorname{assms}(3) \operatorname{assms}(4)$ show ?thesis unfolding $\langle i 2=i 1+k\rangle$
proof (induct $k$ arbitrary: i1)
case 0
from this(1) have $i 1<$ dim-row $A$ by simp
from - assms(1) this have $f$ i1 $\leq n c$ by (rule pivot-funD, intro refl)
with $\langle n c \leq f$ i1〉 show? case by simp
next
case (Suc $k$ )
from $\operatorname{Suc}(2)$ have $S u c(i 1+k)<$ dim-row $A$ by simp
hence Suc i1 $+k<$ dim-row $A$ by simp
hence Suc i1 < dim-row $A$ by simp
hence $i 1<$ dim-row $A$ by simp
have $n c \leq f$ (Suc i1)
proof -
have $f i 1<f(S u c i 1) \vee f(S u c i 1)=n c$ by (rule pivot-funD, rule refl, fact+)
with $\operatorname{Suc}(3)$ show ?thesis by auto
qed
with $\langle$ Suc $i 1+k<$ dim-row $A\rangle$ have $f($ Suc $i 1+k)=n c$ by (rule Suc(1))
thus? ?ase by simp
qed
qed
lemma pivot-fun-mono-strict:
assumes pivot-fun $A f n c$ and $i 1<i 2$ and $i 2<\operatorname{dim}$-row $A$ and $f i 1<n c$ shows $f i 1<f i 2$
proof -
from $\operatorname{assms}(2)$ have $i 2-i 1 \neq 0$ and $i 2=i 1+(i 2-i 1)$ by simp-all
then obtain $k$ where $k \neq 0$ and $i 2=i 1+k$..
from this(1) assms(3) assms(4) show ?thesis unfolding $\langle i 2=i 1+k\rangle$ proof (induct $k$ arbitrary: i1)
case 0
thus ?case by simp
next
case (Suc k)
from $\operatorname{Suc}(3)$ have $\operatorname{Suc}(i 1+k)<$ dim-row $A$ by simp
hence Suc i1 $+k<\operatorname{dim}$-row $A$ by simp
hence Suc i1 <dim-row $A$ by simp
hence $i 1<$ dim-row $A$ by simp
have $*$ : $f$ i1 $<f$ (Suc i1)
proof -
have $f i 1<f(S u c ~ i 1) \vee f(S u c ~ i 1)=n c$ by (rule pivot-funD, rule refl, fact+)
with $\operatorname{Suc}(4)$ show ?thesis by auto

```
    qed
    show ?case
    proof (simp, cases k=0)
        case True
        show f i1<f(Suc (i1 + k)) by (simp add: True *)
    next
        case False
        have f(Suc i1) \leqf(Suc i1 +k)
        proof (cases f (Suc i1)<nc)
            case True
            from False «Suc i1 + k< dim-row A`True have f (Suc i1) < f (Suc i1
+ k) by (rule Suc(1))
            thus?thesis by simp
        next
            case False
            hence nc \leqf (Suc i1) by simp
            from assms(1)-\langleSuc i1 +k<dim-row A> this have f(Suc i1 + k)=nc
                by (rule pivot-fun-stabilizes[where ?f =f], simp)
            moreover have f(Suc i1)=nc by (rule pivot-fun-stabilizes[where ? f=f],
fact, rule le-refl, fact+)
            ultimately show ?thesis by simp
            qed
            also have ... =f (i1 + Suc k) by simp
            finally have f(Suc i1)\leqf(i1 + Suc k).
            with * show fi1<f(Suc (i1 + k)) by simp
        qed
    qed
qed
lemma pivot-fun-mono:
    assumes pivot-fun A f nc and i1 \leqi2 and i2 < dim-row A
    shows fi1 \leq f i2
proof -
    from assms(2) have i1<i2 \vee i1 = i2 by auto
    thus ?thesis
    proof
        assume i1<i2
        show ?thesis
        proof (cases f i1 < nc)
            case True
        from assms(1)<i1 < i2`assms(3) this have fi1<fiQ by (rule pivot-fun-mono-strict)
            thus ?thesis by simp
        next
            case False
            hence nc\leqfi1 by simp
            from assms(1) - this have f i1 = nc
            proof (rule pivot-fun-stabilizes[where ?f=f], simp)
                    from assms(2) assms(3) show i1 < dim-row A by (rule le-less-trans)
            qed
```

```
            moreover have fi2 = nc by (rule pivot-fun-stabilizes[where ?f=f], fact+)
            ultimately show ?thesis by simp
        qed
    next
        assume i1 = i2
        thus ?thesis by simp
    qed
qed
lemma row-echelon-carrier:
    assumes A\in carrier-mat nr nc
    shows row-echelon A \in carrier-mat nr nc
proof -
    from assms have dim-row A = nr by simp
    let ?B = 1m}(\mathrm{ dim-row }A
    note assms
    moreover have ? B \in carrier-mat nr nr by (simp add: {dim-row A = nr`)
    moreover from surj-pair obtain }\mp@subsup{A}{}{\prime}\mp@subsup{B}{}{\prime}\mathrm{ where *: gauss-jordan A ? B = ( A', B')
by metis
    ultimately have }\mp@subsup{A}{}{\prime}\in\mathrm{ carrier-mat nr nc by (rule gauss-jordan-carrier)
    thus ?thesis by (simp add: row-echelon-def *)
qed
lemma dim-row-echelon[simp]:
    shows dim-row (row-echelon A) = dim-row A and dim-col (row-echelon A) =
dim-col A
proof -
    have }A\in\mathrm{ carrier-mat (dim-row A) (dim-col A) by simp
    hence row-echelon A\incarrier-mat (dim-row A) (dim-col A) by (rule row-echelon-carrier)
    thus dim-row (row-echelon A) = dim-row A and dim-col (row-echelon A)}
dim-col A by simp-all
qed
lemma row-echelon-transform:
    obtains P where P\inUnits (ring-mat TYPE('a::field) (dim-row A) b) and
row-echelon A = P*A
proof -
    let ?B = 1m (dim-row }A
    have }A\in\mathrm{ carrier-mat (dim-row A) (dim-col A) by simp
    moreover have ? B \in carrier-mat (dim-row A) (dim-row A) by simp
    moreover from surj-pair obtain }\mp@subsup{A}{}{\prime}\mp@subsup{B}{}{\prime}\mathrm{ where *: gauss-jordan A ? B = ( }\mp@subsup{A}{}{\prime},\mp@subsup{B}{}{\prime}
by metis
    ultimately have \existsP\inUnits (ring-mat TYPE('a) (dim-row A)b). A' = P*A
    A B'=P* ? B
    by (rule gauss-jordan-transform)
    then obtain P where P E Units (ring-mat TYPE ('a) (dim-row A) b) and **:
A'}=P*A\wedge\mp@subsup{B}{}{\prime}=P*?B.
    from this(1) show ?thesis
    proof
```

```
    from ** have }\mp@subsup{A}{}{\prime}=P*A.
    thus row-echelon A = P*A by (simp add:row-echelon-def *)
    qed
qed
lemma row-space-row-echelon[simp]: row-space (row-echelon A) = row-space A
proof -
    obtain P where *: P U Units (ring-mat TYPE('a::field) (dim-row A) Nil) and
**: row-echelon A=P*A
    by (rule row-echelon-transform)
    from * have row-space ( }P*A)=\mathrm{ row-space A by (rule row-space-mult-unit)
    thus ?thesis by (simp only: **)
qed
lemma row-echelon-pivot-fun:
    obtains f}\mathrm{ where pivot-fun (row-echelon A) f(dim-col (row-echelon A))
proof -
    let ?B = 1m}(\mathrm{ dim-row }A
    have }A\in\mathrm{ carrier-mat (dim-row A) (dim-col A) by simp
    moreover from surj-pair obtain }\mp@subsup{A}{}{\prime}\mp@subsup{B}{}{\prime}\mathrm{ where *: gauss-jordan A ? B = ( }\mp@subsup{A}{}{\prime},\mp@subsup{B}{}{\prime}
by metis
    ultimately have row-echelon-form A' by (rule gauss-jordan-row-echelon)
    then obtain f}\mathrm{ where pivot-fun }\mp@subsup{A}{}{\prime}f(\mathrm{ dim-col }\mp@subsup{A}{}{\prime})\mathrm{ unfolding row-echelon-form-def
    hence pivot-fun (row-echelon A) f (dim-col (row-echelon A)) by (simp add:
row-echelon-def *)
    thus ?thesis ..
qed
lemma distinct-nzrows-row-echelon: distinct (nzrows (row-echelon A))
    unfolding nzrows-def
proof (rule distinct-filterI, simp del: dim-row-echelon)
    let ?B = row-echelon A
    fix i j::nat
    assume }i<j\mathrm{ and j<dim-row? B
    hence }i\not=j\mathrm{ and }i<dim-row ?B by simp-all
    assume ri: row ?B i\not= Ov (dim-col ?B) and rj: row ?B j\not= 0v (dim-col ?B)
    obtain f}\mathrm{ where pf: pivot-fun ?B f (dim-col ?B) by (fact row-echelon-pivot-fun)
    from rj have fj< dim-col ?B by (simp only: row-not-zero-iff-pivot-fun[OF pf
<j<dim-row ?B`])
    from - pf \langlej< dim-row ?B\rangle this <i<dim-row ? B\rangle\langlei\not=j\rangle have *: ?B $$ (i,f
j) =0
    by (rule pivot-funD(5), intro refl)
    show row ?B i\not= row ?B j
    proof
        assume row ?B i= row ?B j
    hence row ?B i$(fj)= row ?B j$(fj) by simp
    with <i<dim-row ?B\rangle\langlej< dim-row ?B\rangle\langlef j<dim-col ?B> have ?B $$ (i,f
j)=?B$$(j,fj) by simp
```

```
    also from - pf <j < dim-row ?B\rangle\langlef j<dim-col ?B\rangle have ... = 1 by (rule
pivot-funD, intro refl)
    finally show False by (simp add:*)
    qed
qed
```


### 15.3 Converting Between Polynomials and Macaulay Matrices

```
definition poly-to-row :: 'a list }=>('a\mp@subsup{#}{0}{\prime}'b::zero) ) = 'b vec where
```

definition poly-to-row :: 'a list }=>('a\mp@subsup{\#}{0}{\prime}'b::zero) ) = 'b vec where
poly-to-row ts p = vec-of-list (map (lookup p) ts)
definition polys-to-mat :: 'a list }=>('a\mp@subsup{=>}{0}{\prime}'b::zero) list => 'b mat where
polys-to-mat ts ps = mat-of-rows (length ts) (map (poly-to-row ts) ps)
definition list-to-fun :: 'a list }=>\mathrm{ ('b::zero) list }=>\mp@subsup{}{}{\prime}a=>\mp@subsup{}{}{\prime}'b\mathrm{ where
list-to-fun ts cs t=(case map-of (zip ts cs) t of Some c=>c|None = 0)
definition list-to-poly :: 'a list }=>\mp@subsup{}{}{\prime}b\mathrm{ list }=>('a=\mp@subsup{|}{0}{\prime}'b::zero) where
list-to-poly ts cs = Abs-poly-mapping (list-to-fun ts cs)
definition row-to-poly :: 'a list }=>\mp@subsup{'}{}{\prime}b\mathrm{ vec }=>('a=\mp@subsup{|}{0}{\prime}'b::zero) where
row-to-poly ts r = list-to-poly ts (list-of-vec r)
definition mat-to-polys :: 'a list }=>\mp@subsup{'}{}{\prime}b\mathrm{ mat }=>('a=\mp@subsup{\#}{0}{\prime}'b::zero) list wher
mat-to-polys ts A = map (row-to-poly ts) (rows A)
lemma dim-poly-to-row: dim-vec (poly-to-row ts p)= length ts
by (simp add: poly-to-row-def)
lemma poly-to-row-index:
assumes i< length ts
shows poly-to-row ts p$i= lookup p (ts!i)
    by (simp add: poly-to-row-def vec-of-list-index assms)
context term-powerprod
begin
lemma poly-to-row-scalar-mult:
    assumes keys p\subseteq set ts
    shows row-to-poly ts (c\cdotv}(\mathrm{ poly-to-row ts p)) =c cp
proof -
    have eq: (vec (length ts) (\lambdai. c * poly-to-row ts p$ i))=
(vec (length ts) (\lambdai.c * lookup p (ts!i)))
by (rule vec-cong, rule, simp only: poly-to-row-index)
have *: list-to-fun ts (list-of-vec (c c}v(\mathrm{ (poly-to-row ts p))) = ( }\lambdat.c*\mathrm{ cookup p t)
proof (rule, simp add: list-to-fun-def smult-vec-def dim-poly-to-row eq,
simp add: map-upt[of \lambdax.c* lookup p x] map-of-zip-map, rule)
fix }

```
```

    assume t\not\in set ts
    with assms(1) have t & keys p by auto
    thus c*lookup pt=0 by (simp add: in-keys-iff)
    qed
    have **: lookup (Abs-poly-mapping (list-to-fun ts (list-of-vec ( c vv (poly-to-row ts
    p))))) =
(\lambdat.c* lookup pt)
proof (simp only: *, rule Abs-poly-mapping-inverse, simp, rule finite-subset, rule,
simp)
fix }
assume c* lookup pt\not=0
hence lookup p t\not=0 using mult-not-zero by blast
thus t\in keys p by (simp add: in-keys-iff)
qed (fact finite-keys)
show ?thesis unfolding row-to-poly-def
by (rule poly-mapping-eqI) (simp only:list-to-poly-def ** lookup-map-scale)
qed
lemma poly-to-row-to-poly:
assumes keys p\subseteq set ts
shows row-to-poly ts (poly-to-row ts p) = (p::'t =\mp@subsup{0}{0}{\prime}}\mp@subsup{}{}{\prime}b::semiring-1
proof -
have 1 v
thus ?thesis using poly-to-row-scalar-mult[OF assms, of 1] by simp
qed
lemma lookup-list-to-poly: lookup (list-to-poly ts cs) = list-to-fun ts cs
unfolding list-to-poly-def
proof (rule Abs-poly-mapping-inverse, rule, rule finite-subset)
show {x. list-to-fun ts cs }x\not=0}\subseteq\mathrm{ set ts
proof (rule, simp)
fix }
assume list-to-fun ts cs t\not=0
then obtain c where map-of (zip ts cs) t=Some c unfolding list-to-fun-def
by fastforce
thus t\in set ts by (meson in-set-zipE map-of-SomeD)
qed
qed simp
lemma list-to-fun-Nil [simp]: list-to-fun [] cs = 0
by (simp only: zero-fun-def, rule, simp add: list-to-fun-def)
lemma list-to-poly-Nil [simp]: list-to-poly [] cs = 0
by (rule poly-mapping-eqI, simp add: lookup-list-to-poly)
lemma row-to-poly-Nil [simp]: row-to-poly [] r=0
by (simp only: row-to-poly-def, fact list-to-poly-Nil)
lemma lookup-row-to-poly:

```
```

    assumes distinct ts and dim-vec r = length ts and i< length ts
    shows lookup (row-to-poly ts r) (ts!i) =r $i
    proof (simp only: row-to-poly-def lookup-list-to-poly)
from assms(2) assms(3) have i<dim-vec r by simp
have map-of (zip ts (list-of-vec r)) (ts!i)=Some ((list-of-vec r)!i)
by (rule map-of-zip-nth, simp-all only: length-list-of-vec assms(2), fact, fact)
also have ... = Some (r \$ i) by (simp only:list-of-vec-index)
finally show list-to-fun ts (list-of-vec r) (ts !i)=r\$ i by (simp add: list-to-fun-def)
qed
lemma keys-row-to-poly: keys (row-to-poly ts r)\subseteq set ts
proof
fix }
assume t\in keys (row-to-poly ts r)
hence lookup (row-to-poly ts r) t = 0 by (simp add: in-keys-iff)
thus t\in set ts
proof (simp add: row-to-poly-def lookup-list-to-poly list-to-fun-def del: lookup-not-eq-zero-eq-in-keys
split:option.splits)
fix }
assume map-of (zip ts (list-of-vec r)) t=Some c
thus t\in set ts by (meson in-set-zipE map-of-SomeD)
qed
qed
lemma lookup-row-to-poly-not-zeroE:
assumes lookup (row-to-poly ts r) t\not=0
obtains i}\mathrm{ where }i<length ts and t=ts!
proof -
from assms have t\inkeys (row-to-poly ts r) by (simp add: in-keys-iff)
have}t\in\mathrm{ set ts by (rule, fact, fact keys-row-to-poly)
then obtain i where i< length ts and t=ts!i by (metis in-set-conv-nth)
thus ?thesis ..
qed
lemma row-to-poly-zero [simp]: row-to-poly ts ( }0v(\mathrm{ length ts ) ) = (0::'t 盾'b::zero)
proof -
have eq: map ( }\lambda\mathrm{ -. 0::'b) [0..<length ts] = map ( }\lambda\mathrm{ -. 0) ts by (simp add: map-replicate-const)
show ?thesis
by (simp add: row-to-poly-def zero-vec-def, rule poly-mapping-eqI,
simp add: lookup-list-to-poly list-to-fun-def eq map-of-zip-map)
qed
lemma row-to-poly-zeroD:
assumes distinct ts and dim-vec r = length ts and row-to-poly ts r = 0
shows }r=\mp@subsup{O}{v}{}(\mathrm{ length ts)
proof (rule, simp-all add: assms(2))
fix }
assume i< length ts
from assms(3) have 0 = lookup (row-to-poly ts r) (ts!i) by simp

```
```

    also from assms(1) assms(2) <i< length ts` have ... =r $ i by (rule lookup-row-to-poly)
    finally show r $ i=0 by simp
    qed
lemma row-to-poly-inj:
assumes distinct ts and dim-vec r1 = length ts and dim-vec r2 = length ts
and row-to-poly ts r1 = row-to-poly ts r2
shows r1 = r2
proof (rule, simp-all add: assms(2) assms(3))
fix i
assume i< length ts
have r1 \$ i = lookup (row-to-poly ts r1) (ts ! i)
by (simp only: lookup-row-to-poly[OF assms(1) assms(2)<i<length ts>])
also from assms(4) have ... = lookup (row-to-poly ts r2) (ts!i) by simp
also from assms(1) assms(3) <i < length ts` have ... = r2 $ i by (rule lookup-row-to-poly)     finally show r1 $ i=r2 $ i. qed lemma row-to-poly-vec-plus:     assumes distinct ts and length ts = n     shows row-to-poly ts (vec n (f1 + f2)) = row-to-poly ts (vec n f1) + row-to-poly ts (vec n f2) proof (rule poly-mapping-eqI)     fix }     show lookup (row-to-poly ts (vec n (f1 + f2))) t=                 lookup (row-to-poly ts (vec n f1) + row-to-poly ts (vec n f2)) t     (is lookup ?l t = lookup (?r1 + ?r2) t)     proof (cases t\in set ts)     case True     then obtain j where j:j< length ts and t:t=ts!j by (metis in-set-conv-nth)     have d1: dim-vec (vec n f1) = length ts and d2:dim-vec (vec n f2) = length ts         and da: dim-vec (vec n (f1 + f2)) = length ts by (simp-all add: assms(2))     from j have j': j< n by (simp only:assms(2))     show ?thesis             by (simp only: t lookup-add lookup-row-to-poly[OF assms(1) d1 j]                     lookup-row-to-poly[OF assms(1) d2 j] lookup-row-to-poly[OF assms(1) da j] index-vec[OF j],                 simp only: plus-fun-def)     next     case False     with keys-row-to-poly[of ts vec n (f1 + f2)] keys-row-to-poly[of ts vec n f1]                 keys-row-to-poly[of ts vec n f2] have t & keys ?l and t & keys ?r1 and t\not\in keys ?r2                 by auto     from this(2) this(3) have t & keys (?r1 + ?r2)             by (meson Poly-Mapping.keys-add UnE in-mono)     with «t & keys ?l` show ?thesis by (simp add: in-keys-iff)
qed

```

\section*{qed}
lemma row-to-poly-vec-sum:
assumes distinct ts and length \(t s=n\)
shows row-to-poly ts (vec \(\left.n\left(\lambda j . \sum i \in I . f i j\right)\right)=\left(\left(\sum i \in I\right.\right.\). row-to-poly ts (vec \(n\) \((f i))):: ' t \Rightarrow_{0}\) 'b::comm-monoid-add)
proof (cases finite I)
case True
thus ?thesis
proof (induct I)
case empty
thus ?case by (simp add: zero-vec-def[symmetric] assms(2)[symmetric])
next
case (insert \(x\) I)
have row-to-poly ts (vec \(n\left(\lambda j\right.\). \(\sum i \in\) insert \(\left.\left.x I . f i j\right)\right)=\) row-to-poly ts (vec \(n\) \(\left.\left(\lambda j . f x j+\left(\sum i \in I . f i j\right)\right)\right)\)
by (simp add: insert(1) insert(2))
also have \(\ldots=\) row-to-poly ts (vec \(n\left(f x+\left(\lambda j .\left(\sum i \in I . f i j\right)\right)\right)\) ) by (simp only: plus-fun-def)
also from assms have \(\ldots=\) row-to-poly ts (vec \(n(f x))+\) row-to-poly ts (vec \(\left.n\left(\lambda j .\left(\sum i \in I . f i j\right)\right)\right)\)
by (rule row-to-poly-vec-plus)
also have \(\ldots=\) row-to-poly ts (vec \(n(f x))+\left(\sum i \in I\right.\). row-to-poly ts (vec \(n(f\) i)))
by ( \(\operatorname{simp}\) only: \(\operatorname{insert(3))}\)
also have \(\ldots=\left(\sum i \in\right.\) insert \(x\) I. row-to-poly ts \((\) vec \(\left.n(f i))\right)\)
by (simp add: insert(1) insert(2))
finally show ?case .
qed
next
case False
thus ?thesis by (simp add: zero-vec-def[symmetric] assms(2)[symmetric])
qed
lemma row-to-poly-smult:
assumes distinct ts and dim-vec \(r=\) length \(t s\)
shows row-to-poly ts \((c \cdot v r)=c \cdot(\) row-to-poly ts \(r)\)
proof (rule poly-mapping-eqI, simp only: lookup-map-scale)
fix \(t\)
show lookup (row-to-poly ts \(\left.\left(c \cdot{ }_{v} r\right)\right) t=c *\) lookup (row-to-poly ts \(r\) ) \(t\) (is lookup ?l \(t=c *\) lookup ? \(r t)\)
proof (cases \(t \in\) set ts)
case True
then obtain \(j\) where \(j: j<l e n g t h ~ t s ~ a n d ~ t: t=t s!j\) by (metis in-set-conv-nth)
from assms(2) have dm: dim-vec \((c \cdot v r)=\) length ts by simp
from \(j\) have \(j^{\prime}: j<\) dim-vec \(r\) by (simp only: assms(2))
show ?thesis
by (simp add: t lookup-row-to-poly[OF assms j] lookup-row-to-poly[OF assms(1) dm j] index-smult-vec (1)[OF j])
```

    next
        case False
        with keys-row-to-poly[of ts c \cdotv r] keys-row-to-poly[of ts r] have
        t\not\in keys ?l and t}\not\inkeys ?r by aut
    thus ?thesis by (simp add: in-keys-iff)
    qed
    qed
lemma poly-to-row-Nil [simp]: poly-to-row [] p = vec 0f
proof -
have dim-vec (poly-to-row [] p)=0 by (simp add: dim-poly-to-row)
thus ?thesis by auto
qed
lemma polys-to-mat-Nil [simp]: polys-to-mat ts [] = mat 0 (length ts) f
by (simp add: polys-to-mat-def mat-eq-iff)
lemma dim-row-polys-to-mat[simp]: dim-row (polys-to-mat ts ps)= length ps
by (simp add: polys-to-mat-def)
lemma dim-col-polys-to-mat[simp]: dim-col (polys-to-mat ts ps) = length ts
by (simp add: polys-to-mat-def)
lemma polys-to-mat-index:
assumes i< length ps and j< length ts

    shows (polys-to-mat ts ps) $$ (i,j) = lookup (ps!i) (ts!j)
    by (simp add: polys-to-mat-def index-mat(1)[OF assms] mat-of-rows-def nth-map[OF
    assms(1)],
rule poly-to-row-index, fact)
lemma row-polys-to-mat:
assumes i< length ps
shows row (polys-to-mat ts ps) i = poly-to-row ts (ps!i)
proof -
have row (polys-to-mat ts ps) i = (map (poly-to-row ts) ps)!i unfolding
polys-to-mat-def
proof (rule mat-of-rows-row)
from assms show i< length (map (poly-to-row ts) ps) by simp
next
show map (poly-to-row ts) ps!i\in carrier-vec (length ts) unfolding nth-map[OF
assms]
by (rule carrier-vecI, fact dim-poly-to-row)
qed
also from assms have ... = poly-to-row ts (ps!i) by (rule nth-map)
finally show ?thesis.
qed
lemma col-polys-to-mat:
assumes j< length ts

```
```

    shows col (polys-to-mat ts ps) j = vec-of-list (map (\lambdap.lookup p (ts!j)) ps)
    by (simp add: vec-of-list-alt col-def, rule vec-cong, rule refl, simp add: polys-to-mat-index
    assms)
lemma length-mat-to-polys[simp]: length (mat-to-polys ts A) = dim-row A
by (simp add: mat-to-polys-def mat-to-list-def)
lemma mat-to-polys-nth:
assumes }i<\mathrm{ dim-row A
shows (mat-to-polys ts A)!i= row-to-poly ts (row A i)
proof -
from assms have i< length (rows A) by (simp only:length-rows)
thus?thesis by (simp add: mat-to-polys-def)
qed
lemma Keys-mat-to-polys: Keys (set (mat-to-polys ts A))\subseteq set ts
proof
fix }
assume t\inKeys (set (mat-to-polys ts A))
then obtain p where p\in set (mat-to-polys ts A) and t: t\in keys p by (rule
in-KeysE)
from this(1) obtain i where i< length (mat-to-polys ts A) and p: p=
(mat-to-polys ts A)!i
by (metis in-set-conv-nth)
from this(1) have i< dim-row A by simp
with p have p= row-to-poly ts (row A i) by (simp only: mat-to-polys-nth)
with t have t\in keys (row-to-poly ts (row A i)) by simp
also have .. \subseteq set ts by (fact keys-row-to-poly)
finally show }t\in\mathrm{ set ts.
qed
lemma polys-to-mat-to-polys:
assumes Keys (set ps)\subseteq set ts
shows mat-to-polys ts (polys-to-mat ts ps) = (ps::('t 质 'b::semiring-1) list)
unfolding mat-to-polys-def mat-to-list-def
proof (rule nth-equalityI, simp-all)
fix }
assume i< length ps
have *: keys (ps!i)\subseteq set ts
using <i < length ps` assms keys-subset-Keys nth-mem by blast     show row-to-poly ts (row (polys-to-mat ts ps) i)=ps!i     by (simp only: row-polys-to-mat[OF<i< length ps`] poly-to-row-to-poly[OF *])
qed
lemma mat-to-polys-to-mat:
assumes distinct ts and length ts = dim-col A
shows (polys-to-mat ts (mat-to-polys ts A)) = A
proof
fix ij

```
assume \(i\) ：\(i<d i m\)－row \(A\) and \(j: j<\operatorname{dim}-\operatorname{col} A\)
hence \(i^{\prime}: i<\) length（mat－to－polys ts \(A\) ）and \(j^{\prime}: j<\) length ts by（simp，simp only：assms（2））
have \(r\) ：dim－vec（row \(A i)=\) length \(t s\) by（simp add：assms（2））
show polys－to－mat ts（mat－to－polys ts A）\＄\＄\((i, j)=A \$ \$(i, j)\)
by（simp only：polys－to－mat－index［OF \(\left.i^{\prime} j^{\prime}\right]\) mat－to－polys－nth \([O F<i<\) dim－row A〉］
lookup－row－to－poly［OF assms（1）rij index－row（1）［OFij］）
qed（simp－all add：assms）

\section*{15．4 Properties of Macaulay Matrices}

\section*{lemma row－to－poly－vec－times：}

\section*{assumes distinct ts and length ts \(=\) dim－col \(A\)}
shows row－to－poly ts \(\left(v_{v^{*}} A\right)=\left(\left(\sum i=0 . .<\right.\right.\) dim－row \(A .(v \$ i) \cdot(\) row－to－poly ts （row \(A\) ）））：：＇\(t \Rightarrow_{0}{ }^{\prime} b::\) comm－semiring－ 0 ）
proof（simp add：mult－vec－mat－def scalar－prod－def row－to－poly－vec－sum［OF assms］， rule sum．cong，rule）
fix \(i\)
assume \(i \in\{0 . .<\) dim－row \(A\}\)
hence \(i<\) dim－row \(A\) by simp
have dim－vec（row \(A i)=\) length \(t s\) by（simp add：assms（2））
have \(*: \operatorname{vec}(\operatorname{dim}-\operatorname{col} A)(\lambda j\) ．col \(A j \$ i)=\operatorname{vec}(\operatorname{dim}-\operatorname{col} A)(\lambda j . A \$ \$(i, j))\)
by（rule vec－cong，rule refl，simp add：\(\langle i<\operatorname{dim}\)－row \(A\rangle\) ）
have vec \((\operatorname{dim}-c o l A)(\lambda j . v \$ i * \operatorname{col} A j \$ i)=v \$ i \cdot v \operatorname{vec}(\operatorname{dim}-\operatorname{col} A)(\lambda j . \operatorname{col}\) A \(j \$ i)\)
by（simp only：vec－scalar－mult－fun）
also have \(\ldots=v \$ i \cdot v(\) row \(A i)\) by（simp only：＊row－def［symmetric］）
finally show row－to－poly ts（vec（dim－col \(A)(\lambda j . v \$ i * \operatorname{col} A j \$ i))=\) （ \(v \$ i) \cdot(\) row－to－poly ts（row \(A i)\) ）
by（simp add：row－to－poly－smult［OF assms（1）〈dim－vec（row A i）\(=\) length ts〉］） qed
lemma vec－times－polys－to－mat：
assumes Keys \((\) set \(p s) \subseteq\) set ts and \(v \in\) carrier－vec（length ps）
shows row－to－poly ts \(\left(v_{v^{*}}\right.\)（polys－to－mat ts ps））\(=\left(\sum(c, p) \leftarrow z i p\right.\)（list－of－vec \(\left.v\right)\) ps．\(c \cdot p\) ）
（is ？l \(=? r\) ）
proof－
from assms have \(*\) ：dim－vec \(v=\) length ps by（simp only：carrier－dim－vec）
have eq：map（ \(\lambda i . v \cdot \operatorname{col}(\) polys－to－mat ts ps）\(i)[0 . .<\) length \(t s]=\) \(\operatorname{map}(\lambda s . v \cdot(v e c-o f-l i s t(\operatorname{map}(\lambda p\) ．lookup p s）ps \())\) ）ts
proof（rule nth－equalityI，simp－all）
fix \(i\)
assume \(i<\) length \(t s\)
hence col（polys－to－mat ts ps）\(i=\) vec－of－list（map \((\lambda p\) ．lookup \(p(t s!i)) p s)\)
by（rule col－polys－to－mat）
thus \(v \cdot \operatorname{col}(\) polys－to－mat ts ps）\(i=v \cdot \operatorname{map-vec}(\lambda p\) ．lookup \(p(t s!i))(v e c-o f-l i s t\) \(p s)\)
```

    by simp
    qed
    show ?thesis
    proof (rule poly-mapping-eqI, simp add: mult-vec-mat-def row-to-poly-def lookup-list-to-poly
eq list-to-fun-def map-of-zip-map lookup-sum-list o-def, intro conjI impI)
fix }
assume t\in set ts
have v | vec-of-list (map ( }\lambda\mathrm{ p. lookup pt) ps)=
( }\sum(c,p)\leftarrowzip (list-of-vec v) ps.lookup (c\cdotp)t
proof (simp add: scalar-prod-def vec-of-list-index)
have (\sumi=0..<length ps.v \$ i*lookup (ps!i)t)=
(\sumi=0..<length ps. (list-of-vec v)!i*lookup (ps!i)t)
by (rule sum.cong, rule refl, simp add:*)
also have ... = (\sum(c,p)\leftarrowzip (list-of-vec v) ps.c*lookup p t)
by (simp only: sum-set-upt-eq-sum-list, rule sum-list-upt-zip, simp only:
length-list-of-vec *)
finally show ( \sumi=0..<length ps.v \$ i* lookup (ps!i)t)=
(\sum(c,p)\leftarrowzip (list-of-vec v) ps.c*lookup pt).
qed
thus v • map-vec ( }\lambda\mathrm{ p.lookup p t) (vec-of-list ps)=
(\sumx\leftarrowzip (list-of-vec v) ps.lookup (case x of (c,x) =>c\cdotx)t)
by (metis (mono-tags, lifting) case-prod-conv cond-case-prod-eta vec-of-list-map)
next
fix }
assume t\not\in set ts
with assms(1) have t\not\in Keys (set ps) by auto
have }(\sum(c,p)\leftarrowzip (list-of-vec v) ps.lookup (c\cdotp)t)=
proof (rule sum-list-zeroI, rule, simp)
fix }
assume x ( ( (c,p).c* lookup pt)'set (zip (list-of-vec v) ps)
then obtain c p where cp:(c,p)\in set (zip (list-of-vec v) ps)
and x:x=c* lookup p t by auto
from cp have p\in set ps by (rule set-zip-rightD)
with <t \not\in Keys (set ps)\rangle have t\not\in keys p by (auto intro: in-KeysI)
thus }x=0\mathrm{ by (simp add: x in-keys-iff)
qed
thus (\sumx\leftarrowzip (list-of-vec v) ps. lookup (case x of (c,x)=>c}\=x)t)=
by (metis (mono-tags, lifting) case-prod-conv cond-case-prod-eta)
qed
qed
lemma row-space-subset-phull:
assumes Keys (set ps)\subseteq set ts
shows row-to-poly ts 'row-space (polys-to-mat ts ps)\subseteq phull (set ps)
(is ?r }\subseteq\mathrm{ ? ?h)
proof
fix q
assume q\in?r
then obtain x where x1:x\in row-space (polys-to-mat ts ps)

```
and q1: \(q=\) row-to-poly ts \(x\)..
from x1 obtain \(v\) where \(v: v \in\) carrier-vec (dim-row (polys-to-mat ts ps)) and \(x: x=v_{v} *\) polys-to-mat ts ps
by (rule row-spaceE)
from \(v\) have \(v \in\) carrier-vec (length ps) by (simp only: dim-row-polys-to-mat)
thm vec-times-polys-to-mat
with \(x q 1\) have \(q: q=\left(\sum(c, p) \leftarrow\right.\) zip (list-of-vec \(\left.v\right)\) ps. \(\left.c \cdot p\right)\)
by (simp add: vec-times-polys-to-mat[OF assms])
show \(q \in\) ? \(h\) unfolding \(q\) by (rule phull.span-listI)
qed
lemma phull-subset-row-space:
assumes Keys \((\) set ps) \(\subseteq\) set ts
shows phull (set ps) \(\subseteq\) row-to-poly ts 'row-space (polys-to-mat ts ps)
(is ? \(h \subseteq ? r\) )
proof
fix \(q\)
assume \(q \in ? h\)
then obtain \(c s\) where \(l\) : length \(c s=\) length \(p s\) and \(q: q=\left(\sum(c, p) \leftarrow z i p\right.\) cs \(p s\).
\(c \cdot p\) )
by (rule phull.span-listE)
let \(? v=v e c-o f-l i s t ~ c s\)
from \(l\) have \(*: ? v \in\) carrier-vec (length \(p s\) ) by (simp only: carrier-dim-vec dim-vec-of-list)
let \(? q=? v_{v^{*}}\) polys-to-mat ts ps
show \(q \in\) ? \(r\)
proof
show \(q\) = row-to-poly ts ? \(q\)
by (simp add: vec-times-polys-to-mat [OF assms *] q list-vec)
next
show ?q \(\in\) row-space (polys-to-mat ts ps) by (rule row-spaceI, rule)
qed
qed
lemma row-space-eq-phull:
assumes Keys \((\) set \(p s) \subseteq\) set ts
shows row-to-poly ts ' row-space (polys-to-mat ts ps) = phull (set ps)
by (rule, rule row-space-subset-phull, fact, rule phull-subset-row-space, fact)
lemma row-space-row-echelon-eq-phull:
assumes Keys \((\) set ps) \(\subseteq\) set ts
shows row-to-poly ts 'row-space (row-echelon (polys-to-mat ts ps)) \(=\) phull (set \(p s)\)
by (simp add: row-space-eq-phull[OF assms])
lemma phull-row-echelon:
assumes Keys \((\) set ps) \(\subseteq\) set ts and distinct ts
shows phull (set (mat-to-polys ts (row-echelon (polys-to-mat ts ps)))) = phull (set ps)
```

proof -
have len-ts:length ts = dim-col (row-echelon (polys-to-mat ts ps)) by simp
have *: Keys (set (mat-to-polys ts (row-echelon (polys-to-mat ts ps)))) \subseteq set ts
by (fact Keys-mat-to-polys)
show ?thesis
by (simp only: row-space-eq-phull[OF *, symmetric] mat-to-polys-to-mat[OF
assms(2) len-ts],
rule row-space-row-echelon-eq-phull, fact)
qed
lemma pmdl-row-echelon:
assumes Keys (set ps)\subseteq set ts and distinct ts
shows pmdl (set (mat-to-polys ts (row-echelon (polys-to-mat ts ps)))) = pmdl
(set ps)
(is ?l = ?r)
proof
show ?l \subseteq?r
by (rule pmdl.span-subset-spanI, rule subset-trans, rule phull.span-superset,
simp only: phull-row-echelon[OF assms] phull-subset-module)
next
show ?r \subseteq?l
by (rule pmdl.span-subset-spanI, rule subset-trans, rule phull.span-superset,
simp only: phull-row-echelon[OF assms, symmetric] phull-subset-module)
qed
end
context ordered-term
begin
lemma lt-row-to-poly-pivot-fun:
assumes card S = dim-col (A::'b::semiring-1 mat) and pivot-fun A f (dim-col
A)
and i<dim-row A and fi<dim-col A
shows lt ((mat-to-polys (pps-to-list S)A)!i)=(pps-to-list S)!(fi)
proof -
let ?ts = pps-to-list S
have len-ts:length ?ts = dim-col A by (simp add: length-pps-to-list assms(1))
show ?thesis
proof (simp add: mat-to-polys-nth[OF assms(3)], rule lt-eqI)
have lookup (row-to-poly ?ts (row A i)) (?ts!fi)=(row A i)\$ (f i)
by (rule lookup-row-to-poly, fact distinct-pps-to-list, simp-all add: len-ts
assms(4))

    also have ... = A $$ (i,fi) using assms(3) assms(4) by simp
    also have ... = 1 by (rule pivot-funD, rule refl, fact+)
    finally show lookup (row-to-poly ?ts (row A i)) (?ts ! fi)\not=0 by simp
    next
    fix u
    assume a: lookup (row-to-poly ?ts (row A i)) u\not=0
    ```
```

    then obtain j where j:j< length ?ts and u:u=?ts!j
    by (rule lookup-row-to-poly-not-zeroE)
    from j have j<card S and j<dim-col A by (simp only:length-pps-to-list,
    simp only:len-ts)
from a have 0 flookup (row-to-poly ?ts (row A i)) (?ts ! j) by (simp add: u)
also have lookup (row-to-poly ?ts (row A i)) (?ts!j)=(row A i) \$ j
by (rule lookup-row-to-poly, fact distinct-pps-to-list, simp add: len-ts, fact)

    finally have A$$(i,j)\not=0 using assms(3)<j<dim-col A> by simp
    from - <j< card S> show }u\mp@subsup{\preceq}{t}{}\mathrm{ ?ts ! fi unfolding u
    proof (rule pps-to-list-nth-leI)
        show fi\leqj
        proof (rule ccontr)
            assume }\negfi\leq
            hence j<fi by simp
            have A$$ (i,j) = 0 by (rule pivot-funD, rule refl, fact+)
            with}<A$$(i,j)\not=0\rangle\mathrm{ show False ..
        qed
    qed
    qed
    qed
lemma lc-row-to-poly-pivot-fun:
assumes card S = dim-col (A::'b::semiring-1 mat) and pivot-fun A f (dim-col
A)
and }i<dim-row A and fi<dim-col A
shows lc ((mat-to-polys (pps-to-list S)A)!i)=1
proof -
let ?ts = pps-to-list S
have len-ts:length ?ts = dim-col A by (simp only:length-pps-to-list assms(1))
have lookup (row-to-poly ?ts (row A i)) (?ts!fi)=(row A i) \$ (f i)
by (rule lookup-row-to-poly, fact distinct-pps-to-list, simp-all add: len-ts assms(4))

    also have ... =A $$ (i,fi) using assms(3) assms(4) by simp
    finally have eq: lookup (row-to-poly ?ts (row A i)) (?ts !f i)=A$$ (i,fi).
    show ?thesis
    by (simp only:lc-def lt-row-to-poly-pivot-fun[OF assms], simp only: mat-to-polys-nth[OF
    assms(3)] eq,
rule pivot-funD, rule refl, fact+)
qed
lemma lt-row-to-poly-pivot-fun-less:
assumes card $S=\operatorname{dim}$-col ( $A::$ 'b::semiring-1 mat) and pivot-fun $A f$ (dim-col A)
and $i 1<i 2$ and $i 2<\operatorname{dim}-r o w A$ and $f i 1<\operatorname{dim}-c o l A$ and $f i 2<\operatorname{dim}-c o l ~ A$ shows (pps-to-list $S)!(f i 2) \prec_{t}(p p s$-to-list $S)!(f i 1)$
proof -
let ?ts $=p p s$-to-list $S$
have len-ts: length ?ts $=$ dim-col $A$ by (simp add: length-pps-to-list assms(1))
from $\operatorname{assms}(3) \operatorname{assms}(4)$ have $i 1<\operatorname{dim}$-row $A$ by simp
show ?thesis

