

Gröbner Bases, Macaulay Matrices and Dubé's Degree Bounds

Alexander Maletzky*

September 13, 2023

Abstract

This entry formalizes the connection between Gröbner bases and Macaulay matrices (sometimes also referred to as ‘generalized Sylvester matrices’). In particular, it contains a method for computing Gröbner bases, which proceeds by first constructing some Macaulay matrix of the initial set of polynomials, then row-reducing this matrix, and finally converting the result back into a set of polynomials. The output is shown to be a Gröbner basis if the Macaulay matrix constructed in the first step is sufficiently large. In order to obtain concrete upper bounds on the size of the matrix (and hence turn the method into an effectively executable algorithm), Dubé’s degree bounds on Gröbner bases are utilized; consequently, they are also part of the formalization.

Contents

1	Introduction	4
1.1	Future Work	4
2	Degree Sections of Power-Products	4
3	Utility Definitions and Lemmas about Degree Bounds for Gröbner Bases	7
4	Computing Gröbner Bases by Triangularizing Macaulay Matrices	8
4.1	Gröbner Bases	9
4.2	Bounds	9
5	Integer Binomial Coefficients	10
5.1	Sums	12
5.2	Inequalities	12
5.3	Backward Difference Operator	13

*Funded by the Austrian Science Fund (FWF): grant no. P 29498-N31

6	Integer Polynomial Functions	14
6.1	Definition and Basic Properties	14
6.2	Closure Properties	15
7	Monomial Modules	16
7.1	Sets of Monomials	17
7.2	Modules	17
7.3	Reduction	17
7.4	Gröbner Bases	18
8	Preliminaries	19
8.1	Sets	20
8.2	Sums	20
8.3	<i>count-list</i>	20
8.4	<i>listset</i>	20
9	Direct Decompositions and Hilbert Functions	22
9.1	Direct Decompositions	22
9.2	Direct Decompositions and Vector Spaces	25
9.3	Homogeneous Sets of Polynomials with Fixed Degree	26
9.4	Interpreting Polynomial Rings as Vector Spaces over the Coefficient Field	28
9.5	(Projective) Hilbert Function	28
10	Cone Decompositions	29
10.1	More Properties of Reduced Gröbner Bases	29
10.2	Quotient Ideals	30
10.3	Direct Decompositions of Polynomial Rings	30
10.4	Basic Cone Decompositions	32
10.5	Splitting w.r.t. Ideals	40
10.6	Function <i>split</i>	42
10.7	Splitting Ideals	45
10.8	Exact Cone Decompositions	46
10.9	Functions <i>shift</i> and <i>exact</i>	50
11	Dubé's Degree-Bound for Homogeneous Gröbner Bases	57
11.1	Hilbert Function and Hilbert Polynomial	58
11.2	Dubé's Bound	59
12	Sample Computations of Gröbner Bases via Macaulay Matrices	64
12.1	Combining <i>Groebner-Macaulay.Groebner-Macaulay</i> and <i>Groebner-Macaulay.Dube-Bound</i>	65
12.2	Preparations	65

12.2.1	Connection between $(x \Rightarrow_0 a) \Rightarrow_0 b$ and $(x, a) pp \Rightarrow_0 b$	66
12.2.2	Locale <i>pp-powerprod</i>	68
12.3	Computations	72

1 Introduction

The formalization consists of two main parts:

- The connection between Gröbner bases and Macaulay matrices (or ‘generalized Sylvester matrices’), due to Wiesinger-Widi [4]. In particular, this includes a method for computing Gröbner bases via Macaulay matrices.
- Dubé’s upper bounds on the degrees of Gröbner bases [1]. These bounds are not only of theoretical interest, but are also necessary to turn the above-mentioned method for computing Gröbner bases into an actual algorithm.

For more information about this formalization, see the accompanying papers [2] (Dubé’s bound) and [3] (Macaulay matrices).

1.1 Future Work

This formalization could be extended by formalizing improved degree bounds for special input. For instance, Wiesinger-Widi in [4] obtains much smaller bounds if the initial set of polynomials only consists of two binomials.