```
by (rule pps-to-list-nth-lessI, rule pivot-fun-mono-strict \([\) where ? \(f=f]\), fact, fact, fact, fact, \(\operatorname{simp}\) only: \(\operatorname{assms}(1) \operatorname{assms}(6))\)
qed
lemma \(l t\)-row-to-poly-pivot-fun-eqD:
assumes card \(S=\) dim-col ( \(A::\) 'b::semiring-1 mat) and pivot-fun \(A f\) (dim-col A)
and \(i 1<\operatorname{dim}\)-row \(A\) and \(i 2<\operatorname{dim}\)-row \(A\) and \(f i 1<\operatorname{dim}-\operatorname{col} A\) and \(f i 2<\) dim-col \(A\)
and \((\) pps-to-list \(S)!(f i 1)=(\) pps-to-list \(S)!(f i 2)\)
shows \(i 1=i 2\)
proof (rule linorder-cases)
assume \(i 1<i 2\)
from \(\operatorname{assms}(1) \operatorname{assms}(2)\) this assms(4) assms(5) \(\operatorname{assms}(6)\) have
(pps-to-list \(S\) ) ! (fi2) \(\prec_{t}\) (pps-to-list \(S\) ) ! (f i1) by (rule lt-row-to-poly-pivot-fun-less)
with assms(7) show ?thesis by auto

\section*{next}
assume \(i 2<i 1\)
from \(\operatorname{assms}(1) \operatorname{assms}(2)\) this assms(3) assms(6) assms(5) have
(pps-to-list \(S)!(f i 1) \prec_{t}(\) pps-to-list \(S)!(f i 2)\) by (rule lt-row-to-poly-pivot-fun-less)
with assms(7) show ?thesis by auto

\section*{qed}
lemma lt-row-to-poly-pivot-in-keysD:
assumes card \(S=\) dim-col ( \(A::^{\prime} b::\) semiring-1 mat) and pivot-fun \(A f\) (dim-col A)
and \(i 1<\) dim-row \(A\) and \(i 2<d i m\)-row \(A\) and \(f i 1<\operatorname{dim}-c o l A\)
and (pps-to-list \(S)!(f\) i1 \() \in\) keys \(((\) mat-to-polys \((\) pps-to-list \(S) A)!i 2)\)
shows \(i 1=i 2\)
proof (rule ccontr)
assume \(i 1 \neq i 2\)
hence \(i 2 \neq i 1\) by \(\operatorname{simp}\)
let ?ts \(=p p s\)-to-list \(S\)
have len-ts: length ?ts \(=\) dim-col \(A\) by (simp only: length-pps-to-list assms(1))
from \(\operatorname{assms}(6)\) have \(0 \neq\) lookup (row-to-poly ?ts (row A i2)) (?ts ! (f i1))
by (auto simp: mat-to-polys-nth[OF assms(4)])
also have lookup (row-to-poly ?ts (row A i2) ) (?ts ! (fi1)) \(=(\) row A i2) \(\$(f\) i1)
by (rule lookup-row-to-poly, fact distinct-pps-to-list, simp-all add: len-ts assms(5))
finally have \(A \$ \$(i 2, f i 1) \neq 0\) using \(\operatorname{assms}(4) \operatorname{assms}(5)\) by simp
moreover have \(A \$ \$(\) i2, fi1) \(=0\) by (rule pivot-funD(5), rule refl, fact + )
ultimately show False ..
qed
lemma lt-row-space-pivot-fun:
assumes card \(S=\) dim-col ( \(A::\) 'b:: \{comm-semiring- 0, semiring-1-no-zero-divisors \} mat)
and pivot-fun \(A f(\operatorname{dim}-c o l A)\) and \(p \in\) row-to-poly (pps-to-list \(S)\) 'row-space \(A\) and \(p \neq 0\)
```

    shows lt p \inlt-set (set (mat-to-polys (pps-to-list S) A))
    proof -
let ?ts = pps-to-list S
let ?I = {0..<dim-row A}
have len-ts:length ?ts = dim-col A by (simp add: length-pps-to-list assms(1))
from assms(3) obtain x where x fow-space A and p: p = row-to-poly?ts x
from this(1) obtain v}\mathrm{ where }v\in\operatorname{carrier-vec (dim-row A) and x:x= v}\mp@subsup{v}{*}{*}
by (rule row-spaceE)
have p': p=(\sumi\in?I. (v \$ i) \cdot(row-to-poly ?ts (row A i)))
unfolding px by (rule row-to-poly-vec-times, fact distinct-pps-to-list, fact
len-ts)
have lt (\sumi=0..<dim-row A. (v\$i) · (row-to-poly ?ts (row A i)))
|lt-set ((\lambdai. (v \$ i) · (row-to-poly ?ts (row A i)))'{0..<dim-row A})
proof (rule lt-sum-distinct-in-lt-set, rule, simp add: p'[symmetric] <p}\not=0\rangle
fix i1 i2
let ?p1 = (v \$ i1)}\cdot(\mathrm{ row-to-poly ?ts (row A i1))
let ?p2 = (v \$ i2) \cdot(row-to-poly ?ts (row A i2))
assume i1 \in?I and i2 \in?I
hence i1 < dim-row A and i2 < dim-row A by simp-all
assume ?p1 }\not=
hence v\$ i1 \not=0 and row-to-poly ?ts (row A i1) }=0\mathrm{ by auto
hence row A i1 }\not=\mp@subsup{0}{v}{}\mathrm{ (length ?ts) by auto
hence f i1 < dim-col A
by (simp add:len-ts row-not-zero-iff-pivot-fun[OF assms(2)<i1 < dim-row
A>])
have lt ?p1 = lt (row-to-poly ?ts (row A i1)) by (rule lt-map-scale, fact)
also have ... =lt ((mat-to-polys ?ts A)! i1) by (simp only: mat-to-polys-nth[OF
<i < dim-row A>])
also have ... =?ts ! (f i1) by (rule lt-row-to-poly-pivot-fun, fact+)
finally have lt1:lt ?p1 = ?ts ! (f i1).
assume ?p2 }=
hence v\$ i2 \not=0 and row-to-poly ?ts (row A i2) }=0\mathrm{ by auto
hence row A i2 }=\mp@subsup{0}{v}{}\mathrm{ (length ?ts) by auto
hence f i2 < dim-col A
by (simp add: len-ts row-not-zero-iff-pivot-fun[OF assms(2)<i2 < dim-row
A>])
have lt ?p2 = lt (row-to-poly ?ts (row A i2)) by (rule lt-map-scale, fact)
also have ... = lt ((mat-to-polys ?ts A)!i2) by (simp only: mat-to-polys-nth[OF
<i2 < dim-row A〉])
also have ... = ?ts ! (f i2) by (rule lt-row-to-poly-pivot-fun, fact+)
finally have lt2:lt ?p2 = ?ts ! (f i2) .
assume lt ?p1 = lt ?p2
with assms(1) assms(2)<i1 < dim-row A〉\langlei2 < dim-row A〉\langlef i1 < dim-col

```
```

A\rangle\langlef iQ < dim-col A>

```
    show \(i 1=\) i2 unfolding lt1 lt2 by (rule lt-row-to-poly-pivot-fun-eqD)
    qed
    also have \(\ldots \subseteq l t\)-set \(((\lambda i\). row-to-poly ?ts \((\) row \(A i))\) ' \(\{0 . .<\) dim-row \(A\})\)
    proof
    fix \(s\)
        assume \(s \in l\)-set \(((\lambda i .(v \$ i) \cdot(\) row-to-poly ?ts \((\) row \(A i)))\) ' \(\{0 . .<\) dim-row
A\})
    then obtain \(f\)
            where \(f \in(\lambda i .(v \$ i) \cdot(\) row-to-poly?ts (row \(A i)))\) ' \(\{0 . .<\operatorname{dim}\)-row \(A\}\)
            and \(f \neq 0\) and \(l t f=s\) by (rule lt-setE)
    from this(1) obtain \(i\) where \(i \in\{0 . .<\) dim-row \(A\}\)
            and \(f: f=(v \$ i) \cdot(\) row-to-poly ?ts (row A i)) ..
    from this(2) \(\langle f \neq 0\rangle\) have \(v \$ i \neq 0\) and \(* *\) : row-to-poly ?ts (row \(A\) ) \(\neq 0\)
by auto
    from \(\langle l t f=s\rangle\) have \(s=l t((v \$ i) \cdot(\) row-to-poly ?ts \((\) row \(A i)))\) by \((\) simp
only: f)
            also from \(\langle v \$ i \neq 0\rangle\) have \(\ldots=l t\) (row-to-poly ?ts (row \(A i\) )) by (rule
lt-map-scale)
    finally have \(s: s=l t\) (row-to-poly ?ts (row A i)).
    show \(s \in l\)-set \(((\lambda i\). row-to-poly ?ts (row \(A i))\) ' \(\{0 . .<\) dim-row \(A\})\)
                unfolding \(s\) by (rule lt-setI, rule, rule refl, fact+)
    qed
    also have \(\ldots=l\)-set \(((\lambda r\). row-to-poly ?ts \(r)\) ' (row \(A \cdot\{0 . .<\) dim-row \(A\}))\)
    by (simp only: image-comp o-def)
    also have \(\ldots=l\)-set (set (map ( \(\lambda\) r. row-to-poly ?ts r) (map (row \(A\) ) \([0 . .<\) dim-row
A])))
    by (metis image-set set-upt)
    also have \(\ldots=\) lt-set (set (mat-to-polys ?ts A)) by (simp only: mat-to-polys-def
rows-def)
    finally show ?thesis unfolding \(p^{\prime}\).
qed

\subsection*{15.5 Functions Macaulay-mat and Macaulay-list}
```

definition Macaulay-mat :: ('t $\left.\Rightarrow_{0}{ }^{\prime} b\right)$ list $\Rightarrow{ }^{\prime} b::$ field mat
where Macaulay-mat $p s=$ polys-to-mat (Keys-to-list ps) ps

```
definition Macaulay-list :: (' \(\left.t \nRightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\left({ }^{\prime} t \nRightarrow_{0}{ }^{\prime} b::\right.\) field \()\) list
    where Macaulay-list \(p s=\)
            filter \((\lambda p . p \neq 0)\) (mat-to-polys (Keys-to-list ps) (row-echelon
(Macaulay-mat ps)))
lemma dim-Macaulay-mat[simp]:
    dim-row \((\) Macaulay-mat ps) \(=\) length \(p s\)
    dim-col \((\) Macaulay-mat ps \()=\operatorname{card}(\) Keys \((\) set ps \())\)
    by (simp-all add: Macaulay-mat-def length-Keys-to-list)
lemma Macaulay-list-Nil [simp]: Macaulay-list []\(=\left([]::\left(' t \Rightarrow_{0}{ }^{\prime} b::\right.\right.\) field) list) (is ?l
```

= -)
proof -
have length ?l \leq length (mat-to-polys (Keys-to-list ([]::('t =00'b) list))
(row-echelon (Macaulay-mat ([]::('t =\mp@subsup{0}{0}{\prime}}\mp@subsup{}{}{\prime}b) list)))
unfolding Macaulay-list-def by (fact length-filter-le)
also have ... = 0 by simp
finally show ?thesis by simp
qed
lemma set-Macaulay-list:
set (Macaulay-list ps)=
set (mat-to-polys (Keys-to-list ps)(row-echelon (Macaulay-mat ps))) - {0}
by (auto simp add: Macaulay-list-def)
lemma Keys-Macaulay-list: Keys (set (Macaulay-list ps))\subseteq Keys (set ps)
proof -
have Keys (set (Macaulay-list ps))\subseteq set (Keys-to-list ps)
by (simp only: set-Macaulay-list Keys-minus-zero, fact Keys-mat-to-polys)
also have ... = Keys (set ps) by (fact set-Keys-to-list)
finally show ?thesis .
qed
lemma in-Macaulay-listE:
assumes p\in set (Macaulay-list ps)
and pivot-fun (row-echelon (Macaulay-mat ps))f(dim-col (row-echelon (Macaulay-mat
ps)))
obtains i where i<dim-row (row-echelon (Macaulay-mat ps))
and p=(mat-to-polys (Keys-to-list ps)(row-echelon (Macaulay-mat ps)))!i
and fi<dim-col (row-echelon (Macaulay-mat ps))
proof -
let ?ts = Keys-to-list ps
let ?A = Macaulay-mat ps
let ?E = row-echelon ?A
from assms(1) have p\in set (mat-to-polys ?ts ?E) - {0} by (simp add:
set-Macaulay-list)
hence p\in set (mat-to-polys ?ts ?E) and p\not=0 by auto
from this(1) obtain i where i< length (mat-to-polys ?ts ?E) and p: p=
(mat-to-polys ?ts ?E)!i
by (metis in-set-conv-nth)
from this(1) have i< dim-row ?E and i<dim-row?A by simp-all
from this(1) p show ?thesis
proof
from \langlep\not=0\rangle have 0}\not=(\mathrm{ mat-to-polys ?ts ?E) ! i by (simp only: p)
also have (mat-to-polys ?ts ?E)!i row-to-poly ?ts (row ?E i)
by (simp only:Macaulay-list-def mat-to-polys-nth[OF <i<dim-row ?E>])
finally have *: row-to-poly ?ts (row ?E i)}\not=0\mathrm{ by simp
have row ?E i\not= 0v (length ?ts)

```
```

    proof
        assume row ?E i= Ov (length ?ts)
        with * show False by simp
    qed
    hence row ?E i\not= Ov (dim-col ?E) by (simp add: length-Keys-to-list)
    thus fi< dim-col ?E
        by (simp only: row-not-zero-iff-pivot-fun[OF assms(2)<i<dim-row ?E>])
    qed
    qed
lemma phull-Macaulay-list: phull (set (Macaulay-list ps)) = phull (set ps)
proof -
have *: Keys (set ps)\subseteq set (Keys-to-list ps)
by (simp add: set-Keys-to-list)
have phull (set (Macaulay-list ps)) =
phull (set (mat-to-polys (Keys-to-list ps) (row-echelon (Macaulay-mat ps))))
by (simp only: set-Macaulay-list phull.span-Diff-zero)
also have ... = phull (set ps)
by (simp only: Macaulay-mat-def phull-row-echelon[OF * distinct-Keys-to-list])
finally show ?thesis.
qed
lemma pmdl-Macaulay-list: pmdl (set (Macaulay-list ps)) = pmdl (set ps)
proof -
have *:Keys (set ps)\subseteqset (Keys-to-list ps)
by (simp add: set-Keys-to-list)
have pmdl (set (Macaulay-list ps)) =
pmdl (set (mat-to-polys (Keys-to-list ps) (row-echelon (Macaulay-mat ps))))
by (simp only: set-Macaulay-list pmdl.span-Diff-zero)
also have ... = pmdl (set ps)
by (simp only: Macaulay-mat-def pmdl-row-echelon[OF * distinct-Keys-to-list])
finally show ?thesis .
qed
lemma Macaulay-list-is-monic-set: is-monic-set (set (Macaulay-list ps))
proof (rule is-monic-setI)
let ?ts = Keys-to-list ps
let ?E = row-echelon (Macaulay-mat ps)
fix p
assume p\in set (Macaulay-list ps)
obtain h where pivot-fun ?E h (dim-col ?E) by (rule row-echelon-pivot-fun)
with}\langlep\in\operatorname{set (Macaulay-list ps)\rangle obtain i where i< dim-row ?E
and p:p=(mat-to-polys ?ts ?E)!i and hi< dim-col ?E
by (rule in-Macaulay-listE)
show lc p = 1 unfolding p Keys-to-list-eq-pps-to-list
by (rule lc-row-to-poly-pivot-fun, simp, fact+)
qed

```
```

lemma Macaulay-list-not-zero: 0 \& set (Macaulay-list ps)
by (simp add: Macaulay-list-def)
lemma Macaulay-list-distinct-lt:
assumes }x\in\operatorname{set}(Macaulay-list ps) and y set (Macaulay-list ps
and }x\not=
shows lt x\not=lt y
proof
let ?S = Keys (set ps)
let ?ts = Keys-to-list ps
let ?E = row-echelon (Macaulay-mat ps)
assume lt x = lt y
obtain h}\mathrm{ where pf: pivot-fun ?E h (dim-col ?E) by (rule row-echelon-pivot-fun)
with assms(1) obtain i1 where i1 < dim-row ?E
and x: x = (mat-to-polys ?ts ?E)!i1 and h i1 < dim-col ?E
by (rule in-Macaulay-listE)
from assms(2) pf obtain i2 where i2 < dim-row ? E
and y:y=(mat-to-polys ?ts ?E)!i2 and hi2 < dim-col ?E
by (rule in-Macaulay-listE)
have lt x = ?ts !(h i1)
by (simp only: x Keys-to-list-eq-pps-to-list, rule lt-row-to-poly-pivot-fun, simp,
fact+)
moreover have lt y = ?ts ! (h i2)
by (simp only: y Keys-to-list-eq-pps-to-list, rule lt-row-to-poly-pivot-fun, simp,
fact+)
ultimately have ?ts ! (h i1) = ?ts ! (h iQ) by (simp only: <lt x = lt y>)
hence pps-to-list (Keys (set ps))! h i1 = pps-to-list (Keys (set ps))!h i2
by (simp only:Keys-to-list-eq-pps-to-list)
have i1 = i2
proof (rule lt-row-to-poly-pivot-fun-eqD)
show card ?S = dim-col ?E by simp
qed fact+
hence }x=y\mathrm{ by (simp only: x y)
with }\langlex\not=y\rangle\mathrm{ show False ..
qed
lemma Macaulay-list-lt:
assumes p\in phull (set ps) and p\not=0
obtains g}\mathrm{ where g}\operatorname{get (Macaulay-list ps) and g\not=0 and lt p=lt g
proof -
let ?S = Keys (set ps)
let ?ts = Keys-to-list ps
let ?E = row-echelon (Macaulay-mat ps)
let ?gs = mat-to-polys ?ts ?E
have finite?S by (rule finite-Keys, rule)

```
```

have ?S $\subseteq$ set ?ts by (simp only: set-Keys-to-list)
from $\operatorname{assms}(1)\langle ? S \subseteq$ set ?ts〉 have $p \in$ row-to-poly?ts' row-space ?E
by (simp only: Macaulay-mat-def row-space-row-echelon-eq-phull[symmetric])
hence $p \in$ row-to-poly (pps-to-list ?S) 'row-space ? E
by (simp only: Keys-to-list-eq-pps-to-list)
obtain $f$ where pivot-fun ?E $f($ dim-col ?E) by (rule row-echelon-pivot-fun)
have lt $p \in l t$-set (set ?gs) unfolding Keys-to-list-eq-pps-to-list
by (rule lt-row-space-pivot-fun, simp, fact+)
then obtain $g$ where $g \in$ set ?gs and $g \neq 0$ and lt $g=l t p$ by (rule lt-setE)
show ?thesis
proof
from $\langle g \in$ set $? g s\rangle\langle g \neq 0\rangle$ show $g \in$ set (Macaulay-list ps) by (simp add:
set-Macaulay-list)
next
from $\langle l t g=l t p\rangle$ show $l t p=l t g$ by $\operatorname{simp}$
qed fact
qed
end
end

```

\section*{16 Faugère's F4 Algorithm}

\author{
theory \(F_{4}\) \\ imports Macaulay-Matrix Algorithm-Schema \\ begin
}

This theory implements Faugère's F4 algorithm based on \(g d\)-term.gb-schema-direct.

\subsection*{16.1 Symbolic Preprocessing}
context gd-term
begin
definition sym-preproc-aux-term1 :: ('a \(\Rightarrow\) nat \() \Rightarrow\left(\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right.\right.\) list \(\times{ }^{\prime} t\) list \(\times{ }^{\prime} t\) list \(\times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \() \times\)
\[
\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \text { list } \times{ }^{\prime} t \text { list } \times{ }^{\prime} t \text { list } \times\left({ }^{\prime} t \Rightarrow_{0}\right.\right.
\]
'b) list)) set
where sym-preproc-aux-term1 \(d=\)
\(\left\{\left((g s 1, k s 1, t s 1, f s 1),\left(g s 2::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right.\right.\right.\) list, ks2, ts2, fs2)). \(\exists\) t2 \(\in\) set ts2.
\(\forall t 1 \in\) set ts1. t1 \(\left.\prec_{t} t 2\right\}\)
definition sym-preproc-aux-term2 :: \(\left({ }^{\prime} a \Rightarrow\right.\) nat \() \Rightarrow\left(\left({ }^{\prime} t \Rightarrow{ }_{0}{ }^{\prime} b::\right.\right.\) zero \()\) list \(\times\) 't list \(\times\) 't list \(\times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \() \times\)
\[
\left(\left({ }^{\prime} t \Rightarrow_{0}^{\prime} b\right) \text { list } \times{ }^{\prime} t \text { list } \times{ }^{\prime} t \text { list } \times\left({ }^{\prime} t \Rightarrow_{0}\right.\right.
\]
'b) list)) set
where sym-preproc-aux-term2 \(d=\)
\(\left\{\left((g s 1, k s 1, t s 1, f s 1),\left(g s 2::\left(' t \Rightarrow_{0}{ }^{\prime} b\right)\right.\right.\right.\) list, ks2, ts2, fs2)\() . g s 1=g s 2 \wedge\) dgrad-set-le d (pp-of-term'set ts1) (pp-of-term
' (Keys \((\) set gs2) \(\cup\) set ts2 \())\}\)
definition sym-preproc-aux-term
where sym-preproc-aux-term \(d=\) sym-preproc-aux-term1 \(d \cap\) sym-preproc-aux-term2 d
lemma wfp-on-ord-term-strict:
assumes dickson-grading d
shows wfp-on \(\left(\prec_{t}\right)\) (pp-of-term -'dgrad-set d m)
proof (rule wfp-onI-min)
fix \(x Q\)
assume \(x \in Q\) and \(Q \subseteq p p\)-of-term - ' dgrad-set \(d m\)
from wf-dickson-less-v \([\) OF assms, of \(m]\langle x \in Q\rangle\) obtain \(z\)
where \(z \in Q\) and \(*: \bigwedge y\). dickson-less-v \(d m y z \Longrightarrow y \notin Q\) by (rule wfE-min[to-pred], blast)
from this \((1)\langle Q \subseteq p p\)-of-term -'dgrad-set \(d m>\) have \(z \in p p\)-of-term -'dgrad-set
d \(m\)..
show \(\exists z \in Q . \forall y \in p p\)-of-term - \({ }^{\prime}\) dgrad-set \(d m . y \prec_{t} z \longrightarrow y \notin Q\)
proof (intro bexI ballI impI, rule *)
fix \(y\)
assume \(y \in p p\)-of-term -' dgrad-set \(d m\) and \(y \prec_{t} z\)
from this \((1)\langle z \in p p\)-of-term \(-‘ d g r a d\)-set \(d m\rangle\) have \(d(p p\)-of-term \(y) \leq m\)
and \(d(p p\)-of-term \(z) \leq m\)
by (simp-all add: dgrad-set-def)
thus dickson-less-v \(d m y z\) using \(\left\langle y \prec_{t} z\right\rangle\) by (rule dickson-less-vI)
qed fact
qed
lemma sym-preproc-aux-term1-wf-on:
assumes dickson-grading \(d\)
shows wfp-on \((\lambda x y .(x, y) \in\) sym-preproc-aux-term1 \(d\) ) \(\{x\). set (fst (snd (snd \(x))) \subseteq\) pp-of-term - ' dgrad-set d m\}
proof (rule wfp-onI-min)
let \(? B=p p\)-of-term - \({ }^{\text {' }}\) dgrad-set \(d m\)
let ? \(A=\left\{x::\left(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right.\right.\) list \(\times{ }^{\prime} t\) list \(\times{ }^{\prime} t\) list \(\times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \()\). set \((f s t\) (snd \((\) snd \(x))) \subseteq ? B\}\)
have \(A\)-sub-Pow: set 'fst'snd'snd' ? \(A \subseteq\) Pow ?B by auto
fix \(x Q\)
assume \(x \in Q\) and \(Q \subseteq ? A\)
let \(? Q=\{\) ord-term-lin.Max \((\) set \((f s t(\) snd \((\) snd \(q)))) \mid q \cdot q \in Q \wedge\) fst (snd (snd q) \() \neq[]\}\)
show \(\exists z \in Q . \forall y \in\{x\). set \((f\) st \((\) snd \((\) snd \(x))) \subseteq ? B\} .(y, z) \in\) sym-preproc-aux-term1 \(d \longrightarrow y \notin Q\)
proof (cases \(\exists z \in Q\). fst \((\) snd \((\) snd \(z))=[])\)
case True
then obtain \(z\) where \(z \in Q\) and \(f s t(\operatorname{snd}(\) snd \(z))=[] .\).
show ?thesis
proof (intro bexI ballI impI)
fix \(y\)
assume \((y, z) \in\) sym-preproc-aux-term1 \(d\)
then obtain \(t\) where \(t \in \operatorname{set}(f s t(s n d(s n d z)))\) unfolding sym-preproc-aux-term1-def by auto
with \(\langle f s t(\) snd \((\) snd \(z))=[]\rangle\) show \(y \notin Q\) by simp
qed fact
next
case False
hence \(*: q \in Q \Longrightarrow f\) st \((\) snd \((\) snd \(q)) \neq[]\) for \(q\) by blast
with \(\langle x \in Q\rangle\) have \(f\) st (snd (snd \(x)\) ) \(\neq[]\) by simp
from assms have wfp-on \(\left(\prec_{t}\right)\) ?B by (rule wfp-on-ord-term-strict)
moreover from \(\langle x \in Q\rangle\langle f s t(\) snd \((\) snd \(x)) \neq[]\rangle\)
have ord-term-lin.Max \((\) set \((\) fst \((\) snd \((\operatorname{snd} x)))) \in ? Q\) by blast
moreover have ? \(Q \subseteq\) ? \(B\)
proof (rule, simp, elim exE conjE, simp)
fix \(a b c d 0\)
assume \((a, b, c, d 0) \in Q\) and \(c \neq[]\)
from this (1) 〈Q \(\subseteq\) ? \(A\rangle\) have \((a, b, c, d 0) \in ? A\)..
hence \(p\)-of-term' set \(c \subseteq d g r a d\)-set \(d m\) by auto
moreover have pp-of-term (ord-term-lin.Max (set c)) \(\in\) pp-of-term'set \(c\)
proof
from \(\langle c \neq[]\rangle\) show ord-term-lin.Max \((\) set \(c) \in\) set \(c\) by simp
qed (fact refl)
ultimately show pp-of-term (ord-term-lin.Max (set c)) \(\in\) dgrad-set d m ..
qed
ultimately obtain \(t\) where \(t \in ? Q\) and \(\min : \wedge s . s \prec_{t} t \Longrightarrow s \notin ? Q\) by (rule wfp-onE-min) blast
from this \((1)\) obtain \(z\) where \(z \in Q\) and \(f s t(\operatorname{snd}(\operatorname{snd} z)) \neq[]\)
and \(t: t=\) ord-term-lin.Max \((\) set \((f s t(\) snd \((s n d z))))\) by blast
show ?thesis
proof (intro bexI ballI impI, rule)
fix \(y\)
assume \(y \in ? A\) and \((y, z) \in\) sym-preproc-aux-term1 \(d\) and \(y \in Q\)
from this(2) obtain \(t^{\prime}\) where \(t^{\prime} \in \operatorname{set}(f s t(\) snd \((\) snd \(z)))\)
and \(* *: \bigwedge s . s \in \operatorname{set}(f s t(\) snd \((\) snd \(y))) \Longrightarrow s \prec_{t} t^{\prime}\)
unfolding sym-preproc-aux-term1-def by auto
from \(\langle y \in Q\rangle\) have \(f\) st (snd (snd \(y)) \neq[]\) by (rule *)
with \(\langle y \in Q\rangle\) have ord-term-lin.Max (set \((\) fst \((\) snd \((\) snd \(y)))) \in ? Q\) (is ?s \(\in\) -)
by blast
from \(\langle f s t(\) snd \((\operatorname{snd} y)) \neq[]\rangle\) have \(? s \in \operatorname{set}(f s t(\) snd \((\) snd \(y)))\) by simp
hence? \(\prec_{t} t^{\prime}\) by (rule **)
also from \(\left\langle t^{\prime} \in \operatorname{set}(\right.\) fst \((\) snd \((\) snd \(\left.z)))\right\rangle\) have \(t^{\prime} \preceq_{t} t\) unfolding \(t\)
using \(\langle f s t(\) snd \((\) snd \(z)) \neq[]>\) by simp
finally have ?s \(\notin ? Q\) by (rule min)
```

        from this <?s \in?Q> show False ..
    qed fact
    qed
    qed
lemma sym-preproc-aux-term-wf:
assumes dickson-grading d
shows wf (sym-preproc-aux-term d)
proof (rule wfI-min)
fix }x::(('t>\mp@subsup{|}{0}{\prime}'b) list \times 't list \times 't list \times (' ' 看 'b) list) and Q
assume x }\in
let ?A = Keys (set (fst x)) \cup set (fst (snd (snd x)))
have finite?A by (simp add: finite-Keys)
hence finite (pp-of-term' ?A) by (rule finite-imageI)
then obtain m}\mathrm{ where pp-of-term '?A }\subseteqdgrad-set d m by (rule dgrad-set-exhaust
hence A:?A }\subseteqpp\mathrm{ -of-term -' dgrad-set d m by blast
let ?B = pp-of-term -'dgrad-set d m
let ?Q}={q\inQ.Keys (set (fst q))\cup set (fst (snd (snd q)))\subseteq?B
from assms have wfp-on ( }\lambdaxy.(x,y)\in\mathrm{ sym-preproc-aux-term1 d) {x. set (fst
(snd (snd x)))\subseteq?B}
by (rule sym-preproc-aux-term1-wf-on)
moreover from <x\inQ> A have }x\in?Q\mathrm{ by simp
moreover have ?Q \subseteq{x. set (fst (snd (snd x))) \subseteq?B} by auto
ultimately obtain z}\mathrm{ where z \& ?Q
and *: \bigwedgey. (y,z)\in sym-preproc-aux-term1 d\Longrightarrowy\not\in?Q by (rule wfp-onE-min)
blast
from this(1) have z\inQ and Keys (set (fst z))\cup set (fst (snd (snd z)))\subseteq?B
by simp-all
from this(2) have a: pp-of-term'(Keys (set (fst z)) \cup set (fst (snd (snd z))))
\subseteq d g r a d - s e t ~ d ~ m ~
by blast
show \existsz\inQ.\forally. (y,z)\in sym-preproc-aux-term d \longrightarrowy\not\inQ
proof (intro bexI allI impI)
fix }
assume (y,z)\in sym-preproc-aux-term d
hence (y,z) \in sym-preproc-aux-term1 d and (y,z)\in sym-preproc-aux-term2
d
by (simp-all add: sym-preproc-aux-term-def)
from this(2) have fst y=fst z
and dgrad-set-le d (pp-of-term'set (fst (snd (snd y)))) (pp-of-term'(Keys
(set (fst z))\cup set (fst (snd (snd z)))))
by (auto simp add: sym-preproc-aux-term2-def)
from this(2) a have pp-of-term' (set (fst (snd (snd y))))\subseteq dgrad-set d m
by (rule dgrad-set-le-dgrad-set)
hence Keys (set (fst y)) \cup set (fst (snd (snd y)))\subseteq?B
using a by (auto simp add:<fst y = fst z>)
moreover from <(y,z)\in sym-preproc-aux-term1 d> have y }\not\in?Q\mathrm{ by (rule *)
ultimately show y}\not\inQ\mathrm{ by simp
qed fact

```

\section*{qed}
primrec sym-preproc-addnew :: (' \(t \Rightarrow 0\) ' \(b::\) semiring- 1 ) list \(\Rightarrow{ }^{\prime} t\) list \(\Rightarrow\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\) ' \(t \Rightarrow\)
\[
\left(' t \text { list } \times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \text { list }\right) \text { where }
\]
sym-preproc-addnew [] vs \(f s-=(v s, f s) \mid\)
sym-preproc-addnew \((g \# g s)\) vs fs \(v=\)
(if lt \(g\) addst \(v\) then
(let \(f=\) monom-mult 1 (pp-of-term \(v-l p g) g\) in
sym-preproc-addnew gs (merge-wrt \(\left(\succ_{t}\right)\) vs (keys-to-list (tail f))) (insert-list
\(f f s) v\)
)
else
sym-preproc-addnew gs vs fs \(v\)
)
lemma fst-sym-preproc-addnew-less:
assumes \(\wedge u . u \in\) set \(v s \Longrightarrow u \prec_{t} v\)
and \(u \in\) set (fst (sym-preproc-addnew gs vs \(f s v)\) )
shows \(u \prec_{t} v\)
using assms
proof (induct gs arbitrary: fs vs)
case Nil
from \(\operatorname{Nil(2)}\) have \(u \in\) set vs by simp
thus ?case by (rule Nil(1))
next
case (Cons g gs)
from Cons(3) show ?case
proof (simp add: Let-def split: if-splits)
let \(? t=p p\)-of-term \(v-l p g\)
assume lt \(g\) addst \(v\)
assume \(u \in\) set (fst (sym-preproc-addnew gs
(merge-wrt \(\left(\succ_{t}\right)\) vs (keys-to-list (tail (monom-mult 1 ?t
\(g))\) )
(insert-list (monom-mult 1 ?t g) fs) v))
with - show ?thesis
proof (rule Cons(1))
fix \(u\)
assume \(u \in \operatorname{set}\left(\right.\) merge-wrt \(\left(\succ_{t}\right)\) vs (keys-to-list (tail (monom-mult 1 ?t g)))) hence \(u \in\) set vs \(\vee u \in\) keys (tail (monom-mult 1 ?t g))
by (simp add: set-merge-wrt keys-to-list-def set-pps-to-list)
thus \(u \prec_{t} v\)
proof
assume \(u \in\) set vs
thus ?thesis by (rule Cons(2))
next
assume \(u \in\) keys (tail (monom-mult 1 ?t g))
hence \(u \prec_{t}\) lt (monom-mult 1 ?t \(g\) ) by (rule keys-tail-less-lt)
also have \(\ldots \preceq_{t}\) ? \(t \oplus l t g\) by (rule lt-monom-mult-le)
```

                also from <lt g addst v> have ... = v
                    by (metis add-diff-cancel-right' adds-termE pp-of-term-splus)
            finally show ?thesis .
        qed
    qed
    next
        assume u\in set (fst (sym-preproc-addnew gs vs fs v))
        with Cons(2) show ?thesis by (rule Cons(1))
    qed
    qed
lemma fst-sym-preproc-addnew-dgrad-set-le:
assumes dickson-grading d
shows dgrad-set-le d (pp-of-term' set (fst (sym-preproc-addnew gs vs fs v)))
(pp-of-term'(Keys (set gs)\cup insert v (set vs)))
proof (induct gs arbitrary: fs vs)
case Nil
show ?case by (auto intro: dgrad-set-le-subset)
next
case (Cons g gs)
show ?case
proof (simp add: Let-def, intro conjI impI)
assume lt g addst v
let ?t = pp-of-term v-lpg
let ?vs=merge-wrt (}\mp@subsup{\succ}{t}{})\mathrm{ ) vs (keys-to-list (tail (monom-mult 1 ?t g)))
let ?fs = insert-list (monom-mult 1 ?t g) fs
from Cons have dgrad-set-le d (pp-of-term' set (fst (sym-preproc-addnew gs
?vs ?fs v)))
(pp-of-term'(Keys (insert g(set gs)) \cup insert v(set
vs)))
proof (rule dgrad-set-le-trans)
show dgrad-set-le d (pp-of-term'(Keys (set gs) \cup insert v (set ?vs)))
(pp-of-term'(Keys (insert g (set gs)) \cup insert v (set vs)))
unfolding dgrad-set-le-def set-merge-wrt set-keys-to-list
proof (intro ballI)
fix }
assume s \in pp-of-term'(Keys (set gs) U insert v (set vs U keys (tail
(monom-mult 1 ?t g))))
hence s f pp-of-term '(Keys (set gs) \cup insert v (set vs)) \cup pp-of-term'
keys (tail (monom-mult 1 ?t g))
by auto
thus \existst\inpp-of-term'(Keys (insert g(set gs))\cup insert v (set vs)).d s\leq
d t
proof
assume s f pp-of-term '(Keys (set gs) \cup insert v (set vs))
thus ?thesis by (auto simp add: Keys-insert)
next
assume s f pp-of-term'keys (tail (monom-mult 1 ?t g))
hence s\inpp-of-term'keys (monom-mult 1 ?t g) by (auto simp add:

```
keys-tail)
from this keys-monom-mult-subset have \(s \in\) pp-of-term' \((\oplus)\) ?t' keys \(g\) by blast
then obtain \(u\) where \(u \in\) keys \(g\) and \(s: s=p p\)-of-term ( ?t \(\oplus u\) ) by blast have \(d s=d\) ? \(t \vee d s=d\) ( \(p p\)-of-term \(u\) ) unfolding \(s\) pp-of-term-splus using dickson-gradingD1[OF assms] by auto
thus ?thesis
proof
from \(\left\langle l t g a d d s_{t} v\right\rangle\) have \(l p g\) adds pp-of-term \(v\) by (simp add: adds-term-def)
assume \(d s=d\) ? \(t\)
also from assms \(\langle l p g\) adds pp-of-term \(v\rangle\) have \(\ldots \leq d\) (pp-of-term \(v\) ) by (rule dickson-grading-minus)
finally show?thesis by blast
next
assume \(d s=d\) ( \(p\) p-of-term \(u\) )
moreover from \(\langle u \in\) keys \(g\rangle\) have \(u \in\) Keys (insert \(g\) (set gs)) by (simp add: Keys-insert)
ultimately show ?thesis by auto
qed
qed
qed
qed
thus dgrad-set-le d (pp-of-term' set (fst (sym-preproc-addnew gs ?vs ?fs v)))
(insert (pp-of-term v) (pp-of-term' (Keys (insert g (set gs)) \(\cup\) set \(v s)\) ))
by \(\operatorname{simp}\)
next
from Cons show dgrad-set-le d (pp-of-term'set (fst (sym-preproc-addnew gs vs \(f s\) v))
(insert (pp-of-term \(v)(p p-o f-t e r m '(\) Keys \((\) insert \(g(\) set gs))
\(\cup\) set \(v s))\) )
proof (rule dgrad-set-le-trans)
show dgrad-set-le d (pp-of-term ' \((\) Keys \((\) set gs \() \cup\) insert \(v(\) set vs \()))\)
(insert (pp-of-term v) (pp-of-term' (Keys (insert g (set gs))
\(\cup\) set \(v s))\) )
by (rule dgrad-set-le-subset, auto simp add: Keys-def)
qed
qed
qed
lemma components-fst-sym-preproc-addnew-subset:
component-of-term'set (fst (sym-preproc-addnew gs vs fs v)) \(\subseteq\) component-of-term
' (Keys \((\) set \(g s) \cup\) insert \(v(\) set vs \())\)
proof (induct gs arbitrary: fs vs)
case Nil
show ?case by (auto intro: dgrad-set-le-subset)
next
case (Cons g gs)
```

show ?case
proof (simp add: Let-def, intro conjI impI)
assume lt g addst v
let ?t = pp-of-term v-lpg
let ?vs = merge-wrt (}\mp@subsup{\succ}{t}{})\mathrm{ ) vs (keys-to-list (tail (monom-mult 1 ?t g)))
let ?fs = insert-list (monom-mult 1 ?t g) fs
from Cons have component-of-term'set (fst (sym-preproc-addnew gs ?vs ?fs