2 Degree Sections of Power-Products

theory *Degree-Section*

imports *Polynomials.MPoly-PM*

begin

definition *deg-sect* :: 'x set \Rightarrow nat \Rightarrow ('x::countable \Rightarrow_0 nat) set
where *deg-sect* X d = .[X] \cap {t. *deg-pm* t = d}

definition *deg-le-sect* :: 'x set \Rightarrow nat \Rightarrow ('x::countable \Rightarrow_0 nat) set
where *deg-le-sect* X d = (\bigcup d0 \leq d. *deg-sect* X d0)

lemma *deg-sectI*: t \in .[X] \Longrightarrow *deg-pm* t = d \Longrightarrow t \in *deg-sect* X d
{*proof*}

lemma *deg-sectD*:

assumes t \in *deg-sect* X d

shows t \in .[X] **and** *deg-pm* t = d

{*proof*}

lemma *deg-le-sect-alt*: *deg-le-sect* X d = .[X] \cap {t. *deg-pm* t \leq d}
{*proof*}

lemma *deg-le-sectI*: t \in .[X] \Longrightarrow *deg-pm* t \leq d \Longrightarrow t \in *deg-le-sect* X d

$\langle \text{proof} \rangle$

lemma *deg-le-sectD*:

assumes $t \in \text{deg-le-sect } X \ d$

shows $t \in \cdot[X]$ **and** $\text{deg-pm } t \leq d$

$\langle \text{proof} \rangle$

lemma *deg-sect-zero [simp]*: $\text{deg-sect } X \ 0 = \{0\}$

$\langle \text{proof} \rangle$

lemma *deg-sect-empty*: $\text{deg-sect } \{\} \ d = (\text{if } d = 0 \text{ then } \{0\} \text{ else } \{\})$

$\langle \text{proof} \rangle$

lemma *deg-sect-singleton [simp]*: $\text{deg-sect } \{x\} \ d = \{\text{Poly-Mapping.single } x \ d\}$

$\langle \text{proof} \rangle$

lemma *deg-le-sect-zero [simp]*: $\text{deg-le-sect } X \ 0 = \{0\}$

$\langle \text{proof} \rangle$

lemma *deg-le-sect-empty [simp]*: $\text{deg-le-sect } \{\} \ d = \{0\}$

$\langle \text{proof} \rangle$

lemma *deg-le-sect-singleton*: $\text{deg-le-sect } \{x\} \ d = \text{Poly-Mapping.single } x \ \{\dots d\}$

$\langle \text{proof} \rangle$

lemma *deg-sect-mono*: $X \subseteq Y \implies \text{deg-sect } X \ d \subseteq \text{deg-sect } Y \ d$

$\langle \text{proof} \rangle$

lemma *deg-le-sect-mono-1*: $X \subseteq Y \implies \text{deg-le-sect } X \ d \subseteq \text{deg-le-sect } Y \ d$

$\langle \text{proof} \rangle$

lemma *deg-le-sect-mono-2*: $d1 \leq d2 \implies \text{deg-le-sect } X \ d1 \subseteq \text{deg-le-sect } X \ d2$

$\langle \text{proof} \rangle$

lemma *zero-in-deg-le-sect*: $0 \in \text{deg-le-sect } n \ d$

$\langle \text{proof} \rangle$

lemma *deg-sect-disjoint*: $d1 \neq d2 \implies \text{deg-sect } X \ d1 \cap \text{deg-sect } Y \ d2 = \{\}$

$\langle \text{proof} \rangle$

lemma *deg-le-sect-deg-sect-disjoint*: $d1 < d2 \implies \text{deg-le-sect } Y \ d1 \cap \text{deg-sect } X \ d2$

$= \{\}$

$\langle \text{proof} \rangle$

lemma *deg-sect-Suc*:

$\text{deg-sect } X \ (\text{Suc } d) = (\bigcup x \in X. (+) (\text{Poly-Mapping.single } x \ 1) \ \{\dots d\}) \ (\text{is } ?A = ?B)$

$\langle \text{proof} \rangle$

lemma *deg-sect-insert*:

$deg-sect (insert\ x\ X)\ d = (\bigcup d0 \leq d. (+) (Poly-Mapping.single\ x\ (d - d0))) \text{ ‘}$
 $deg-sect\ X\ d0)$
(is ?A = ?B)
<proof>

lemma *deg-le-sect-Suc*: $deg-le-sect\ X\ (Suc\ d) = deg-le-sect\ X\ d \cup deg-sect\ X\ (Suc\ d)$
<proof>

lemma *deg-le-sect-Suc-2*:

$deg-le-sect\ X\ (Suc\ d) = insert\ 0\ (\bigcup x \in X. (+) (Poly-Mapping.single\ x\ 1)) \text{ ‘}$
 $deg-le-sect\ X\ d)$
(is ?A = ?B)
<proof>

lemma *finite-deg-sect*:

assumes *finite X*
shows *finite ((deg-sect X d)::('x::countable \Rightarrow nat) set)*
<proof>

corollary *finite-deg-le-sect*: $finite\ X \implies finite\ ((deg-le-sect\ X\ d)::('x::countable \Rightarrow nat)\ set)$
<proof>

lemma *keys-subset-deg-le-sectI*:

assumes $p \in P[X]$ and *poly-deg p \leq d*
shows $keys\ p \subseteq deg-le-sect\ X\ d$
<proof>

lemma *binomial-symmetric-plus*: $(n + k)\ choose\ n = (n + k)\ choose\ k$
<proof>

lemma *card-deg-sect*:

assumes *finite X* and $X \neq \{\}$
shows $card\ (deg-sect\ X\ d) = (d + (card\ X - 1))\ choose\ (card\ X - 1)$
<proof>

corollary *card-deg-sect-Suc*:

assumes *finite X*
shows $card\ (deg-sect\ X\ (Suc\ d)) = (d + card\ X)\ choose\ (Suc\ d)$
<proof>

corollary *card-deg-le-sect*:

assumes *finite X*
shows $card\ (deg-le-sect\ X\ d) = (d + card\ X)\ choose\ card\ X$
<proof>

end

3 Utility Definitions and Lemmas about Degree Bounds for Gröbner Bases

theory *Degree-Bound-Utils*

imports *Groebner-Bases.Groebner-PM*

begin

context *pm-powerprod*

begin

definition *is-GB-cofactor-bound* :: $((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b::\text{field}) \text{ set} \Rightarrow \text{nat} \Rightarrow \text{bool}$
where *is-GB-cofactor-bound* $F b \longleftrightarrow$
 $(\exists G. \text{punit.is-Groebner-basis } G \wedge \text{ideal } G = \text{ideal } F \wedge (\bigcup g:G. \text{indets } g) \subseteq$
 $(\bigcup f:F. \text{indets } f) \wedge$
 $(\forall g \in G. \exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. q f * f) \wedge (\forall f \in F'. \text{poly-deg}$
 $(q f * f) \leq b))$

definition *is-hom-GB-bound* :: $((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b::\text{field}) \text{ set} \Rightarrow \text{nat} \Rightarrow \text{bool}$
where *is-hom-GB-bound* $F b \longleftrightarrow ((\forall f \in F. \text{homogeneous } f) \longrightarrow (\forall g \in \text{punit.reduced-GB}$
 $F. \text{poly-deg } g \leq b))$

lemma *is-GB-cofactor-boundI*:

assumes *punit.is-Groebner-basis* G **and** *ideal* $G = \text{ideal } F$ **and** $\bigcup (\text{indets } 'G)$
 $\subseteq \bigcup (\text{indets } 'F)$
and $\bigwedge g. g \in G \implies \exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. q f * f) \wedge$
 $(\forall f \in F'. \text{poly-deg } (q f * f) \leq b)$
shows *is-GB-cofactor-bound* $F b$
 $\langle \text{proof} \rangle$

lemma *is-GB-cofactor-boundE*:

fixes $F :: ((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b::\text{field}) \text{ set}$
assumes *is-GB-cofactor-bound* $F b$
obtains G **where** *punit.is-Groebner-basis* G **and** *ideal* $G = \text{ideal } F$ **and** $\bigcup (\text{indets}$
 $'G) \subseteq \bigcup (\text{indets } 'F)$
and $\bigwedge g. g \in G \implies \exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. q f * f) \wedge$
 $(\forall f. \text{indets } (q f) \subseteq \bigcup (\text{indets } 'F) \wedge \text{poly-deg } (q f * f) \leq b \wedge$
 $(f \notin F' \longrightarrow q f = 0))$
 $\langle \text{proof} \rangle$

lemma *is-GB-cofactor-boundE-Polys*:

fixes $F :: ((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b::\text{field}) \text{ set}$
assumes *is-GB-cofactor-bound* $F b$ **and** $F \subseteq P[X]$
obtains G **where** *punit.is-Groebner-basis* G **and** *ideal* $G = \text{ideal } F$ **and** $G \subseteq$
 $P[X]$
and $\bigwedge g. g \in G \implies \exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. q f * f) \wedge$
 $(\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq b \wedge (f \notin F' \longrightarrow q f$
 $= 0))$
 $\langle \text{proof} \rangle$

lemma *is-GB-cofactor-boundE-finite-Polys*:
fixes $F :: ('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b::\text{field}$ *set*
assumes *is-GB-cofactor-bound* F b **and** *finite* F **and** $F \subseteq P[X]$
obtains G **where** *punit.is-Groebner-basis* G **and** *ideal* $G = \text{ideal } F$ **and** $G \subseteq P[X]$
and $\bigwedge g. g \in G \implies \exists q. g = (\sum_{f \in F} q f * f) \wedge (\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq b)$
 $\langle \text{proof} \rangle$

lemma *is-GB-cofactor-boundI-subset-zero*:
assumes $F \subseteq \{0\}$
shows *is-GB-cofactor-bound* F b
 $\langle \text{proof} \rangle$