```
\(v)) \subseteq\)
                            component-of-term' \((\) Keys \((\) insert \(g(\) set gs \()) \cup\) insert \(v(\) set \(v s))\)
    proof (rule subset-trans)
    show component-of-term ' Keys \((\) set gs \() \cup\) insert \(v(\) set ? \(v s)) \subseteq\)
            component-of-term' (Keys \((\) insert \(g(\) set gs \()) \cup\) insert \(v(\) set vs \())\)
        unfolding set-merge-wrt set-keys-to-list
    proof
        fix \(k\)
            assume \(k \in\) component-of-term ' \((\) Keys \((\) set gs) \(\cup\) insert \(v\) (set vs \(\cup\) keys
(tail (monom-mult 1 ?t g))))
            hence \(k \in\) component-of-term ' \((\) Keys \((\) set gs \() \cup\) insert \(v(\) set vs \()) \cup\)
component-of-term' keys (tail (monom-mult 1 ?t g))
            by auto
            thus \(k \in\) component-of-term ' (Keys (insert \(g(\) set gs \()) \cup\) insert \(v(\) set vs \())\)
            proof
            assume \(k \in\) component-of-term ' \((\) Keys \((\) set gs \() \cup\) insert \(v(\) set vs \())\)
            thus ?thesis by (auto simp add: Keys-insert)
        next
            assume \(k \in\) component-of-term' keys (tail (monom-mult 1 ?t g))
            hence \(k \in\) component-of-term'keys (monom-mult 1 ?t \(g\) ) by (auto simp
add: keys-tail)
            from this keys-monom-mult-subset have \(k \in\) component-of-term ' \((\oplus)\) ?t
' keys \(g\) by blast
            also have \(\ldots \subseteq\) component-of-term' keys \(g\) using component-of-term-splus
by fastforce
            finally show ?thesis by (simp add: image-Un Keys-insert)
            qed
        qed
    qed
    thus component-of-term'set (fst (sym-preproc-addnew gs ?vs ?fs \(v)\) ) \(\subseteq\)
                insert (component-of-term v) (component-of-term ' (Keys (insert g (set
\(g s)) \cup\) set \(v s))\)
        by simp
    next
    from Cons show component-of-term'set (fst (sym-preproc-addnew gs vs fs v))
\(\subseteq\)
                                    insert (component-of-term v) (component-of-term ' (Keys (insert g
\((\) set \(g s)) \cup\) set \(v s))\)
    proof (rule subset-trans)
        show component-of-term' \((\) Keys \((\) set gs \() \cup\) insert \(v(\) set vs \()) \subseteq\)
                insert (component-of-term \(v\) ) (component-of-term ' (Keys (insert g (set
\(g s)) \cup\) set \(v s))\)
```

        by (auto simp add: Keys-def)
        qed
    qed
    qed
lemma fst-sym-preproc-addnew-superset: set vs \subseteq set (fst (sym-preproc-addnew gs
vs fs v))
proof (induct gs arbitrary:vs fs)
case Nil
show ?case by simp
next
case (Cons g gs)
show ?case
proof (simp add: Let-def, intro conjI impI)
let ?t = pp-of-term v-lpg
define f}\mathrm{ where f= monom-mult 1 ?t g
have set vs \subseteqset (merge-wrt (}\mp@subsup{\succ}{t}{})\mathrm{ )vs (keys-to-list (tail f))) by (auto simp add:
set-merge-wrt)
thus set vs \subseteqset (fst (sym-preproc-addnew gs
(merge-wrt (}\mp@subsup{\succ}{t}{})\mathrm{ ) vs (keys-to-list (tail f))) (insert-list ffs)
v))
using Cons by (rule subset-trans)
next
show set vs \subseteq set (fst (sym-preproc-addnew gs vs fs v)) by (fact Cons)
qed
qed
lemma snd-sym-preproc-addnew-superset: set fs \subseteq set (snd (sym-preproc-addnew
gs vs fs v))
proof (induct gs arbitrary:vs fs)
case Nil
show ?case by simp
next
case (Cons g gs)
show ?case
proof (simp add: Let-def, intro conjI impI)
let ?t = pp-of-term v - lp g
define f}\mathrm{ where f= monom-mult 1 ?t g
have set fs \subseteqset (insert-list ffs) by (auto simp add: set-insert-list)
thus set fs \subseteq set (snd (sym-preproc-addnew gs
(merge-wrt (}\mp@subsup{\succ}{t}{})\mathrm{ vs (keys-to-list (tail f))) (insert-list f fs)
v))
using Cons by (rule subset-trans)
next
show set fs \subseteqset (snd (sym-preproc-addnew gs vs fs v)) by (fact Cons)
qed
qed
lemma in-snd-sym-preproc-addnewE:

```
```

    assumes p\in set (snd (sym-preproc-addnew gs vs fs v))
    assumes 1:p\in set fs \Longrightarrow thesis
    assumes 2: \g s.g\in set gs \Longrightarrowp= monom-mult 1s g\Longrightarrow thesis
    shows thesis
    using assms
    proof (induct gs arbitrary: vs fs thesis)
case Nil
from Nil(1) have p\in set fs by simp
thus ?case by (rule Nil(2))
next
case (Cons g gs)
from Cons(2) show ?case
proof (simp add: Let-def split: if-splits)
define f}\mathrm{ where f= monom-mult 1(pp-of-term v-lpg)g
define ts' where ts' = merge-wrt (}\mp@subsup{\succ}{t}{})\mathrm{ ) vs (keys-to-list (tail f))
define fs'' where fs'}\mp@subsup{}{}{\prime}=\mathrm{ insert-list f fs
assume p}\in\mathrm{ set (snd (sym-preproc-addnew gs ts' fs'v))
thus?thesis
proof (rule Cons(1))
assume p\in set fs'
hence p=f\veep\in set fs by (simp add: fs'-def set-insert-list)
thus ?thesis
proof
assume p=f
have g}\operatorname{set}(g\#gs) by sim
from this }\langlep=f\rangle\mathrm{ show ?thesis unfolding f-def by (rule Cons(4))
next
assume p \in set fs
thus ?thesis by (rule Cons(3))
qed
next
fix hs
assume h set gs
hence h}\in\operatorname{set}(g\#gs) by sim
moreover assume p= monom-mult 1s h
ultimately show thesis by (rule Cons(4))
qed
next
assume p\in set (snd (sym-preproc-addnew gs vs fs v))
moreover note Cons(3)
moreover have h\in set gs \Longrightarrowp= monom-mult 1sh\Longrightarrowthesis for hs
proof -
assume h f set gs
hence }h\in\operatorname{set}(g\#gs) by sim
moreover assume p= monom-mult 1s h
ultimately show thesis by (rule Cons(4))
qed
ultimately show ?thesis by (rule Cons(1))
qed

```

\section*{qed}
lemma sym-preproc-addnew-pmdl:
pmdl (set gs \(\cup\) set (snd (sym-preproc-addnew gs vs fs \(v)\) )) \(=\) pmdl (set gs \(\cup\) set fs)
(is pmdl (set gs \(\cup ? l)=? r)\)
proof
have set \(g s \subseteq\) set \(g s \cup\) set \(f s\) by simp
also have \(\ldots \subseteq\) ?r by (fact pmdl.span-superset)
finally have set \(g s \subseteq\) ? \(r\).
moreover have ?l \(\subseteq\) ?r
proof
fix \(p\)
assume \(p \in\) ?l
thus \(p \in\) ? \(r\)
proof (rule in-snd-sym-preproc-addnewE)
assume \(p \in\) set \(f s\)
hence \(p \in\) set \(g s \cup\) set fs by simp
thus ?thesis by (rule pmdl.span-base)
next
fix \(g s\)
assume \(g \in\) set gs and \(p: p=\) monom-mult 1 sg
from this (1) 〈set \(g s \subseteq ? r\rangle\) have \(g \in ? r\)..
thus ?thesis unfolding \(p\) by (rule pmdl-closed-monom-mult)
qed
qed
ultimately have set gs \(\cup ?\urcorner\) ? \(\subseteq\) by blast
thus pmdl (set gs \(\cup ? l) \subseteq\) ?r by (rule pmdl.span-subset-spanI)
next
from snd-sym-preproc-addnew-superset have set gs \(\cup\) set \(f s \subseteq\) set \(g s \cup\) ?l by blast
thus \(? r \subseteq p m d l(\) set \(g s \cup ? l)\) by (rule pmdl.span-mono)
qed
lemma Keys-snd-sym-preproc-addnew:
Keys \((\) set \((\) snd \((\) sym-preproc-addnew gs vs fs \(v))) \cup\) insert \(v(\) set \(v s)=\)
Keys \((\) set \(f s) \cup\) insert \(v\) (set ( \(f s t\) (sym-preproc-addnew gs vs ( \(f s::\left(' t \Rightarrow_{0}{ }^{\prime} b::\right.\) semiring-1-no-zero-divisors)
list) \(v\) ))
proof (induct gs arbitrary: vs fs)
case Nil
show ? case by simp
next
case (Cons g gs)
from Cons have eq: insert \(v\) (Keys (set (snd (sym-preproc-addnew gs ts'fs'v)))
\(\cup\) set \(\left.t s^{\prime}\right)=\)
insert \(v\) (Keys (set fs \(\left.{ }^{\prime}\right) \cup\) set (fst (sym-preproc-addnew gs ts \(s^{\prime} f s^{\prime}\)
v)))
for \(t s^{\prime} f s^{\prime}\) by \(\operatorname{simp}\)
show ?case
```

    proof (simp add: Let-def eq, rule)
    assume \(l t g a d d s_{t} v\)
    let \(? t=p p\)-of-term \(v-l p g\)
    define \(f\) where \(f=\) monom-mult 1 ?t \(g\)
    define \(t s^{\prime}\) where \(t s^{\prime}=\) merge-wrt \(\left(\succ_{t}\right)\) vs (keys-to-list (tail f))
    define \(f s^{\prime}\) where \(f s^{\prime}=\) insert-list \(f f s\)
    have keys \((\) tail \(f)=\) keys \(f-\{v\}\)
    proof (cases \(g=0\) )
        case True
        hence \(f=0\) by (simp add: \(f\)-def)
        thus ?thesis by simp
    next
        case False
        hence lt \(f=\) ?t \(\oplus l t g\) by (simp add: \(f\)-def lt-monom-mult)
        also from \(\left\langle l t g a d d s_{t} v\right\rangle\) have \(\ldots=v\)
            by (metis add-diff-cancel-right' adds-termE pp-of-term-splus)
        finally show ?thesis by (simp add: keys-tail)
    qed
    hence \(t s^{\prime}:\) set \(t s^{\prime}=\) set \(v s \cup(\) keys \(f-\{v\})\)
        by (simp add: ts'-def set-merge-wrt set-keys-to-list)
    have \(f s^{\prime}\) : set \(f s^{\prime}=\operatorname{insert} f(\) set \(f s)\) by (simp add: \(f s^{\prime}\)-def set-insert-list)
    hence \(f \in\) set \(f s^{\prime}\) by simp
    from this snd-sym-preproc-addnew-superset have \(f \in\) set (snd (sym-preproc-addnew
    $\left.g s t s^{\prime} f s^{\prime} v\right)$ )..
hence keys $f \subseteq$ Keys (set (snd (sym-preproc-addnew gs ts' $\mathrm{fs}^{\prime}$ v)) ) by (rule
keys-subset-Keys)
hence insert $v\left(\right.$ Keys $\left(\right.$ set $\left(\right.$ snd $\left(\right.$ sym-preproc-addnew gs ts $\left.\left.\left.s^{\prime} f s^{\prime} v\right)\right)\right) \cup$ set vs $)=$
insert $v\left(\right.$ Keys (set (snd (sym-preproc-addnew gs ts' $\left.\left.\left.f s^{\prime} v\right)\right)\right) \cup$ set ts $\left.{ }^{\prime}\right)$
by (auto simp add: ts')
also have $\ldots=$ insert $v\left(\right.$ Keys $\left(\right.$ set $\left.f s^{\prime}\right) \cup$ set (fst (sym-preproc-addnew gs ts ${ }^{\prime}$
$\left.f_{s}{ }^{\prime} v\right)$ )
by (fact eq)
also have $\ldots=\operatorname{insert} v\left(\right.$ Keys $($ set $f s) \cup$ set $\left(f s t\left(\right.\right.$ sym-preproc-addnew gs ts' fs ${ }^{\prime}$
v)))
proof -
\{
fix $u$
assume $u \neq v$ and $u \in$ keys $f$
hence $u \in$ set $t s^{\prime}$ by (simp add: $t s^{\prime}$ )
from this fst-sym-preproc-addnew-superset have $u \in$ set (fst (sym-preproc-addnew
$\left.g s t s^{\prime} f s^{\prime} v\right)$ )..
\}
thus ?thesis by (auto simp add: $f s^{\prime}$ Keys-insert)
qed
finally show insert $v\left(\right.$ Keys $\left(\right.$ set $\left(\right.$ snd $\left(\right.$ sym-preproc-addnew gs ts' $\left.\left.\left.f s^{\prime} v\right)\right)\right) \cup$ set
$v s)=$
insert $v\left(\right.$ Keys $($ set $f s) \cup$ set $\left.\left(f s t\left(s y m-p r e p r o c-a d d n e w ~ g s t s^{\prime} f s^{\prime} v\right)\right)\right)$.
qed
qed

```
```

lemma sym-preproc-addnew-complete:
assumes g}\in\mathrm{ set gs and lt gaddst v
shows monom-mult 1 (pp-of-term v - lp g)g\in set (snd (sym-preproc-addnew
gs vs fs v))
using assms(1)
proof (induct gs arbitrary:vs fs)
case Nil
thus ?case by simp
next
case (Cons h gs)
let ?t = pp-of-term v-lpg
show ?case
proof (cases h=g)
case True
show ?thesis
proof (simp add: True assms(2) Let-def)
define f}\mathrm{ where f= monom-mult 1 ?t g
define ts' where ts'= merge-wrt (}\mp@subsup{\succ}{t}{})\mathrm{ )vs (keys-to-list (tail (monom-mult 1
?t g)))
have f}\in\mathrm{ set (insert-list f fs) by (simp add: set-insert-list)
with snd-sym-preproc-addnew-superset show f}\in\mathrm{ set (snd (sym-preproc-addnew
gs ts'(insert-list ffs)v)) ..
qed
next
case False
with Cons(2) have g}\in\mathrm{ set gs by simp
hence *: monom-mult 1 ?t g set (snd (sym-preproc-addnew gs ts' fs'v)) for
ts' fs'
by (rule Cons(1))
show ?thesis by (simp add: Let-def *)
qed
qed
function sym-preproc-aux :: ('t =\mp@subsup{0}{0}{\prime}}\mathrm{ 'b::semiring-1) list }=>\mp@subsup{}{}{\prime}t list => ('t list > ('t
\#0'b) list) =>
('t list }\times(\mp@subsup{}{}{\prime}t=\mp@subsup{=}{0}{\prime}'b) list) wher
sym-preproc-aux gs ks (vs,fs)=
(if vs = [] then
(ks, fs)
else
let v = ord-term-lin.max-list vs; vs' = removeAll v vs in
sym-preproc-aux gs (ks @ [v]) (sym-preproc-addnew gs vs' fs v)
)
by pat-completeness auto
termination proof -
from ex-dgrad obtain d::'a m nat where dg: dickson-grading d ..

```

```

'b) list) }

```
\[
\left.\left({ }^{\prime} t \Rightarrow_{0}^{\prime} b\right) \text { list } \times{ }^{\prime} t \text { list } \times{ }^{\prime} t \text { list } \times\left({ }^{\prime} t \Rightarrow_{0}^{\prime} b\right) \text { list }\right) \text { set }
\]
show ?thesis
proof
from \(d g\) show \(w f\) ? \(R\) by (rule sym-preproc-aux-term-wf)
next
fix \(g s::\left(' t \nRightarrow_{0} ' b\right)\) list and ks vs \(f s v v s^{\prime}\)
assume \(v s \neq[]\) and \(v=\) ord-term-lin.max-list \(v s\) and \(v s^{\prime}: v s^{\prime}=\) removeAll \(v\) vs
from \(\operatorname{this}(1,2)\) have \(v: v=\) ord-term-lin.Max (set vs)
by (simp add: ord-term-lin.max-list-Max)
obtain vs0 fs0 where eq: sym-preproc-addnew gs vs' fs \(v=(v s 0, f s 0)\) by
fastforce
show \(((g s, k s @[v]\), sym-preproc-addnew gs vs' fs \(v),(g s, k s, v s, f s)) \in ? R\)
proof (simp add: eq sym-preproc-aux-term-def sym-preproc-aux-term1-def sym-preproc-aux-term2-def, intro conjI bexI ballI)
fix \(w\)
assume \(w \in\) set vs0
show \(w \prec_{t} v\)
proof (rule fst-sym-preproc-addnew-less)
fix \(u\)
assume \(u \in\) set \(v s^{\prime}\)
thus \(u \prec_{t} v\) unfolding \(v s^{\prime} v\) set-removeAll using ord-term-lin.antisym-conv1
by fastforce
next
from \(\langle w \in\) set \(v s 0\rangle\) show \(w \in \operatorname{set}(f s t\) (sym-preproc-addnew gs vs' fs \(v))\) by
(simp add: eq)
qed
next
from \(\langle v s \neq[]\rangle\) show \(v \in\) set \(v s\) by ( \(\operatorname{simp}\) add: \(v\) )
next
from \(d g\) have dgrad-set-le d (pp-of-term'set (fst (sym-preproc-addnew gs vs' fs \(v)\) ))
\[
(p p-o f \text {-term' }(\text { Keys }(\text { set gs }) \cup \text { insert } v(\text { set vs'})))
\]
by (rule fst-sym-preproc-addnew-dgrad-set-le)
moreover have insert \(v\left(\right.\) set \(\left.v s^{\prime}\right)=\) set vs by (auto simp add: vs' \(\left.v\langle v s \neq[]\rangle\right)\)
ultimately show dgrad-set-le d (pp-of-term' set vs0) (pp-of-term' (Keys \((\) set \(g s) \cup\) set \(v s))\)
by (simp add: eq)
qed
qed
qed
lemma sym-preproc-aux-Nil: sym-preproc-aux gs ks \(([], f s)=(k s, f s)\)
by \(\operatorname{simp}\)
lemma sym-preproc-aux-sorted:
assumes sorted-wrt \(\left(\succ_{t}\right)(v \# v s)\)
shows sym-preproc-aux gs \(k s(v \# v s, f s)=\) sym-preproc-aux gs (ks @ [v])
(sym-preproc-addnew gs vs fs \(v\) )
proof -
from assms have \(*: u \in\) set \(v s \Longrightarrow u \prec_{t} v\) for \(u\) by simp
have ord-term-lin.max-list \((v \#\) vs \()=\) ord-term-lin.Max \((\) set \((v \# v s))\)
by (simp add: ord-term-lin.max-list-Max del: ord-term-lin.max-list.simps)
also have \(\ldots=v\)
proof (rule ord-term-lin.Max-eqI)
fix \(s\)
assume \(s \in \operatorname{set}(v \# v s)\)
hence \(s=v \vee s \in\) set vs by simp
thus \(s \preceq_{t} v\)
proof
assume \(s=v\)
thus ?thesis by simp
next
assume \(s \in\) set vs
hence \(s \prec_{t} v\) by (rule *)
thus ?thesis by simp
qed
next
show \(v \in \operatorname{set}(v \# v s)\) by \(\operatorname{simp}\)
qed rule
finally have eq1: ord-term-lin.max-list \((v \# v s)=v\).
have eq2: removeAll \(v(v \# v s)=v s\)
proof (simp, rule removeAll-id, rule)
assume \(v \in\) set vs
hence \(v \prec_{t} v\) by (rule *)
thus False ..
qed
show ?thesis by (simp only: sym-preproc-aux.simps eq1 eq2 Let-def, simp)
qed
lemma sym-preproc-aux-induct [consumes 0, case-names base rec]:
assumes base: \(\bigwedge k s f s . P k s[] f s(k s, f s)\)
and rec: \(\bigwedge k s\) vs fs \(v v^{\prime}\). vs \(\neq[] \Longrightarrow v=\) ord-term-lin.Max (set vs) \(\Longrightarrow v s^{\prime}=\) removeAll v vs \(\Longrightarrow\)
\(P(k s @[v])(f s t(\) sym-preproc-addnew gs vs' fs \(v))(\) snd (sym-preproc-addnew \(\left.g s v s^{\prime} f s v\right)\) )
(sym-preproc-aux gs (ks@ [v]) (sym-preproc-addnew gs vs' fs v))
Pks vs fs (sym-preproc-aux gs (ks @ [v]) (sym-preproc-addnew gs vs' fs \(v\) ))
shows \(P\) ks vs fs (sym-preproc-aux gs ks (vs, fs))
proof -
from ex-dgrad obtain \(d::^{\prime} a \Rightarrow\) nat where dg: dickson-grading \(d\)..
let ? \(R=(\) sym-preproc-aux-term \(d)::\left(\left(\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\right.\right.\) list \(\times{ }^{\prime} t\) list \(\times{ }^{\prime} t\) list \(\times\left(^{\prime} t \Rightarrow_{0}\right.\)
'b) list) \(\times\)
\(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\times{ }^{\prime} t\) list \(\times{ }^{\prime} t\) list \(\times\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \()\) set
define args where args \(=(g s, k s, v s, f s)\)
from \(d g\) have \(w f\) ? \(R\) by (rule sym-preproc-aux-term-wf)
hence \(f s t\) args \(=g s \Longrightarrow P(f s t(s n d \operatorname{args}))(f s t(s n d(s n d \operatorname{args})))(\) snd (snd (snd
\(\arg s))\) )
\[
(\text { sym-preproc-aux gs }(\text { fst }(\text { snd args }))(\text { snd }(\text { snd args })))
\]

\section*{proof induct}
fix \(x\)
assume \(I H^{\prime}: \bigwedge y .(y, x) \in\) sym-preproc-aux-term \(d \Longrightarrow f s t y=g s \Longrightarrow\)
\(P(f s t(\) snd \(y))(\) fst \((\) snd \((\) snd \(y)))(\) snd (snd (snd \(y)))\)
(sym-preproc-aux gs (fst (snd y)) (snd (snd y)))
assume \(f\) st \(x=g s\)
then obtain \(x 0\) where \(x: x=(g s, x 0)\) by (meson eq-fst-iff)
obtain \(k s x 1\) where \(x 0: x 0=(k s, x 1)\) by (meson case-prodE case-prodIL)
obtain \(v s f_{s}\) where \(x 1: x 1=(v s, f s)\) by (meson case-prodE case-prodI2)
from \(I H^{\prime}\) have \(I H: \bigwedge k s^{\prime} n .\left(\left(g s, k s^{\prime}, n\right),(g s, k s, v s, f s)\right) \in\) sym-preproc-aux-term \(d \Longrightarrow\)
\(P k s^{\prime}(f s t n)(\) snd \(n)\left(\right.\) sym-preproc-aux gs \(\left.k s^{\prime} n\right)\)
unfolding \(x x 0 x 1\) by fastforce
show \(P(\) fst \((\) snd \(x))(f s t(\) snd \((\operatorname{snd} x)))(\) snd \((\) snd \((\) snd \(x)))\)
(sym-preproc-aux gs \((\) fst \((\) snd \(x))\) (snd \((\) snd \(x)))\)
proof (simp add: x x0 x1 Let-def, intro conjI impI)
show \(P k s[] f s(k s, f s)\) by (fact base)
next
assume \(v s \neq[]\)
define \(v\) where \(v=\) ord-term-lin.max-list vs
from \(\langle v s \neq[]\rangle\) have \(v\)-alt: \(v=\) ord-term-lin.Max (set vs) unfolding \(v\)-def by (rule ord-term-lin.max-list-Max)
define \(v s^{\prime}\) where \(v s^{\prime}=\) removeAll \(v\) vs
show \(P\) ks vs fs (sym-preproc-aux gs (ks @ [v]) (sym-preproc-addnew gs vs' fs
v))
proof (rule rec, fact \(\langle v s \neq[]\rangle\), fact \(v\)-alt, fact vs'-def)
let \(? n=\) sym-preproc-addnew gs \(v s^{\prime}\) fs \(v\)
obtain \(v s 0 f_{s} 0\) where eq: ? \(n=\left(v s 0, f_{s} 0\right)\) by fastforce
show \(P\) (ks @ \([v]\) ) (fst ? \(n\) ) (snd ?n) (sym-preproc-aux gs (ks @ \([v]\) ) ?n) proof (rule IH,
simp add: eq sym-preproc-aux-term-def sym-preproc-aux-term1-def
sym-preproc-aux-term2-def,
intro conjI bexI ballI)
fix \(s\)
assume \(s \in\) set vs0
show \(s \prec_{t} v\)
proof (rule fst-sym-preproc-addnew-less)
fix \(u\)
assume \(u \in\) set \(v s^{\prime}\)
thus \(u \prec_{t} v\) unfolding \(v s^{\prime}\)-def \(v\)-alt set-removeAll using ord-term-lin.antisym-conv1 by fastforce
next
from \(\langle s \in\) set \(v s 0\rangle\) show \(s \in \operatorname{set}(f s t(\) sym-preproc-addnew gs vs' fs \(v))\)
by (simp add: eq)
qed
next
from \(\langle v s \neq[]\rangle\) show \(v \in\) set \(v s\) by (simp add: v-alt)

\section*{next}
from \(d g\) have dgrad-set-le \(d\) (pp-of-term'set (fst (sym-preproc-addnew gs \(\left.\left.v s^{\prime} f s v\right)\right)\) )
\[
(p p \text {-of-term ‘ }(\text { Keys }(\text { set gs }) \cup \text { insert } v(\text { set vs'})))
\]
by (rule fst-sym-preproc-addnew-dgrad-set-le)
moreover have insert \(v\left(\right.\) set \(\left.v s^{\prime}\right)=\) set \(v s\) by (auto simp add: vs'-def v-alt \(\langle v s \neq[]\rangle)\)
ultimately show dgrad-set-le d (pp-of-term'set vsO) (pp-of-term' (Keys \((\) set \(g s) \cup\) set \(v s))\)
by (simp add: eq)
qed
qed
qed
qed
thus ?thesis by (simp add: args-def)
qed
lemma fst-sym-preproc-aux-sorted-wrt:
assumes sorted-wrt \(\left(\succ_{t}\right) k s\) and \(\bigwedge k v . k \in\) set \(k s \Longrightarrow v \in\) set \(v s \Longrightarrow v \prec_{t} k\)
shows sorted-wrt \(\left(\succ_{t}\right)\) (fst (sym-preproc-aux gs ks (vs, \(\left.f s\right)\) ))
using assms
proof (induct gs ks vs fs rule: sym-preproc-aux-induct)
case (base ks fs)
from base(1) show? case by simp
next
case (rec ks vs fs v vs')
from \(\operatorname{rec}(1)\) have \(v \in\) set \(v s\) by ( \(\operatorname{simp} a d d: \operatorname{rec}(2))\)
from \(\operatorname{rec}(1)\) have \(*: \bigwedge u . u \in\) set \(v s^{\prime} \Longrightarrow u \prec_{t} v\) unfolding rec (2, 3) set-removeAll using ord-term-lin.antisym-conv3 by force
show ?case
proof (rule rec(4))
show sorted-wrt \(\left(\succ_{t}\right)(k s @[v])\)
proof (simp add: sorted-wrt-append rec(5), rule)
fix \(k\)
assume \(k \in\) set \(k s\)
from this \(\langle v \in\) set \(v s\rangle\) show \(v \prec_{t} k\) by (rule rec(6))
qed
next
fix \(k u\)
assume \(k \in \operatorname{set}(k s @[v])\) and \(u \in \operatorname{set}\left(f s t\left(s y m-p r e p r o c-a d d n e w ~ g s ~ v s^{\prime} f s v\right)\right)\)
from \(*\) this(2) have \(u \prec_{t} v\) by (rule fst-sym-preproc-addnew-less)
from \(\langle k \in \operatorname{set}(k s @[v])\rangle\) have \(k \in\) set \(k s \vee k=v\) by auto
thus \(u \prec_{t} k\)
proof
assume \(k \in\) set \(k s\)
from this \(\langle v \in\) set \(v s\rangle\) have \(v \prec_{t} k\) by (rule rec(6))
with \(\left\langle u \prec_{t} v\right\rangle\) show ?thesis by simp
next
assume \(k=v\)
```

        with }\langleu\mp@subsup{\prec}{t}{}v\rangle\mathrm{ show ?thesis by simp
        qed
        qed
    qed
lemma fst-sym-preproc-aux-complete:
assumes Keys (set (fs::(' }t=\mp@subsup{|}{0}{\prime}'b::semiring-1-no-zero-divisors) list)) = set ks U
set vs
shows set (fst (sym-preproc-aux gs ks (vs,fs))) = Keys (set (snd (sym-preproc-aux
gs ks (vs, fs))))
using assms
proof (induct gs ks vs fs rule: sym-preproc-aux-induct)
case (base ks fs)
thus ?case by simp
next
case (rec ks vs fs v vs')
from rec(1) have v\in set vs by (simp add: rec(2))
hence eq: insert v (set vs')= set vs by (auto simp add: rec(3))
also from rec(5) have ...\subseteq Keys (set fs) by simp
also from snd-sym-preproc-addnew-superset have ... \subseteq Keys (set (snd (sym-preproc-addnew
gs vs'fs v)))
by (rule Keys-mono)
finally have ... = .. \cup(insert v (set vs')) by blast
also have ... = Keys (set fs)\cup insert v (set (fst (sym-preproc-addnew gs vs'fs
v)))
by (fact Keys-snd-sym-preproc-addnew)
also have ... = (set ks \cup(insert v (set vs')))\cup(insert v(set (fst (sym-preproc-addnew
gs vs'fs v))))
by (simp only: rec(5) eq)
also have ... = set (ks@ @v])\cup(set vs'\cup set (fst (sym-preproc-addnew gs vs' fs
v))) by auto
also from fst-sym-preproc-addnew-superset have ... = set (ks @ [v]) \cup set (fst
(sym-preproc-addnew gs vs' fs v))
by blast
finally show ?case by (rule rec(4))
qed
lemma snd-sym-preproc-aux-superset: set fs }\subseteq\mathrm{ set (snd (sym-preproc-aux gs ks (vs,
fs)))
proof (induct fs rule: sym-preproc-aux-induct)
case (base ks fs)
show ?case by simp
next
case (rec ks vs fs v vs')
from snd-sym-preproc-addnew-superset rec(4) show ?case by (rule subset-trans)
qed
lemma in-snd-sym-preproc-auxE:
assumes p\in set (snd (sym-preproc-aux gs ks (vs,fs)))

```
```

    assumes 1:p\in set fs \Longrightarrow thesis
    assumes 2: }\gt.g\in\mathrm{ set gs }\Longrightarrowp=\mathrm{ monom-mult 1tg thesis
    shows thesis
    using assms
    proof (induct gs ks vs fs arbitrary: thesis rule: sym-preproc-aux-induct)
case (base ks fs)
from base(1) have p\in set fs by simp
thus ?case by (rule base(2))
next
case (rec ks vs fs v vs')
from rec(5) show ?case
proof (rule rec(4))
assume p\in set (snd (sym-preproc-addnew gs vs'fs v))
thus ?thesis
proof (rule in-snd-sym-preproc-addnewE)
assume p\in set fs
thus ?thesis by (rule rec(6))
next
fix gs
assume g}\in\mathrm{ set gs and p= monom-mult 1sg
thus ?thesis by (rule rec(7))
qed
next
fix gt
assume g}\mathrm{ fet gs and p= monom-mult 1tg
thus ?thesis by (rule rec(7))
qed
qed
lemma snd-sym-preproc-aux-pmdl:
pmdl (set gs \cup set (snd (sym-preproc-aux gs ks (ts,fs)))) = pmdl (set gs \cup set
fs)
proof (induct fs rule: sym-preproc-aux-induct)
case (base ks fs)
show ?case by simp
next
case (rec ks vs fs v vs')
from rec(4) sym-preproc-addnew-pmdl show ?case by (rule trans)
qed
lemma snd-sym-preproc-aux-dgrad-set-le:
assumes dickson-grading d and set vs \subseteq Keys (set (fs::(' }t\mp@subsup{=>}{0}{\prime}\mp@subsup{'}{}{\prime}b::semiring-1-no-zero-divisors)
list))
shows dgrad-set-le d (pp-of-term` Keys (set (snd (sym-preproc-aux gs ks (vs,
fs)))))(pp-of-term'Keys (set gs \cup set fs))
using assms(2)
proof (induct fs rule: sym-preproc-aux-induct)
case (base ks fs)
show ?case by (rule dgrad-set-le-subset, simp add: Keys-Un image-Un)

```
```

next
case (rec ks vs fs v vs')
let ?n = sym-preproc-addnew gs vs' fs v
from rec(1) have v\in set vs by (simp add: rec(2))
hence set-vs: insert v (set vs') = set vs by (auto simp add: rec(3))
from rec(5) have eq: Keys (set fs) \cup(Keys (set gs) \cup set vs)=Keys (set gs) \cup
Keys (set fs)
by blast
have dgrad-set-le d (pp-of-term'Keys (set (snd (sym-preproc-aux gs (ks @ [v])
?n))))
(pp-of-term`Keys (set gs \cup set (snd ?n)))
proof (rule rec(4))
have set (fst ?n)\subseteqKeys (set (snd ?n)) \cup insert v(set vs')
by (simp only: Keys-snd-sym-preproc-addnew, blast)
also have ... = Keys (set (snd ?n)) \cup(set vs) by (simp only: set-vs)
also have ... \subseteqKeys (set (snd ?n))
proof -
{
fix u
assume u}\in\mathrm{ set vs
with rec(5) have u\inKeys (set fs)..
then obtain f}\mathrm{ where f}\in\mathrm{ set fs and u}\in\mathrm{ keys f by (rule in-KeysE)
from this(1) snd-sym-preproc-addnew-superset have f\in set (snd ?n) ..
with }\langleu\in\mathrm{ keys f> have u}\in\mathrm{ Keys (set (snd ?n)) by (rule in-KeysI)
}
thus ?thesis by auto
qed
finally show set (fst ?n)\subseteqKeys (set (snd ?n)).
qed
also have dgrad-set-le d ... (pp-of-term'Keys (set gs \cup set fs))
proof (simp only: image-Un Keys-Un dgrad-set-le-Un, rule)
show dgrad-set-le d (pp-of-term'Keys (set gs)) (pp-of-term'Keys (set gs) U
pp-of-term ' Keys (set fs))
by (rule dgrad-set-le-subset, simp)
next
have dgrad-set-le d (pp-of-term'Keys (set (snd ?n))) (pp-of-term'(Keys (set
fs) \cup insert v (set (fst ?n))))
by (rule dgrad-set-le-subset, auto simp only: Keys-snd-sym-preproc-addnew[symmetric])
also have dgrad-set-le d ... (pp-of-term'Keys (set fs)\cup pp-of-term'(Keys (set
gs) \cup insert v(set vs}\mp@subsup{}{}{\prime}))
proof (simp only: dgrad-set-le-Un image-Un, rule)
show dgrad-set-le d (pp-of-term' Keys (set fs))
(pp-of-term'Keys (set fs)\cup(pp-of-term'Keys (set gs) \cup pp-of-term'
insert v(set vs')))
by (rule dgrad-set-le-subset, blast)
next
have dgrad-set-le d (pp-of-term'{v})(pp-of-term'(Keys (set gs) U insert v
(set vs')))
by (rule dgrad-set-le-subset, simp)

```
moreover from assms(1) have dgrad-set-le d (pp-of-term'set (fst ? \(n\) )) ( \(p\)-of-term' \((\) Keys \((\) set gs \() \cup\) insert \(v(\) set vs' \())\) )
by (rule fst-sym-preproc-addnew-dgrad-set-le)
ultimately have dgrad-set-le d (pp-of-term' \((\{v\} \cup\) set \((f s t\) ? \(n))\) ) (pp-of-term ' \(\left(\right.\) Keys \((\) set gs \() \cup\) insert \(v\left(\right.\) set \(\left.\left.\left.v s^{\prime}\right)\right)\right)\)
by (simp only: dgrad-set-le-Un image-Un)
also have dgrad-set-le d (pp-of-term' (Keys (set gs) \(\cup\) insert \(v(\) set vs' \()\) )) \((p p\)-of-term' \((\) Keys \((\) set \(f s) \cup(\) Keys \((\) set gs \() \cup\) insert \(v\) \(\left(\right.\) set \(\left.\left.\left.v s^{\prime}\right)\right)\right)\) )
by (rule dgrad-set-le-subset, blast)
finally show dgrad-set-le \(d\) ( \(p p\)-of-term 'insert \(v(\) set \((f s t\) ? \(n))\) )
( \(p\) p-of-term'Keys \((\) set \(f s) \cup(p p-o f-t e r m ‘ K e y s ~(s e t ~\)
\(g s) \cup p p\)-of-term' insert \(v\left(\right.\) set \(\left.\left.\left.v s^{\prime}\right)\right)\right)\)
by (simp add: image-Un)
qed
finally show dgrad-set-le d (pp-of-term'Keys (set (snd ?n))) (pp-of-term' Keys \((\) set gs) \(\cup\) pp-of-term 'Keys \((\) set fs \()\) )
by (simp only: set-vs eq, metis eq image-Un)
qed
finally show ?case .
qed
lemma components-snd-sym-preproc-aux-subset:
assumes set vs \(\subseteq\) Keys (set (fs::('t \(\Rightarrow_{0}{ }^{\prime} b::\) semiring-1-no-zero-divisors \()\) list \()\) )
shows component-of-term'Keys (set (snd (sym-preproc-aux gs ks (vs, fs)))) \(\subseteq\) component-of-term'Keys (set gs \(\cup\) set fs)
using assms
proof (induct fs rule: sym-preproc-aux-induct)
case (base \(k s f s\) )
show ?case by (simp add: Keys-Un image-Un)
next
case (rec ks vs fs v vs')
let \({ }^{2} n=\) sym-preproc-addnew gs \(v s\) ' fs \(v\)
from \(\operatorname{rec}(1)\) have \(v \in\) set \(v s\) by (simp add: rec(2))
hence set-vs: insert \(v\left(\right.\) set \(\left.v s^{\prime}\right)=\) set \(v s\) by (auto simp add: rec(3))
from rec (5) have eq: Keys \((\) set fs \() \cup(\) Keys \((\) set gs \() \cup\) set vs \()=\) Keys \((\) set gs \() \cup\)
Keys (set fs)
by blast
have component-of-term 'Keys (set (snd (sym-preproc-aux gs (ks @ [v]) ?n)) ) \(\subseteq\)
component-of-term'Keys (set gs \(\cup\) set (snd ?n))
proof (rule rec(4))
have set \((f s t ? n) \subseteq\) Keys \((\) set \((\) snd \(? n)) \cup\) insert \(v(\) set vs \()\)
by (simp only: Keys-snd-sym-preproc-addnew, blast)
also have \(\ldots=\) Keys \((\) set \((\) snd ? \(n)) \cup(\) set vs) by (simp only: set-vs)
also have \(\ldots \subseteq\) Keys \((\) set \((\) snd ? \(n)\) )
proof -
\{
fix \(u\)
assume \(u \in\) set vs
```

            with rec(5) have u\inKeys (set fs) ..
            then obtain f}\mathrm{ where f}\in\mathrm{ set fs and u}\in\mathrm{ keys f by (rule in-KeysE)
            from this(1) snd-sym-preproc-addnew-superset have f}\in\mathrm{ set (snd ?n) ..
            with }\langleu\inkeys f> have u\inKeys (set (snd ?n)) by (rule in-KeysI
        }
            thus ?thesis by auto
        qed
        finally show set (fst ? n)\subseteqKeys (set (snd ?n)).
    qed
    also have ...\subseteq component-of-term'Keys (set gs \cup set fs)
    proof (simp only: image-Un Keys-Un Un-subset-iff, rule, fact Un-upper1)
    have component-of-term'Keys (set (snd ?n)) \subseteqcomponent-of-term '(Keys
    (set fs) \cupinsert v(set (fst ?n)))
by (auto simp only: Keys-snd-sym-preproc-addnew[symmetric])
also have ... \subseteq component-of-term'Keys (set fs) \cup component-of-term '(Keys
(set gs) \cup insert v (set vs'))
proof (simp only:Un-subset-iff image-Un, rule, fact Un-upper1)
have component-of-term'{v}\subseteqcomponent-of-term'(Keys (set gs) \cup insert
v(set vs'))
by simp
moreover have component-of-term'set (fst ?n) \subseteqcomponent-of-term '(Keys
(set gs) \cup insert v(set vs'))
by (rule components-fst-sym-preproc-addnew-subset)
ultimately have component-of-term'' }{v}\cup\mathrm{ set (fst ? n) ) ¢ component-of-term
'(Keys (set gs) \cup insert v (set vs'))
by (simp only:Un-subset-iff image-Un)
also have component-of-term '(Keys (set gs) \cup insert v (set vs'))\subseteq
component-of-term'(Keys (set fs) \cup(Keys (set gs) \cup insert v
(set vs')))
by blast
finally show component-of-term'insert v (set (fst ?n))\subseteq
component-of-term'Keys (set fs) \cup
(component-of-term'Keys (set gs) \cup component-of-term' insert
v(set vs'))
by (simp add: image-Un)
qed
finally show component-of-term 'Keys (set (snd ?n)) \subseteq
component-of-term' Keys (set gs) \cup component-of-term'Keys (set
fs)
by (simp only: set-vs eq, metis eq image-Un)
qed
finally show ?case .
qed
lemma snd-sym-preproc-aux-complete:
assumes }\bigwedge\mp@subsup{u}{}{\prime}\mp@subsup{g}{}{\prime}.\mp@subsup{u}{}{\prime}\in\mathrm{ Keys (set fs) }\Longrightarrow\mp@subsup{u}{}{\prime}\not\in\mathrm{ set vs }\Longrightarrow\mp@subsup{g}{}{\prime}\in\mathrm{ set gs }\Longrightarrowlt g
addst}\mp@subsup{|}{}{\prime}
monom-mult 1 (pp-of-term u' - lp g') g}\mp@subsup{g}{}{\prime}\in\mathrm{ set fs
assumes u\inKeys (set (snd (sym-preproc-aux gs ks (vs,fs)))) and g set gs