lemma *is-hom-GB-boundI*:
 $(\bigwedge g. (\bigwedge f. f \in F \implies \text{homogeneous } f) \implies g \in \text{punit.reduced-GB } F \implies \text{poly-deg } g \leq b) \implies \text{is-hom-GB-bound } F$ b
 $\langle \text{proof} \rangle$

lemma *is-hom-GB-boundD*:
 $\text{is-hom-GB-bound } F$ $b \implies (\bigwedge f. f \in F \implies \text{homogeneous } f) \implies g \in \text{punit.reduced-GB } F \implies \text{poly-deg } g \leq b$
 $\langle \text{proof} \rangle$

The following is the main theorem in this theory. It shows that a bound for Gröbner bases of homogenized input sets is always also a cofactor bound for the original input sets.

lemma (**in** *extended-ord-pm-powerprod*) *hom-GB-bound-is-GB-cofactor-bound*:
assumes *finite* X **and** $F \subseteq P[X]$ **and** *extended-ord.is-hom-GB-bound* (*homogenize* F) b
shows *is-GB-cofactor-bound* F b
 $\langle \text{proof} \rangle$

end

end

4 Computing Gröbner Bases by Triangularizing Macaulay Matrices

theory *Groebner-Macaulay*
imports *Groebner-Bases.Macaulay-Matrix* *Groebner-Bases.Groebner-PM Degree-Section* *Degree-Bound-Utils*
begin

Relationship between Gröbner bases and Macaulay matrices, following [4].

4.1 Gröbner Bases

lemma (in *gd-term*) *Macaulay-list-is-GB*:

assumes *is-Groebner-basis* G **and** $\text{pmdl } (\text{set } ps) = \text{pmdl } G$ **and** $G \subseteq \text{phull } (\text{set } ps)$

shows *is-Groebner-basis* $(\text{set } (\text{Macaulay-list } ps))$

<proof>

4.2 Bounds

context *pm-powerprod*

begin

context

fixes $X :: 'x \text{ set}$

assumes *fin-X*: *finite* X

begin

definition *deg-shifts* $:: \text{nat} \Rightarrow (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b) \text{ list} \Rightarrow (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b :: \text{semiring-1}) \text{ list}$

where $\text{deg-shifts } d \text{ fs} = \text{concat } (\text{map } (\lambda f. (\text{map } (\lambda t. \text{punit.monom-mult } 1 \text{ t } f) (\text{punit.pps-to-list } (\text{deg-le-sect } X (d - \text{poly-deg } f)))))) \text{ fs}$

lemma *set-deg-shifts*:

$\text{set } (\text{deg-shifts } d \text{ fs}) = (\bigcup_{f \in \text{set } \text{fs}} (\lambda t. \text{punit.monom-mult } 1 \text{ t } f) \text{ ' } (\text{deg-le-sect } X (d - \text{poly-deg } f)))$

<proof>

corollary *set-deg-shifts-singleton*:

$\text{set } (\text{deg-shifts } d [f]) = (\lambda t. \text{punit.monom-mult } 1 \text{ t } f) \text{ ' } (\text{deg-le-sect } X (d - \text{poly-deg } f))$

<proof>

lemma *deg-shifts-superset*: $\text{set } \text{fs} \subseteq \text{set } (\text{deg-shifts } d \text{ fs})$

<proof>

lemma *deg-shifts-mono*:

assumes $\text{set } \text{fs} \subseteq \text{set } \text{gs}$

shows $\text{set } (\text{deg-shifts } d \text{ fs}) \subseteq \text{set } (\text{deg-shifts } d \text{ gs})$

<proof>

lemma *ideal-deg-shifts [simp]*: $\text{ideal } (\text{set } (\text{deg-shifts } d \text{ fs})) = \text{ideal } (\text{set } \text{fs})$

<proof>

lemma *thm-2-3-6*:

assumes $\text{set } \text{fs} \subseteq P[X]$ **and** *is-GB-cofactor-bound* $(\text{set } \text{fs}) \text{ b}$

shows *punit.is-Groebner-basis* $(\text{set } (\text{punit.Macaulay-list } (\text{deg-shifts } \text{b } \text{fs})))$

<proof>

lemma *thm-2-3-7*:
assumes $set\ fs \subseteq P[X]$ **and** *is-GB-cofactor-bound* ($set\ fs$) b
shows $1 \in ideal\ (set\ fs) \longleftrightarrow 1 \in set\ (punit.Macaulay-list\ (deg-shifts\ b\ fs))$ (**is**
 $?L \longleftrightarrow ?R$)
 $\langle proof \rangle$

end

lemma *thm-2-3-6-indets*:
assumes *is-GB-cofactor-bound* ($set\ fs$) b
shows *punit.is-Groebner-basis* ($set\ (punit.Macaulay-list\ (deg-shifts\ (\bigcup (indets\ ' (set\ fs)))\ b\ fs)))$)
 $\langle proof \rangle$

lemma *thm-2-3-7-indets*:
assumes *is-GB-cofactor-bound* ($set\ fs$) b
shows $1 \in ideal\ (set\ fs) \longleftrightarrow 1 \in set\ (punit.Macaulay-list\ (deg-shifts\ (\bigcup (indets\ ' (set\ fs)))\ b\ fs))$
 $\langle proof \rangle$

end

end

5 Integer Binomial Coefficients

theory *Binomial-Int*
imports *Complex-Main*
begin

lemma *upper-le-binomial*:
assumes $0 < k$ **and** $k < n$
shows $n \leq n\ choose\ k$
 $\langle proof \rangle$

Restore original sort constraints:
 $\langle ML \rangle$

lemma *gbinomial-0-left*: $0\ gchoose\ k = (if\ k = 0\ then\ 1\ else\ 0)$
 $\langle proof \rangle$

lemma *gbinomial-eq-0-int*:
assumes $n < k$
shows $(int\ n)\ gchoose\ k = 0$
 $\langle proof \rangle$

corollary *gbinomial-eq-0*: $0 \leq a \implies a < int\ k \implies a\ gchoose\ k = 0$
 $\langle proof \rangle$

lemma *int-binomial*: $\text{int } (n \text{ choose } k) = (\text{int } n) \text{ gchoose } k$
 ⟨proof⟩

lemma *falling-fact-pochhammer*: $\text{prod } (\lambda i. a - \text{int } i) \{0..<k\} = (-1) \wedge k * \text{pochhammer } (-a) k$
 ⟨proof⟩

lemma *falling-fact-pochhammer'*: $\text{prod } (\lambda i. a - \text{int } i) \{0..<k\} = \text{pochhammer } (a - \text{int } k + 1) k$
 ⟨proof⟩

lemma *gbinomial-int-pochhammer*: $(a::\text{int}) \text{ gchoose } k = (-1) \wedge k * \text{pochhammer } (-a) k \text{ div fact } k$
 ⟨proof⟩

lemma *gbinomial-int-pochhammer'*: $a \text{ gchoose } k = \text{pochhammer } (a - \text{int } k + 1) k \text{ div fact } k$
 ⟨proof⟩

lemma *fact-dvd-pochhammer*: $\text{fact } k \text{ dvd } \text{pochhammer } (a::\text{int}) k$
 ⟨proof⟩

lemma *gbinomial-int-negated-upper*: $(a \text{ gchoose } k) = (-1) \wedge k * ((\text{int } k - a - 1) \text{ gchoose } k)$
 ⟨proof⟩

lemma *gbinomial-int-mult-fact*: $\text{fact } k * (a \text{ gchoose } k) = (\prod i = 0..<k. a - \text{int } i)$
 ⟨proof⟩

corollary *gbinomial-int-mult-fact'*: $(a \text{ gchoose } k) * \text{fact } k = (\prod i = 0..<k. a - \text{int } i)$
 ⟨proof⟩

lemma *gbinomial-int-binomial*:
 $a \text{ gchoose } k = (\text{if } 0 \leq a \text{ then } \text{int } ((\text{nat } a) \text{ choose } k) \text{ else } (-1::\text{int}) \wedge k * \text{int } ((k + (\text{nat } (-a)) - 1) \text{ choose } k))$
 ⟨proof⟩

corollary *gbinomial-nneg*: $0 \leq a \implies a \text{ gchoose } k = \text{int } ((\text{nat } a) \text{ choose } k)$
 ⟨proof⟩

corollary *gbinomial-neg*: $a < 0 \implies a \text{ gchoose } k = (-1::\text{int}) \wedge k * \text{int } ((k + (\text{nat } (-a)) - 1) \text{ choose } k)$
 ⟨proof⟩

lemma *of-int-gbinomial*: $\text{of-int } (a \text{ gchoose } k) = (\text{of-int } a :: 'a::\text{field-char-0}) \text{ gchoose } k$
 ⟨proof⟩

lemma *uminus-one-gbinomial* [simp]: $(- 1 :: \text{int}) \text{ gchoose } k = (- 1) ^ k$
 ⟨proof⟩

lemma *gbinomial-int-Suc-Suc*: $(x + 1 :: \text{int}) \text{ gchoose } (\text{Suc } k) = (x \text{ gchoose } k) + (x \text{ gchoose } (\text{Suc } k))$
 ⟨proof⟩

corollary *plus-Suc-gbinomial*:

$(x + (1 + \text{int } k)) \text{ gchoose } (\text{Suc } k) = ((x + \text{int } k) \text{ gchoose } k) + ((x + \text{int } k) \text{ gchoose } (\text{Suc } k))$
 (is ?l = ?r)
 ⟨proof⟩

lemma *gbinomial-int-n-n* [simp]: $(\text{int } n) \text{ gchoose } n = 1$
 ⟨proof⟩

lemma *gbinomial-int-Suc-n* [simp]: $(1 + \text{int } n) \text{ gchoose } n = 1 + \text{int } n$
 ⟨proof⟩

lemma *zbinomial-eq-0-iff* [simp]: $a \text{ gchoose } k = 0 \iff (0 \leq a \wedge a < \text{int } k)$
 ⟨proof⟩

5.1 Sums

lemma *gchoose-rising-sum-nat*: $(\sum_{j \leq n} \text{int } j + \text{int } k \text{ gchoose } k) = (\text{int } n + \text{int } k + 1) \text{ gchoose } (\text{Suc } k)$
 ⟨proof⟩

lemma *gchoose-rising-sum*:

assumes $0 \leq n$ — Necessary condition.

shows $(\sum_{j=0..n} j + \text{int } k \text{ gchoose } k) = (n + \text{int } k + 1) \text{ gchoose } (\text{Suc } k)$
 ⟨proof⟩

5.2 Inequalities

lemma *binomial-mono*:

assumes $m \leq n$

shows $m \text{ choose } k \leq n \text{ choose } k$

⟨proof⟩

lemma *binomial-plus-le*:

assumes $0 < k$

shows $(m \text{ choose } k) + (n \text{ choose } k) \leq (m + n) \text{ choose } k$

⟨proof⟩

lemma *binomial-ineq-1*: $2 * ((n + i) \text{ choose } k) \leq n \text{ choose } k + ((n + 2 * i) \text{ choose } k)$

⟨proof⟩

lemma *gbinomial-int-nonneg*:

assumes $0 \leq (x::int)$
shows $0 \leq x \text{ gchoose } k$
 $\langle proof \rangle$

lemma *gbinomial-int-mono*:
assumes $0 \leq x$ **and** $x \leq (y::int)$
shows $x \text{ gchoose } k \leq y \text{ gchoose } k$
 $\langle proof \rangle$

lemma *gbinomial-int-plus-le*:
assumes $0 < k$ **and** $0 \leq x$ **and** $0 \leq (y::int)$
shows $(x \text{ gchoose } k) + (y \text{ gchoose } k) \leq (x + y) \text{ gchoose } k$
 $\langle proof \rangle$

lemma *binomial-int-ineq-1*:
assumes $0 \leq x$ **and** $0 \leq (y::int)$
shows $2 * (x + y \text{ gchoose } k) \leq x \text{ gchoose } k + ((x + 2 * y) \text{ gchoose } k)$
 $\langle proof \rangle$

corollary *binomial-int-ineq-2*:
assumes $0 \leq y$ **and** $y \leq (x::int)$
shows $2 * (x \text{ gchoose } k) \leq x - y \text{ gchoose } k + (x + y \text{ gchoose } k)$
 $\langle proof \rangle$

corollary *binomial-int-ineq-3*:
assumes $0 \leq y$ **and** $y \leq 2 * (x::int)$
shows $2 * (x \text{ gchoose } k) \leq y \text{ gchoose } k + (2 * x - y \text{ gchoose } k)$
 $\langle proof \rangle$

5.3 Backward Difference Operator

definition *bw-diff* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a::\{ab\text{-group-add,one}\}$
where $bw\text{-diff } f x = f x - f (x - 1)$

lemma *bw-diff-const* [*simp*]: $bw\text{-diff } (\lambda\cdot. c) = (\lambda\cdot. 0)$
 $\langle proof \rangle$

lemma *bw-diff-id* [*simp*]: $bw\text{-diff } (\lambda x. x) = (\lambda\cdot. 1)$
 $\langle proof \rangle$

lemma *bw-diff-plus* [*simp*]: $bw\text{-diff } (\lambda x. f x + g x) = (\lambda x. bw\text{-diff } f x + bw\text{-diff } g x)$
 $\langle proof \rangle$

lemma *bw-diff-uminus* [*simp*]: $bw\text{-diff } (\lambda x. - f x) = (\lambda x. - bw\text{-diff } f x)$
 $\langle proof \rangle$

lemma *bw-diff-minus* [*simp*]: $bw\text{-diff } (\lambda x. f x - g x) = (\lambda x. bw\text{-diff } f x - bw\text{-diff } g x)$