```
and \(l t g a d d s_{t} u\)
shows monom-mult ( \(1::\) 'b::semiring-1-no-zero-divisors) (pp-of-term \(u-l p g) g\) \(\epsilon\)
set (snd (sym-preproc-aux gs ks (vs, fs)))
using assms
proof (induct fs rule: sym-preproc-aux-induct)
case (base ks fs)
from base(2) have \(u \in\) Keys (set fs) by simp
from this - base(3, 4) have monom-mult 1 (pp-of-term \(u-l p g) g \in \operatorname{set} f s\)
proof (rule base(1))
show \(u \notin\) set [] by simp
qed
thus?case by simp
next
case (rec ks vs fs v vs')
from \(\operatorname{rec}(1)\) have \(v \in\) set \(v s\) by ( \(\operatorname{simp}\) add: rec(2))
hence set-ts: set \(v s=\) insert \(v\left(\right.\) set \(\left.v s^{\prime}\right)\) by (auto simp add: rec(3))
let \(? n=\) sym-preproc-addnew gs \(v s^{\prime} f s v\)
from \(-\operatorname{rec}(6,7,8)\) show ?case
proof (rule rec(4))
fix \(v^{\prime} g^{\prime}\)
assume \(v^{\prime} \in\) Keys \((\) set \((s n d ? n))\) and \(v^{\prime} \notin\) set \((f s t ? n)\) and \(g^{\prime} \in\) set \(g s\) and \(l t\)
\(g^{\prime}\) adds \(s_{t} v^{\prime}\)
from this(1) Keys-snd-sym-preproc-addnew have \(v^{\prime} \in\) Keys \((\) set \(f s) \cup\) insert \(v\) (set (fst ?n))
by blast
with \(\left\langle v^{\prime} \notin \operatorname{set}(f s t ? n)\right\rangle\) have disj: \(v^{\prime} \in\) Keys (set fs) \(\vee v^{\prime}=v\) by blast
show monom-mult 1 (pp-of-term \(\left.v^{\prime}-l p g^{\prime}\right) g^{\prime} \in\) set (snd ? \(n\) )
proof (cases \(v^{\prime}=v\) )
case True
from \(\left\langle g^{\prime} \in\right.\) set \(\left.g s\right\rangle\left\langle l t g^{\prime} a d d s_{t} v^{\prime}\right\rangle\) show ?thesis
unfolding True by (rule sym-preproc-addnew-complete)
next
case False
with disj have \(v^{\prime} \in\) Keys (set fs) by simp
moreover have \(v^{\prime} \notin\) set vs
proof
assume \(v^{\prime} \in\) set \(v s\)
hence \(v^{\prime} \in\) set \(v s^{\prime}\) using False by (simp add: rec(3))
with \(f\) st-sym-preproc-addnew-superset have \(v^{\prime} \in\) set (fst ?n) ..
with \(\left\langle v^{\prime} \notin \operatorname{set}(f s t ? n)\right\rangle\) show False ..
qed
ultimately have monom-mult 1 (pp-of-term \(\left.v^{\prime}-l p g^{\prime}\right) g^{\prime} \in\) set \(f s\)
using \(\left\langle g^{\prime} \in\right.\) set \(\left.g s\right\rangle\left\langle l t g^{\prime}\right.\) addst \(\left.v^{\prime}\right\rangle\) by (rule rec(5)) with snd-sym-preproc-addnew-superset show ?thesis ..
qed
qed
qed
definition sym-preproc :: (' \(t \Rightarrow_{0}{ }^{\prime} b::\) semiring-1) list \(\Rightarrow\left(^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\left({ }^{\prime} t\right.\) list \(\times\) ( \(' t \Rightarrow_{0}{ }^{\prime} b\) ) list)
where sym-preproc gs \(f s=\) sym-preproc-aux gs [] (Keys-to-list fs, fs)
lemma sym-preproc-Nil [simp]: sym-preproc gs []\(=([],[])\)
by (simp add: sym-preproc-def)
lemma fst-sym-preproc:
fst (sym-preproc gs fs) \(=\) Keys-to-list (snd (sym-preproc gs ( \(f s::\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) semiring-1-no-zero-divisors) \()\)
list)))
proof -
let \(? a=f s t(\) sym-preproc \(g s f s)\)
let \(? b=\) Keys-to-list (snd (sym-preproc gs fs))
have antisymp \(\left(\succ_{t}\right)\) unfolding antisymp-def by fastforce
have irreflp \(\left(\succ_{t}\right)\) by (simp add: irreflp-def)
moreover have transp \(\left(\succ_{t}\right)\) unfolding transp-def by fastforce
moreover have s1: sorted-wrt \(\left(\succ_{t}\right)\) ?a unfolding sym-preproc-def
by (rule fst-sym-preproc-aux-sorted-wrt, simp-all)
ultimately have d1: distinct ?a by (rule distinct-sorted-wrt-irrefl)
have s2: sorted-wrt \(\left(\succ_{t}\right)\) ?b by (fact Keys-to-list-sorted-wrt)
with \(\left\langle\right.\) irreflp \(\left.\left(\succ_{t}\right)\right\rangle\left\langle\right.\) transp \(\left.\left(\succ_{t}\right)\right\rangle\) have d2: distinct ?b by (rule distinct-sorted-wrt-irrefl)
from 〈antisymp \(\left.\left(\succ_{t}\right)\right\rangle\) s1 d1 s2 d2 show ?thesis
proof (rule sorted-wrt-distinct-set-unique)
show set ?a \(=\) set ?b unfolding set-Keys-to-list sym-preproc-def
by (rule fst-sym-preproc-aux-complete, simp add: set-Keys-to-list)
qed
qed
lemma snd-sym-preproc-superset: set \(f s \subseteq\) set (snd (sym-preproc gs fs))
by (simp only: sym-preproc-def snd-conv, fact snd-sym-preproc-aux-superset)
lemma in-snd-sym-preprocE:
assumes \(p \in \operatorname{set}\) (snd (sym-preproc gs fs))
assumes 1: \(p \in\) set \(f s \Longrightarrow\) thesis
assumes 2: \(\bigwedge g t . g \in\) set \(g s \Longrightarrow p=\) monom-mult \(1 t g \Longrightarrow\) thesis
shows thesis
using assms unfolding sym-preproc-def snd-conv by (rule in-snd-sym-preproc-auxE)
lemma snd-sym-preproc-pmdl: pmdl \((\) set \(g s \cup\) set \((\) snd \((\) sym-preproc gs \(f s)))=\) \(p m d l(\) set \(g s \cup\) set \(f s)\)
unfolding sym-preproc-def snd-conv by (fact snd-sym-preproc-aux-pmdl)
lemma snd-sym-preproc-dgrad-set-le:
assumes dickson-grading d
shows dgrad-set-le d (pp-of-term'Keys (set (snd (sym-preproc gs fs))))
(pp-of-term'Keys (set gs \(\cup\) set ( \(f s::\left(' t \nRightarrow_{0}\right.\) 'b::semiring-1-no-zero-divisors)
list)))
unfolding sym-preproc-def snd-conv using assms
proof (rule snd-sym-preproc-aux-dgrad-set-le)
show set \((\) Keys-to-list \(f s) \subseteq\) Keys (set fs) by (simp add: set-Keys-to-list)
qed
corollary snd-sym-preproc-dgrad-p-set-le:
assumes dickson-grading d
shows dgrad-p-set-le d (set (snd (sym-preproc gs fs))) (set gs \(\cup\) set \(\left(f s::\left({ }^{\prime} t \Rightarrow_{0}\right.\right.\) 'b:::semiring-1-no-zero-divisors) list))
unfolding dgrad-p-set-le-def
proof -
from assms show dgrad-set-le d (pp-of-term'Keys (set (snd (sym-preproc gs \(f s)))\) ) (pp-of-term 'Keys \((\) set \(g s \cup\) set \(f s))\)
by (rule snd-sym-preproc-dgrad-set-le)
qed
lemma components-snd-sym-preproc-subset:
component-of-term'Keys (set (snd (sym-preproc gs fs))) \(\subseteq\) component-of-term'Keys (set gs \(\cup\) set ( \(f s::\left(' t \nRightarrow_{0}{ }^{\prime} b::\right.\) semiring-1-no-zero-divisors)
list))
unfolding sym-preproc-def snd-conv
by (rule components-snd-sym-preproc-aux-subset, simp add: set-Keys-to-list)
lemma snd-sym-preproc-complete:
assumes \(v \in\) Keys (set (snd (sym-preproc gs fs))) and \(g \in\) set gs and lt \(g\) adds \(s_{t}\) \(v\)
shows monom-mult ( \(1::\) 'b::semiring-1-no-zero-divisors) ( \(p p\)-of-term \(v-l p g\) ) g \(\in \operatorname{set}(\) snd (sym-preproc gs fs))
using - assms unfolding sym-preproc-def snd-conv
proof (rule snd-sym-preproc-aux-complete)
fix \(u^{\prime}\) and \(g^{\prime}:::^{\prime} t \Rightarrow{ }^{\prime} b\)
assume \(u^{\prime} \in\) Keys (set fs) and \(u^{\prime} \notin\) set (Keys-to-list fs)
thus monom-mult 1 (pp-of-term \(\left.u^{\prime}-l p g^{\prime}\right) g^{\prime} \in\) set \(f s\) by (simp add: set-Keys-to-list)
qed
end

\section*{16.2 lin-red}
context ordered-term
begin
definition lin-red :: (' \(t \Rightarrow_{0}{ }^{\prime} b::\) field \()\) set \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\) bool where lin-red \(F\) p \(q \equiv(\exists f \in F\). red-single \(p q f 0)\)
lin-red is a restriction of red, where the reductor \((f)\) may only be multiplied by a constant factor, i. e. where the power-product is 0 .

\section*{lemma lin-redI:}
assumes \(f \in F\) and red-single \(p q f 0\)
shows lin-red \(F p q\)
unfolding lin-red-def using assms ..
lemma lin-redE:
assumes lin-red \(F\) p \(q\)
obtains \(f:::^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\) field where \(f \in F\) and red-single \(p q f 0\)
proof -
from assms obtain \(f\) where \(f \in F\) and \(t\) : red-single p qf 0 unfolding lin-red-def by blast
thus ?thesis ..
qed
lemma lin-red-imp-red:
assumes lin-red \(F p q\)
shows red \(F p q\)
proof -
from assms obtain \(f\) where \(f \in F\) and red-single \(p q f 0\) by (rule lin-redE)
thus ?thesis by (rule red-setI)
qed
lemma lin-red-Un: lin-red \((F \cup G) p q=(\) lin-red \(F p q \vee\) lin-red \(G p q)\)
proof
assume lin-red \((F \cup G) p q\)
then obtain \(f\) where \(f \in F \cup G\) and \(r\) : red-single \(p q f 0\) by (rule lin-redE)
from this(1) show lin-red \(F p q \vee\) lin-red \(G p q\)
proof
assume \(f \in F\)
from this \(r\) have lin-red \(F p q\) by (rule lin-redI)
thus ?thesis..
next
assume \(f \in G\)
from this \(r\) have lin-red \(G p q\) by (rule lin-redI)
thus ?thesis ..
qed
next
assume lin-red \(F p q \vee\) lin-red \(G p q\)
thus lin-red \((F \cup G) p q\)
proof
assume lin-red \(F p q\)
then obtain \(f\) where \(f \in F\) and \(r\) : red-single \(p q f 0\) by (rule lin-redE)
from this(1) have \(f \in F \cup G\) by simp
from this \(r\) show ?thesis by (rule lin-redI)
next
assume lin-red \(G p q\)
then obtain \(g\) where \(g \in G\) and \(r\) : red-single \(p q g 0\) by (rule lin-red \(E\) )
from this(1) have \(g \in F \cup G\) by simp
from this \(r\) show ?thesis by (rule lin-redI)
qed
qed
```

lemma lin-red-imp-red-rtrancl:
assumes(lin-red F)** pq
shows (red F)** p q
using assms
proof induct
case base
show ?case ..
next
case (step y z)
from step(2) have red Fyz by (rule lin-red-imp-red)
with step(3) show ?case ..
qed
lemma phull-closed-lin-red:
assumes phull B\subseteqphull A and p\in phull A and lin-red B pq
shows q\in phull A
proof -
from assms(3) obtain f}\mathrm{ where f}\inB\mathrm{ and red-single pqf0 by (rule lin-redE)
hence q: q=p-(lookup p (lt f)/lcf).f
by (simp add: red-single-def term-simps map-scale-eq-monom-mult)
have }q-p\in\mathrm{ phull B
by (simp add: q, rule phull.span-neg, rule phull.span-scale, rule phull.span-base,
fact <f \in B`)
with assms(1) have q-p\in phull A ..
from this assms(2) have (q-p)+p\in phull A by (rule phull.span-add)
thus ?thesis by simp
qed

```

\subsection*{16.3 Reduction}
definition Macaulay-red \(::{ }^{\prime} t\) list \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) field \()\) list
where Macaulay-red vs \(f s=\)
(let lts \(=\) map lt \((\) filter \((\lambda p . p \neq 0)\) fs \()\) in
filter \((\lambda p . p \neq 0 \wedge l t p \notin\) set lts) (mat-to-polys vs (row-echelon (polys-to-mat vs \(f s)\) ))
)
Macaulay-red vs \(f s\) auto-reduces (w.r.t. lin-red) the given list \(f s\) and returns those non-zero polynomials whose leading terms are not in lt-set (set fs). Argument \(v s\) is expected to be Keys-to-list \(f s\); this list is passed as an argument to Macaulay-red, because it can be efficiently computed by symbolic preprocessing.
```

lemma Macaulay-red-alt:
Macaulay-red (Keys-to-list fs) $f s=$ filter $(\lambda p$. lt $p \notin l t$-set (set fs)) (Macaulay-list
fs)
proof -
have $\{x \in$ set fs. $x \neq 0\}=$ set $f s-\{0\}$ by blast
thus ?thesis by (simp add: Macaulay-red-def Macaulay-list-def Macaulay-mat-def
lt-set-def Let-def)

```

\section*{qed}
lemma set-Macaulay-red:
set (Macaulay-red (Keys-to-list fs) fs) \(=\) set (Macaulay-list fs) \(-\{p\).lt \(p \in\) lt-set ( set fs) \}
by (auto simp add: Macaulay-red-alt)
lemma Keys-Macaulay-red: Keys \((\) set (Macaulay-red \((\) Keys-to-list fs) fs) \() \subseteq\) Keys ( set fs)
proof -
have Keys \((\) set \((\) Macaulay-red \((\) Keys-to-list \(f s) f s)) \subseteq\) Keys \((\) set (Macaulay-list fs))
unfolding set-Macaulay-red by (fact Keys-minus)
also have \(\ldots \subseteq\) Keys \((\) set \(f s\) ) by (fact Keys-Macaulay-list) finally show? thesis.
qed
end
context \(g d\)-term
begin
lemma Macaulay-red-reducible:
assumes \(f \in\) phull (set \(f s\) ) and \(F \subseteq\) set \(f s\) and lt-set \(F=l t\)-set (set fs)
shows (lin-red \((F \cup\) set \((\) Macaulay-red \((\) Keys-to-list \(f s) f s)))^{* *} f 0\)
proof -
define \(A\) where \(A=F \cup\) set \((\) Macaulay-red \((\) Keys-to-list \(f s) f s)\)
have phull-A: phull \(A \subseteq\) phull (set fs)
proof (rule phull.span-subset-spanI, simp add: A-def, rule)
have \(F \subseteq\) phull \(F\) by (rule phull.span-superset)
also from \(\operatorname{assms}(2)\) have \(\ldots \subseteq\) phull (set fs) by (rule phull.span-mono)
finally show \(F \subseteq\) phull (set fs).
next
have set (Macaulay-red (Keys-to-list fs) \(f s\) ) \(\subseteq\) set (Macaulay-list fs)
by (auto simp add: set-Macaulay-red)
also have \(\ldots \subseteq\) phull (set (Macaulay-list fs)) by (rule phull.span-superset)
also have \(\ldots=\) phull \((\) set \(f s)\) by (rule phull-Macaulay-list)
finally show set (Macaulay-red (Keys-to-list fs) fs) \(\subseteq\) phull (set fs) .
qed
have \(l t-A: p \in \operatorname{phull}(\) set \(f s) \Longrightarrow p \neq 0 \Longrightarrow(\bigwedge g . g \in A \Longrightarrow g \neq 0 \Longrightarrow l t g=l t\)
\(p \Longrightarrow\) thesis \(\Longrightarrow\) thesis
for \(p\) thesis
proof -
assume \(p \in\) phull (set fs) and \(p \neq 0\)
then obtain \(g\) where \(g\)-in: \(g \in \operatorname{set}\) (Macaulay-list fs) and \(g \neq 0\) and \(l t p=\) lt \(g\)
by (rule Macaulay-list-lt)
```

    assume \(*: \bigwedge g . g \in A \Longrightarrow g \neq 0 \Longrightarrow\) lt \(g=\) lt \(p \Longrightarrow\) thesis
    show ?thesis
    proof (cases \(g \in \operatorname{set}(\) Macaulay-red (Keys-to-list fs) fs))
        case True
        hence \(g \in A\) by (simp add: \(A\)-def)
        from this \(\langle g \neq 0\rangle\langle l t p=l t g\rangle[\) symmetric \(]\) show ?thesis by (rule *)
    next
        case False
        with \(g\)-in have lt \(g \in l t\)-set (set fs) by (simp add: set-Macaulay-red)
        also have \(\ldots=l\) t-set \(F\) by (simp only: assms(3))
    finally obtain \(g^{\prime}\) where \(g^{\prime} \in F\) and \(g^{\prime} \neq 0\) and \(l t g^{\prime}=l t g\) by (rule lt-setE)
    from this(1) have \(g^{\prime} \in A\) by (simp add: A-def)
        moreover note \(\left\langle g^{\prime} \neq 0\right\rangle\)
        moreover have \(l t g^{\prime}=l t p\) by (simp only: \(\left.\langle l t p=l t g\rangle\left\langle l t g^{\prime}=l t g\right\rangle\right)\)
        ultimately show ?thesis by (rule *)
    qed
    qed
from assms(2) finite-set have finite $F$ by (rule finite-subset)
from this finite-set have fin-A: finite $A$ unfolding $A$-def by (rule finite-UnI)
from ex-dgrad obtain $d::^{\prime} a \Rightarrow$ nat where $d g$ : dickson-grading $d$..
from fin- $A$ have finite (insert $f A$ )..
then obtain $m$ where insert $f(\subseteq d g r a d-p$-set $d m$ by (rule dgrad-p-set-exhaust)
hence $A$-sub: $A \subseteq d g r a d$-p-set $d m$ and $f \in d g r a d$ - $p$-set $d m$ by simp-all
from $d g$ have $w f P$ (dickson-less-p $d m$ ) by (rule wf-dickson-less-p)
from this assms $(1)\langle f \in$ dgrad-p-set $d m\rangle$ show (lin-red $A)^{* *} f 0$
proof (induct f)
fix $p$
assume $I H: \bigwedge q$. dickson-less-p d $m q p \Longrightarrow q \in$ phull (set fs) $\Longrightarrow q \in$ dgrad- $p$-set
$d m \Longrightarrow$
$(\operatorname{lin-red} A)^{* *} q 0$
and $p \in$ phull $($ set $f s)$ and $p \in d g r a d-p-s e t ~ d ~ m$
show (lin-red $A$ )** p 0
proof (cases $p=0$ )
case True
thus ?thesis by simp
next
case False
with $\langle p \in \operatorname{phull}($ set $f s)\rangle$ obtain $g$ where $g \in A$ and $g \neq 0$ and $l t g=l t p$
by (rule lt-A)
define $q$ where $q=p$ - monom-mult (lc $p / l c g$ ) $0 g$
from $\langle g \in A\rangle$ have $l r$ : lin-red A pq
proof (rule lin-redI)
show red-single $p$ q g 0
by (simp add: red-single-def $\langle l t ~ g=l t p\rangle l c-d e f[s y m m e t r i c] ~ q-d e f\langle g \neq 0\rangle$
lc-not- 0 [OF False] term-simps)
qed
moreover have (lin-red $A)^{* *}$ q 0

```
```

    proof -
            from lr have red: red A pq by (rule lin-red-imp-red)
            with dg A-sub <p\indgrad-p-set d m> have q\indgrad-p-set d m by (rule
    dgrad-p-set-closed-red)
moreover from red have q}\mp@subsup{\prec}{p}{}p\mathrm{ by (rule red-ord)
ultimately have dickson-less-p d m q p using <p \in dgrad-p-set d m>
by (simp add: dickson-less-p-def)
moreover from phull-A <p\in phull (set fs)> lr have q \in phull (set fs)
by (rule phull-closed-lin-red)
ultimately show ?thesis using <q\indgrad-p-set d m> by (rule IH)
qed
ultimately show ?thesis by fastforce
qed
qed
qed
primrec pdata-pairs-to-list :: ('t, 'b::field, 'c) pdata-pair list }=>('t>\mp@subsup{|}{0}{\prime}'b) lis
where
pdata-pairs-to-list [] = []|
pdata-pairs-to-list (p\# ps)=
(let f=fst (fst p);g= fst (snd p);lf = lp f;lg=lp g;l=lcslf lg in
(monom-mult (1 / lc f) (l-lf)f) \# (monom-mult (1 / lc g) (l - lg)g) \#
(pdata-pairs-to-list ps)
)
lemma in-pdata-pairs-to-listI1:
assumes (f,g) \in set ps
shows monom-mult (1 / lc (fst f)) ((lcs (lp (fst f)) (lp (fst g))) - (lp (fst f)))
(fst f) \in set (pdata-pairs-to-list ps) (is ?m \in -)
using assms
proof (induct ps)
case Nil
thus ?case by simp
next
case (Cons p ps)
from Cons(2) have p=(f,g)\vee (f,g)\in set ps by auto
thus ?case
proof
assume p=(f,g)
show ?thesis by (simp add: <p = (f,g)\rangle Let-def)
next
assume (f,g)\in set ps
hence ?m \in set (pdata-pairs-to-list ps) by (rule Cons(1))
thus ?thesis by (simp add: Let-def)
qed
qed
lemma in-pdata-pairs-to-listI2:
assumes (f,g)\in set ps

```
```

    shows monom-mult (1 / lc (fst g)) ((lcs (lp (fst f)) (lp (fst g))) - (lp (fst g)))
                            (fst g) \in set (pdata-pairs-to-list ps) (is ?m \in -)
    using assms
    proof (induct ps)
case Nil
thus ?case by simp
next
case (Cons p ps)
from Cons(2) have p=(f,g)\vee (f,g)\in set ps by auto
thus ?case
proof
assume p=(f,g)
show ?thesis by (simp add: }\langlep=(f,g)\rangle\mathrm{ Let-def)
next
assume (f,g)\in set ps
hence ?m \in set (pdata-pairs-to-list ps) by (rule Cons(1))
thus ?thesis by (simp add: Let-def)
qed
qed
lemma in-pdata-pairs-to-listE:
assumes h\in set (pdata-pairs-to-list ps)
obtains fg}\mathrm{ where (f,g) e set ps V (g,f) f set ps
and h=monom-mult (1 / lc (fst f)) ((lcs (lp (fst f)) (lp (fst g))) - (lp (fst
f))) (fst f)
using assms
proof (induct ps arbitrary: thesis)
case Nil
from Nil(2) show ?case by simp
next
case (Cons p ps)
let ?f = fst (fst p)
let ?g=fst (snd p)
let ?lf = lp ?f
let ?lg = lp ?g
let ?l = lcs ?lf ?lg
from Cons(3) have h= monom-mult (1 / lc ?f) (?l - ?lf) ?f \vee h=monom-mult
(1 / lc ?g) (?l - ?lg) ?g V
h\in set (pdata-pairs-to-list ps)
by (simp add: Let-def)
thus?case
proof (elim disjE)
assume h: h = monom-mult (1 / lc ?f) (?l - ?lf) ?f
have (fst p, snd p) \in set ( }p\#\mathrm{ \# ps) by simp
hence (fst p, snd p) \in set (p\# ps)\vee (snd p, fst p) \in set (p\# ps)..
from this h show ?thesis by (rule Cons(2))
next
assume h:h=monom-mult (1 / lc ?g) (?l - ?lg) ?g
have (fst p, snd p) \in set ( p\# ps) by simp

```
```

    hence (snd p, fst p)\in set (p# ps)\vee (fst p, snd p) \in set (p# ps)..
    moreover from h have h=monom-mult (1 / lc ?g) ((lcs ?lg ?lf) - ?lg) ?g
        by (simp only:lcs-comm)
    ultimately show ?thesis by (rule Cons(2))
    next
assume h-in: h\in set (pdata-pairs-to-list ps)
obtain fg}\mathrm{ where (f,g) \& set ps V (g,f) \& set ps
and h:h= monom-mult (1 / lc (fst f)) ((lcs (lp (fst f)) (lp (fst g))) - (lp
(fst f))) (fst f)
by (rule Cons(1), assumption, intro h-in)
from this(1) have (f,g)\in set ( }p\#ps)\vee(g,f)\in\operatorname{set}(p\#ps) by aut
from this h show ?thesis by (rule Cons(2))
qed
qed
definition f4-red-aux :: ('t, 'b::field, 'c) pdata list \# ('t, 'b, 'c) pdata-pair list =>
(' }t=
where f4-red-aux bs ps=
(let aux = sym-preproc (map fst bs) (pdata-pairs-to-list ps) in Macaulay-red
(fst aux) (snd aux))
$f_{4}$-red-aux only takes two arguments, since it does not distinguish between those elements of the current basis that are known to be a Gröbner basis (called gs in Groebner-Bases.Algorithm-Schema) and the remaining ones.
lemma f4-red-aux-not-zero: $0 \notin$ set (f4-red-aux bs ps)
by (simp add: f4-red-aux-def Let-def fst-sym-preproc set-Macaulay-red set-Macaulay-list)
lemma f4-red-aux-irredudible:
assumes $h \in$ set (f4-red-aux bs ps) and $b \in$ set bs and fst $b \neq 0$
shows $\neg l t(f s t b) a d d s_{t} l t h$
proof
from $\operatorname{assms}(1) f 4$-red-aux-not-zero have $h \neq 0$ by metis
hence lt $h \in$ keys $h$ by (rule lt-in-keys)
also from assms (1) have $\ldots \subseteq$ Keys (set (f4-red-aux bs ps)) by (rule keys-subset-Keys)
also have $\ldots \subseteq$ Keys (set (snd (sym-preproc (map fst bs) (pdata-pairs-to-list
$p s)$ ))
(is - $\subseteq$ Keys (set ?s)) by (simp only: f4-red-aux-def Let-def fst-sym-preproc
Keys-Macaulay-red)
finally have lt $h \in$ Keys (set ?s).
moreover from assms(2) have fst $b \in$ set (map fst bs) by auto
moreover assume $a$ : lt (fst b) adds ${ }_{t}$ lt $h$
ultimately have monom-mult $1(l p h-l p(f s t b))(f s t b) \in$ set ?s (is ? $m \in-)$
by (rule snd-sym-preproc-complete)
from $\operatorname{assms}(3)$ have $? m \neq 0$ by (simp add: monom-mult-eq-zero-iff)
with $\langle ? m \in$ set ? s $\rangle$ have $l t ? m \in l t$-set (set ?s) by (rule lt-setI)
moreover from $\operatorname{assms}(3) a$ have $l t ? m=l t h$
by (simp add: lt-monom-mult, metis add-diff-cancel-right' adds-termE pp-of-term-splus)
ultimately have $l t h \in l t$-set (set ?s) by simp
moreover from $\operatorname{assms}(1)$ have $l t h \notin l t$-set (set ?s)

```
```

    by (simp add: f4-red-aux-def Let-def fst-sym-preproc set-Macaulay-red)
    ultimately show False by simp
    qed
lemma f4-red-aux-dgrad-p-set-le:
assumes dickson-grading d
shows dgrad-p-set-le d (set (f4-red-aux bs ps)) (args-to-set ([], bs, ps))
unfolding dgrad-p-set-le-def dgrad-set-le-def
proof
fix }
assume s \in pp-of-term'Keys (set (f4-red-aux bs ps))
also have ... \subseteqpp-of-term' Keys (set (snd (sym-preproc (map fst bs) (pdata-pairs-to-list
ps))))
(is - \subseteqpp-of-term'Keys (set ?s))
by (rule image-mono, simp only: f4-red-aux-def Let-def fst-sym-preproc Keys-Macaulay-red)
finally have s\inpp-of-term'Keys (set ?s).
with snd-sym-preproc-dgrad-set-le[OF assms] obtain t
where t\inpp-of-term' Keys (set (map fst bs)\cup set (pdata-pairs-to-list ps))
and ds\leqdt
by (rule dgrad-set-leE)
from this(1) have t\inpp-of-term'Keys (fst'set bs) \vee t\inpp-of-term'Keys
(set (pdata-pairs-to-list ps))
by (simp add: Keys-Un image-Un)
thus \existst\inpp-of-term'Keys(args-to-set ([],bs,ps)).ds\leqdt
proof
assume t f pp-of-term 'Keys (fst' set bs)
also have ... \subseteqpp-of-term'Keys (args-to-set ([],bs, ps))
by (rule image-mono, rule Keys-mono, auto simp add: args-to-set-alt)
finally have t\inpp-of-term'Keys (args-to-set ([],bs, ps)).
with <d s\leqd t\rangle show ?thesis ..
next
assume t f pp-of-term' Keys (set (pdata-pairs-to-list ps))
then obtain p where p\in set (pdata-pairs-to-list ps) and t\inpp-of-term 'keys
p
by (auto elim: in-KeysE)
from this(1) obtain fg where disj: (f,g)\in set ps \vee (g,f)\in set ps
and p: p = monom-mult (1 / lc (fst f)) ((lcs (lp (fst f)) (lp (fst g))) - (lp
(fst f))) (fst f)
by (rule in-pdata-pairs-to-listE)
from disj have fst f\in\operatorname{args-to-set ([],bs,ps)}\wedgefst g}\in\operatorname{args-to-set ([],bs,ps)
proof
assume (f,g)\in set ps
hence f\infst' set ps and g\in snd ' set ps by force+
hence fst f\infst 'fst' set ps and fst g\infst'snd ' set ps by simp-all
thus ?thesis by (simp add: args-to-set-def image-Un)
next
assume (g,f) \in set ps
hence f}\in\mathrm{ snd ' set ps and g}\infst' set ps by force
hence fst f\infst' snd ' set ps and fst g\infst'fst' set ps by simp-all

```
```

        thus ?thesis by (simp add: args-to-set-def image-Un)
    qed
    hence fst f}\in\operatorname{args-to-set ([],bs,ps) and fst g}\in\operatorname{args-to-set ([],bs,ps) by
    simp-all
hence keys-f: keys (fst f)\subseteqKeys (args-to-set ([],bs, ps))
and keys-g: keys (fst g)\subseteq Keys (args-to-set ([],bs, ps))
by (auto intro!: keys-subset-Keys)
let ?lf = lp (fst f)
let ?lg=lp(fst g)
define l where l=lcs ?lf ?lg
have pp-of-term'keys p\subseteqpp-of-term'((\oplus)(lcs ?lf ?lg - ?lf)'keys (fst f))
unfolding }
using keys-monom-mult-subset by (rule image-mono)
with «t \inpp-of-term' keys p> have t f pp-of-term '((\oplus) (l-?lf)' keys (fst
f)) unfolding l-def ..
then obtain t' where t'
using pp-of-term-splus by fastforce
from this(1) have fst f}\not=0\mathrm{ by auto
show ?thesis
proof (cases fst g=0)
case True
hence ?lg = 0 by (simp add:lt-def min-term-def term-simps)
hence l=?lf by (simp add:l-def lcs-zero lcs-comm)
hence }t=\mp@subsup{t}{}{\prime}\mathrm{ by (simp add: t)
with <d s\leqd t> have ds\leqd t' by simp
moreover from <t' \in pp-of-term' keys (fst f)> keys-f have t'\inpp-of-term
` Keys (args-to-set ([], bs, ps))
by blast
ultimately show ?thesis ..
next
case False
have d t=d (l-?lf) \veed t=d t'
by (auto simp add: t dickson-gradingD1[OF assms])
thus ?thesis
proof
assume dt=d(l-?lf)
also from assms have ... \leq ord-class.max (d ?lf) (d ?lg)
unfolding l-def by (rule dickson-grading-lcs-minus)
finally have ds\leqd?lf \vee d s\leqd?lg using <ds \leqdt\rangle by auto
thus ?thesis
proof
assume d s\leqd?lf
moreover have lt (fst f)\in Keys (args-to-set ([],bs, ps))
by (rule, rule lt-in-keys, fact+)
ultimately show ?thesis by blast
next
assume ds\leqd?lg
moreover have lt (fst g) \in Keys (args-to-set ([], bs, ps))
by (rule, rule lt-in-keys, fact+)

```
```

                    ultimately show ?thesis by blast
            qed
        next
            assume d t=d t'
            with }\langleds\leqdt\rangle\mathrm{ have ds}s=d\mp@subsup{t}{}{\prime}\mathrm{ by simp
            moreover from <t'\inpp-of-term ' keys (fst f)> keys-f have t' \inpp-of-term
    ' Keys (args-to-set ([], bs, ps))
by blast
ultimately show ?thesis ..
qed
qed
qed
qed
lemma components-f4-red-aux-subset:
component-of-term'Keys (set (f4-red-aux bs ps)) \subseteqcomponent-of-term' Keys
(args-to-set ([],bs,ps))
proof
fix }
assume k}k\mathrm{ component-of-term'Keys (set (f4-red-aux bs ps))
also have ... \subseteq component-of-term'Keys (set (snd (sym-preproc (map fst bs)
(pdata-pairs-to-list ps))))
by (rule image-mono, simp only: f4-red-aux-def Let-def fst-sym-preproc Keys-Macaulay-red)
also have ...\subseteq component-of-term'Keys (set (map fst bs)\cup set (pdata-pairs-to-list
ps))
by (fact components-snd-sym-preproc-subset)
finally have k component-of-term 'Keys (fst'set bs) \cup component-of-term'
Keys (set (pdata-pairs-to-list ps))
by (simp add: image-Un Keys-Un)
thus k\incomponent-of-term'Keys (args-to-set ([],bs, ps))
proof
assume k component-of-term ' Keys (fst' set bs)
also have ... \subseteqcomponent-of-term' Keys (args-to-set ([],bs,ps))
by (rule image-mono, rule Keys-mono, auto simp add: args-to-set-alt)
finally show }k\in\mathrm{ component-of-term'Keys (args-to-set ([],bs,ps)).
next
assume k component-of-term 'Keys (set (pdata-pairs-to-list ps))
then obtain p where p\in set (pdata-pairs-to-list ps) and k\in component-of-term
` keys p
by (auto elim: in-KeysE)
from this(1) obtain fg where disj: (f,g)\in set ps\vee (g,f) \in set ps
and p:p = monom-mult (1 / lc (fst f)) ((lcs (lp (fst f)) (lp (fst g))) - (lp
(fst f))) (fst f)
by (rule in-pdata-pairs-to-listE)
from disj have fst f}\in\operatorname{args-to-set ([],bs, ps)
by (simp add: args-to-set-alt, metis fst-conv image-eqI snd-conv)
hence fst f}\in\mathrm{ args-to-set ([], bs, ps) by simp
hence keys-f: keys (fst f)\subseteq Keys (args-to-set ([],bs, ps))
by (auto intro!: keys-subset-Keys)

```
let \(? l f=l p\left(f_{s t} f\right)\)
let \(? l g=l p(f s t g)\)
define \(l\) where \(l=l c s ? l f ? l g\)
have component-of-term 'keys \(p \subseteq\) component-of-term ' ( \((\oplus)\) (lcs ?lf ?lg - ?lf)
‘ keys (fst f))
unfolding \(p\) using keys-monom-mult-subset by (rule image-mono)
with \(\langle k \in\) component-of-term'keys \(p\rangle\) have \(k \in\) component-of-term ' \(((\oplus)(l\)
- ?(f)'keys (fst f))
unfolding \(l\)-def ..
hence \(k \in\) component-of-term ' keys (fst f) using component-of-term-splus by fastforce
with keys-f show \(k \in\) component-of-term 'Keys (args-to-set ([], bs, ps)) by blast
qed
qed
lemma pmdl-f4-red-aux: set \((f 4\)-red-aux bs ps) \(\subseteq p m d l(\) args-to-set \(([], b s, p s))\)
proof -
have set \(\left(f_{4}\right.\)-red-aux bs \(\left.p s\right) \subseteq\)
set (Macaulay-list (snd (sym-preproc (map fst bs) (pdata-pairs-to-list ps))))
by (auto simp add: \(\mathrm{f}_{4}\)-red-aux-def Let-def fst-sym-preproc set-Macaulay-red)
also have \(\ldots \subseteq p m d l\) (set (Macaulay-list (snd (sym-preproc (map fst bs) (pdata-pairs-to-list
\(p s)\) )))
by (fact pmdl.span-superset)
also have \(\ldots=\) pmdl (set (snd (sym-preproc (map fst bs) (pdata-pairs-to-list \(p s)\) ))
by (fact pmdl-Macaulay-list)
also have \(\ldots \subseteq p m d l(\) set (map fst bs) \(\cup\)
set (snd (sym-preproc (map fst bs) (pdata-pairs-to-list ps))))
by (rule pmdl.span-mono, blast)
also have \(\ldots=p m d l(\) set \((\) map \(f s t b s) \cup\) set \((\) pdata-pairs-to-list ps \())\)
by (fact snd-sym-preproc-pmdl)
also have..\(\subseteq p m d l(\) args-to-set ( \([, b s, p s))\)
proof (rule pmdl.span-subset-spanI, simp only: Un-subset-iff, rule conjI)
have set (map fst bs) \(\subseteq\) args-to-set ( [ ], bs, ps) by (auto simp add: args-to-set-def)
also have \(\ldots \subseteq p m d l\) (args-to-set ( []\(, b s, p s)\) ) by (rule pmdl.span-superset)
finally show set \((\) map fst \(b s) \subseteq p m d l(\) args-to-set \(([], b s, p s))\).
next
show set \((\) pdata-pairs-to-list ps) \(\subseteq p m d l(\) args-to-set \(([], b s, p s))\)
proof
fix \(p\)
assume \(p \in\) set (pdata-pairs-to-list ps)
then obtain \(f g\) where \((f, g) \in\) set \(p s \vee(g, f) \in\) set \(p s\)
and \(p: p=\) monom-mult \((1 / l c(f s t f))((l c s(l p(f s t f))(l p(f s t g)))-(l p\)
(fst f))) (fst f)
by (rule in-pdata-pairs-to-listE)
from this(1) have \(f \in f s t\) ' set \(p s \cup\) snd ' set \(p s\) by force
hence \(f s t f \in\) args-to-set ( []\(, b s, p s)\) by (auto simp add: args-to-set-alt)
hence \(f_{s t} f \in \operatorname{pmdl}\) (args-to-set ([], bs, ps)) by (rule pmdl.span-base)
```

            thus p pmdl (args-to-set ([], bs, ps)) unfolding p by (rule pmdl-closed-monom-mult)
            qed
    qed
    finally show ?thesis .
    qed
lemma f4-red-aux-phull-reducible:
assumes set ps\subseteq set bs \times set bs
and f\in phull (set (pdata-pairs-to-list ps))
shows (red (fst' set bs U set (f4-red-aux bs ps)))** f 0
proof -
define fs where fs = snd (sym-preproc (map fst bs) (pdata-pairs-to-list ps))
have set (pdata-pairs-to-list ps)\subseteq set fs unfolding fs-def by (fact snd-sym-preproc-superset)
hence phull (set (pdata-pairs-to-list ps)) \subseteq phull (set fs) by (rule phull.span-mono)
with assms(2) have f-in: f\in phull (set fs) ..
have eq: (set fs) \cup set (f4-red-aux bs ps)=(set fs)\cup set (Macaulay-red (Keys-to-list
fs) fs)
by (simp add: f4-red-aux-def fs-def Let-def fst-sym-preproc)
have (lin-red ((set fs) \cup set (f4-red-aux bs ps)))** f 0
by (simp only: eq, rule Macaulay-red-reducible, fact f-in, fact subset-refl, fact
refl)
thus ?thesis
proof induct
case base
show ?case ..
next
case (step y z)
from step(2) have red (fst'set bs U set (f4-red-aux bs ps)) y z unfolding
lin-red-Un
proof
assume lin-red (set fs) yz
then obtain a where }a\in\mathrm{ set fs and r: red-single y z a 0 by (rule lin-redE)
from this(1) obtain b ct where b\infst'set bs and a: a = monom-mult c
tb unfolding fs-def
proof (rule in-snd-sym-preprocE)
assume *: \b c t. b fst'set bs \Longrightarrowa= monom-mult c t b\Longrightarrow thesis
assume a \in set (pdata-pairs-to-list ps)
then obtain fg}\mathrm{ where (f,g) \& set ps }\vee(g,f)\in\mathrm{ set ps
and a: a = monom-mult (1 / lc (fst f)) ((lcs (lp (fst f)) (lp (fst g))) - (lp
(fst f))) (fst f)
by (rule in-pdata-pairs-to-listE)
from this(1) have f\infst'set ps U snd ' set ps by force
with assms(1) have f}\in\mathrm{ set bs by fastforce
hence fst f}\inf\mathrm{ fst' set bs by simp
from this a show ?thesis by (rule *)
next
fix gs
assume *: \b c t. b f fst' set bs \Longrightarrowa= monom-mult c t b \Longrightarrow thesis

```
```

            assume g\in set (map fst bs)
            hence g}\infst'set bs by sim
            moreover assume a= monom-mult 1sg
            ultimately show ?thesis by (rule *)
            qed
                            from r have c\not=0 and b\not=0 by (simp-all add: a red-single-def monom-mult-eq-zero-iff)
            from }r\mathrm{ have red-single yzbt
            by (simp add: a red-single-def monom-mult-eq-zero-iff lt-monom-mult[OF <c
    
# 0\rangle\langleb\not=0\rangle]

                                    monom-mult-assoc term-simps)
            with }\langleb\infst' set bs` have red (fst' set bs) y z by (rule red-setI)
            thus ?thesis by (rule red-unionI1)
    next
            assume lin-red (set (f4-red-aux bs ps)) y z
            hence red (set (f4-red-aux bs ps)) y z by (rule lin-red-imp-red)
            thus ?thesis by (rule red-unionI2)
    qed
    with step(3) show ?case ..
    qed
    qed
corollary f4-red-aux-spoly-reducible:
assumes set ps\subseteqset bs \times set bs and (p,q)\in set ps
shows (red (fst 'set bs \cup set (f4-red-aux bs ps)))** (spoly (fst p) (fst q)) 0
using assms(1)
proof (rule f4-red-aux-phull-reducible)
let ?lt = lp (fst p)
let ?lq = lp (fst q)
let ?l = lcs ?lt ?lq
let ?p = monom-mult (1 / lc (fst p)) (?l - ?lt) (fst p)
let ?q = monom-mult (1 / lc (fst q)) (?l - ?lq) (fst q)
from assms(2) have ?p\in set (pdata-pairs-to-list ps) and ?q { set (pdata-pairs-to-list
ps)
by (rule in-pdata-pairs-to-listI1, rule in-pdata-pairs-to-listI2)
hence ?p f phull (set (pdata-pairs-to-list ps)) and ?q \in phull (set (pdata-pairs-to-list
ps))
by (auto intro: phull.span-base)
hence ?p - ?q\in phull (set (pdata-pairs-to-list ps)) by (rule phull.span-diff)
thus spoly (fst p) (fst q) \in phull (set (pdata-pairs-to-list ps))
by (simp add: spoly-def Let-def phull.span-zero lc-def split: if-split)
qed
definition f4-red :: ('t, 'b::field, 'c::default, 'd) complT
where f4-red gs bs ps sps data = (map (\lambdah. (h, default)) (f4-red-aux (gs @ bs)
sps), snd data)
lemma fst-set-fst-f4-red: fst' set (fst (f4-red gs bs ps sps data)) = set (f4-red-aux
(gs @ bs) sps)
by (simp add: f4-red-def, force)