<proof>

lemma *bw-diff-const-pow*: $(bw\text{-diff} \overset{\sim}{\sim} k) (\lambda\cdot. c) = (if\ k = 0\ then\ \lambda\cdot. c\ else\ (\lambda\cdot. 0))$
<proof>

lemma *bw-diff-id-pow*:
 $(bw\text{-diff} \overset{\sim}{\sim} k) (\lambda x. x) = (if\ k = 0\ then\ (\lambda x. x)\ else\ if\ k = 1\ then\ (\lambda\cdot. 1)\ else\ (\lambda\cdot. 0))$
<proof>

lemma *bw-diff-plus-pow* [*simp*]:
 $(bw\text{-diff} \overset{\sim}{\sim} k) (\lambda x. f\ x + g\ x) = (\lambda x. (bw\text{-diff} \overset{\sim}{\sim} k) f\ x + (bw\text{-diff} \overset{\sim}{\sim} k) g\ x)$
<proof>

lemma *bw-diff-uminus-pow* [*simp*]: $(bw\text{-diff} \overset{\sim}{\sim} k) (\lambda x. - f\ x) = (\lambda x. - (bw\text{-diff} \overset{\sim}{\sim} k) f\ x)$
<proof>

lemma *bw-diff-minus-pow* [*simp*]:
 $(bw\text{-diff} \overset{\sim}{\sim} k) (\lambda x. f\ x - g\ x) = (\lambda x. (bw\text{-diff} \overset{\sim}{\sim} k) f\ x - (bw\text{-diff} \overset{\sim}{\sim} k) g\ x)$
<proof>

lemma *bw-diff-sum-pow* [*simp*]:
 $(bw\text{-diff} \overset{\sim}{\sim} k) (\lambda x. (\sum i \in I. f\ i\ x)) = (\lambda x. (\sum i \in I. (bw\text{-diff} \overset{\sim}{\sim} k) (f\ i)\ x))$
<proof>

lemma *bw-diff-gbinomial*:
assumes $0 < k$
shows $bw\text{-diff} (\lambda x::int. (x + n)\ gchoose\ k) = (\lambda x. (x + n - 1)\ gchoose\ (k - 1))$
<proof>

lemma *bw-diff-gbinomial-pow*:
 $(bw\text{-diff} \overset{\sim}{\sim} l) (\lambda x::int. (x + n)\ gchoose\ k) =$
 $(if\ l \leq k\ then\ (\lambda x. (x + n - int\ l)\ gchoose\ (k - l))\ else\ (\lambda\cdot. 0))$
<proof>

end

6 Integer Polynomial Functions

theory *Poly-Fun*
imports *Binomial-Int HOL-Computational-Algebra.Polynomial*
begin

6.1 Definition and Basic Properties

definition *poly-fun* :: $(int \Rightarrow int) \Rightarrow bool$

where $\text{poly-fun } f \iff (\exists p::\text{rat poly. } \forall a. \text{rat-of-int } (f a) = \text{poly } p (\text{rat-of-int } a))$

lemma poly-funI : $(\bigwedge a. \text{rat-of-int } (f a) = \text{poly } p (\text{rat-of-int } a)) \implies \text{poly-fun } f$
 $\langle \text{proof} \rangle$

lemma poly-funE :
assumes $\text{poly-fun } f$
obtains p **where** $\bigwedge a. \text{rat-of-int } (f a) = \text{poly } p (\text{rat-of-int } a)$
 $\langle \text{proof} \rangle$

lemma poly-fun-eqI :
assumes $\text{poly-fun } f$ **and** $\text{poly-fun } g$ **and** $\text{infinite } \{a. f a = g a\}$
shows $f = g$
 $\langle \text{proof} \rangle$

corollary poly-fun-eqI-ge :
assumes $\text{poly-fun } f$ **and** $\text{poly-fun } g$ **and** $\bigwedge a. b \leq a \implies f a = g a$
shows $f = g$
 $\langle \text{proof} \rangle$

corollary poly-fun-eqI-gr :
assumes $\text{poly-fun } f$ **and** $\text{poly-fun } g$ **and** $\bigwedge a. b < a \implies f a = g a$
shows $f = g$
 $\langle \text{proof} \rangle$