```
```

lemma rcp-spec-f4-red: rcp-spec f4-red
proof (rule rcp-specI)
fix gs bs::('t, 'b, 'c) pdata list and ps sps and data::nat \times 'd
show 0 \& fst ' set (fst (f4-red gs bs ps sps data))
by (simp add: fst-set-fst-f4-red f4-red-aux-not-zero)
next
fix gs bs::('t, 'b, 'c) pdata list and ps sps hb and data::nat }\times\mathrm{ 'd
assume h\in set (fst (f4-red gs bs ps sps data)) and b\in set gs U set bs
from this(1) have fst h\infst'set (fst (f4-red gs bs ps sps data)) by simp
hence fst h \in set (f4-red-aux (gs @ bs) sps) by (simp only: fst-set-fst-f4-red)
moreover from <b bet gs \cup set bs> have b\in set (gs @ bs) by simp
moreover assume fst b\not=0
ultimately show }\neglt (fst b) addst lt (fst h) by (rule f4-red-aux-irredudible
next
fix gs bs::('t, 'b, 'c) pdata list and ps sps and d::'a => nat and data::nat × 'd
assume dickson-grading d
hence dgrad-p-set-le d (set (f4-red-aux (gs @ bs) sps)) (args-to-set ([],gs @ bs,
sps))
by (fact f4-red-aux-dgrad-p-set-le)
also have ... = args-to-set (gs, bs, sps) by (simp add: args-to-set-alt image-Un)
finally show dgrad-p-set-le d (fst ' set (fst (f4-red gs bs ps sps data))) (args-to-set
(gs, bs, sps))
by (simp only: fst-set-fst-f4-red)
next
fix gs bs::('t, 'b, 'c) pdata list and ps sps and data::nat × 'd
have component-of-term'Keys(set (f4-red-aux (gs @ bs) sps)) \subseteq
component-of-term 'Keys(args-to-set ([], gs @ bs, sps))
by (fact components-f4-red-aux-subset)
also have ... = component-of-term'Keys (args-to-set (gs, bs, sps))
by (simp add: args-to-set-alt image-Un)
finally show component-of-term ' Keys (fst 'set (fst (f4-red gs bs ps sps data)))
\subseteq
component-of-term' Keys (args-to-set (gs, bs, sps))
by (simp only: fst-set-fst-f4-red)
next
fix gs bs::('t, 'b, 'c) pdata list and ps sps and data::nat \times 'd
have set (f4-red-aux (gs @ bs) sps)\subseteqpmdl (args-to-set ([],gs @ bs, sps))
by (fact pmdl-f4-red-aux)
also have .. = pmdl (args-to-set (gs, bs, sps)) by (simp add: args-to-set-alt
image-Un)
finally have fst 'set (fst (f4-red gs bs ps sps data))\subseteqpmdl (args-to-set (gs, bs,
sps))
by (simp only: fst-set-fst-f4-red)
moreover {
fix pq::('t, 'b,'c) pdata
assume set sps \subseteq set bs }\times(\mathrm{ set gs U set bs)
hence set sps\subseteq set (gs @ bs) > set (gs @ bs) by fastforce
moreover assume (p,q)\in set sps

```
```

    ultimately have (red (fst'set (gs@ bs)\cup set (f4-red-aux (gs @ bs) sps)))**
    (spoly (fst p) (fst q)) 0
by (rule f4-red-aux-spoly-reducible)
}
ultimately show
fst'set (fst (f4-red gs bs ps sps data))\subseteqpmdl (args-to-set (gs,bs, sps))}
(}\forall(p,q)\inset sps
set sps \subseteq set bs \times (set gs \cup set bs) \longrightarrow
(red (fst'(set gs \cup set bs)\cupfst' set (fst (f4-red gs bs ps sps data))))**
(spoly (fst p) (fst q)) 0)
by (auto simp add: image-Un fst-set-fst-f4-red)
qed
lemmas compl-struct-f4-red = compl-struct-rcp[OF rcp-spec-f4-red]
lemmas compl-pmdl-f4-red = compl-pmdl-rcp[OF rcp-spec-f4-red]
lemmas compl-conn-f4-red = compl-conn-rcp[OF rcp-spec-f4-red]

```

\subsection*{16.4 Pair Selection}
primrec 44 -sel-aux \(::{ }^{\prime} a \Rightarrow(' t, ' b:: z e r o, ' c)\) pdata-pair list \(\Rightarrow\left(' t, ' b,{ }^{\prime} c\right)\) pdata-pair list where
\[
f_{4} \text {-sel-aux }-[]=[] \mid
\]
f4-sel-aux t \((p \# p s)=\)
```

(if (lcs (lp (fst (fst p))) (lp (fst (snd p)))) = t then
p \# (f4-sel-aux t ps)
else
[]
)

```
lemma \(f 4\)-sel-aux-subset: set \((f 4\)-sel-aux \(t \mathrm{ps}) \subseteq\) set \(p s\)
    by (induct ps, auto)
primrec \(f_{4}\)-sel \(::\left({ }^{\prime} t,{ }^{\prime} b::\right.\) zero, \(\left.{ }^{\prime} c,{ }^{\prime} d\right)\) sel \(T\) where
    f4-sel gs bs [] data \(=[] \mid\)
    f4-sel gs bs \((p \# p s) d a t a=p \#(f 4-s e l-a u x(l c s(l p(f s t(f s t p)))(l p(f s t)(s n d\)
p)))) ps
lemma sel-spec-f4-sel: sel-spec f4-sel
proof (rule sel-specI)
    fix gs bs :: ('t, 'b, 'c) pdata list and ps::('t, 'b, 'c) pdata-pair list and data::nat
    \(\times{ }^{\prime} d\)
    assume \(p s \neq[]\)
    then obtain \(p p s^{\prime}\) where \(p s\) : \(p s=p \# p s^{\prime}\) by (meson list.exhaust)
    show \(f 4\)-sel gs bs ps data \(\neq[] \wedge\) set \((f 4\)-sel gs bs ps data \() \subseteq\) set ps
    proof
    show 44 -sel gs bs ps data \(\neq[]\) by (simp add: ps)
    next
        from 44 -sel-aux-subset show set (f4-sel gs bs ps data) \(\subseteq\) set ps by (auto simp
add: ps)
qed
qed

\subsection*{16.5 The F4 Algorithm}

The F4 algorithm is just \(g b\)-schema-direct with parameters instantiated by suitable functions.
lemma struct-spec-f4: struct-spec f4-sel add-pairs-canon add-basis-canon f4-red using sel-spec-f4-sel ap-spec-add-pairs-canon ab-spec-add-basis-sorted compl-struct-f4-red by (rule struct-specI)
definition \(f_{4}\)-aux \(::\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)\) pdata list \(\Rightarrow\) nat \(\times n a t \times{ }^{\prime} d \Rightarrow\left(' t,{ }^{\prime} b,{ }^{\prime} c\right)\) pdata list \(\Rightarrow\)
\(\left(' t, ' b,{ }^{\prime} c\right)\) pdata-pair list \(\Rightarrow\left({ }^{\prime} t\right.\), ' \(b::\) field, \({ }^{\prime} c::\) default \()\) pdata list
where \(f_{4}\)-aux \(=\) gb-schema-aux \(f_{4}\)-sel add-pairs-canon add-basis-canon \(f_{4}\)-red
lemmas \(f_{4}\)-aux-simps \([\) code \(]=g b\)-schema-aux-simps \(\left[\right.\) OF struct-spec-f4, folded \(f_{4}\)-aux-def \(]\)
definition \(f_{4}::\left({ }^{\prime} t,{ }^{\prime} b,{ }^{\prime} c\right)\) pdata' list \(\Rightarrow{ }^{\prime} d \Rightarrow\left({ }^{\prime} t\right.\), 'b::field, ' \(c::\) default \()\) pdata' list where \(f_{4}=\) gb-schema-direct \(f_{4}\)-sel add-pairs-canon add-basis-canon \(f_{4}\)-red
lemmas \(f_{4}\)-simps \([\) code \(]=g b\)-schema-direct-def[off4-sel add-pairs-canon add-basis-canon f4-red, folded f4-def f4-aux-def]
lemmas \(f_{4}\)-isGB \(=\) gb-schema-direct-isGB[OF struct-spec-f4 compl-conn-f4-red, folded f4-def]
lemmas \(f_{4}\)-pmdl \(=g b\)-schema-direct-pmdl[ \([O F\) struct-spec-f4 compl-pmdl-f4-red, folded f4-def]

\subsection*{16.5.1 Special Case: punit}
lemma (in gd-term) struct-spec-f4-punit: punit.struct-spec punit.f4-sel add-pairs-punit-canon punit.add-basis-canon punit.f4-red
using punit.sel-spec-f4-sel ap-spec-add-pairs-punit-canon ab-spec-add-basis-sorted punit.compl-struct-f4-red
by (rule punit.struct-specI)
definition \(f 4\)-aux-punit \(::\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)\) pdata list \(\Rightarrow\) nat \(\times\) nat \(\times{ }^{\prime} d \Rightarrow\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)\) pdata list \(\Rightarrow\)
\(\left({ }^{\prime} a, ~ ' b,{ }^{\prime} c\right)\) pdata-pair list \(\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b:: f i e l d,{ }^{\prime} c::\right.\) default \()\) pdata list
where \(f_{4}\)-aux-punit \(=\) punit.gb-schema-aux punit.f4-sel add-pairs-punit-canon punit.add-basis-canon punit.f4-red
lemmas \(f_{4}\)-aux-punit-simps \([\) code \(]=\) punit.gb-schema-aux-simps \([\) OF struct-spec-f4-punit, folded f4-aux-punit-def]
definition \(f_{4}\)-punit \(::\left({ }^{\prime} a,{ }^{\prime} b,{ }^{\prime} c\right)\) pdata' list \(\Rightarrow{ }^{\prime} d \Rightarrow\left({ }^{\prime} a,{ }^{\prime} b::\right.\) field, \({ }^{\prime} c::\) default \()\) pdata \({ }^{\prime}\) list
where \(f_{4}\)-punit \(=\) punit.gb-schema-direct punit.f4-sel add-pairs-punit-canon punit.add-basis-canon punit.f4-red
lemmas \(f_{4}\)-punit-simps \([\) code \(]=\) punit.gb-schema-direct-def[of punit.f4-sel add-pairs-punit-canon punit.add-basis-canon punit.f4-red, folded f4-punit-def
f4-aux-punit-def]
lemmas \(f_{4}\)-punit-is \(G B=\) punit.gb-schema-direct-isGB[OF struct-spec-f4-punit punit.compl-conn-f4-red, folded f4-punit-def]
lemmas f4-punit-pmdl \(^{4}=\) punit.gb-schema-direct-pmdl \([O F\) struct-spec-f4-punit punit.compl-pmdl-f4-red, folded f4-punit-def]
end
end

\section*{17 Sample Computations with the F4 Algorithm}
theory F4-Examples
imports F4 Algorithm-Schema-Impl Jordan-Normal-Form.Gauss-Jordan-IArray-Impl
Code-Target-Rat
begin
We only consider scalar polynomials here, but vector-polynomials could be handled, too.

\subsection*{17.1 Preparations}
primrec remdups-wrt-rev \(::\left({ }^{\prime} a \Rightarrow{ }^{\prime} b\right) \Rightarrow\) 'a list \(\Rightarrow{ }^{\prime} b\) list \(\Rightarrow\) ' \(a\) list where
remdups-wrt-rev \(f[]\) vs \(=[] \mid\)
remdups-wrt-rev \(f(x \#\) xs \()\) vs \(=\)
(let \(f x=f x\) in if List.member vs \(f x\) then remdups-wrt-rev \(f\) xs vs else \(x \#\) (remdups-wrt-rev fxs \((f x \#\) vs \())\) )
```

lemma remdups-wrt-rev-notin: $v \in$ set $v s \Longrightarrow v \notin f$ 'set (remdups-wrt-rev $f$ xs vs)
proof (induct xs arbitrary: vs)
case Nil
show ?case by simp
next
case (Cons $x$ xs)
from $\operatorname{Cons}(2)$ have 1:v $\neq f$ 'set (remdups-wrt-rev $f$ xs vs) by (rule Cons(1))
from Cons(2) have $v \in \operatorname{set}(f x \# v s)$ by simp
hence 2: v $\notin f^{\prime}$ set (remdups-wrt-rev $f$ xs ( $f x \#$ vs)) by (rule Cons(1))
from Cons(2) show ?case by (auto simp: Let-def 1 2 List.member-def)
qed
lemma distinct-remdups-wrt-rev: distinct (map $f$ (remdups-wrt-rev $f$ xs vs))
proof (induct xs arbitrary: vs)

```
```

    case Nil
    show ?case by simp
    next
case (Cons x xs)
show ?case by (simp add: Let-def Cons(1) remdups-wrt-rev-notin)
qed
lemma map-of-remdups-wrt-rev':
map-of (remdups-wrt-rev fst xs vs) k= map-of (filter ( }\lambdax.fst x\not\in set vs) xs)
proof (induct xs arbitrary: vs)
case Nil
show ?case by simp
next
case (Cons x xs)
show ?case
proof (simp add: Let-def List.member-def Cons, intro impI)
assume k}=f\mathrm{ fst x
have map-of (filter (\lambday.fst y f fst x}\wedge fst y \# set vs) xs)
map-of (filter (\lambday.fst y\not=fst x)(filter (\lambday.fst y \& set vs) xs))
by (simp only: filter-filter conj-commute)
also have ... = map-of (filter (\lambday. fst y \# set vs) xs) |' {y. y f fst x} by (rule
map-of-filter)
finally show map-of (filter (\lambday. fst y}=f\mathrm{ fst }x\wedgefst y \& set vs) xs) k
map-of (filter ( }\lambday.fst y\not\in set vs) xs)
by (simp add: restrict-map-def «k\not= fst x〉)
qed
qed
corollary map-of-remdups-wrt-rev: map-of (remdups-wrt-rev fst xs []) = map-of
xs
by (rule ext, simp add: map-of-remdups-wrt-rev')
lemma (in term-powerprod) compute-list-to-poly [code]:
list-to-poly ts cs = distro DRLEX (remdups-wrt-rev fst (zip ts cs) [])
by (rule poly-mapping-eqI,
simp add: lookup-list-to-poly list-to-fun-def distro-def oalist-of-list-ntm-def
oa-ntm.lookup-oalist-of-list distinct-remdups-wrt-rev lookup-dflt-def map-of-remdups-wrt-rev)
lemma (in ordered-term) compute-Macaulay-list [code]:
Macaulay-list ps=
(let ts = Keys-to-list ps in
filter ( }\lambdap.p\not=0)(mat-to-polys ts (row-echelon (polys-to-mat ts ps)))
)
by (simp add: Macaulay-list-def Macaulay-mat-def Let-def)
declare conversep-iff [code]
derive (eq) ceq poly-mapping
derive (no) ccompare poly-mapping

```
derive (dlist) set-impl poly-mapping
derive (no) cenum poly-mapping
derive (eq) ceq rat
derive (no) ccompare rat
derive (dlist) set-impl rat
derive (no) cenum rat
global-interpretation punit \(^{\prime}\) : gd-powerprod ord-pp-punit cmp-term ord-pp-strict-punit cmp-term
rewrites punit.adds-term \(=(a d d s)\)
and punit.pp-of-term \(=(\lambda x . x)\)
and punit.component-of-term \(=(\lambda-.())\)
and punit.monom-mult \(=\) monom-mult-punit
and punit.mult-scalar \(=\) mult-scalar-punit
and punit'.punit.min-term \(=\) min-term-punit
and punit'.punit.lt \(=l t\)-punit cmp-term
and punit'.punit.lc \(=l c\)-punit cmp-term
and punit' \({ }^{\prime}\) punit.tail \(=\) tail-punit cmp-term
and punit'. punit.ord- \(p=\) ord-p-punit cmp-term
and punit'.punit.ord-strict- \(p=\) ord-strict-p-punit cmp-term
and punit'.punit.keys-to-list \(=\) keys-to-list-punit cmp-term
for cmp-term :: ('a::nat, 'b::\{nat,add-wellorder\}) pp nat-term-order
defines max-punit \(=\) punit'. .ordered-powerprod-lin.max
and max-list-punit \(=\) punit \({ }^{\prime}\).ordered-powerprod-lin.max-list
and find-adds-punit \(=\) punit' \({ }^{\prime}\).punit.find-adds
and trd-aux-punit \(=\) punit' \({ }^{\prime}\).punit.trd-aux
and trd-punit \(=\) punit \({ }^{\prime} \cdot\) punit.trd
and spoly-punit \(=\) punit \({ }^{\prime}\).punit.spoly
and count-const-lt-components-punit \(=\) punit \({ }^{\prime}\).punit.count-const-lt-components
and count-rem-components-punit \(=\) punit' \({ }^{\prime}\).punit.count-rem-components
and const-lt-component-punit \(=\) punit'.punit.const-lt-component
and full-gb-punit \(=\) punit \({ }^{\prime} \cdot\) punit.full-gb
and add-pairs-single-sorted-punit \(=\) punit \({ }^{\prime} \cdot\) punit.add-pairs-single-sorted
and add-pairs-punit \(=\) punit \({ }^{\prime} \cdot\) punit.add-pairs
and canon-pair-order-aux-punit \(=\) punit'.punit.canon-pair-order-aux
and canon-basis-order-punit \(=\) punit' \({ }^{\prime}\).punit.canon-basis-order
and new-pairs-sorted-punit \(=\) punit'.punit.new-pairs-sorted
and product-crit-punit \(=\) punit \({ }^{\prime}\).punit.product-crit
and chain-ncrit-punit \(=\) punit' \(\cdot\).punit.chain-ncrit
and chain-ocrit-punit \(=\) punit' .punit.chain-ocrit
and apply-icrit-punit \(=\) punit' \({ }^{\prime}\) punit.apply-icrit
and apply-ncrit-punit \(=\) punit'.punit.apply-ncrit
and apply-ocrit-punit \(=\) punit' \({ }^{\prime}\).punit.apply-ocrit
and Keys-to-list-punit \(=\) punit \({ }^{\prime} \cdot\) punit. Keys-to-list
and sym-preproc-addnew-punit \(=\) punit'.punit.sym-preproc-addnew
and sym-preproc-aux-punit \(=\) punit' \({ }^{\prime}\) punit.sym-preproc-aux
and sym-preproc-punit \(=\) punit'.punit.sym-preproc
and Macaulay-mat-punit \(=\) punit' \(\cdot\) punit.Macaulay-mat
and Macaulay-list-punit \(=\) punit'.punit.Macaulay-list
and pdata-pairs-to-list-punit \(=\) punit \({ }^{\prime}\).punit.pdata-pairs-to-list
and Macaulay-red-punit \(=\) punit \({ }^{\prime}\).punit.Macaulay-red
and f \(_{4}\)-sel-aux-punit \(=\) punit' \(\cdot\) punit.f4-sel-aux
and f4-sel-punit \(=\) punit' \(\cdot\) punit.f4-sel
and \(f_{4}\)-red-aux-punit \(=\) punit \({ }^{\prime}\).punit.f4-red-aux
and f4-red-punit \(=\) punit \({ }^{\prime}\).punit.f4-red
and \(f_{4}\)-aux-punit \(=\) punit \({ }^{\prime}\).punit. \(f_{4}\)-aux-punit
and \(f_{4}\)-punit \(=\) punit' \({ }^{\prime}\) punit.f4-punit
subgoal by (fact gd-powerprod-ord-pp-punit)
subgoal by (fact punit-adds-term)
subgoal by (simp add: id-def)
subgoal by (fact punit-component-of-term)
subgoal by (simp only: monom-mult-punit-def)
subgoal by (simp only: mult-scalar-punit-def)
subgoal using min-term-punit-def by fastforce
subgoal by (simp only: lt-punit-def ord-pp-punit-alt)
subgoal by (simp only: lc-punit-def ord-pp-punit-alt)
subgoal by (simp only: tail-punit-def ord-pp-punit-alt)
subgoal by (simp only: ord-p-punit-def ord-pp-strict-punit-alt)
subgoal by (simp only: ord-strict-p-punit-def ord-pp-strict-punit-alt)
subgoal by (simp only: keys-to-list-punit-def ord-pp-punit-alt)
done

\subsection*{17.2 Computations}
experiment begin interpretation trivariate \(_{0}-r a t\).

\section*{lemma}
lt-punit DRLEX \(\left(X^{2} * Z^{\wedge} 3+3 * X^{2} * Y\right)=\operatorname{sparse}_{0}[(0,2),(2,3)]\)
by eval

\section*{lemma}
lc-punit DRLEX \(\left(X^{2} * Z^{\wedge} 3+3 * X^{2} * Y\right)=1\)
by eval
lemma
tail-punit DRLEX \(\left(X^{2} * Z^{\wedge} 3+3 * X^{2} * Y\right)=3 * X^{2} * Y\)
by eval

\section*{lemma}
ord-strict-p-punit DRLEX \(\left(X^{2} * Z^{\wedge} 4-2 * Y^{\wedge} 3 * Z^{2}\right)\left(X^{2} * Z^{\wedge} 7+2 *\right.\) \(Y^{\wedge} 3 * Z^{2}\) )
by eval

\section*{lemma}
f4-punit DRLEX
[

```

        (Y'2*Z+2* Z^ 3,())
    [ () =
        (X ( 
        ( (X * * Z^4-2*Y^3* Z
        ( Y' *Z+2* Z^ 3,()),
        ( X'* Y^4*Z+4*Y^ 5*Z,())
    ]
    by eval
    ```
lemma
    f4-punit DRLEX
        \(\left(X^{2}+Y^{2}+Z^{2}-1,()\right)\),
        \((X * Y-Z-1,())\),
        \(\left(Y^{2}+X,()\right)\),
    \(\left(Z^{2}+X,()\right)\)
    ] ()\(=\)
    [
    (1, ())
    ]
    by eval
end
value \([\) code \(]\) length (f4-punit DRLEX (map \((\lambda p .(p,()))\left((\operatorname{cyclic} D R L E X 4)::\left(-\Rightarrow_{0}\right.\right.\) rat) list)) ())
value [code] length (f4-punit DRLEX (map \((\lambda p .(p,()))((k a t s u r a ~ D R L E X ~ 2)::(-\) \(\Rightarrow_{0}\) rat) list)) ())
end

\section*{18 Syzygies of Multivariate Polynomials}
```

theory Syzygy
imports Groebner-Bases More-MPoly-Type-Class
begin

```

In this theory we first introduce the general concept of syzygies in modules, and then provide a method for computing Gröbner bases of syzygy modules of lists of multivariate vector-polynomials. Since syzygies in this context are themselves represented by vector-polynomials, this method can be applied repeatedly to compute bases of syzygy modules of syzygies, and so on.
instance nat :: comm-powerprod ..

\subsection*{18.1 Syzygy Modules Generated by Sets}
```

context module
begin
definition rep :: ('b =00'a) =>'b
where rep r = (\sumv\inkeys r. lookup r v*s v)
definition represents :: 'b set }=>('b=>\mp@subsup{0}{0}{\prime}'a)=>'b b boo
where represents Brx \longleftrightarrow (keys r\subseteqB\wedge local.rep r=x)
definition syzygy-module :: 'b set }=>('b=\mp@subsup{=}{0}{\prime}'a) se
where syzygy-module B}={\begin{array}{l}{\mathrm{ s.local.represents B s 0}}}
end

```
hide-const (open) real-vector.rep real-vector.represents real-vector.syzygy-module
context module
begin
lemma rep-monomial [simp]: rep (monomial \(c x)=c * s x\)
proof -
    have sub: keys (monomial \(c x) \subseteq\{x\}\) by simp
    have rep (monomial \(c x)=\left(\sum v \in\{x\}\right.\). lookup (monomial \(\left.c x\right) v * s v\) ) unfolding
rep-def
    by (rule sum.mono-neutral-left, simp, fact sub, simp)
    also have \(\ldots=c * s x\) by simp
    finally show ?thesis.
qed
lemma rep-zero [simp]: rep \(0=0\)
    by (simp add: rep-def)
lemma rep-uminus [simp]: rep \((-r)=-\) rep \(r\)
    by (simp add: keys-uminus sum-negf rep-def)
lemma rep-plus: rep \((r+s)=\) rep \(r+\) rep \(s\)
proof -
    from finite-keys finite-keys have fin: finite (keys \(r \cup\) keys s) by (rule finite-UnI)
    from fin have eq1: ( \(\sum v \in\) keys \(r \cup\) keys s. lookup \(\left.r v * s\right)=\left(\sum v \in\right.\) keys \(r\). lookup
\(r v * s v)\)
    proof (rule sum.mono-neutral-right)
        show \(\forall v \in\) keys \(r \cup\) keys \(s-k e y s\) r. lookup \(r v * s v=0\) by (simp add:
in-keys-iff)
    qed \(\operatorname{simp}\)
    from fin have eq2: \(\left(\sum v \in k e y s ~ r \cup\right.\) keys s. lookup \(\left.s v * s v\right)=\left(\sum v \in k e y s\right.\) s. lookup
\(s v * s v\) )
    proof (rule sum.mono-neutral-right)
    show \(\forall v \in\) keys \(r \cup\) keys \(s-k e y s ~ s . l o o k u p ~ s v * s v=0\) by (simp add: in-keys-iff)
qed \(\operatorname{simp}\)
have rep \((r+s)=\left(\sum v \in k e y s(r+s)\right.\). lookup \(\left.(r+s) v * s v\right)\) by (simp only: rep-def)
also have \(\ldots=\left(\sum v \in\right.\) keys \(r \cup\) keys s. lookup \(\left.(r+s) v * s v\right)\)
proof (rule sum.mono-neutral-left)
show \(\forall i \in k e y s r \cup\) keys \(s-k e y s(r+s)\). lookup \((r+s) i * s i=0\) by (simp add: in-keys-iff)
qed (auto simp: Poly-Mapping.keys-add)
also have \(\ldots=\left(\sum v \in k e y s r \cup\right.\) keys s. lookup \(\left.r v * s v\right)+\left(\sum v \in k e y s r \cup\right.\) keys \(s\). lookup s \(v * s v\) )
by (simp add: lookup-add scale-left-distrib sum.distrib)
also have \(\ldots=\) rep \(r+\) rep \(s\) by (simp only: eq1 eq2 rep-def)
finally show ?thesis .
qed
lemma rep-minus: rep \((r-s)=\) rep \(r-r e p s\)
proof -
from finite-keys finite-keys have fin: finite (keys \(r \cup\) keys s) by (rule finite-UnI)
from fin have eq1: ( \(\sum v \in\) keys \(r \cup\) keys s. lookup \(\left.r v * s v\right)=\left(\sum v \in\right.\) keys \(r\). lookup \(r v * s v)\)
proof (rule sum.mono-neutral-right)
show \(\forall v \in\) keys \(r \cup\) keys \(s-k e y s r\). lookup \(r v * s v=0\) by (simp add: in-keys-iff)
qed \(\operatorname{simp}\)
from fin have eq2: \(\left(\sum v \in k e y s r \cup\right.\) keys \(s\). lookup \(\left.s v * s v\right)=\left(\sum v \in k e y s\right.\) s. lookup \(s v * s v\) )
proof (rule sum.mono-neutral-right)
show \(\forall v \in\) keys \(r \cup\) keys \(s-k e y s\) s. lookup \(s v * s v=0\) by (simp add: in-keys-iff) qed \(\operatorname{simp}\)
have rep \((r-s)=\left(\sum v \in k e y s(r-s)\right.\). lookup \(\left.(r-s) v * s v\right)\) by (simp only: rep-def)
also from fin keys-minus have \(\ldots=\left(\sum v \in k e y s ~ r \cup\right.\) keys \(s\). lookup \((r-s) v * s\) \(v)\)
proof (rule sum.mono-neutral-left)
show \(\forall i \in\) keys \(r \cup\) keys \(s-k e y s(r-s)\). lookup \((r-s) i * s i=0\) by (simp add: in-keys-iff)
qed
also have \(\ldots=\left(\sum v \in\right.\) keys \(r \cup\) keys s. lookup \(\left.r v * s v\right)-\left(\sum v \in k e y s r \cup\right.\) keys \(s\). lookup s \(v * s v\) )
by (simp add: lookup-minus scale-left-diff-distrib sum-subtractf)
also have \(\ldots=\) rep \(r\) - rep \(s\) by (simp only: eq1 eq2 rep-def)
finally show ?thesis.
qed
lemma rep-smult: rep (monomial \(c 0 * r)=c * s\) rep \(r\) proof -
have \(l\) : lookup (monomial c \(0 * r\) ) \(v=c *(\) lookup \(r v)\) for \(v\)
unfolding mult-map-scale-conv-mult[symmetric] by (rule map-lookup, simp)
have sub: keys (monomial c \(0 * r\) ) \(\subseteq\) keys \(r\)
```

    by (metis l lookup-not-eq-zero-eq-in-keys mult-zero-right subsetI)
    have rep (monomial c 0*r)=(\sumv\inkeys(monomial c 0 * r). lookup (monomial
    c0*r)v*sv)
by (simp only: rep-def)
also from finite-keys sub have ... = (\sumv\inkeys r.lookup (monomial c 0 * r)v
*s v)
proof (rule sum.mono-neutral-left)
show }\forallv\inkeys r - keys (monomial c 0*r). lookup (monomial c 0*r) v*
v=0 by (simp add: in-keys-iff)
qed
also have ... = c*s(\sumv\inkeys r. lookup r v*s v) by (simp add: l scale-sum-right)
also have ... =c*s rep r by (simp add: rep-def)
finally show ?thesis.
qed
lemma rep-in-span: rep r f span (keys r)
unfolding rep-def by (fact sum-in-spanI)
lemma spanE-rep:
assumes }x\in\operatorname{span}
obtains r where keys r\subseteqB and x = rep r
proof -
from assms obtain A q where finite A and A\subseteqB and x:x = (\suma\inA.qa
*s a) by (rule spanE)
define r where r=Abs-poly-mapping ( }\lambdak.qk\mathrm{ when }k\inA
have 1: lookup r = ( }\lambdak.qk\mathrm{ when }k\inA)\mathrm{ unfolding r-def
by (rule Abs-poly-mapping-inverse, simp add:〈finite A〉)
have 2: keys r\subseteqA by (auto simp: in-keys-iff 1)
show ?thesis
proof
have }x=(\suma\inA.lookup r a *s a) unfolding x by (rule sum.cong, simp-all
add: 1)
also from〈finite A〉2 have ... = (\suma\inkeys r.lookup r a *s a)
proof (rule sum.mono-neutral-right)
show }\foralla\inA - keys r. lookup r a*s a=0 by (simp add: in-keys-iff
qed
finally show }x=rep r by (simp only: rep-def
next
from 2 <A\subseteqB\rangle show keys r\subseteqB by (rule subset-trans)
qed
qed
lemma representsI:
assumes keys r\subseteqB and rep r=x
shows represents Brx
unfolding represents-def using assms by blast
lemma representsD1:

```
```

    assumes represents B r x
    shows keys r\subseteqB
    using assms unfolding represents-def by blast
    lemma representsD2:
assumes represents B r x
shows }x=rep
using assms unfolding represents-def by blast
lemma represents-mono:
assumes represents Brx and B\subseteqA
shows represents A r x
proof (rule representsI)
from assms(1) have keys r\subseteqB by (rule representsD1)
thus keys r\subseteqA using assms(2) by (rule subset-trans)
next
from assms(1) have x rep r by (rule representsD2)
thus rep r = x by (rule HOL.sym)
qed
lemma represents-self:represents {x} (monomial 1 x) x
proof -
have sub: keys (monomial (1::'a) x)\subseteq{x} by simp
moreover have rep (monomial (1::'a) x)=x by simp
ultimately show ?thesis by (rule representsI)
qed
lemma represents-zero: represents B 0 0
by (rule representsI, simp-all)
lemma represents-plus:
assumes represents A rx and represents B s y
shows represents }(A\cupB)(r+s)(x+y
proof -
from assms(1) have r: keys r\subseteqA and x:x = rep r by (rule representsD1,
rule representsD2)
from assms(2) have s: keys s\subseteqB and y: y = rep s by (rule representsD1, rule
representsD2)
show ?thesis
proof (rule representsI)
from rs have keys r \ keys s\subseteqA\cupB by blast
thus keys (r+s)\subseteqA\cupB
by (meson Poly-Mapping.keys-add subset-trans)
qed (simp add: rep-plus x y)
qed
lemma represents-uminus:
assumes represents B r x
shows represents B (-r) (-x)

```
```

proof -
from assms have r: keys r\subseteqB and x: x = rep r by (rule representsD1, rule
representsD2)
show ?thesis
proof (rule representsI)
from r show keys (-r)\subseteqB by (simp only: keys-uminus)
qed (simp add: x)
qed
lemma represents-minus:
assumes represents Arx and represents B s y
shows represents (A\cupB)(r-s) (x-y)
proof -
from assms(1) have r: keys r\subseteqA and x: x = rep r by (rule representsD1,
rule representsD2)
from assms(2) have s: keys s\subseteqB and y: y=rep s by (rule representsD1, rule
representsD2)
show ?thesis
proof (rule representsI)
from r s have keys r \ keys s\subseteqA\cupB by blast
with keys-minus show keys (r-s)\subseteqA\cupB by (rule subset-trans)
qed (simp only: rep-minus x y)
qed
lemma represents-scale:
assumes represents B r x
shows represents B (monomial c 0*r)(c*s x)
proof -
from assms have r:keys r\subseteqB and x:x= rep r by (rule representsD1, rule
representsD2)
show ?thesis
proof (rule representsI)
have l:lookup (monomial c 0 * r) v=c*(lookup r v) for v
unfolding mult-map-scale-conv-mult[symmetric] by (rule map-lookup, simp)
have sub: keys (monomial c 0*r)\subseteq keys r
by (metis l lookup-not-eq-zero-eq-in-keys mult-zero-right subsetI)
thus keys (monomial c 0*r)\subseteqB using r by (rule subset-trans)
qed (simp only: rep-smult x)
qed
lemma represents-in-span:
assumes represents Brx
shows }x\in\operatorname{span}
proof -
from assms have r: keys r\subseteqB and x:x=rep r by (rule representsD1, rule
representsD2)
have }x\in\operatorname{span}(keys r) unfolding x by (fact rep-in-span)
also from r have ... \subseteq span B by (rule span-mono)
finally show ?thesis.

```

\section*{qed}
lemma syzygy-module-iff: \(s \in\) syzygy-module \(B \longleftrightarrow\) represents \(B\) s 0 by (simp add: syzygy-module-def)
lemma syzygy-moduleI:
assumes represents \(B\) s 0
shows \(s \in\) syzygy-module \(B\)
unfolding syzygy-module-iff using assms.
lemma syzygy-moduleD:
assumes \(s \in\) syzygy-module \(B\)
shows represents \(B\) s 0
using assms unfolding syzygy-module-iff .
lemma zero-in-syzygy-module: \(0 \in\) syzygy-module \(B\)
using represents-zero by (rule syzygy-moduleI)
lemma syzygy-module-closed-plus:
assumes s1 \(\in\) syzygy-module \(B\) and s2 \(\in\) syzygy-module \(B\)
shows \(s 1+s 2 \in\) syzygy-module \(B\)
proof -
from \(\operatorname{assms}(1)\) have represents \(B\) s1 0 by (rule syzygy-moduleD)
moreover from assms(2) have represents \(B\) s2 0 by (rule syzygy-moduleD)
ultimately have represents \((B \cup B)(s 1+s \mathcal{Q})(0+0)\) by (rule represents-plus)
hence represents \(B(s 1+s 2) 0\) by simp
thus ?thesis by (rule syzygy-moduleI)
qed
lemma syzygy-module-closed-minus:
assumes s1 \(\in\) syzygy-module \(B\) and s2 \(\in\) syzygy-module \(B\)
shows s1-s2 \(\in\) syzygy-module \(B\)
proof -
from \(\operatorname{assms}(1)\) have represents \(B\) s1 0 by (rule syzygy-moduleD)
moreover from assms(2) have represents \(B\) s2 0 by (rule syzygy-moduleD)
ultimately have represents \((B \cup B)(s 1-s 2)(0-0)\) by (rule represents-minus)
hence represents \(B(s 1-s 2) 0\) by simp
thus ?thesis by (rule syzygy-moduleI)
qed
lemma syzygy-module-closed-times-monomial:
assumes \(s \in\) syzygy-module \(B\)
shows monomial c \(0 * s \in\) syzygy-module \(B\)
proof -
from \(\operatorname{assms}(1)\) have represents \(B s 0\) by (rule syzygy-moduleD)
hence represents \(B\) (monomial \(c 0 * s)(c * s 0)\) by (rule represents-scale)
hence represents \(B\) (monomial \(c 0 * s) 0\) by simp
thus ?thesis by (rule syzygy-moduleI)
qed
end
context term-powerprod
begin
lemma keys-rep-subset:
assumes \(u \in\) keys (pmdl.rep \(r\) )
obtains \(t v\) where \(t \in\) Keys (Poly-Mapping.range \(r\) ) and \(v \in\) Keys (keys \(r\) ) and
\(u=t \oplus v\)
proof -
note assms
also have keys (pmdl.rep \(r) \subseteq(\bigcup v \in\) keys r. keys (lookup r \(v \odot v)\) ) by (simp add: pmdl.rep-def keys-sum-subset)
finally obtain \(v 0\) where \(v 0 \in\) keys \(r\) and \(u \in k e y s\) (lookup \(r v 0 \odot v 0\) ) ..
from this(2) obtain \(t v\) where \(t \in\) keys (lookup \(r v 0\) ) and \(v \in\) keys \(v 0\) and \(u\) \(=t \oplus v\)
by (rule in-keys-mult-scalarE)
show ?thesis
proof
from \(\langle v 0 \in\) keys \(r\rangle\) have lookup \(r v 0 \in\) Poly-Mapping.range \(r\) by (rule
in-keys-lookup-in-range)
with \(\langle t \in\) keys (lookup \(r v 0\) ) 〉show \(t \in\) Keys (Poly-Mapping.range \(r\) ) by (rule in-KeysI)
next
from \(\langle v \in\) keys \(v 0\rangle\langle v 0 \in\) keys \(r\rangle\) show \(v \in\) Keys (keys \(r\) ) by (rule in-KeysI) qed fact
qed
lemma rep-mult-scalar: pmdl.rep (punit.monom-mult c \(0 r\) ) \(=c \odot p m d l . r e p r\)
unfolding punit.mult-scalar-monomial[symmetric] punit-mult-scalar by (fact pmdl.rep-smult)
lemma represents-mult-scalar:
assumes pmdl.represents \(B\) r x
shows pmdl.represents \(B\) (punit.monom-mult c \(0 r)(c \odot x)\)
unfolding punit.mult-scalar-monomial[symmetric] punit-mult-scalar using assms
by (rule pmdl.represents-scale)
lemma syzygy-module-closed-map-scale: \(s \in\) pmdl.syzygy-module \(B \Longrightarrow c \cdot s \in\) pmdl.syzygy-module \(B\)
unfolding map-scale-eq-times by (rule pmdl.syzygy-module-closed-times-monomial)
lemma phull-syzygy-module: phull (pmdl.syzygy-module B) \(=\) pmdl.syzygy-module B
unfolding phull.span-eq-iff
apply (rule phull.subspaceI)
subgoal by (fact pmdl.zero-in-syzygy-module)
subgoal by (fact pmdl.syzygy-module-closed-plus)
subgoal by (fact syzygy-module-closed-map-scale)
done
end

\subsection*{18.2 Polynomial Mappings on List-Indices}
definition pm-of-idx-pm :: ('a list) \(\Rightarrow\left(n a t \Rightarrow \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow{ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b::\) zero
where pm-of-idx-pm xs \(f=\) Abs-poly-mapping ( \(\lambda x\). lookup \(f\) (Min \(\{i . i<\) length \(x s \wedge x s!i=x\})\) when \(x \in\) set \(x s)\)
definition \(i d x\)-pm-of-pm \(::\left({ }^{\prime} a\right.\) list \() \Rightarrow\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow n a t \Rightarrow{ }_{0}{ }^{\prime} b::\) zero
where idx-pm-of-pm xs \(f=\) Abs-poly-mapping ( \(\lambda i\). lookup \(f(x s!i\) ) when \(i<\) length \(x s\) )
lemma lookup-pm-of-idx-pm:
lookup \((\) pm-of-idx-pm xs \(f)=(\lambda x\). lookup \(f(\operatorname{Min}\{i . i<\) length \(x s \wedge x s!i=x\})\) when \(x \in\) set \(x s\) )
unfolding pm-of-idx-pm-def by (rule Abs-poly-mapping-inverse, simp)
lemma lookup-pm-of-idx-pm-distinct:
assumes distinct xs and \(i<\) length \(x s\)
shows lookup (pm-of-idx-pm xs f) (xs !i) lookup fi
proof -
from assms have \(\{j . j<\) length \(x s \wedge x s!j=x s!i\}=\{i\}\) using distinct-Ex1 nth-mem by fastforce
moreover from assms(2) have \(x s!i \in\) set \(x s\) by (rule nth-mem)
ultimately show ?thesis by (simp add: lookup-pm-of-idx-pm)
qed
lemma keys-pm-of-idx-pm-subset: keys (pm-of-idx-pm xs \(f\) ) \(\subseteq\) set xs proof
fix \(t\)
assume \(t \in\) keys (pm-of-idx-pm xs \(f\) )
hence lookup (pm-of-idx-pm xs f) \(t \neq 0\) by (simp add: in-keys-iff)
thus \(t \in\) set \(x s\) by (simp add: lookup-pm-of-idx-pm)
qed
lemma range-pm-of-idx-pm-subset: Poly-Mapping.range (pm-of-idx-pm xs \(f\) ) \(\subseteq\) lookup \(f\) ' \(\{0 . .<\) length \(x s\}-\{0\}\) proof
fix \(c\)
assume \(c \in\) Poly-Mapping.range (pm-of-idx-pm xs f)
then obtain \(t\) where \(t: t \in\) keys (pm-of-idx-pm xs f) and \(c: c=\) lookup ( \(p m\)-of-idx-pm xs f) \(t\)
by (metis Diffe imageE insertCI not-in-keys-iff-lookup-eq-zero range.rep-eq)
from \(t\) keys-pm-of-idx-pm-subset have \(t \in\) set xs ..
hence \(c 1: c=\) lookup \(f(\operatorname{Min}\{i . i<\) length \(x s \wedge x s!i=t\})\) by (simp add: lookup-pm-of-idx-pm c)
show \(c \in\) lookup \(f\) ' \(\{0 . .<\) length \(x s\}-\{0\}\)

\section*{proof (intro DiffI image-eqI)}
from \(\langle t \in\) set \(x s\rangle\) obtain \(i\) where \(i<\) length \(x s\) and \(t=x s!i\) by (metis in-set-conv-nth)
have finite \(\{i . i<\) length \(x s \wedge x s!i=t\}\) by simp
moreover from \(\langle i<\) length \(x s\rangle\langle t=x s!i\rangle\) have \(\{i . i<\) length \(x s \wedge x s!i=\) \(t\} \neq\{ \}\) by auto
ultimately have Min \(\{i . i<\) length \(x s \wedge x s!i=t\} \in\{i . i<\) length \(x s \wedge x s\) \(!i=t\}\)
by (rule Min-in)
thus Min \(\{i . i<\) length \(x s \wedge x s!i=t\} \in\{0 . .<\) length \(x s\}\) by simp
next
from \(t\) show \(c \notin\{0\}\) by (simp add: c in-keys-iff)
qed (fact c1)
qed
corollary range-pm-of-idx-pm-subset': Poly-Mapping.range (pm-of-idx-pm xs \(f\) ) \(\subseteq\) Poly-Mapping.range \(f\)
using range-pm-of-idx-pm-subset
proof (rule subset-trans)
show lookup \(f\) ' \(\{0 . .<\) length \(x s\}-\{0\} \subseteq\) Poly-Mapping.range \(f\) by (transfer, auto)
qed
lemma pm-of-idx-pm-zero [simp]: pm-of-idx-pm xs \(0=0\)
by (rule poly-mapping-eqI, simp add: lookup-pm-of-idx-pm)
lemma pm-of-idx-pm-plus: pm-of-idx-pm xs \((f+g)=p m\)-of-idx-pm \(x s f+p m\)-of-idx-pm xs \(g\)
by (rule poly-mapping-eqI, simp add: lookup-pm-of-idx-pm lookup-add when-def)
lemma pm-of-idx-pm-uminus: pm-of-idx-pm xs \((-f)=-p m\)-of-idx-pm xs \(f\)
by (rule poly-mapping-eqI, simp add: lookup-pm-of-idx-pm when-def)
lemma pm-of-idx-pm-minus: pm-of-idx-pm xs \((f-g)=p m-o f-i d x-p m\) xs \(f-\) pm-of-idx-pm xs g
by (rule poly-mapping-eqI, simp add: lookup-pm-of-idx-pm lookup-minus when-def)
lemma pm-of-idx-pm-monom-mult: pm-of-idx-pm xs (punit.monom-mult c \(0 f\) ) = punit.monom-mult c 0 (pm-of-idx-pm xs f)
by (rule poly-mapping-eqI, simp add: lookup-pm-of-idx-pm punit.lookup-monom-mult-zero when-def)
lemma pm-of-idx-pm-monomial:
assumes distinct xs
shows pm-of-idx-pm xs (monomial ci) \(=(\) monomial \(c(x s!i)\) when \(i<l e n g t h\) xs)
proof -
from assms have \(*:\{i . i<\) length \(x s \wedge x s!i=x s!j\}=\{j\}\) if \(j<\) length \(x s\) for \(j\)
```

    using distinct-Ex1 nth-mem that by fastforce
    show ?thesis
    proof (cases i< length xs)
    case True
    have pm-of-idx-pm xs (monomial c i) = monomial c (xs!i)
    proof (rule poly-mapping-eqI)
        fix }
        show lookup (pm-of-idx-pm xs (monomial c i)) k= lookup (monomial c (xs !
    i)) }
proof (cases xs !i=k)
case True
with <i< length xs\rangle have k\in set xs by auto
thus ?thesis by (simp add: lookup-pm-of-idx-pm lookup-single *[OF<i<
length xs>] True[symmetric])
next
case False
have lookup (pm-of-idx-pm xs (monomial c i)) k=0
proof (cases k f set xs)
case True
then obtain j where j<length xs and k=xs!j by (metis in-set-conv-nth)
with False have i\not= Min {i.i<length xs ^ xs !i=k}
by (auto simp: <k=xs! j〉*[OF< < < length xs>])
thus ?thesis by (simp add: lookup-pm-of-idx-pm True lookup-single)
next
case False
thus ?thesis by (simp add: lookup-pm-of-idx-pm)
qed
with False show ?thesis by (simp add: lookup-single)
qed
qed
with True show ?thesis by simp
next
case False
have pm-of-idx-pm xs (monomial c i)=0
proof (rule poly-mapping-eqI, simp add: lookup-pm-of-idx-pm when-def, rule)
fix }
assume k f set xs
then obtain j where j<length xs and k=xs!j by (metis in-set-conv-nth)
with False have i\not= Min {i.i<length xs \wedge xs!i=k}
by (auto simp: <k=xs! j>*[OF< < < length xs\rangle])
thus lookup (monomial c i) (Min {i. i< length xs ^xs!i=k})=0
by (simp add: lookup-single)
qed
with False show ?thesis by simp
qed
qed
lemma pm-of-idx-pm-take:
assumes keys f\subseteq{0..<j}

```
```

    shows pm-of-idx-pm (take j xs) f= pm-of-idx-pm xs f
    proof (rule poly-mapping-eqI)
fix i
let ?xs = take j xs
let ?A = {k. k< length xs ^ xs ! k=i}
let ?B}={k.k<length xs \wedgek<j^xs!k=i
have A-fin: finite ?A and B-fin: finite ?B by fastforce+
have A-ne: i}\mathrm{ set xs ఋ?A ={} by (simp add: in-set-conv-nth)
have B-ne: i\in set ?xs \Longrightarrow?B }\not={}\mathrm{ by (auto simp add: in-set-conv-nth)
define m1 where m1= Min ?A
define m2 where m2 = Min ? B
have m1:m1\in?A if i\in set xs
unfolding m1-def by (rule Min-in, fact A-fin, rule A-ne, fact that)
have m2: m2 \in?B if }i\in\mathrm{ set ?xs
unfolding m2-def by (rule Min-in, fact B-fin, rule B-ne, fact that)
show lookup (pm-of-idx-pm (take j xs) f) i = lookup (pm-of-idx-pm xs f) i
proof (cases i f set ?xs)
case True
hence i\in set xs using set-take-subset ..
hence m1\in?A by (rule m1)
hence m1< length xs and xs!m1=i by simp-all
from True have m2 \in?B by (rule m2)
hence m2 < length xs and m2 < j and xs!m2 = i by simp-all
hence m2 \in?A by simp
with A-fin have m1\leqm2 unfolding m1-def by (rule Min-le)
with <m2< j> have m1<j by simp
with <m1<length xs\rangle\langlexs!m1= i\rangle have m1\in?B by simp
with B-fin have m2 \leqm1 unfolding m2-def by (rule Min-le)
with <m1\leqm2` have m1=m2 by (rule le-antisym)     with True «i \in set xs` show ?thesis by (simp add: lookup-pm-of-idx-pm m1-def
m2-def cong:conj-cong)
next
case False
thus ?thesis
proof (simp add: lookup-pm-of-idx-pm when-def m1-def[symmetric], intro impI)
assume i\in set xs
hence m1\in?A by (rule m1)
hence m1< length xs and xs!m1=i by simp-all
have m1 \& keys f
proof
assume m1 \in keys f
hence m1\in{0..<j} using assms ..
hence m1<j by simp
with <m1 < length xs` have m1 < length ?xs by simp
hence ?xs!m1 \in set ?xs by (rule nth-mem)
with <m1< j\rangle have i\in set ?xs by (simp add:<xs!m1= i\rangle)
with False show False ..
qed
thus lookup f m1 = 0 by (simp add: in-keys-iff)

```
```

    qed
    qed
    qed
lemma lookup-idx-pm-of-pm: lookup (idx-pm-of-pm xs f)=(\lambdai.lookup f (xs!i)
when i < length xs)
unfolding idx-pm-of-pm-def by (rule Abs-poly-mapping-inverse, simp)
lemma keys-idx-pm-of-pm-subset: keys (idx-pm-of-pm xs f)\subseteq{0..<length xs }
proof
fix }
assume i f keys (idx-pm-of-pm xs f)
hence lookup (idx-pm-of-pm xs f) i\not=0 by (simp add: in-keys-iff)
thus i\in{0..<length xs} by (simp add: lookup-idx-pm-of-pm)
qed
lemma idx-pm-of-pm-zero [simp]: idx-pm-of-pm xs 0 = 0
by (rule poly-mapping-eqI, simp add: lookup-idx-pm-of-pm)
lemma idx-pm-of-pm-plus: idx-pm-of-pm xs (f+g) =idx-pm-of-pm xs f+idx-pm-of-pm
xs g
by (rule poly-mapping-eqI, simp add: lookup-idx-pm-of-pm lookup-add when-def)
lemma idx-pm-of-pm-minus: idx-pm-of-pm xs (f - g) = idx-pm-of-pm xs f -
idx-pm-of-pm xs g
by (rule poly-mapping-eqI, simp add: lookup-idx-pm-of-pm lookup-minus when-def)
lemma pm-of-idx-pm-of-pm:
assumes keys f\subseteq set xs
shows pm-of-idx-pm xs (idx-pm-of-pm xs f) =f
proof (rule poly-mapping-eqI, simp add: lookup-pm-of-idx-pm when-def, intro conjI
impI)
fix }
assume k f set xs
define i}\mathrm{ where i= Min {i. i< length xs ^xs!i=k}
have finite {i.i< length xs ^xs!i=k} by simp
moreover from <k\in set xs> have {i.i< length xs ^xs!i=k}\not={}
by (simp add: in-set-conv-nth)
ultimately have i\in{i.i< length xs }\wedgexs!i=k} unfolding i-def by (rul
Min-in)
hence i< length xs and xs ! i=k by simp-all
thus lookup (idx-pm-of-pm xs f) i= lookup fk by (simp add: lookup-idx-pm-of-pm)
next
fix }
assume k \& set xs
with assms show lookup fk=0 by (auto simp: in-keys-iff)
qed
lemma idx-pm-of-pm-of-idx-pm:

```
```

    assumes distinct xs and keys f\subseteq{0..<length xs }
    shows idx-pm-of-pm xs (pm-of-idx-pm xs f) =f
    proof (rule poly-mapping-eqI)
fix }
show lookup (idx-pm-of-pm xs (pm-of-idx-pm xs f)) i=lookup fi
proof (cases i < length xs)
case True
with assms(1) show ?thesis by (simp add: lookup-idx-pm-of-pm lookup-pm-of-idx-pm-distinct)
next
case False
hence i\not\in{0..<length xs } by simp
with assms(2) have i\not\in keys f by blast
with False show ?thesis by (simp add: in-keys-iff lookup-idx-pm-of-pm)
qed
qed

```

\subsection*{18.3 POT Orders}
context ordered-term
begin
definition is-pot-ord :: bool
where \(i s\)-pot-ord \(\longleftrightarrow(\forall u v\). component-of-term \(u<\) component-of-term \(v \longrightarrow\) \(u \prec_{t} v\) )
lemma is-pot-ordI:
assumes \(\bigwedge u v\). component-of-term \(u<\) component-of-term \(v \Longrightarrow u \prec_{t} v\) shows is-pot-ord
unfolding is-pot-ord-def using assms by blast
lemma is-pot-ordD:
assumes is-pot-ord and component-of-term \(u<\) component-of-term \(v\)
shows \(u \prec_{t} v\)
using assms unfolding is-pot-ord-def by blast
lemma is-pot-ordD2:
assumes is-pot-ord and \(u \preceq_{t} v\)
shows component-of-term \(u \leq\) component-of-term \(v\)
proof (rule ccontr)
assume \(\neg\) component-of-term \(u \leq\) component-of-term \(v\)
hence component-of-term \(v<\) component-of-term \(u\) by simp
with assms(1) have \(v \prec_{t} u\) by (rule is-pot-ordD)
with assms(2) show False by simp
qed
lemma is-pot-ord:
assumes is-pot-ord
shows \(u \preceq_{t} v \longleftrightarrow\) (component-of-term \(u<\) component-of-term \(v \vee\)
(component-of-term \(u=\) component-of-term \(v \wedge\) pp-of-term \(u \preceq\)
```

pp-of-term v))(is ?l \longleftrightarrow ?r)
proof
assume ?l
with assms have component-of-term u\leqcomponent-of-term v by (rule is-pot-ordD2)
hence component-of-term u< component-of-term v\vee component-of-term u=
component-of-term v
by (simp add: order-class.le-less)
thus ?r
proof
assume component-of-term u< component-of-term v
thus ?r ..
next
assume 1: component-of-term u= component-of-term v
moreover have pp-of-term u\preceqpp-of-term v
proof (rule ccontr)
assume \neg pp-of-term u\preceqpp-of-term v
hence 2: pp-of-term v\preceqpp-of-term u and 3: pp-of-term u\not=pp-of-term v
by simp-all
from 1 have component-of-term v\leqcomponent-of-term u by simp
with 2 have v}\mp@subsup{\preceq}{t}{}u\mathrm{ by (rule ord-termI)
with 〈?l` have u=v by simp
with 3 show False by simp
qed
ultimately show ?r by simp
qed
next
assume ?r
thus?l
proof
assume component-of-term u< component-of-term v
with assms have }u\mp@subsup{\prec}{t}{}v\mathrm{ by (rule is-pot-ordD)
thus ?l by simp
next
assume component-of-term u=component-of-term v ^ pp-of-term u\preceq pp-of-term
v
hence pp-of-term u\preceq pp-of-term v and component-of-term u}\leqcompo
nent-of-term v by simp-all
thus ?l by (rule ord-termI)
qed
qed
definition map-component :: ('k
where map-component fv=term-of-pair (pp-of-term v,f (component-of-term
v))
lemma pair-of-map-component [term-simps]:
pair-of-term (map-component fv)=(pp-of-term v,f(component-of-term v))
by (simp add: map-component-def pair-term)

```
```

lemma pp-of-map-component [term-simps]: pp-of-term (map-component $f v$ ) $=$
pp-of-term v
by (simp add: pp-of-term-def pair-of-map-component)
lemma component-of-map-component [term-simps]:
component-of-term (map-component $f v)=f$ (component-of-term $v$ )
by (simp add: component-of-term-def pair-of-map-component)
lemma map-component-term-of-pair [term-simps]:
map-component $f$ (term-of-pair $(t, k))=$ term-of-pair $(t, f k)$
by (simp add: map-component-def term-simps)
lemma map-component-comp: map-component $f$ (map-component $g x)=$ map-component
( $\lambda k . f(g k)) x$
by (simp add: map-component-def term-simps)
lemma map-component-id [term-simps]: map-component $(\lambda k . k) x=x$
by (simp add: map-component-def term-simps)
lemma map-component-inj:
assumes inj $f$ and map-component $f u=$ map-component $f v$
shows $u=v$
proof -
from $\operatorname{assms}(2)$ have term-of-pair $(p p$-of-term $u, f($ component-of-term $u))=$
term-of-pair (pp-of-term $v, f$ (component-of-term $v$ ))
by (simp only: map-component-def)
hence $(p p$-of-term $u, f($ component-of-term $u))=(p p$-of-term $v, f($ component-of-term
v))
by (rule term-of-pair-injective)
hence 1 : pp-of-term $u=p p$-of-term $v$ and $f$ (component-of-term $u)=f$ (component-of-term
$v)$ by simp-all
from assms(1) this(2) have component-of-term $u=$ component-of-term $v$ by
(rule injD)
with 1 show ?thesis by (metis term-of-pair-pair)
qed
end

```

\subsection*{18.4 Gröbner Bases of Syzygy Modules}
```

locale $g d$-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::graded-dickson-powerprod $\times$ nat $)$
and term-of-pair::(' $a \times n a t) \Rightarrow$ '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

```

In order to compute a Gröbner basis of the syzygy module of a list \(b s\) of polynomials, one first needs to "lift" \(b s\) to a new list \(b s^{\prime}\) by adding further components, compute a Gröbner basis \(g s\) of \(b s^{\prime}\), and then filter out those elements of \(g s\) whose only non-zero components are those that were newly added to \(b s\). Function init-syzygy-list takes care of constructing \(b s^{\prime}\), and function filter-syzygy-basis does the filtering. Function proj-orig-basis, finally, projects the Gröbner basis \(g s\) of \(b s^{\prime}\) to a Gröbner basis of the original list \(b s\).
```

definition lift-poly-syz :: nat $\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow$ nat $\Rightarrow\left({ }^{\prime} t \Rightarrow{ }_{0}{ }^{\prime} b::\right.$ semiring-1 $)$
where lift-poly-syz n b $i=$ Abs-poly-mapping
( $\lambda x$. if pair-of-term $x=(0, i)$ then 1
else if $n \leq$ component-of-term $x$ then lookup $b$ (map-component $(\lambda k$.
$k-n) x$ )
else 0)

```
definition proj-poly-syz :: nat \(\Rightarrow\left({ }^{\prime} t \Rightarrow{ }_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) semiring- 1\()\)
    where proj-poly-syz \(n b=\) Poly-Mapping.map-key \((\lambda x\). map-component \((\lambda k . k+\)
n) \(x\) ) \(b\)
definition cofactor-list-syz :: nat \(\Rightarrow\left({ }^{\prime} t \Rightarrow{ }_{0}{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} a \Rightarrow_{0}{ }^{\prime} b::\right.\) semiring-1 \()\) list
    where cofactor-list-syz \(n b=\operatorname{map}(\lambda i\). proj-poly \(i b)[0 . .<n]\)
definition init-syzygy-list :: ( \(\left.{ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) semiring-1 \()\) list
    where init-syzygy-list bs \(=\) map-idx (lift-poly-syz (length bs)) bs 0
definition proj-orig-basis :: nat \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right.\) ::semiring-1 \()\) list
    where proj-orig-basis \(n\) bs \(=\) map (proj-poly-syz \(n\) ) bs
definition filter-syzygy-basis :: nat \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) semiring-1 \()\) list
    where filter-syzygy-basis \(n\) bs \(=[b \leftarrow b s\). component-of-term' keys \(b \subseteq\{0 . .<n\}]\)
definition syzygy-module-list :: \(\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b\right)\) list \(\Rightarrow\left({ }^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\right.\) comm-ring-1) set
    where syzygy-module-list bs = atomize-poly 'idx-pm-of-pm bs'pmdl.syzygy-module
(set bs)

\subsection*{18.4.1 lift-poly-syz}
lemma keys-lift-poly-syz-aux:
\(\{x\). (if pair-of-term \(x=(0, i)\) then 1
else if \(n \leq\) component-of-term \(x\) then lookup b (map-component \((\lambda k . k-n\) )
x)
else 0\() \neq 0\} \subseteq\) insert (term-of-pair \((0, i))\) (map-component \((\lambda k . k+n)\) '
keys b)
(is ? \(l \subseteq ? r\) ) for \(b::^{\prime} t \Rightarrow_{0}{ }^{\prime} b::\) semiring- 1
proof
fix \(x::^{\prime} t\)
assume \(x \in\) ?l
hence (if pair-of-term \(x=(0, i)\) then 1 else if \(n \leq\) component-of-term \(x\) then
```

lookup b (map-component ( }\lambdak.k-n) x) else 0) \not=
by simp
hence pair-of-term x = (0,i)\vee( n\leq component-of-term x ^lookup b (map-component
(\lambdak.k-n) x) =0)
by (simp split: if-split-asm)
thus }x\in\mathrm{ ?r
proof
assume pair-of-term x = (0,i)
hence ( }0,i)=\mathrm{ pair-of-term x by (rule sym)
hence x = term-of-pair ( }0,i\mathrm{ ) by (simp add: term-pair)
thus ?thesis by simp
next
assume n\leq component-of-term x ^ lookup b (map-component (\lambdak.k-n) x)
\not=0
hence n\leq component-of-term x and 2: map-component ( }\lambdak.k-n)x\inkey
b
by (auto simp: in-keys-iff)
from this(1) have 3: map-component ( }\lambdak.k-n+n)x=x by (simp add
map-component-def term-simps)
from 2 have map-component ( }\lambdak.k+n)\mathrm{ (map-component ( }\lambdak.k-n)x)
map-component ( }\lambdak.k+n)`keys b
by (rule imageI)
with 3 have x\in map-component ( }\lambdak.k+n)'keys b by (simp add: map-component-comp
thus ?thesis by simp
qed
qed
lemma lookup-lift-poly-syz:
lookup (lift-poly-syz n b i)=
(\lambdax. if pair-of-term x = (0,i) then 1 else if n \leqcomponent-of-term x then
lookup b (map-component ( }\lambdak.k-n)x) else 0
unfolding lift-poly-syz-def
proof (rule Abs-poly-mapping-inverse)
from finite-keys have finite (map-component ( }\lambdak.k+n)\mathrm{ ' keys b) ..
hence finite (insert (term-of-pair (0,i)) (map-component ( }\lambdak.k+n)'keys b)
by (rule finite.insertI)
with keys-lift-poly-syz-aux
have finite {x. (if pair-of-term x= (0,i) then 1
else if n\leq component-of-term x then lookup b (map-component
(\lambdak.k-n) x)
else 0)}\not=0
by (rule finite-subset)
thus ( }\lambdax\mathrm{ . if pair-of-term x = (0, i) then 1
else if n \leq component-of-term x then lookup b (map-component ( }\lambdak.

- n) x)
else 0) \in
{f. finite {x.fx\not=0}} by simp
qed

```
```

corollary lookup-lift-poly-syz-alt:
lookup (lift-poly-syz n b i) (term-of-pair ( }t,j))
(if (t,j) = (0, i) then 1 else if n \leq j then lookup b (term-of-pair (t, j-
n)) else 0)
by (simp only: lookup-lift-poly-syz term-simps)
lemma keys-lift-poly-syz:
keys (lift-poly-syz nbi)= insert (term-of-pair (0,i)) (map-component ( }\lambdak.k
n)'keys b)
proof
have keys (lift-poly-syz n b i)\subseteq
{x.(if pair-of-term x = (0,i) then 1
else if n\leq component-of-term x then lookup b (map-component ( }\lambdak.

- n) x)
else 0) }=0
(is - \subseteq?A)
proof
fix }
assume x \in keys(lift-poly-syz n b i)
hence lookup (lift-poly-syz n b i) x\not=0 by (simp add: in-keys-iff)
thus }x\in\mathrm{ ?A by (simp add: lookup-lift-poly-syz)
qed
also note keys-lift-poly-syz-aux
finally show keys (lift-poly-syz n b i)\subseteq insert (term-of-pair (0,i)) (map-component
(\lambdak.k+n)'keys b).
next
show insert (term-of-pair (0, i)) (map-component ( }\lambdak.k+n)`keys b)\subseteq key
(lift-poly-syz n b i)
proof (simp, rule)
have lookup (lift-poly-syz n b i) (term-of-pair (0, i)) = 0 by (simp add:
lookup-lift-poly-syz-alt)
thus term-of-pair (0,i)\in keys(lift-poly-syz n b i) by (simp add: in-keys-iff)
next
show map-component ( }\lambdak.k+n)'keys b\subseteqkeys(lift-poly-syz n b i
proof (rule, elim imageE, simp)
fix }
assume x keys b
hence lookup (lift-poly-syz n b i) (map-component ( }\lambdak.k+n)x)\not=
by (simp add: in-keys-iff lookup-lift-poly-syz-alt map-component-def term-simps)
thus map-component ( }\lambdak.k+n)x\in\mathrm{ keys (lift-poly-syz n b i) by (simp add:
in-keys-iff)
qed
qed
qed

```
18.4.2 proj-poly-syz
lemma inj-map-component-plus: inj (map-component \((\lambda k . k+n)\) )
proof (rule injI)
fix \(x y\)
have inj ( \(\lambda k:: n a t . k+n\) ) by (simp add: inj-def)
moreover assume map-component \((\lambda k . k+n) x=\) map-component \((\lambda k . k+\) n) \(y\)
ultimately show \(x=y\) by (rule map-component-inj)
qed
lemma lookup-proj-poly-syz: lookup (proj-poly-syz n \(p\) ) \(x=\) lookup \(p\) (map-component \((\lambda k . k+n) x)\)
by (simp add: proj-poly-syz-def map-key.rep-eq[OF inj-map-component-plus])
lemma lookup-proj-poly-syz-alt:
lookup (proj-poly-syz n p) (term-of-pair \((t, i))=\) lookup \(p(\) term-of-pair \((t, i+\) n))
by (simp add: lookup-proj-poly-syz map-component-term-of-pair)
lemma keys-proj-poly-syz: keys (proj-poly-syz \(n\) p) \(=\) map-component \((\lambda k . k+n)\)
-' keys \(p\)
by (simp add: proj-poly-syz-def keys-map-key[OF inj-map-component-plus])
lemma proj-poly-syz-zero [simp]: proj-poly-syz \(n 0=0\)
by (rule poly-mapping-eqI, simp add: lookup-proj-poly-syz)
lemma proj-poly-syz-plus: proj-poly-syz \(n(p+q)=\) proj-poly-syz \(n p+\) proj-poly-syz \(n q\)
by (simp add: proj-poly-syz-def map-key-plus[OF inj-map-component-plus])
lemma proj-poly-syz-sum: proj-poly-syz \(n(\operatorname{sum} f A)=\left(\sum a \in A\right.\). proj-poly-syz \(n(f\) a))
by (rule fun-sum-commute, simp-all add: proj-poly-syz-plus)
lemma proj-poly-syz-sum-list: proj-poly-syz \(n\) (sum-list xs) \(=\) sum-list (map (proj-poly-syz
n) \(x s\) )
by (rule fun-sum-list-commute, simp-all add: proj-poly-syz-plus)
lemma proj-poly-syz-monom-mult:
proj-poly-syz \(n\) (monom-mult ctp) \(\begin{aligned} & \text { monom-mult } \operatorname{ct}(\text { proj-poly-syz } n ~ p) ~\end{aligned}\)
by (rule poly-mapping-eqI,
simp add: lookup-proj-poly-syz lookup-monom-mult term-simps adds-pp-def sminus-def)
lemma proj-poly-syz-mult-scalar:
proj-poly-syz \(n\) (mult-scalar q \(p\) ) \(=\) mult-scalar \(q\) (proj-poly-syz \(n\) p)
by (rule fun-mult-scalar-commute, simp-all add: proj-poly-syz-plus proj-poly-syz-monom-mult)
lemma proj-poly-syz-lift-poly-syz:
assumes \(i<n\)
shows proj-poly-syz \(n\) (lift-poly-syz \(n\) p \(i\) ) \(=p\)
proof (rule poly-mapping-eqI, simp add: lookup-proj-poly-syz lookup-lift-poly-syz
```

term-simps map-component-comp,
rule, elim conjE)
fix $x:: ' t$
assume component-of-term $x+n=i$
hence $n \leq i$ by simp
with assms show lookup p $x=1$ by simp
qed
lemma proj-poly-syz-eq-zero-iff: proj-poly-syz n $p=0 \longleftrightarrow$ (component-of-term'
keys $p \subseteq\{0 . .<n\}$ )
unfolding keys-eq-empty[symmetric] keys-proj-poly-syz
proof
assume map-component $(\lambda k . k+n)-‘$ keys $p=\{ \}$ (is ? $A=\{ \})$
show component-of-term' keys $p \subseteq\{0 . .<n\}$
proof (rule, rule ccontr)
fix $i$
assume $i \in$ component-of-term' keys $p$
then obtain $x$ where $x: x \in$ keys $p$ and $i: i=$ component-of-term $x$..
assume $i \notin\{0 . .<n\}$
hence $i-n+n=i$ by simp
hence 1: map-component $(\lambda k . k-n+n) x=x$ by (simp add: map-component-def
i term-simps)
have map-component $(\lambda k . k-n) x \in ? A$ by (rule vimageI2, simp add:
map-component-comp $x$ 1)
thus False by (simp add: <?A = \{\}〉)
qed
next
assume $a$ : component-of-term' keys $p \subseteq\{0 . .<n\}$
show map-component $(\lambda k . k+n)-‘$ keys $p=\{ \}$ (is ? $A=\{ \})$
proof (rule ccontr)
assume $? A \neq\{ \}$
then obtain $x$ where $x \in$ ? A by blast
hence map-component $(\lambda k . k+n) x \in$ keys $p$ by (rule vimageD)
with $a$ have component-of-term (map-component $(\lambda k . k+n) x) \in\{0 . .<n\}$
by blast
thus False by (simp add: term-simps)
qed
qed
lemma component-of-lt-ge:
assumes is-pot-ord and proj-poly-syz $n p \neq 0$
shows $n \leq$ component-of-term (lt $p$ )
proof -
from assms(2) have $\neg$ component-of-term' keys $p \subseteq\{0 . .<n\}$ by (simp add:
proj-poly-syz-eq-zero-iff)
then obtain $i$ where $i \in$ component-of-term'keys $p$ and $i \notin\{0 . .<n\}$ by
fastforce
from this(1) obtain $x$ where $x \in$ keys $p$ and $i: i=$ component-of-term $x$..
from this(1) have $x \preceq_{t}$ lt $p$ by (rule lt-max-keys)

```
with \(\operatorname{assms}(1)\) have component-of-term \(x \leq\) component-of-term (lt p) by (rule is-pot-ordD2)
with \(\langle i \notin\{0 . .<n\}\rangle\) show ?thesis by (simp add: i)
qed
lemma lt-proj-poly-syz:
assumes is-pot-ord and proj-poly-syz n \(p \neq 0\)
shows \(l t\) (proj-poly-syz \(n p)=\) map-component \((\lambda k . k-n)(l t p)(\) is \(-=? l)\)
proof -
from component-of-lt-ge[OF assms]
have component-of-term (lt p) - \(n+n=\) component-of-term (lt p) by simp
hence eq: map-component \((\lambda k . k-n+n)(l t p)=l t p\) by (simp add: map-component-def term-simps)
show ?thesis
proof (rule lt-eqI)
have lookup (proj-poly-syz n p) ?l = lc p
by (simp add: lc-def lookup-proj-poly-syz term-simps map-component-comp eq)
also have...\(\neq 0\)
proof (rule lc-not-0, rule)
assume \(p=0\)
hence proj-poly-syz \(n p=0\) by simp
with assms(2) show False ..
qed
finally show lookup (proj-poly-syz \(n\) p) ?l \(\neq 0\).
next
fix \(x\)
assume lookup (proj-poly-syz n p) \(x \neq 0\)
hence map-component \((\lambda k . k+n) x \in\) keys \(p\) by (simp add: in-keys-iff lookup-proj-poly-syz)
hence map-component \((\lambda k . k+n) x \preceq_{t}\) lt \(p\) by (rule lt-max-keys)
with \(\operatorname{assms}(1)\) show \(x \preceq_{t}\) ?l by (auto simp add: is-pot-ord term-simps)
qed
qed
lemma proj-proj-poly-syz: proj-poly \(k\) (proj-poly-syz \(n p)=\) proj-poly \((k+n) p\)
by (rule poly-mapping-eqI, simp add: lookup-proj-poly lookup-proj-poly-syz-alt)
lemma poly-mapping-eqI-proj-syz:
assumes proj-poly-syz \(n p=\) proj-poly-syz \(n q\)
and \(\bigwedge k . k<n \Longrightarrow\) proj-poly \(k p=\) proj-poly \(k q\)
shows \(p=q\)
proof (rule poly-mapping-eqI-proj)
fix \(k\)
show proj-poly \(k p=\) proj-poly \(k q\)
proof (cases \(k<n\) )
case True
thus ?thesis by (rule assms(2))
next
case False
```

    have proj-poly (k-n+n) p= proj-poly (k-n+n)q
            by (simp only: proj-proj-poly-syz[symmetric] assms(1))
    with False show ?thesis by simp
    qed
    qed

```

\subsection*{18.4.3 cofactor-list-syz}
lemma length-cofactor-list-syz [simp]: length (cofactor-list-syz \(n\) p) \(=n\)
by (simp add: cofactor-list-syz-def)
lemma cofactor-list-syz-nth:
assumes \(i<n\)
shows (cofactor-list-syz \(n\) p) ! \(i=\) proj-poly \(i p\)
by (simp add: cofactor-list-syz-def map-idx-nth assms)
lemma cofactor-list-syz-zero [simp]: cofactor-list-syz \(n 0=\) replicate \(n 0\)
by (rule nth-equalityI, simp-all add: cofactor-list-syz-nth proj-zero)
lemma cofactor-list-syz-plus:
cofactor-list-syz \(n(p+q)=\) map2 \((+)\) (cofactor-list-syz \(n p)(\) cofactor-list-syz \(n\) q)
by (rule nth-equalityI, simp-all add: cofactor-list-syz-nth proj-plus)

\subsection*{18.4.4 init-syzygy-list}
lemma length-init-syzygy-list [simp]: length (init-syzygy-list bs) \(=\) length \(b s\) by (simp add: init-syzygy-list-def)
lemma init-syzygy-list-nth:
assumes \(i<\) length \(b s\)
shows (init-syzygy-list bs)!i=lift-poly-syz (length bs) (bs!i) i
by (simp add: init-syzygy-list-def map-idx-nth[OF assms])
lemma Keys-init-syzygy-list:
Keys \((\) set \((\) init-syzygy-list bs \())=\) map-component \((\lambda k . k+\) length bs)'Keys \((\) set bs \() \cup(\lambda i\).term-of-pair \((0, i))\)
' \(\{0 . .<\) length \(b s\}\)
proof -
have eq1: \((\bigcup b \in\) set \(b s\). map-component \((\lambda k . k+\) length \(b s)\) ' keys \(b)=\)
\((\bigcup i \in\{0 . .<\) length \(b s\}\). map-component \((\lambda k . k+\) length bs)'keys (bs !
i)) by (fact UN-upt[symmetric])
have eq2: ( \(\lambda\) i. term-of-pair \((0, i))\) ' \(\{0 . .<\) length \(b s\}=(\bigcup i \in\{0 . .<\) length \(b s\}\).
\(\{\) term-of-pair \((0, i)\})\)
by auto
show ?thesis
by (simp add: init-syzygy-list-def set-map-idx Keys-def keys-lift-poly-syz im-age-UN
eq1 eq2 UN-Un-distrib[symmetric])

\section*{qed}
lemma pp-of-Keys-init-syzygy-list-subset:
pp-of-term'Keys \((\) set (init-syzygy-list bs)) \(\subseteq\) insert 0 (pp-of-term' Keys (set bs))
by (auto simp add: Keys-init-syzygy-list image-Un rev-image-eqI term-simps)
lemma pp-of-Keys-init-syzygy-list-superset:
pp-of-term'Keys (set bs) \(\subseteq\) pp-of-term'Keys (set (init-syzygy-list bs))
by (simp add: Keys-init-syzygy-list image-Un term-simps image-image)
lemma pp-of-Keys-init-syzygy-list:
assumes \(b s \neq[]\)
shows pp-of-term'Keys (set (init-syzygy-list bs)) \(=\) insert 0 (pp-of-term'Keys (set bs))
proof
show insert \(0(p p\)-of-term 'Keys \((\) set bs) \() \subseteq\) pp-of-term 'Keys (set (init-syzygy-list bs))
proof (simp add: pp-of-Keys-init-syzygy-list-superset)
from assms have \(\{0 . .<\) length \(b s\} \neq\{ \}\) by auto
hence Pair 0 ' \(\{0 . .<\) length \(b s\} \neq\{ \}\) by blast
then obtain \(x:: ' t\) where \(x: x \in(\lambda i\). term-of-pair \((0, i))\) ' \(\{0 . .<\) length \(b s\}\) by blast
hence \(p p\)-of-term' \((\lambda i\). term-of-pair \((0, i))\) ' \(\{0 . .<\) length \(b s\}=\{p p\)-of-term \(x\}\) using image-subset-iff by (auto simp: term-simps)
also from \(x\) have \(\ldots=\{0\}\) using pp-of-term-of-pair by auto
finally show \(0 \in p p\)-of-term'Keys (set (init-syzygy-list bs))
by (simp add: Keys-init-syzygy-list image-Un)
qed
qed (fact pp-of-Keys-init-syzygy-list-subset)
lemma component-of-Keys-init-syzygy-list:
component-of-term 'Keys (set (init-syzygy-list bs)) =
\((+)(\) length \(b s)\) ' component-of-term'Keys \((\) set bs \() \cup\{0 . .<\) length bs \(\}\)
by (simp add: Keys-init-syzygy-list image-Un image-comp o-def ac-simps term-simps)
lemma proj-lift-poly-syz:
assumes \(j<n\)
shows proj-poly \(j\) (lift-poly-syz n p i) \(=(1\) when \(j=i)\)
proof (simp add: when-def, intro conjI impI)
assume \(j=i\)
with assms have \(\neg n \leq i\) by simp
show proj-poly \(i(\) lift-poly-syz n p \(i)=1\)
by (rule poly-mapping-eqI, simp add: lookup-proj-poly lookup-lift-poly-syz-alt \(\prec \neg\) \(n \leq i\) 〉lookup-one)
next
assume \(j \neq i\)
from assms have \(\neg n \leq j\) by simp
show proj-poly \(j\) (lift-poly-syz \(n\) p \(i)=0\)
by (rule poly-mapping-eqI, simp add: lookup-proj-poly lookup-lift-poly-syz-alt \(\prec \neg\) \(n \leq j\rangle\langle j \neq i\rangle)\)
qed

\subsection*{18.4.5 proj-orig-basis}
lemma length-proj-orig-basis [simp]: length (proj-orig-basis \(n\) bs) \(=\) length bs by (simp add: proj-orig-basis-def)
lemma proj-orig-basis-nth:
assumes \(i<\) length bs
shows (proj-orig-basis \(n\) bs) ! \(i=\) proj-poly-syz \(n(b s!i)\)
by (simp add: proj-orig-basis-def assms)
lemma proj-orig-basis-init-syzygy-list [simp]:
proj-orig-basis (length bs) (init-syzygy-list bs) \(=b s\)
by (rule nth-equalityI, simp-all add: init-syzygy-list-nth proj-orig-basis-nth proj-poly-syz-lift-poly-syz)
lemma set-proj-orig-basis: set (proj-orig-basis \(n\) bs) \(=\) proj-poly-syz \(n\) 'set bs
by (simp add: proj-orig-basis-def)
The following lemma could be generalized from proj-poly-syz to arbitrary module homomorphisms, i. e. functions respecting 0 , addition and scalar multiplication.
lemma pmdl-proj-orig-basis':
pmdl \((\) set \((\) proj-orig-basis \(n b s))=\) proj-poly-syz \(n\) 'pmdl \((\) set bs) \((\) is \(? A=? B)\)
proof
show ? \(A \subseteq ? B\)
proof
fix \(p\)
assume \(p \in p m d l(\) set (proj-orig-basis \(n \quad b s))\)
thus \(p \in\) proj-poly-syz \(n\) ' \(p m d l\) (set bs)
proof (induct rule: pmdl-induct)
case module-0
have \(0=\) proj-poly-syz \(n 0\) by simp
also from pmdl.span-zero have...\(\in\) proj-poly-syz n'pmdl (set bs) by (rule imageI)
finally show ?case .
next
case (module-plus pbct)
from module-plus(2) obtain \(q\) where \(q \in \operatorname{pmdl}(\) set \(b s)\) and \(p: p=\) proj-poly-syz \(n\) q..
from module-plus(3) obtain \(a\) where \(a \in\) set \(b s\) and \(b: b=\) proj-poly-syz \(n\)
\(a\)
unfolding set-proj-orig-basis ..
have \(p+\) monom-mult \(c t b=\) proj-poly-syz \(n(q+\) monom-mult c \(t a)\)
by (simp add: p b proj-poly-syz-monom-mult proj-poly-syz-plus)
also have ... \(\in\) proj-poly-syz \(n\) ' pmdl (set bs)
proof (rule imageI, rule pmdl.span-add)
```

            show monom-mult c t a f pmdl (set bs)
            by (rule pmdl-closed-monom-mult, rule pmdl.span-base, fact)
        qed fact
        finally show ?case .
        qed
    qed
    next
show ?B\subseteq?A
proof
fix }
assume p f proj-poly-syz n` pmdl (set bs)
then obtain q}\mathrm{ where q
from this(1) show p\inpmdl (set (proj-orig-basis n bs)) unfolding p
proof (induct rule: pmdl-induct)
case module-0
have proj-poly-syz n 0 = 0 by simp
also have ... \in pmdl (set (proj-orig-basis n bs)) by (fact pmdl.span-zero)
finally show ?case .
next
case (module-plus q b c t)
have proj-poly-syz n (q + monom-mult c t b) =
proj-poly-syz n q + monom-mult c t (proj-poly-syz n b)
by (simp add: proj-poly-syz-plus proj-poly-syz-monom-mult)
also have ... \in pmdl (set (proj-orig-basis n bs))
proof (rule pmdl.span-add)
show monom-mult c t (proj-poly-syz n b) \in pmdl (set (proj-orig-basis n bs))
proof (rule pmdl-closed-monom-mult, rule pmdl.span-base)
show proj-poly-syz n b \in set (proj-orig-basis n bs)
by (simp add: set-proj-orig-basis, rule imageI, fact)
qed
qed fact
finally show ?case .
qed
qed
qed

```

\subsection*{18.4.6 filter-syzygy-basis}
```

lemma filter-syzygy-basis-alt: filter-syzygy-basis $n b s=[b \leftarrow b s$. proj-poly-syz $n b=$ 0 ]
by (simp add: filter-syzygy-basis-def proj-poly-syz-eq-zero-iff)
lemma set-filter-syzygy-basis:
set (filter-syzygy-basis $n$ bs) $=\{b \in$ set bs. proj-poly-syz $n b=0\}$
by (simp add: filter-syzygy-basis-alt)

```

\subsection*{18.4.7 syzygy-module-list}
```

lemma syzygy-module-listI:

```
assumes \(s^{\prime} \in p m d l . s y z y g y-m o d u l e ~(s e t ~ b s)\) and \(s=\) atomize-poly (idx-pm-of-pm \(b s s^{\prime}\) )
shows \(s \in\) syzygy-module-list bs
unfolding assms(2) syzygy-module-list-def by (intro imageI, fact assms(1))
lemma syzygy-module-listE:
assumes \(s \in\) syzygy-module-list bs
obtains \(s^{\prime}\) where \(s^{\prime} \in p m d l . s y z y g y-m o d u l e ~(s e t ~ b s)\) and \(s=\) atomize-poly (idx-pm-of-pm \(b s s^{\prime}\) )
using assms unfolding syzygy-module-list-def by (elim imageE, simp)
lemma monom-mult-atomize:
monom-mult ct(atomize-poly p) = atomize-poly (MPoly-Type-Class.punit.monom-mult (monomial ct) 0 p)
by (rule poly-mapping-eqI-proj, simp add: proj-monom-mult proj-atomize-poly MPoly-Type-Class.punit.lookup-monom-mult times-monomial-left)
lemma punit-monom-mult-monomial-idx-pm-of-pm:
MPoly-Type-Class.punit.monom-mult (monomial c t) ( \(0:: n a t\) ) (idx-pm-of-pm bs
s) \(=\)
idx-pm-of-pm bs (MPoly-Type-Class.punit.monom-mult (monomial c \(t\) ) ( \(0::\) ' \(t\) \(\Rightarrow_{0}{ }^{\prime} b::\) ring-1) \(s\) )
by (rule poly-mapping-eqI, simp add: MPoly-Type-Class.punit.lookup-monom-mult lookup-idx-pm-of-pm when-def)
lemma syzygy-module-list-closed-monom-mult:
assumes \(s \in\) syzygy-module-list bs
shows monom-mult cts syzygy-module-list bs
proof -
from assms obtain \(s^{\prime}\) where \(s^{\prime}: s^{\prime} \in p m d l\).syzygy-module (set bs)
and \(s: s=\) atomize-poly (idx-pm-of-pm bs \(s^{\prime}\) ) by (rule syzygy-module-listE)
show ?thesis unfolding \(s\)
proof (rule syzygy-module-listI)
from \(s^{\prime}\) show (monomial \(c t\) ) \(\cdot s^{\prime} \in\) pmdl.syzygy-module (set bs)
by (rule syzygy-module-closed-map-scale)
next
show monom-mult \(c t\) (atomize-poly \(\left.\left(i d x-p m-o f-p m b s s^{\prime}\right)\right)=\) atomize-poly (idx-pm-of-pm bs ((monomial ct) \(\left.s^{\prime}\right)\) )
by (simp add: monom-mult-atomize punit-monom-mult-monomial-idx-pm-of-pm MPoly-Type-Class.punit.map-scale-eq-monom-mult)
qed
qed
lemma pmdl-syzygy-module-list [simp]: pmdl (syzygy-module-list bs) = syzygy-module-list bs
proof (rule pmdl-idI)
show \(0 \in\) syzygy-module-list bs
by (rule syzygy-module-listI, fact pmdl.zero-in-syzygy-module, simp add: atom-ize-zero)
```

next
fix s1 s2
assume s1 \in syzygy-module-list bs
then obtain s1' where s1': s1' \in pmdl.syzygy-module (set bs)
and s1:s1 = atomize-poly (idx-pm-of-pm bs s1') by (rule syzygy-module-listE)
assume s2 \in syzygy-module-list bs
then obtain }s\mp@subsup{Q}{}{\prime}\mathrm{ where }s\mp@subsup{\mathcal{R}}{}{\prime}:s\mp@subsup{\mathcal{R}}{}{\prime}\in\mathrm{ pmdl.syzygy-module (set bs)
and s2: s2 = atomize-poly (idx-pm-of-pm bs s2') by (rule syzygy-module-listE)
show s1 + s2 \in syzygy-module-list bs
proof (rule syzygy-module-listI)
from s1' s\mp@subsup{2}{}{\prime}}\mathrm{ show }s\mp@subsup{1}{}{\prime}+s\mp@subsup{Q}{}{\prime}\inpmdl.syzygy-module (set bs
by (rule pmdl.syzygy-module-closed-plus)
next
show s1 + s2 = atomize-poly (idx-pm-of-pmbs (s1'}+s2')
by (simp add: idx-pm-of-pm-plus atomize-plus s1 s2)
qed
qed (fact syzygy-module-list-closed-monom-mult)

```

The following lemma also holds without the distinctness constraint on \(b s\), but then the proof becomes more difficult.
```

lemma syzygy-module-listI':
assumes distinct bs and sum-list (map2 mult-scalar (cofactor-list-syz (length bs)
s) bs)=0
and component-of-term'keys s\subseteq{0..<length bs}
shows s\in syzygy-module-list bs
proof (rule syzygy-module-listI)
show pm-of-idx-pm bs (vectorize-poly s) \in pmdl.syzygy-module (set bs)
proof (rule pmdl.syzygy-moduleI, rule pmdl.representsI)
have ( }\sumv\inkeys (pm-of-idx-pm bs (vectorize-poly s))
mult-scalar (lookup (pm-of-idx-pm bs (vectorize-poly s)) v) v)=
(\sumb\inset bs. mult-scalar (lookup (pm-of-idx-pm bs (vectorize-poly s)) b) b)
by (rule sum.mono-neutral-left, fact finite-set, fact keys-pm-of-idx-pm-subset,
simp add: in-keys-iff)
also have ... = sum-list (map ( }\lambda\mathrm{ b. mult-scalar (lookup (pm-of-idx-pm bs
(vectorize-poly s)) b) b) bs)
by (simp only: sum-code distinct-remdups-id[OF assms(1)])
also have ... = sum-list (map2 mult-scalar (cofactor-list-syz (length bs) s) bs)
proof (rule arg-cong[of - sum-list], rule nth-equalityI, simp-all)
fix }
assume i< length bs
with assms(1) have lookup (pm-of-idx-pm bs (vectorize-poly s)) (bs!i)=
cofactor-list-syz (length bs) s!i
by (simp add: lookup-pm-of-idx-pm-distinct[OF assms(1)] cofactor-list-syz-nth
lookup-vectorize-poly)
thus mult-scalar (lookup (pm-of-idx-pm bs (vectorize-poly s)) (bs!i)) (bs!i)
=
mult-scalar (cofactor-list-syz (length bs) s!i)(bs!i) by (simp only:)
qed
also have ... = 0 by (fact assms(2))

```
```

finally show pmdl.rep (pm-of-idx-pm bs (vectorize-poly s) $)=0$ by (simp only: pmdl.rep-def)
qed (fact keys-pm-of-idx-pm-subset)
next
from assms(3) have keys (vectorize-poly s) $\subseteq\{0 . .<$ length $b s\}$ by (simp add: keys-vectorize-poly)
with $\operatorname{assms}(1)$ have $i d x$-pm-of-pm bs (pm-of-idx-pm bs (vectorize-poly $s))=$ vectorize-poly s
by (rule idx-pm-of-pm-of-idx-pm)
thus $s=$ atomize-poly (idx-pm-of-pm bs (pm-of-idx-pm bs (vectorize-poly s)))
by (simp add: atomize-vectorize-poly)
qed
lemma component-of-syzygy-module-list:
assumes $s \in$ syzygy-module-list bs
shows component-of-term'keys $s \subseteq\{0 . .<$ length $b s\}$
proof -
from assms obtain $s^{\prime}$ where $s: s=$ atomize-poly (idx-pm-of-pm bs $s^{\prime}$ )
by (rule syzygy-module-listE)
have component-of-term'keys $s \subseteq(\bigcup x \in\{0 . .<$ length $b s\} .\{x\})$
by (simp only: s keys-atomize-poly image-UN, rule UN-mono, fact keys-idx-pm-of-pm-subset, auto simp: term-simps)
also have $\ldots=\{0 . .<$ length bs $\}$ by simp
finally show ?thesis.
qed
lemma map2-mult-scalar-proj-poly-syz:
map2 mult-scalar xs (map (proj-poly-syz n) ys) =
map (proj-poly-syz $n \circ(\lambda(x, y)$. mult-scalar $x y))(z i p x s$ ys)
by (rule nth-equalityI, simp-all add: proj-poly-syz-mult-scalar)
lemma map2-times-proj:
map2 (*) xs (map (proj-poly k) ys) $=$ map $($ proj-poly $k \circ(\lambda(x, y) . x \odot y))(z i p$ xs ys)
by (rule nth-equalityI, simp-all add: proj-mult-scalar)

```

Probably the following lemma also holds without the distinctness constraint on \(b s\).
```

lemma syzygy-module-list-subset:
assumes distinct bs
shows syzygy-module-list $b s \subseteq p m d l($ set (init-syzygy-list bs))
proof
let ?as = init-syzygy-list bs
fix $s$
assume $s \in$ syzygy-module-list bs
then obtain $s^{\prime}$ where $s^{\prime}: s^{\prime} \in p m d l$.syzygy-module (set bs)
and $s: s=$ atomize-poly (idx-pm-of-pm bs $s^{\prime}$ ) by (rule syzygy-module-listE)
from $s^{\prime}$ have pmdl.represents (set bs) $s^{\prime} 0$ by (rule pmdl.syzygy-moduleD)
hence keys $s^{\prime} \subseteq$ set bs and 1:0 pmdl.rep $s^{\prime}$

```
```

    by (rule pmdl.representsD1, rule pmdl.representsD2)
    have s= sum-list (map2 mult-scalar (cofactor-list-syz (length bs) s) (init-syzygy-list
    bs))
(is - = ?r)
proof (rule poly-mapping-eqI-proj-syz)
have proj-poly-syz (length bs) ?r =
sum-list (map2 mult-scalar (cofactor-list-syz (length bs) s)
(map (proj-poly-syz (length bs)) (init-syzygy-list
bs)))
by (simp add: proj-poly-syz-sum-list map2-mult-scalar-proj-poly-syz)
also have ... = sum-list (map2 mult-scalar (cofactor-list-syz (length bs) s) bs)
by (simp add: proj-orig-basis-def[symmetric])
also have ... = sum-list (map ( }\lambda\mathrm{ b. mult-scalar (lookup s' b) b) bs)
proof (rule arg-cong[of - sum-list], rule nth-equalityI, simp-all)
fix }
assume i< length bs
with assms(1) have lookup s'(bs!i)= cofactor-list-syz (length bs) s!i
by (simp add: s cofactor-list-syz-nth lookup-idx-pm-of-pm proj-atomize-poly)
thus mult-scalar (cofactor-list-syz (length bs) s!i) (bs!i)=
mult-scalar (lookup s'(bs!i)) (bs!i) by (simp only:)
qed
also have ... =( \sumb\inset bs. mult-scalar (lookup s' b) b)
by (simp only: sum-code distinct-remdups-id[OF assms])
also have ... = (\sumv\inkeys s'. mult-scalar (lookup s'v)v)
by (rule sum.mono-neutral-right, fact finite-set, fact, simp add: in-keys-iff)
also have ... = 0 by (simp add: 1 pmdl.rep-def)
finally have eq: proj-poly-syz (length bs) ?r = 0 .
show proj-poly-syz (length bs) s= proj-poly-syz (length bs) ?r
by (simp add: eq«s \in syzygy-module-list bs` proj-poly-syz-eq-zero-iff compo- nent-of-syzygy-module-list)     next         fix }         assume k< length bs         have proj-poly ks= map2 (*) (cofactor-list-syz (length bs) s) (map (proj-poly k)                                     (init-syzygy-list bs))!k             by (simp add: <k < length bs` init-syzygy-list-nth proj-lift-poly-syz cofac-
tor-list-syz-nth)
also have ... = sum-list (map2 (*) (cofactor-list-syz (length bs) s)
(map (proj-poly k) (init-syzygy-list bs)))
by (rule sum-list-eq-nthI[symmetric],
simp-all add:<k < length bs` init-syzygy-list-nth proj-lift-poly-syz)
also have ... = proj-poly k?r
by (simp add: proj-sum-list map2-times-proj)
finally show proj-poly ks= proj-poly k?r.
qed
also have ...\in pmdl (set (init-syzygy-list bs)) by (fact pmdl.span-listI)
finally show s\inpmdl (set (init-syzygy-list bs)).
qed

```

\subsection*{18.4.8 Cofactors}
lemma map2-mult-scalar-plus:
map2 \((\odot)(\) map2 \((+) x s y s)\) zs \(=\) map2 \((+)(\) map2 \((\odot) x s z s)(\) map2 \((\odot) y s z s)\) by (rule nth-equalityI, simp-all add: mult-scalar-distrib-right)
lemma syz-cofactors:
assumes \(p \in p m d l(\) set (init-syzygy-list bs))
shows proj-poly-syz (length bs) \(p=\) sum-list (map2 mult-scalar (cofactor-list-syz
(length bs) p) bs)
using assms
proof (induct rule: pmdl-induct)
case module-0
show ?case by (simp, rule sum-list-zeroI', simp)
next
case (module-plus p b c t)
from this(3) obtain \(i\) where \(i: i<\) length \(b s\) and \(b: b=(\) init-syzygy-list \(b s)!i\)
unfolding length-init-syzygy-list[symmetric, of bs] by (metis in-set-conv-nth)
have proj-poly-syz (length bs) \((p+\) monom-mult ctb) \(=\)
proj-poly-syz (length bs) \(p+\) monom-mult \(c t(b s!i)\)
by (simp only: proj-poly-syz-plus proj-poly-syz-monom-mult b init-syzygy-list-nth[OF i]
```

            proj-poly-syz-lift-poly-syz[OF i])
    ```
also have \(\ldots=\) sum-list (map2 mult-scalar (cofactor-list-syz (length bs) p)bs) + monom-mult \(c t(b s!i)\) by (simp only: module-plus(2))
also have \(\ldots=\) sum-list (map2 mult-scalar (cofactor-list-syz (length bs) \((p+\) monom-mult \(c t b\) )) bs)
proof (simp add: cofactor-list-syz-plus map2-mult-scalar-plus sum-list-map2-plus)
have proj-b: \(j<\) length \(b s \Longrightarrow\) proj-poly \(j b=(1\) when \(j=i)\) for \(j\)
by (simp add: b init-syzygy-list-nth i proj-lift-poly-syz)
have eq: \(j<\) length \(b s \Longrightarrow\) (map2 mult-scalar (cofactor-list-syz (length bs)
(monom-mult \(c t b)) b s)!j=\)
(monom-mult \(c t(b s!i)\) when \(j=i)\) for \(j\)
by (simp add: cofactor-list-syz-nth proj-monom-mult proj-b mult-scalar-monom-mult when-def)
have sum-list (map2 mult-scalar (cofactor-list-syz (length bs) (monom-mult ct b)) \(b s\) ) \(=\)
(map2 mult-scalar (cofactor-list-syz (length bs) (monom-mult ctb)) bs)!i by (rule sum-list-eq-nthI, simp add: i, simp add: eq del: nth-zip nth-map)
also have \(\ldots=\) mult-scalar (punit.monom-mult ct (proj-poly ib)) (bs!i)
by (simp add: i cofactor-list-syz-nth proj-monom-mult)
also have \(\ldots=\) monom-mult \(c t(b s!i)\)
by (simp add: proj-b i mult-scalar-monomial times-monomial-left[symmetric])
finally show monom-mult ct \(t b s!i)=\)
sum-list (map2 mult-scalar (cofactor-list-syz (length bs) (monom-mult ct
b)) \(b s\) )
by ( simp only:)
qed
finally show ?case .
qed

\subsection*{18.4.9 Modules}
lemma pmdl-proj-orig-basis:
assumes pmdl \((\) set gs) \()=p m d l(\) set (init-syzygy-list bs))
shows pmdl (set (proj-orig-basis (length bs) gs)) \(=\) pmdl \((\) set bs)
by (simp add: pmdl-proj-orig-basis' assms,
simp only: pmdl-proj-orig-basis' \({ }^{[s y m m e t r i c] ~ p r o j-o r i g-b a s i s-i n i t-s y z y g y-l i s t) ~}\)
lemma pmdl-filter-syzygy-basis-subset:
assumes distinct bs and pmdl (set gs) \(=p m d l(\) set \((\) init-syzygy-list bs))
shows pmdl (set (filter-syzygy-basis (length bs) gs)) \(\subseteq\) pmdl (syzygy-module-list
bs)
proof (rule pmdl.span-mono, rule)
fix \(s\)
assume \(s \in\) set (filter-syzygy-basis (length bs) gs)
hence \(s \in\) set gs and eq: proj-poly-syz (length bs) \(s=0\)
by (simp-all add: set-filter-syzygy-basis)
from this(1) have \(s \in p m d l\) (set gs) by (rule pmdl.span-base)
hence \(s \in p m d l\) (set (init-syzygy-list bs)) by (simp only: assms)
hence proj-poly-syz (length bs) \(s=\) sum-list (map2 mult-scalar (cofactor-list-syz (length bs) s) bs)
by (rule syz-cofactors)
hence distinct bs and sum-list (map2 mult-scalar (cofactor-list-syz (length bs)
s) \(b s)=0\)
by (simp-all only: eq assms(1))
moreover from eq have component-of-term'keys \(s \subseteq\{0 . .<\) length bs \(\}\) by (simp only: proj-poly-syz-eq-zero-iff)
ultimately show \(s \in\) syzygy-module-list bs by (rule syzygy-module-listI')
qed
lemma ex-filter-syzygy-basis-adds-lt:
assumes is-pot-ord and distinct bs and is-Groebner-basis (set gs)
and \(p m d l(\) set gs \()=p m d l(\) set \((\) init-syzygy-list bs) \()\)
and \(f \in p m d l\) (syzygy-module-list bs) and \(f \neq 0\)
shows \(\exists g \in\) set (filter-syzygy-basis (length bs) gs). \(g \neq 0 \wedge l t g a d d s_{t} l t f\)
proof -
from assms(5) have \(f \in\) syzygy-module-list bs by simp
also from \(\operatorname{assms}(2)\) have \(\ldots \subseteq p m d l(\) set (init-syzygy-list bs))
by (rule syzygy-module-list-subset)
also have \(\ldots=p m d l(s e t ~ g s)\) by (simp only: assms(4))
finally have \(f \in p m d l\) (set gs) .
with \(\operatorname{assms}(3,6)\) obtain \(g\) where \(g \in\) set \(g s\) and \(g \neq 0\)
and adds: lt \(g\) addst \(l t f\) unfolding \(G B\)-alt-3-finite[OF finite-set] by blast
show ?thesis
proof (intro bexI conjI)
show \(g \in\) set (filter-syzygy-basis (length bs) gs)
proof (simp add: set-filter-syzygy-basis, rule)
show proj-poly-syz (length bs) \(g=0\)
proof (rule ccontr)
assume proj-poly-syz (length bs) \(g \neq 0\)
with assms(1) have length \(b s \leq\) component-of-term (lt g) by (rule compo-nent-of-lt-ge)
also from adds have \(\ldots=\) component-of-term (lt f) by (simp add: adds-term-def)
also have ... < length bs
proof -
from \(\langle f \neq 0\rangle\) have \(l t f \in\) keys \(f\) by (rule lt-in-keys)
hence component-of-term (lt \(f\) ) \(\in\) component-of-term' keys \(f\) by (rule imageI)
also from \(\langle f \in\) syzygy-module-list bs have \(\ldots \subseteq\{0 . .<\) length \(b s\}\)
by (rule component-of-syzygy-module-list)
finally show component-of-term (lt \(f\) ) < length bs by simp

\section*{qed}
finally show False ..
qed
qed fact
qed \(f a c t+\)
qed
lemma pmdl-filter-syzygy-basis:
fixes \(b s:\) :( \(' t \Rightarrow_{0}\) ' \(b::\) field) list
assumes is-pot-ord and distinct bs and is-Groebner-basis (set gs) and \(p m d l(\) set gs \()=p m d l(\) set \((\) init-syzygy-list bs \())\)
shows pmdl (set (filter-syzygy-basis (length bs) gs)) \(=\) syzygy-module-list bs proof -
from finite-set
have pmdl (set (filter-syzygy-basis (length bs) gs)) \(=\) pmdl (syzygy-module-list bs)
proof (rule pmdl-eqI-adds-lt-finite)
from \(\operatorname{assms}(2,4)\)
show pmdl \((\) set \((\) filter-syzygy-basis (length bs) gs)) \(\subseteq p m d l(\) syzygy-module-list bs)
by (rule pmdl-filter-syzygy-basis-subset)

\section*{next}
fix \(f\)
assume \(f \in p m d l\) (syzygy-module-list bs) and \(f \neq 0\)
with assms show \(\exists g \in\) set (filter-syzygy-basis (length bs) gs). \(g \neq 0 \wedge l t g a d d s_{t}\) lt f
by (rule ex-filter-syzygy-basis-adds-lt)
qed
thus?thesis by simp
qed

\subsection*{18.4.10 Gröbner Bases}
lemma proj-orig-basis-isGB:
assumes is-pot-ord and is-Groebner-basis (set gs) and pmdl (set gs) \(=p m d l\) (set (init-syzygy-list bs))
shows is-Groebner-basis (set (proj-orig-basis (length bs) gs))
unfolding GB-alt-3-finite[OF finite-set]
proof (intro ballI impI)
fix \(f\)
assume \(f \in p m d l\) (set (proj-orig-basis (length bs) gs))
also have \(\ldots=\) proj-poly-syz (length bs) 'pmdl (set gs) by (fact pmdl-proj-orig-basis')
finally obtain \(h\) where \(h \in p m d l\) (set gs) and \(f: f=\) proj-poly-syz (length bs)
\(h\)..
assume \(f \neq 0\)
with \(\operatorname{assms}(1)\) have \(l t f: l t f=\) map-component ( \(\lambda k . k\) - length bs) (lt \(h\) ) un-
folding \(f\)
by (rule lt-proj-poly-syz)
from \(\langle f \neq 0\rangle\) have \(h \neq 0\) by (auto simp add: \(f\) )
with \(\operatorname{assms}(2)\langle h \in \operatorname{pmdl}(\) set \(g s)\rangle\) obtain \(g\) where \(g \in\) set \(g s\) and \(g \neq 0\)
and \(l t g\) adds \(s_{t}\) lt \(h\) unfolding GB-alt-3-finite[OF finite-set \(]\) by blast
from this(3) have 1: component-of-term (lt g) = component-of-term (lt h) and 2: pp-of-term (lt g) adds pp-of-term (lt h) by (simp-all add: adds-term-def)
let \(? g=\) proj-poly-syz (length bs) \(g\)
have ? \(g \neq 0\)
proof (simp add: proj-poly-syz-eq-zero-iff, rule)
assume component-of-term'keys \(g \subseteq\{0 . .<\) length \(b s\}\)
from \(\operatorname{assms}(1)\langle f \neq 0\rangle\) have length \(b s \leq\) component-of-term (lt \(h\) )
unfolding \(f\) by (rule component-of-lt-ge)
hence component-of-term (lt g) \(\notin\{0 . .<\) length \(b s\}\) by (simp add: 1)
moreover from \(\langle g \neq 0\rangle\) have \(l t g \in\) keys \(g\) by (rule lt-in-keys)
ultimately show False using <component-of-term'keys \(g \subseteq\{0 . .<\) length bs \(\}\) 〉
by blast
qed
with \(\operatorname{assms}(1)\) have ltg: lt ?g = map-component ( \(\lambda k . k-l e n g t h ~ b s)\) (lt g) by
(rule lt-proj-poly-syz)
show \(\exists g \in\) set (proj-orig-basis (length bs) gs). \(g \neq 0 \wedge l t g a d d s_{t} l t f\)
proof (intro bexI conjI)
show lt ?g addst lt \(f\) by (simp add: ltf ltg adds-term-def 12 term-simps)
next
show ? \(g \in\) set (proj-orig-basis (length bs) gs)
unfolding set-proj-orig-basis using \(\langle g \in\) set gs〉 by (rule imageI)
qed fact
qed
lemma filter-syzygy-basis-isGB:
assumes is-pot-ord and distinct bs and is-Groebner-basis (set gs)
and \(p m d l(\) set gs) \(=p m d l(\) set \((\) init-syzygy-list bs \())\)
shows is-Groebner-basis (set (filter-syzygy-basis (length bs) gs))
unfolding GB-alt-3-finite[OF finite-set]
proof (intro balli impI)
fix \(f:: ' t \Rightarrow_{0}{ }^{\prime} b\)
assume \(f \neq 0\)
assume \(f \in p m d l\) (set (filter-syzygy-basis (length bs) gs))
also from assms have \(\ldots=\) syzygy-module-list bs by (rule pmdl-filter-syzygy-basis)
finally have \(f \in p m d l\) (syzygy-module-list bs) by simp
```

    from assms this <f }\not=0\mathrm{ \>
    show }\existsg\inset (filter-syzygy-basis(length bs) gs). g\not=0\wedgelt g addst lt 
    by (rule ex-filter-syzygy-basis-adds-lt)
    qed
end
end

```

\section*{19 Sample Computations of Syzygies}

\author{
theory Syzygy-Examples \\ imports Buchberger Algorithm-Schema-Impl Syzygy Code-Target-Rat begin
}

\subsection*{19.1 Preparations}

We must define the following four 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 monom-mult-pprod :: 'c::semiring-0 $\Rightarrow\left({ }^{\prime} a:: n a t,{ }^{\prime} b:: n a t\right) p p \Rightarrow\left(\left(\left({ }^{\prime} a,{ }^{\prime} b\right)\right.\right.$
$\left.p p \times n a t) \Rightarrow{ }_{0}{ }^{\prime} c\right) \Rightarrow-$
where monom-mult-pprod $=$ pprod.monom-mult
definition mult-scalar-pprod :: (('a::nat, 'b::nat) pp $\Rightarrow_{0}{ }^{\prime} c::$ semiring- 0$) \Rightarrow\left(\left({ }^{\prime} a\right.\right.$,
'b) $\left.p p \times n a t) \Rightarrow{ }_{0}{ }^{\prime} c\right) \Rightarrow$ -
where mult-scalar-pprod $=$ pprod.mult-scalar
definition adds-term-pprod $::\left(\left({ }^{\prime} a:: n a t\right.\right.$, ' $\left.\left.b:: n a t\right) p p \times-\right) \Rightarrow-$
where adds-term-pprod $=$ pprod.adds-term
lemma (in gd-term) compute-trd-aux [code]:
trd-aux fs pro
(if is-zero $p$ then
$r$
else
case find-adds $f s$ (lt p) of
None $\Rightarrow$ trd-aux fs (tail p) (plus-monomial-less r (lc p) (lt p))
| Some $f \Rightarrow$ trd-aux fs (tail $p$ - monom-mult (lc p/lc f) (lp p-lpf) (tail
f)) $r$
)
by (simp only: trd-aux.simps[of fs $p$ r] plus-monomial-less-def is-zero-def)
locale gd-nat-inf-term $=$ gd-nat-term pair-of-term term-of-pair cmp-term
for pair-of-term::'t::nat-term $\Rightarrow$ ('a::\{nat-term,graded-dickson-powerprod $\} \times$
nat)

```
and term-of-pair::('a \(\times n a t) \Rightarrow{ }^{\prime} t\)
and cmp-term
begin
sublocale aux: gd-inf-term pair-of-term term-of-pair
\(\lambda s t\). le-of-nat-term-order cmp-term (term-of-pair ( \(s\), the-min)) (term-of-pair ( \(t\), the-min))
\(\lambda s t\). lt-of-nat-term-order cmp-term (term-of-pair (s, the-min)) (term-of-pair ( \(t\), the-min))
le-of-nat-term-order cmp-term
lt-of-nat-term-order cmp-term ..
definition lift-keys :: nat \(\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b\right)\) oalist-ntm \(\Rightarrow\left({ }^{\prime} t,{ }^{\prime} b::\right.\) semiring-0) oalist-ntm where lift-keys \(i x s=\) oalist-of-list-ntm (map-raw ( \(\lambda k v\). (map-component \(((+) i)\) \((f s t k v)\), snd \(k v))(\) list-of-oalist-ntm \(x s))\)
lemma list-of-oalist-lift-keys:
list-of-oalist-ntm (lift-keys \(i\) xs \()=(\) map-raw \((\lambda k v\). (map-component \(((+) i)(f s t\) \(k v\) ), snd \(k v\) )) (list-of-oalist-ntm \(x s)\) )
unfolding lift-keys-def oops
Regardless of whether the above lemma holds (which might be the case) or not, we can use lift-keys in computations. Now, however, it is implemented rather inefficiently, because the list resulting from the application of map-raw is sorted again. That should not be a big problem though, since lift-keys is applied only once to every input polynomial before computing syzygies.
```

lemma lookup-lift-keys-plus:
lookup (MP-oalist (lift-keys ixs)) (term-of-pair $(t, i+k))=$ lookup (MP-oalist
xs) (term-of-pair $(t, k)$ )
(is ?l=?r)
proof -
let ?f $=\lambda k v::^{\prime} t \times{ }^{\prime} b$. (map-component $((+) i)(f s t k v)$, snd $\left.k v\right)$
obtain $x s^{\prime}$ ox where $x s$ : list-of-oalist-ntm xs $=\left(x s^{\prime}\right.$, ox) by fastforce
from oalist-inv-list-of-oalist-ntm[of xs] have inv: ko-ntm.oalist-inv-raw ox xs ${ }^{\prime}$
by (simp add: xs ko-ntm.oalist-inv-def nat-term-compare-inv-conv)
let ? rel $=$ ko.lt (key-order-of-nat-term-order-inv ox)
have irreflp?rel by (simp add: irreflp-def)
moreover have transp ?rel by (simp add: lt-of-nat-term-order-alt)
moreover from oa-ntm.list-of-oalist-sorted [of xs]
have sorted-wrt (ko.lt (key-order-of-nat-term-order-inv ox)) (map fst xs') by
( simp add: xs)
ultimately have dist1: distinct (map fst $x s^{\prime}$ ) by (rule distinct-sorted-wrt-irrefl)
have $1: u=v$ if map-component $((+) i) u=$ map-component $((+) i) v$ for $u v$
proof -
have $i n j((+)$ i) by (simp add: inj-def)
thus ?thesis using that by (rule map-component-inj)
qed
have dist2: distinct (map fst (map-pair ( $\lambda k v$. (map-component $((+) i)(f s t k v)$,
snd $k v)$ ) $x s^{\prime}$ ))

```
by (rule ko-ntm.distinct-map-pair, fact dist1, simp add: 1)
have ?l = lookup-dflt (map-pair ?f xs') (term-of-pair \((t, i+k))\)
by (simp add: oa-ntm.lookup-def lift-keys-def xs oalist-of-list-ntm-def list-of-oalist-OAlist-ntm ko-ntm.lookup-pair-sort-oalist'[OF dist2])
also have \(\ldots=\) lookup-dflt (map-pair ?f xs') \((f s t(\) ?f (term-of-pair \((t, k), b)))\)
by (simp add: map-component-term-of-pair)
also have \(\ldots=\operatorname{snd}\left(\right.\) ?f (term-of-pair \((t, k)\), lookup-dflt xs \({ }^{\prime}(\) term-of-pair \(\left.(t, k))\right)\) )
by (rule ko-ntm.lookup-dftt-map-pair, fact dist1, auto intro: 1)
also have \(\ldots=\) ?r by (simp add: oa-ntm.lookup-def xs ko-ntm.lookup-dflt-eq-lookup-pair[OF inv])
finally show ?thesis.
qed
lemma keys-lift-keys-subset:
keys \((\) MP-oalist (lift-keys \(i x s)) \subseteq(\) map-component \(((+)\) i))'keys (MP-oalist xs)
(is \(? l \subseteq ? r\) )
proof -
let ?f \(=\lambda k v::^{\prime} t \times\) 'b. (map-component \(((+) i)(f s t ~ k v)\), snd kv)
obtain \(x s^{\prime}\) ox where xs: list-of-oalist-ntm \(x s=\left(x s^{\prime}\right.\), ox) by fastforce
let ?rel \(=k o . l t\) (key-order-of-nat-term-order-inv ox)
have irreflp ?rel by (simp add: irreflp-def)
moreover have transp ?rel by (simp add: lt-of-nat-term-order-alt)
moreover from oa-ntm.list-of-oalist-sorted[of xs]
have sorted-wrt (ko.lt (key-order-of-nat-term-order-inv ox)) (map fst xs') by (simp add: xs)
ultimately have dist1: distinct (map fst \(x s^{\prime}\) ) by (rule distinct-sorted-wrt-irrefl)
have \(1: u=v\) if map-component \(((+) i) u=\) map-component \(((+) i) v\) for \(u v\)
proof -
have inj \(((+)\) i) by (simp add: inj-def)
thus ?thesis using that by (rule map-component-inj)
qed
have dist2: distinct (map fst (map-pair ( \(\lambda k v\). (map-component \(((+) i)(f s t k v)\), snd \(k v\) )) \(\left.x s^{\prime}\right)\) )
by (rule ko-ntm.distinct-map-pair, fact dist1, simp add: 1)
have ?l \(\subseteq f s t\) ' set \((f s t\) (map-raw ?f (list-of-oalist-ntm xs)))
by (auto simp: keys-MP-oalist lift-keys-def oalist-of-list-ntm-def list-of-oalist-OAlist-ntm xs
ko-ntm.set-sort-oalist[OF dist2])
also from ko-ntm.map-raw-subset have \(\ldots \subseteq f s t\) ' ?f ' set (fst (list-of-oalist-ntm xs))
by (rule image-mono)
also have \(\ldots \subseteq\) ? \(r\) by (simp add: keys-MP-oalist image-image)
finally show? thesis.
qed
end
global-interpretation pprod': gd-nat-inf-term \(\lambda x::\left({ }^{\prime} a, ~ ' b\right) ~ p p \times n a t . x \lambda x\). x cmp-term
rewrites pprod.pp-of-term \(=f s t\)
and pprod.component-of-term \(=\) snd
and pprod.splus \(=\) splus-pprod
and pprod.monom-mult \(=\) monom-mult-pprod
and pprod.mult-scalar \(=\) mult-scalar-pprod
and pprod.adds-term \(=\) adds-term-pprod
for cmp-term :: (('a::nat, 'b::nat) pp \(\times\) nat) nat-term-order
defines shift-map-keys-pprod \(=\) pprod' \({ }^{\prime}\).shift-map-keys
and lift-keys-pprod \(=\) pprod'.lift-keys
and min-term-pprod \(=\) pprod'.min-term
and \(l t-\) pprod \(=\) pprod \(^{\prime} . l t\)
and \(l c-\) pprod \(=p p r o d^{\prime} . l c\)
and tail-pprod \(=\) pprod \({ }^{\prime}\). tail
and comp-opt-p-pprod \(=\) pprod \(^{\prime}\).comp-opt-p
and ord- \(p\)-pprod \(=\) pprod \(^{\prime}\). ord \(-p\)
and ord-strict-p-pprod \(=\) pprod' \(^{\prime}\). ord-strict- \(p\)
and find-adds-pprod \(=\) pprod' \(^{\prime}\).find-adds
and trd-aux-pprod \(=\) pprod' \({ }^{\prime}\).trd-aux
and \(\operatorname{trd}\)-pprod \(=\) pprod \(^{\prime} . t r d\)
and spoly-pprod \(=\) pprod \(^{\prime}\).spoly
and count-const-lt-components-pprod \(=\) pprod'.count-const-lt-components
and count-rem-components-pprod \(=\) pprod'. count-rem-components
and const-lt-component-pprod \(=\) pprod'.const-lt-component
and full-gb-pprod \(=\) pprod \(^{\prime} \cdot f u l l-g b\)
and keys-to-list-pprod \(=\) pprod'. .keys-to-list
and Keys-to-list-pprod \(=\) pprod' \(^{\prime}\).Keys-to-list
and add-pairs-single-sorted-pprod \(=\) pprod'.add-pairs-single-sorted
and add-pairs-pprod \(=\) pprod' \(^{\prime} . a d d-p a i r s\)
and canon-pair-order-aux-pprod \(=\) pprod'.canon-pair-order-aux
and canon-basis-order-pprod \(=\) pprod'. canon-basis-order
and new-pairs-sorted-pprod \(=\) pprod'.new-pairs-sorted
and component-crit-pprod \(=\) pprod'.component-crit
and chain-ncrit-pprod \(=\) pprod \(^{\prime}\).chain-ncrit
and chain-ocrit-pprod \(=\) pprod' \(^{\prime}\).chain-ocrit
and apply-icrit-pprod \(=\) pprod \(^{\prime}\). apply-icrit
and apply-ncrit-pprod \(=\) pprod \(^{\prime}\). apply-ncrit
and apply-ocrit-pprod \(=\) pprod \(^{\prime}\).apply-ocrit
and trdsp-pprod \(=\) pprod'.trdsp
and \(g b\)-sel-pprod \(=\) pprod \(^{\prime} . g b\)-sel
and \(g b\)-red-aux-pprod \(=\) pprod \(^{\prime} \cdot g b\)-red-aux
and \(g b\)-red-pprod \(=\) pprod \(^{\prime} \cdot g b\)-red
and \(g b\)-aux-pprod \(=\) pprod \(^{\prime} . g b-a u x\)
and \(g b-\) pprod \(=\) pprod \(^{\prime} \cdot g b\)
and filter-syzygy-basis-pprod \(=\) pprod' \({ }^{\prime}\) aux.filter-syzygy-basis
and init-syzygy-list-pprod \(=\) pprod' \({ }^{\prime}\).aux.init-syzygy-list
and lift-poly-syz-pprod \(=\) pprod'.aux.lift-poly-syz
and map-component-pprod \(=\) pprod \({ }^{\prime}\).map-component
subgoal by (rule gd-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: monom-mult-pprod-def)
subgoal by (simp only: mult-scalar-pprod-def)
subgoal by (simp only: adds-term-pprod-def)
done

```
lemma compute-adds-term-pprod [code]:
        adds-term-pprod \(u v=(\) snd \(u=\) snd \(v \wedge\) adds-pp-add-linorder \((f\) st \(u)(\) 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-shift-map-keys-pprod [code abstract]:
list-of-oalist-ntm (shift-map-keys-pprod \(t f\) xs \()=\) map-raw \((\lambda(k, v)\). (splus-pprod
\(t k, f v)\) ) (list-of-oalist-ntm xs)
by (simp add: pprod'.list-of-oalist-shift-keys case-prod-beta')
lemma compute-trd-pprod [code]: trd-pprod to fs \(p=\) trd-aux-pprod to fs \(p\) (change-ord to 0)
by (simp only: pprod'.trd-def change-ord-def)
lemmas \([\) code \(]=\) conversep-iff
lemma POT-is-pot-ord: pprod'.is-pot-ord (TYPE('a::nat)) (TYPE('b::nat)) (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)
definition \(V e c_{0}::\) nat \(\Rightarrow\left(\left({ }^{\prime} a\right.\right.\), nat \(\left.) p p \Rightarrow_{0}{ }^{\prime} b\right) \Rightarrow\left(\left({ }^{\prime} a:: n a t\right.\right.\), nat \(\left.) p p \times n a t\right) \Rightarrow_{0}\) ' \(b::\) semiring- 1 where

Vec \({ }_{0}\) i \(p=\) mult-scalar-pprod \(p\) (Poly-Mapping.single \(\left.(0, i) 1\right)\)
definition syzygy-basis to bs \(=\)
filter-syzygy-basis-pprod (length bs) (map fst (gb-pprod (POT to) (map ( \(\lambda p .(p\), ())) (init-syzygy-list-pprod bs)) ()))
thm pprod'.aux.filter-syzygy-basis-isGB[OF POT-is-pot-ord]
lemma lift-poly-syz-MP-oalist [code]:
lift-poly-syz-pprod \(n\) (MP-oalist xs) \(i=M P\)-oalist (OAlist-insert-ntm \(((0, i), 1)\)
(lift-keys-pprod \(n\) xs ))
proof (rule poly-mapping-eqI, simp add: pprod'.aux.lookup-lift-poly-syz del: MP-oalist.rep-eq, intro conjI impI)
fix \(v::\left({ }^{\prime} a, ~ ' b\right) p p \times n a t\)
assume \(n \leq\) snd \(v\)
moreover obtain \(t k\) where \(v=(t, k)\) by fastforce
ultimately have \(k\) : \(n+(k-n)=k\) by \(\operatorname{simp}\)
```

    hence \(v: v=(t, n+(k-n))\) by (simp only: \(\langle v=(t, k)\rangle)\)
    assume \(v \neq(0, i)\)
    hence lookup (MP-oalist (OAlist-insert-ntm (( \(0, i\) ), 1) (lift-keys-pprod \(n\) xs \()\) )) v
    $=$
lookup (MP-oalist (lift-keys-pprod $n$ xs)) v by (simp add: oa-ntm.lookup-insert)
also have $\ldots=$ lookup (MP-oalist $x s)(t, k-n)$ by (simp only: v pprod'.lookup-lift-keys-plus)
also have $\ldots=$ lookup (MP-oalist xs) (map-component-pprod $(\lambda k . k-n) v$ )
by (simp add: v pprod'.map-component-term-of-pair)
finally show lookup (MP-oalist xs) (map-component-pprod $(\lambda k . k-n) v)=$
lookup (MP-oalist (OAlist-insert-ntm ( $(0, i), 1)$ (lift-keys-pprod $n$
xs))) v by (rule HOL.sym)
next
fix $v::\left({ }^{\prime} a, ~ ' b\right) p p \times n a t$
assume $\neg n \leq s n d v$
assume $v \neq(0, i)$
hence lookup (MP-oalist (OAlist-insert-ntm (( $0, i$, 1) (lift-keys-pprod $n$ xs $)$ )) v
$=$
lookup (MP-oalist (lift-keys-pprod $n x s)$ ) v by (simp add: add: oa-ntm.lookup-insert)
also have...$=0$
proof (rule ccontr)
assume lookup (MP-oalist (lift-keys-pprod $n$ xs)) $v \neq 0$
hence $v \in$ keys (MP-oalist (lift-keys-pprod $n$ xs)) by (simp add: in-keys-iff del:
MP-oalist.rep-eq)
also have $\ldots \subseteq$ map-component-pprod $((+) n)$ 'keys (MP-oalist xs)
by (fact pprod'.keys-lift-keys-subset)
finally obtain $u$ where $v=$ map-component-pprod $((+) n) u$..
hence snd $v=n+$ snd $u$ by (simp add: pprod'.component-of-map-component)
with $\langle\neg n \leq$ snd $v\rangle$ show False by simp
qed
finally show lookup (MP-oalist (OAlist-insert-ntm ((0, i), 1) (lift-keys-pprod $n$
$x s))) v=0$.
qed (simp-all add: oa-ntm.lookup-insert)

```

\subsection*{19.2 Computations}
```

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

```

\section*{lemma}
```

syzygy-basis DRLEX [ $\operatorname{Vec}_{0} 0\left(X^{2} * Z^{\wedge} 3+3 * X^{2} * Y\right), \operatorname{Vec}_{0} 0(X * Y * Z$ $\left.\left.+2 * Y^{2}\right)\right]=$
$\left[\operatorname{Vec}_{0} 0\left(C_{0}(1 / 3) * X * Y * Z+C_{0}(2 / 3) * Y^{2}\right)+\operatorname{Vec}_{0} 1\left(C_{0}(-1 / 3)\right.\right.$
$\left.\left.* X^{2} * Z^{へ} 3-X^{2} * Y\right)\right]$
by eval
value [code] syzygy-basis DRLEX $\left[\operatorname{Vec}_{0} 0\left(X^{2} * Z^{\wedge} 3+3 * X^{2} * Y\right), \operatorname{Vec}_{0} 0(X\right.$ $\left.\left.* Y * Z+2 * Y^{2}\right), \operatorname{Vec}_{0} 0(X-Y+3 * Z)\right]$

```

\section*{lemma}
```

map fst (gb-pprod (POT DRLEX) (map $(\lambda p .(p,()))$ (init-syzygy-list-pprod

```

```

X-Y)])) ()) =
Vec}001+\mp@subsup{V}{0}{}0\mp@subsup{c}{0}{}3(\mp@subsup{X}{}{\wedge}4+3*\mp@subsup{X}{}{2}*Y)
Vec}011+Vec⿱\mp@code{3( Y^3+2*X*Z),

```

```

    Vec}021+Veco 3 (Z2 - X - Y)
    Vec}01(\mp@subsup{Z}{}{2}-X-Y)-Veco 2 (Y^ 3 + 2*X*Z),
    Vec}00(\mp@subsup{Z}{}{2}-X-Y)-\mp@subsup{V}{0}{
    ```

```

Z-2* * * Z^3)+
Vec
+3* X'2*Y*Z
]
by eval

```
```

lemma

```

```

Z), Vec}00(\mp@subsup{Z}{}{2}-X-Y)]
Vec}00(Y^3+2*X*Z)- Vec⿱\mp@code{1 ( X^4 + 3* X 2 * Y),
Vec}01(\mp@subsup{Z}{}{2}-X-Y)-Veco 2 ( ( ^^3 + 2* * * Z)
Vec}00(\mp@subsup{Z}{}{2}-X-Y)-Vec⿱\mp@code{2 ( X^4 + 3* X'2*Y),

```

```

Z-2* * * Z^3) +

```

```

+3* X'2*Y* Z')
]
by eval

```
value［code］syzygy－basis DRLEX \(\left[\operatorname{Vec}_{0} 0(X * Y-Z), \operatorname{Vec}_{0} 0(X * Z-Y)\right.\) ， \(\left.\operatorname{Vec}_{0} 0(Y * Z-X)\right]\)

\section*{lemma}
```

map fst (gb-pprod (POT DRLEX) ( $\operatorname{map}(\lambda p .(p,()))$ (init-syzygy-list-pprod
$\left.\left.\left.\left[\operatorname{Vec}_{0} 0(X * Y-Z), \operatorname{Vec}_{0} 0(X * Z-Y), \operatorname{Vec}_{0} 0(Y * Z-X)\right]\right)\right)()\right)=$
[
$V e c_{0} 01+\operatorname{Vec}_{0} 3(X * Y-Z)$,
$V_{0} 11+\operatorname{Vec}_{0} 3(X * Z-Y)$,
$\operatorname{Vec}_{0} 21+\operatorname{Vec}_{0} 3(Y * Z-X)$,
$\operatorname{Vec}_{0} 0(-X * Z+Y)+\operatorname{Vec}_{0} 1(X * Y-Z)$,
$\operatorname{Vec}_{0} 0(-Y * Z+X)+\operatorname{Vec}_{0} 2(X * Y-Z)$,
$\operatorname{Vec}_{0} 1(-Y * Z+X)+\operatorname{Vec}_{0} 2(X * Z-Y)$,
$\operatorname{Vec}_{0} 1(-Y)+\operatorname{Vec}_{0} 2(X)+\operatorname{Vec}_{0} 3\left(Y^{\wedge} 2-X\right.$ へ 2 $)$,
$\operatorname{Vec}_{0} 0(Z)+\operatorname{Vec}_{0} 2(-X)+\operatorname{Vec}_{0} 3\left(X^{\wedge}\right.$ 2-Z へ 2),
$\operatorname{Vec}_{0} 0\left(Y-Y * Z^{\text {へ2 }}\right)+\operatorname{Vec}_{0} 1\left(Y^{\text {へ2 }} * Z-Z\right)+\operatorname{Vec}_{0}$ 2 $\left(Y^{\wedge} 2-Z^{\wedge}\right.$
2),
$\operatorname{Vec}_{0} 0(-Y)+\operatorname{Vec}_{0} 1(-(X * Y))+\operatorname{Vec}_{0} 2\left(X^{\wedge}\right.$ 2 -1$)+\operatorname{Vec}_{0} 3(X-$
$X^{\wedge} 3$ )

```
```

] by eval
lemma
syzygy-basis DRLEX [Vec}0(X*Y-Z), Vec 0 0 (X*Z - Y), Vec 0 0 (Y*
Z-X)]=
[
Vec}00(-X*Z+Y)+ Vec % 1 (X*Y-Z),
Vec}00(-Y*Z+X)+\mp@subsup{Vec}{0}{2}2(X*Y-Z)
Vec}01(-Y*Z+X)+\mp@subsup{Vec}{0}{2}2(X*Z-Y)

```

```

2) by eval
end
end
theory Groebner-PM
imports Polynomials.MPoly-PM Reduced-GB
begin
```

We prove results that hold specifically for Gröbner bases in polynomial rings, where the polynomials really have indeterminates.
```

context pm-powerprod
begin

```
lemmas finite-reduced-GB-Polys \(=\)
    punit.finite-reduced-GB-dgrad-p-set[simplified, OF dickson-grading-varnum, where
\(m=0\), simplified dgrad-p-set-varnum]
lemmas reduced-GB-is-reduced-GB-Polys \(=\)
    punit.reduced-GB-is-reduced-GB-dgrad-p-set[simplified, OF dickson-grading-varnum,
where \(m=0\), simplified dgrad-p-set-varnum]
lemmas reduced-GB-is-GB-Polys \(=\)
    punit.reduced-GB-is-GB-dgrad-p-set[simplified, OF dickson-grading-varnum, where
\(m=0\), simplified dgrad-p-set-varnum]
lemmas reduced-GB-is-auto-reduced-Polys \(=\)
    punit.reduced-GB-is-auto-reduced-dgrad-p-set[simplified, OF dickson-grading-varnum,
where \(m=0\), simplified dgrad-p-set-varnum]
lemmas reduced-GB-is-monic-set-Polys \(=\)
    punit.reduced-GB-is-monic-set-dgrad-p-set[simplified, OF dickson-grading-varnum,
where \(m=0\), simplified dgrad-p-set-varnum]
lemmas reduced-GB-nonzero-Polys \(=\)
    punit.reduced-GB-nonzero-dgrad-p-set[simplified, OF dickson-grading-varnum, where
\(m=0\), simplified dgrad-p-set-varnum]
lemmas reduced-GB-ideal-Polys \(=\)
    punit.reduced-GB-pmdl-dgrad-p-set[simplified, OF dickson-grading-varnum, where
\(m=0\), simplified dgrad-p-set-varnum] lemmas reduced-GB-unique-Polys \(=\)
punit.reduced-GB-unique-dgrad-p-set[simplified, OF dickson-grading-varnum, where \(m=0\), simplified dgrad-p-set-varnum]
lemmas reduced-GB-Polys \(=\)
punit.reduced-GB-dgrad-p-set[simplified, OF dickson-grading-varnum, where \(m=0\), simplified dgrad-p-set-varnum]
lemmas ideal-eq-UNIV-iff-reduced-GB-eq-one-Polys \(=\)
ideal-eq-UNIV-iff-reduced-GB-eq-one-dgrad-p-set[simplified, OF dickson-grading-varnum, where \(m=0\), simplified dgrad-p-set-varnum]

\subsection*{19.3 Univariate Polynomials}
lemma (in -) adds-univariate-linear:
assumes finite \(X\) and card \(X \leq 1\) and \(s \in .[X]\) and \(t \in .[X]\)
obtains \(s\) adds \(t \mid t\) adds \(s\)
proof (cases s adds t)
case True
thus ?thesis..
next
case False
then obtain \(x\) where 1: lookup \(t x<l o o k u p\) s \(x\) by (auto simp: adds-poly-mapping
le-fun-def not-le)
hence \(x \in\) keys \(s\) by (simp add: in-keys-iff)
also from \(\operatorname{assms}(3)\) have \(\ldots \subseteq X\) by (rule PPsD)
finally have \(x \in X\).
have \(t\) adds \(s\) unfolding adds-poly-mapping le-fun-def
proof
fix \(y\)
show lookup \(t y \leq l o o k u p s y\)
proof (cases \(y \in\) keys \(t\) )
case True
also from \(\operatorname{assms}(4)\) have keys \(t \subseteq X\) by (rule PPsD)
finally have \(y \in X\).
with \(\operatorname{assms}(1,2)\langle x \in X\rangle\) have \(x=y\) by (simp add: card-le-Suc0-iff-eq)
with 1 show ?thesis by simp
next
case False
thus ?thesis by (simp add: in-keys-iff)
qed
qed
thus ?thesis..
qed
context
fixes \(X\) :: 'x set
assumes fin- \(X\) : finite \(X\) and card- \(X\) : card \(X \leq 1\)
begin
```

lemma ord-iff-adds-univariate:
assumes }s\in.[X]\mathrm{ and }t\in.[X
shows }s\preceqt\longleftrightarrows\mathrm{ adds }
proof
assume }s\preceq
from fin-X card-X assms show s adds t
proof (rule adds-univariate-linear)
assume t adds s
hence t}\preceqs\mathrm{ by (rule ord-adds)
with }\langles\preceqt\rangle\mathrm{ have }s=
by simp
thus ?thesis by simp
qed
qed (rule ord-adds)
lemma adds-iff-deg-le-univariate:
assumes }s\in.[X]\mathrm{ and }t\in.[X
shows s adds t\longleftrightarrowdeg-pm s}\leq\mathrm{ deg-pm t
proof
assume *:deg-pm s \leq deg-pm t
from fin-X card-X assms show s adds t
proof (rule adds-univariate-linear)
assume t adds s
hence t=s using * by (rule adds-deg-pm-antisym)
thus ?thesis by simp
qed
qed (rule deg-pm-mono)
corollary ord-iff-deg-le-univariate: }s\in.[X]\Longrightarrowt\in.[X]\Longrightarrows\preceqt\longleftrightarrowdeg-pm
s deg-pm t
by (simp only: ord-iff-adds-univariate adds-iff-deg-le-univariate)
lemma poly-deg-univariate:
assumes p}\inP[X
shows poly-deg p=deg-pm(lpp p)
proof (cases p=0)
case True
thus ?thesis by simp
next
case False
hence lp-in: lpp p}\inkeys p by (rule punit.lt-in-keys
also from assms have ...\subseteq.[X] by (rule PolysD)
finally have lpp p\in.[X].
show ?thesis
proof (intro antisym poly-deg-leI)
fix }
assume t\in keys p
hence t\preceqlpp p by (rule punit.lt-max-keys)
moreover from <t\in keys p\rangle\langlekeys p\subseteq.[X]> have t\in.[X] ..

```
```

    ultimately show deg-pm t \leq deg-pm (lpp p) using <lpp p \in.[X]>
        by (simp only: ord-iff-deg-le-univariate)
    next
    from lp-in show deg-pm (lpp p) \leq poly-deg p by (rule poly-deg-max-keys)
    qed
    qed

```
lemma reduced-GB-univariate-cases:
    assumes \(F \subseteq P[X]\)
    obtains \(g\) where \(g \in P[X]\) and \(g \neq 0\) and \(l c f g=1\) and punit.reduced- \(G B F\)
\(=\{g\} \mid\)
    punit.reduced-GB \(F=\{ \}\)
proof (cases punit.reduced-GB \(F=\{ \}\) )
    case True
    thus ?thesis..
next
    case False
    let \(? G=\) punit.reduced \(-G B F\)
    from fin-X assms have ar: punit.is-auto-reduced ? \(G\) and \(0 \notin ? G\) and ? \(G \subseteq\)
\(P[X]\)
    and m: punit.is-monic-set ? \(G\)
    by (rule reduced-GB-is-auto-reduced-Polys, rule reduced-GB-nonzero-Polys, rule
reduced-GB-Polys,
    rule reduced-GB-is-monic-set-Polys)
    from False obtain \(g\) where \(g \in\) ? \(G\) by blast
    with \(\langle 0 \notin ? G\rangle\langle ? G \subseteq P[X]\rangle\) have \(g \neq 0\) and \(g \in P[X]\) by blast +
    from this(1) have \(l p-g: l p p g \in\) keys \(g\) by (rule punit.lt-in-keys)
    also from \(\langle g \in P[X]\rangle\) have \(\ldots \subseteq \cdot[X]\) by (rule Polys \(D\) )
    finally have lpp \(g \in .[X]\).
    note \(\langle g \in P[X]\rangle\langle g \neq 0\rangle\)
    moreover from \(m\langle g \in ? G\rangle\langle g \neq 0\rangle\) have \(l c f g=1\) by (rule punit.is-monic-setD)
    moreover have ? \(G=\{g\}\)
    proof
        show ? \(G \subseteq\{g\}\)
        proof
            fix \(g^{\prime}\)
            assume \(g^{\prime} \in ? G\)
            with \(\langle 0 \notin ? G\rangle\langle ? G \subseteq P[X]\rangle\) have \(g^{\prime} \neq 0\) and \(g^{\prime} \in P[X]\) by blast +
            from this(1) have \(l p-g^{\prime}: l p p g^{\prime} \in\) keys \(g^{\prime}\) by (rule punit.lt-in-keys)
            also from \(\left\langle g^{\prime} \in P[X]\right\rangle\) have \(\ldots \subseteq .[X]\) by (rule Polys \(D\) )
            finally have \(l p p g^{\prime} \in .[X]\).
            have \(g^{\prime}=g\)
            proof (rule ccontr)
            assume \(g^{\prime} \neq g\)
            with \(\langle g \in ? G\rangle\left\langle g^{\prime} \in ? G\right\rangle\) have \(g: g \in ? G-\left\{g^{\prime}\right\}\) and \(g^{\prime}: g^{\prime} \in ? G-\{g\}\)
by blast+
            from fin- \(X\) card- \(X\langle l p p g \in .[X]\rangle\left\langle l p p g^{\prime} \in .[X]\right\rangle\) show False
            proof (rule adds-univariate-linear)
            assume *: lpp gadds lpp \(g^{\prime}\)
```

            from ar \langleg' \in ?G> have \neg punit.is-red (?G - {g'}) g' by (rule
    punit.is-auto-reducedD)
moreover from g<g\not=0`lp-\mp@subsup{g}{}{\prime}* have punit.is-red (?G - {g'}) g
by (rule punit.is-red-addsI[simplified])
ultimately show ?thesis ..
next
assume *: lpp g' adds lpp g
from ar }\langleg\in?G\rangle\mathrm{ have }\neg\mathrm{ punit.is-red (?G - {g})g}\mathrm{ by (rule punit.is-auto-reducedD)
moreover from g}\mp@subsup{g}{}{\prime}\langle\mp@subsup{g}{}{\prime}\not=0\ranglelp-g*\mathrm{ have punit.is-red (?G - {g})g
by (rule punit.is-red-addsI[simplified])
ultimately show ?thesis ..
qed
qed
thus g}\mp@subsup{g}{}{\prime}\in{g}\mathrm{ by simp
qed
next
from }\langleg\in?G\rangle\mathrm{ show {g}}\subseteq??G\mathrm{ by simp
qed
ultimately show ?thesis ..
qed
corollary deg-reduced-GB-univariate-le:
assumes }F\subseteqP[X]\mathrm{ and }f\in\mathrm{ ideal }F\mathrm{ and }f\not=0\mathrm{ and g e punit.reduced-GB F
shows poly-deg g}\leq\mathrm{ poly-deg f
using assms(1)
proof (rule reduced-GB-univariate-cases)
let ?G = punit.reduced-GB F
fix g
assume g'\inP[X] and g}\mp@subsup{g}{}{\prime}\not=0\mathrm{ and }G:?G={\mp@subsup{g}{}{\prime}
from fin-X assms(1) have gb: punit.is-Groebner-basis?G and ideal ?G = ideal
F
and ?G\subseteqP[X]
by (rule reduced-GB-is-GB-Polys, rule reduced-GB-ideal-Polys, rule reduced-GB-Polys)
from assms(2) this(2) have f\inideal ?G by simp
with gb obtain }\mp@subsup{g}{}{\prime\prime}\mathrm{ where g'|}\mp@subsup{g}{}{\prime\prime}\inG\mathrm{ and lpp g" adds lpp f
using assms(3) by (rule punit.GB-adds-lt[simplified])
with assms(4) have lpp g adds lpp f by (simp add: G)
hence deg-pm (lpp g) \leq deg-pm (lpp f) by (rule deg-pm-mono)
moreover from assms(4)<?G\subseteqP[X]> have g\inP[X]..
ultimately have poly-deg g\leqdeg-pm (lpp f) by (simp only: poly-deg-univariate)
also from punit.lt-in-keys have ... s poly-deg f by (rule poly-deg-max-keys) fact
finally show ?thesis.
next
assume punit.reduced-GB F={}
with assms(4) show ?thesis by simp
qed
end

```

\subsection*{19.4 Homogeneity}
```

lemma is-reduced-GB-homogeneous:
assumes $\bigwedge f . f \in F \Longrightarrow$ homogeneous $f$ and punit.is-reduced- $G B G$ and ideal
$G=$ ideal $F$
and $g \in G$
shows homogeneous $g$
proof (rule homogeneousI)
fix $s t$
have 1: deg-pm $u=\operatorname{deg}-p m(l p p g)$ if $u \in k e y s g$ for $u$
proof -
from $\operatorname{assms}(4)$ have $g \in$ ideal $G$ by (rule ideal.span-base)
hence $g \in$ ideal $F$ by (simp only: assms(3))
from that have $u \in$ Keys (hom-components g) by (simp only: Keys-hom-components)
then obtain $q$ where $q: q \in$ hom-components $g$ and $u \in$ keys $q$ by (rule
in-KeysE)
from $\operatorname{assms}(1)\langle g \in$ ideal $F\rangle q$ have $q \in$ ideal $F$ by (rule homogeneous-ideal')
from $\operatorname{assms}(2)$ have punit.is-Groebner-basis $G$ by (rule punit.reduced-GB-D1)
moreover from $\langle q \in$ ideal $F\rangle$ have $q \in$ ideal $G$ by (simp only: assms(3))
moreover from $q$ have $q \neq 0$ by (rule hom-components-nonzero)
ultimately obtain $g^{\prime}$ where $g^{\prime} \in G$ and $g^{\prime} \neq 0$ and adds: lpp $g^{\prime}$ adds $l p p q$
by (rule punit.GB-adds-lt[simplified])
from $\langle q \neq 0\rangle$ have $l p p q \in$ keys $q$ by (rule punit.lt-in-keys)
also from $q$ have $\ldots \subseteq$ Keys (hom-components $g$ ) by (rule keys-subset-Keys)
finally have $\operatorname{lpp} q \in$ keys $g$ by (simp only: Keys-hom-components)
with - $\left\langle g^{\prime} \neq 0\right\rangle$ have red: punit.is-red $\left\{g^{\prime}\right\} g$
using adds by (rule punit.is-red-addsI[simplified]) simp
from assms(2) have punit.is-auto-reduced $G$ by (rule punit.reduced-GB-D2)
hence $\neg$ punit.is-red $(G-\{g\}) g$ using assms(4) by (rule punit.is-auto-reducedD)
with red have $\neg\left\{g^{\prime}\right\} \subseteq G-\{g\}$ using punit.is-red-subset by blast
with $\left\langle g^{\prime} \in G\right\rangle$ have $g^{\prime}=g$ by simp
from $\langle l p p q \in$ keys $g\rangle$ have $l p p q \preceq l p p g$ by (rule punit.lt-max-keys)
moreover from adds have $l p p g \preceq l p p q$
unfolding $\left\langle g^{\prime}=g\right\rangle$ by (rule punit.ord-adds-term[simplified])
ultimately have $e q: l p p q=l p p g$
by $\operatorname{simp}$
from $q$ have homogeneous $q$ by (rule hom-components-homogeneous)
hence deg-pm $u=\operatorname{deg-pm~(lpp~q)~}$
using $\langle u \in$ keys $q\rangle\langle l p p q \in$ keys $q\rangle$ by (rule homogeneous $D$ )
thus ?thesis by (simp only: eq)
qed
assume $s \in$ keys $g$
hence 2: deg-pm $s=\operatorname{deg-pm~(lpp~g)~by~(rule~1)~}$
assume $t \in$ keys $g$
hence deg-pm $t=$ deg-pm (lpp g) by (rule 1)
with 2 show $\operatorname{deg-pm~} s=\operatorname{deg}-p m t$ by $\operatorname{simp}$
qed
lemma lp-dehomogenize:
assumes is-hom-ord $x$ and homogeneous $p$

```
```

    shows lpp (dehomogenize x p) = except (lpp p) {x}
    proof (cases p=0)
case True
thus ?thesis by simp
next
case False
hence lpp p\in keys p by (rule punit.lt-in-keys)
with assms(2) have except (lpp p) {x}\in keys(dehomogenize x p)
by (rule keys-dehomogenizeI)
thus ?thesis
proof (rule punit.lt-eqI-keys)
fix }
assume t\in keys (dehomogenize x p)
then obtain s where s\inkeys p and t:t=except s{x} by (rule keys-dehomogenizeE)
from this(1) have s\preceqlpp p by (rule punit.lt-max-keys)
moreover from assms(2) <s\in keys p〉\langlelpp p \in keys p> have deg-pm s=
deg-pm (lpp p)
by (rule homogeneousD)
ultimately show t}\preceq except (lpp p) {x} using assms(1) by (simp add:
is-hom-ordD)
qed
qed
lemma isGB-dehomogenize:
assumes is-hom-ord x and finite X and G\subseteqP[X] and punit.is-Groebner-basis
G
and }\bigwedgeg.g\inG\Longrightarrow homogeneous g
shows punit.is-Groebner-basis (dehomogenize x'G)
using dickson-grading-varnum
proof (rule punit.isGB-I-adds-lt[simplified])
from assms(2) show finite ( }X-{x}\mathrm{ ) by simp
next
have dehomogenize x ' G\subseteqP[X - {x}]
proof
fix g
assume g\indehomogenize x' }
then obtain g}\mp@subsup{g}{}{\prime}\mathrm{ where }\mp@subsup{g}{}{\prime}\inG\mathrm{ and }g:g=\mathrm{ dehomogenize x }\mp@subsup{g}{}{\prime}.
from this(1) assms(3) have g' }\inP[X].
hence indets }\mp@subsup{g}{}{\prime}\subseteqX by (rule PolysD
have indets g\subseteqindets g' - {x} by (simp only:g indets-dehomogenize)
also from <indets g}\mp@subsup{g}{}{\prime}\subseteqX\rangle\mathrm{ subset-refl have ...`X - {x} by (rule Diff-mono)
finally show g}\inP[X-{x}] by (rule PolysI-alt
qed
thus dehomogenize x' G\subseteq punit.dgrad-p-set (varnum (X - {x})) 0
by (simp only: dgrad-p-set-varnum)
next
fix p
assume p\in ideal (dehomogenize x' G)
then obtain G0 q where G0\subseteq dehomogenize x' G and finite G0 and p:p=

```
\(\left(\sum g \in G 0 . q g * g\right)\)
by（rule ideal．spanE）
from this（1）obtain \(G^{\prime}\) where \(G^{\prime} \subseteq G\) and \(G 0: G 0=\) dehomogenize \(x^{\prime} G^{\prime}\)
and inj：inj－on（dehomogenize x）\(G^{\prime}\) by（rule subset－imageE－inj）
define \(p^{\prime}\) where \(p^{\prime}=\left(\sum g \in G^{\prime} . q\right.\)（dehomogenize \(\left.\left.x g\right) * g\right)\)
have \(p^{\prime} \in\) ideal \(G^{\prime}\) unfolding \(p^{\prime}\)－def by（rule ideal．sum－in－spanI）
also from \(\left\langle G^{\prime} \subseteq G\right\rangle\) have \(\ldots \subseteq\) ideal \(G\) by（rule ideal．span－mono）
finally have \(p^{\prime} \in\) ideal \(G\) ．
with \(\operatorname{assms}(5)\) have homogenize \(x p^{\prime} \in\) ideal \(G\)（is ？p \(\in-\) ）by（rule homoge－ neous－ideal－homogenize）
```

assume $p \in$ punit.dgrad-p-set (varnum $(X-\{x\})) 0$
hence $p \in P[X-\{x\}]$ by (simp only: dgrad- $p$-set-varnum)
hence indets $p \subseteq X-\{x\}$ by (rule PolysD)
hence $x \notin$ indets $p$ by blast
have $p=$ dehomogenize $x p$ by (rule sym) (simp add: $\langle x \notin$ indets $p\rangle$ )
also from inj have $\ldots=$ dehomogenize $x\left(\sum g \in G^{\prime} . q\right.$ (dehomogenize $\left.x g\right) *$
dehomogenize $x g$ )
by (simp add: $p$ G0 sum.reindex)
also have $\ldots=$ dehomogenize $x$ ? $p$
by (simp add: dehomogenize-sum dehomogenize-times $p^{\prime}$-def)
finally have $p: p=$ dehomogenize $x$ ? $p$.
moreover assume $p \neq 0$
ultimately have $? p \neq 0$ by (auto simp del: dehomogenize-homogenize)
with $\operatorname{assms}(4)\langle ? p \in$ ideal $G\rangle$ obtain $g$ where $g \in G$ and $g \neq 0$ and $a d d s: l p p$
$g$ adds lpp ?p
by (rule punit.GB-adds-lt[simplified])
from this(1) have homogeneous $g$ by (rule assms(5))
show $\exists g \in$ dehomogenize $x$ ' $G . g \neq 0 \wedge$ lpp g adds lpp $p$
proof (intro bexI conjI notI)
assume dehomogenize $x g=0$
hence $g=0$ using 〈homogeneous $g$ 〉 by (rule dehomogenize-zeroD)
with $\langle g \neq 0\rangle$ show False ..
next
from assms(1)〈homogeneous $g\rangle$ have $\operatorname{lpp}$ (dehomogenize $x g$ ) $=\operatorname{except}(\operatorname{lpp} g)$
$\{x\}$
by (rule lp-dehomogenize)
also from adds have ... adds except (lpp ?p) $\{x\}$
by (simp add: adds-poly-mapping le-fun-def lookup-except)
also from assms(1) homogeneous-homogenize have $\ldots=\operatorname{lpp}$ (dehomogenize $x$
?p)
by (rule lp-dehomogenize[symmetric])
finally show $l p p$ (dehomogenize $x g$ ) adds $l p p p$ by (simp only: $p$ )
next
from $\langle g \in G\rangle$ show dehomogenize $x g \in$ dehomogenize $x$ ' $G$ by (rule imageI)
qed
qed
end

```
```

context extended-ord-pm-powerprod
begin
lemma extended-ord-lp:
assumes None \& indets p
shows restrict-indets-pp (extended-ord.lpp p)=lpp(restrict-indets p)
proof (cases p=0)
case True
thus ?thesis by simp
next
case False
hence extended-ord.lpp p\in keys p by (rule extended-ord.punit.lt-in-keys)
hence restrict-indets-pp (extended-ord.lpp p)\in restrict-indets-pp 'keys p by (rule
imageI)
also from assms have eq: .. = keys (restrict-indets p) by (rule keys-restrict-indets[symmetric])
finally show ?thesis
proof (rule punit.lt-eqI-keys[symmetric])
fix }
assume t\in keys (restrict-indets p)
then obtain s where s\in keys p and t:t= restrict-indets-pp s unfolding
eq[symmetric] ..
from this(1) have extended-ord s(extended-ord.lpp p) by (rule extended-ord.punit.lt-max-keys)
thus t\preceq restrict-indets-pp (extended-ord.lpp p) by (auto simp: t extended-ord-def)
qed
qed
lemma restrict-indets-reduced-GB:
assumes finite X and F\subseteqP[X]
shows punit.is-Groebner-basis (restrict-indets' extended-ord.punit.reduced-GB
(homogenize None ' extend-indets ' F))
(is ?thesis1)
and ideal (restrict-indets ' extended-ord.punit.reduced-GB (homogenize None`
extend-indets ' F)) = ideal F
(is ?thesis2)
and restrict-indets ' extended-ord.punit.reduced-GB (homogenize None' ex-
tend-indets ' }F)\subseteqP[X
(is ?thesis3)
proof -
let ?F = homogenize None ' extend-indets' F
let ?G = extended-ord.punit.reduced-GB ?F
from assms(1) have finite (insert None (Some' X)) by simp
moreover have ?F \subseteqP[insert None (Some'X)]
proof
fix hf
assume hf \in?F
then obtain f}\mathrm{ where f}\inF\mathrm{ and hf:hf =homogenize None (extend-indets f)
by auto
from this(1) assms(2) have f}\inP[X].

```
hence indets \(f \subseteq X\) by（rule Polys \(D\) ）
hence Some＇indets \(f \subseteq\) Some＇\(X\) by（rule image－mono）
with indets－extend－indets \([o f f]\) have indets（extend－indets \(f\) ）\(\subseteq\) Some＇\(X\) by blast
hence insert None（indets（extend－indets \(f)\) ）\(\subseteq\) insert None（Some＇\(X\) ）by blast
with indets－homogenize－subset have indets hf \(\subseteq\) insert None（Some＇\(X\) ）
unfolding hf by（rule subset－trans）
thus \(h f \in P[\) insert None（Some＇\(X\) ）］by（rule PolysI－alt）
qed
ultimately have \(G\)－sub：？\(G \subseteq P[\) insert None（Some＇\(X\) ）］
and ideal－\(G\) ：ideal ？\(G=\) ideal ？\(F\)
and \(G B-G\) ：extended－ord．punit．is－reduced－\(G B\) ？\(G\)
by（rule extended－ord．reduced－GB－Polys，rule extended－ord．reduced－GB－ideal－Polys， rule extended－ord．reduced－GB－is－reduced－GB－Polys）
show ？thesis3
proof
fix \(g\)
assume \(g \in\) restrict－indets＇？\(G\)
then obtain \(g^{\prime}\) where \(g^{\prime} \in ? G\) and \(g: g=\) restrict－indets \(g^{\prime} .\).
from this（1）G－sub have \(g^{\prime} \in P[\) insert None（Some＇\(X\) ）］．．
hence indets \(g^{\prime} \subseteq\) insert None（Some＇\(X\) ）by（rule PolysD）
have indets \(g \subseteq\) the＇（indets \(g^{\prime}-\{\) None \(\}\) ）by（simp only：\(g\) indets－restrict－indets－subset）
also from＜indets \(g^{\prime} \subseteq\) insert None \(\left.\left(S o m e{ }^{\prime} X\right)\right\rangle\) have \(\ldots \subseteq X\) by auto
finally show \(g \in P[X]\) by（rule PolysI－alt）
qed
from dickson－grading－varnum show ？thesis1
proof（rule punit．isGB－I－adds－lt［simplified］）
from 〈？thesis3〉 show restrict－indets＇？\(G \subseteq\) punit．dgrad－\(p\)－set（varnum X） 0
by（simp only：dgrad－p－set－varnum）
next
fix \(p::\left({ }^{\prime} a \Rightarrow_{0}\right.\) nat \() \Rightarrow_{0}{ }^{\prime} b\)
assume \(p \neq 0\)
assume \(p \in\) ideal（restrict－indets＇？\(G\) ）
hence extend－indets \(p \in\) extend－indets＇ideal（restrict－indets＇？G）by（rule imageI）
also have \(\ldots \subseteq\) ideal（extend－indets＇restrict－indets＇？G）by（fact ex－ tend－indets－ideal－subset）
also have \(\ldots=\) ideal（dehomogenize None＇？G）
by（simp only：image－comp extend－indets－comp－restrict－indets）
finally have \(p\)－in－ideal：extend－indets \(p \in\) ideal（dehomogenize None＇？G）．
assume \(p \in\) punit．dgrad－p－set（varnum X） 0
hence \(p \in P[X]\) by（simp only：dgrad－\(p\)－set－varnum）
have extended－ord．punit．is－Groebner－basis（dehomogenize None＇？G）
using extended－ord－is－hom－ord 〈finite（insert None（Some＇\(X\) ））〉 \(G\)－sub
proof（rule extended－ord．isGB－dehomogenize）
from \(G B-G\) show extended－ord．punit．is－Groebner－basis ？G
```

    by (rule extended-ord.punit.reduced-GB-D1)
    next
        fix g
        assume g}\in
    with - GB-G ideal-G show homogeneous g
    proof (rule extended-ord.is-reduced-GB-homogeneous)
        fix hf
        assume hf \in?F
        then obtain f}\mathrm{ where hf =homogenize None f ..
        thus homogeneous hf by (simp only: homogeneous-homogenize)
    qed
    qed
    moreover note p-in-ideal
    moreover from }\langlep\not=0\rangle have extend-indets p\not=0 by sim
    ultimately obtain g}\mathrm{ where g-in: g}\in\mathrm{ dehomogenize None '?G and g}=
    and adds: extended-ord.lpp g adds extended-ord.lpp (extend-indets p)
    by (rule extended-ord.punit.GB-adds-lt[simplified])
    have None & indets g
    proof
        assume None \in indets g
        moreover from g-in obtain g0 where g=dehomogenize None g0 ..
        ultimately show False using indets-dehomogenize[of None g0] by blast
    qed
    show \existsg\inrestrict-indets`? ?. g}\not=0\wedgelpp g adds lpp 
    proof (intro bexI conjI notI)
    have lpp (restrict-indets g) = restrict-indets-pp (extended-ord.lpp g)
        by (rule sym, intro extended-ord-lp <None # indets g〉)
    also from adds have ... adds restrict-indets-pp (extended-ord.lpp (extend-indets
    p))
by (simp add: adds-poly-mapping le-fun-def lookup-restrict-indets-pp)
also have ... = lpp (restrict-indets (extend-indets p))
proof (intro extended-ord-lp notI)
assume None \in indets (extend-indets p)
thus False by (simp add: indets-extend-indets)
qed
also have ... = lpp p by simp
finally show lpp (restrict-indets g) adds lpp p.
next
from g-in have restrict-indets g\in restrict-indets 'dehomogenize None '?G
by (rule imageI)
also have ... = restrict-indets'?G by (simp only: image-comp restrict-indets-comp-dehomogenize)
finally show restrict-indets g}\in\mathrm{ restrict-indets'?G .
next
assume restrict-indets g=0
with}<None \not\in indets g> extend-restrict-indets have g=0 by fastforc
with }<g\not=0\rangle\mathrm{ show False ..
qed
qed (fact assms(1))

```
```

    from ideal-G show ?thesis2 by (rule ideal-restrict-indets)
    qed
end

```
end

\section*{References}
[1] W. W. Adams and P. Loustaunau. An Introduction to Gröbner Bases. American Mathematical Society, July 1994.
[2] B. Buchberger. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal (An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal). PhD thesis, Mathematical Institute, University of Innsbruck, Austria, 1965. English translation in Journal of Symbolic Computation 41(3-4):475-511, Special Issue on Logic, Mathematics, and Computer Science: Interactions.
[3] B. Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems (An Algorithmic Criterion for the Solvability of an Algebraic System of Equations). Aequationes Mathematicae, pages 374-383, 1970. (English translation in Gröbner Bases and Applications (Proceedings of the International Conference "33 Years of Gröbner Bases", 1998), London Mathematical Society Lecture Note Series 251, Cambridge Univerity Press, 1998, pages 535-545).
[4] B. Buchberger. A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Bases. In E. W. Ng, editor, Symbolic and Algebraic Computation (Proceedings of EUROSAM'79, Marseille, June 26-28), volume 72 of Lecture Notes in Computer Science, pages 3-21. Springer, 1979.
[5] B. Buchberger. Introduction to Gröbner Bases. In B. Buchberger and F. Winkler, editors, Gröbner Bases and Applications, number 251 in London Mathematical Society Lectures Notes Series, pages 3 - 31. Cambridge University Press, 1998.
[6] B. Buchberger. Gröbner Rings in Theorema: A Case Study in Functors and Categories. Technical Report 2003-49, Johannes Kepler University Linz, Spezialforschungsbereich F013, November 2003.
[7] A. Chaieb and M. Wenzel. Context aware Calculation and Deduction: Ring Equalities via Gröbner Bases in Isabelle. In M. Kauers, M. Ker-
ber, R. Miner, and W. Windsteiger, editors, Towards Mechanized Mathematical Assistants (Proceedings of Calculemus'2007, Hagenberg, Austria, June 27-30), volume 4573 of Lecture Notes in Computer Science, pages 27-39. Springer, 2007.
[8] J.-C. Faugère. A New Efficient Algorithm for Computing Gröbner Bases \(\left(F_{4}\right)\). Journal of Pure and Applied Algebra, 139(1):61-88, 1999.
[9] 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 Press, 2002.
[10] J. S. Jorge, V. M. Guilas, and J. L. Freire. Certifying properties of an efficient functional program for computing Gröbner bases. Journal of Symbolic Computation, 44(5):571-582, 2009.
[11] M. Kreuzer and L. Robbiano. Computational Commutative Algebra 1. Springer-Verlag, 2000.
[12] A. Maletzky. Computer-Assisted Exploration of Gröbner Bases Theory in Theorema. PhD thesis, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria, May 2016. To appear.
[13] I. Medina-Bulo, F. Palomo-Lozano, and J.-L. Ruiz-Reina. A verified Common Lisp implementation of Buchberger's algorithm in ACL2. Journal of Symbolic Computation, 45(1):96-123, 2010.
[14] T. Mora. An Introduction to Commutative and Non-Commutative Gröbner Bases. Theoretical Computer Science, 134(1):131-173, 1994.
[15] C. Schwarzweller. Gröbner Bases - Theory Refinement in the Mizar System. In M. Kohlhase, editor, Mathematical Knowledge Management (4th International Conference, Bremen, Germany, July 15-17), volume 3863 of Lecture Notes in Artificial Intelligence, pages 299-314. Springer, 2006.
[16] L. Théry. A Machine-Checked Implementation of Buchberger's Algorithm. Journal of Automated Reasoning, 26(2):107-137, 2001.
[17] F. Winkler and B. Buchberger. A Criterion for Eliminating Unnecessary Reductions in the Knuth-Bendix Algorithm. In J. Demetrovics, G. Katona, and A. Salomaa, editors, Proceedings of Algebra and Logic in Computer Science, Györ, Hungary, volume 42 of Colloquia Mathematica Societatis Janos Bolyai, pages 849-869. North Holland, 1983.```


[^0]:    *Supported by the Austrian Science Fund (FWF): grant no. W1214-N15 (project DK1) and grant no. P 29498-N31