6.2 Closure Properties

lemma poly-fun-const [simp]: $\text{poly-fun } (\lambda-. c)$
 $\langle \text{proof} \rangle$

lemma poly-fun-id [simp]: $\text{poly-fun } (\lambda x. x)$ poly-fun id
 $\langle \text{proof} \rangle$

lemma poly-fun-uminus :
assumes $\text{poly-fun } f$
shows $\text{poly-fun } (\lambda x. - f x)$ **and** $\text{poly-fun } (- f)$
 $\langle \text{proof} \rangle$

lemma $\text{poly-fun-uminus-iff}$ [simp]:
 $\text{poly-fun } (\lambda x. - f x) \iff \text{poly-fun } f \text{ poly-fun } (- f) \iff \text{poly-fun } f$
 $\langle \text{proof} \rangle$

lemma poly-fun-plus [simp]:
assumes $\text{poly-fun } f$ **and** $\text{poly-fun } g$
shows $\text{poly-fun } (\lambda x. f x + g x)$
 $\langle \text{proof} \rangle$

lemma poly-fun-minus [simp]:
assumes $\text{poly-fun } f$ **and** $\text{poly-fun } g$

shows *poly-fun* $(\lambda x. f x - g x)$
<proof>

lemma *poly-fun-times* [*simp*]:
assumes *poly-fun* *f* **and** *poly-fun* *g*
shows *poly-fun* $(\lambda x. f x * g x)$
<proof>

lemma *poly-fun-divide*:
assumes *poly-fun* *f* **and** $\bigwedge a. c \text{ dvd } f a$
shows *poly-fun* $(\lambda x. f x \text{ div } c)$
<proof>

lemma *poly-fun-pow* [*simp*]:
assumes *poly-fun* *f*
shows *poly-fun* $(\lambda x. f x ^ k)$
<proof>

lemma *poly-fun-comp*:
assumes *poly-fun* *f* **and** *poly-fun* *g*
shows *poly-fun* $(\lambda x. f (g x))$ **and** *poly-fun* $(f \circ g)$
<proof>

lemma *poly-fun-sum* [*simp*]: $(\bigwedge i. i \in I \implies \text{poly-fun } (f i)) \implies \text{poly-fun } (\lambda x. (\sum_{i \in I}. f i x))$
<proof>

lemma *poly-fun-prod* [*simp*]: $(\bigwedge i. i \in I \implies \text{poly-fun } (f i)) \implies \text{poly-fun } (\lambda x. (\prod_{i \in I}. f i x))$
<proof>

lemma *poly-fun-pochhammer* [*simp*]: *poly-fun* *f* $\implies \text{poly-fun } (\lambda x. \text{pochhammer } (f x) k)$
<proof>

lemma *poly-fun-gbinomial* [*simp*]: *poly-fun* *f* $\implies \text{poly-fun } (\lambda x. f x \text{ gchoose } k)$
<proof>

end

7 Monomial Modules

theory *Monomial-Module*
imports *Groebner-Bases.Reduced-GB*
begin

Properties of modules generated by sets of monomials, and (reduced) Gröbner bases thereof.

7.1 Sets of Monomials

definition *is-monomial-set* :: ('a \Rightarrow_0 'b::zero) set \Rightarrow bool
where *is-monomial-set* A $\longleftrightarrow (\forall p \in A. \text{is-monomial } p)$

lemma *is-monomial-setI*: $(\bigwedge p. p \in A \Rightarrow \text{is-monomial } p) \Rightarrow \text{is-monomial-set } A$
{proof}

lemma *is-monomial-setD*: *is-monomial-set* A $\Rightarrow p \in A \Rightarrow \text{is-monomial } p$
{proof}

lemma *is-monomial-set-subset*: *is-monomial-set* B $\Rightarrow A \subseteq B \Rightarrow \text{is-monomial-set } A$
{proof}

lemma *is-monomial-set-Un*: *is-monomial-set* (A \cup B) $\longleftrightarrow (\text{is-monomial-set } A \wedge \text{is-monomial-set } B)$
{proof}

7.2 Modules

context *term-powerprod*
begin

lemma *monomial-pmdl*:
assumes *is-monomial-set* B and $p \in \text{pmdl } B$
shows *monomial* (lookup p v) $v \in \text{pmdl } B$
{proof}

lemma *monomial-pmdl-field*:
assumes *is-monomial-set* B and $p \in \text{pmdl } B$ and $v \in \text{keys } (p::\Rightarrow_0 \text{'b::field})$
shows *monomial* c $v \in \text{pmdl } B$
{proof}

end

context *ordered-term*
begin

lemma *keys-monomial-pmdl*:
assumes *is-monomial-set* F and $p \in \text{pmdl } F$ and $t \in \text{keys } p$
obtains f where $f \in F$ and $f \neq 0$ and $lt\ f\ \text{adds}_t\ t$
{proof}

lemma *image-lt-monomial-lt*: $lt\ \text{'monomial } (1::\text{'b::zero-neq-one})\ \text{'lt}\ \text{'F} = lt\ \text{'F}$
{proof}

7.3 Reduction

lemma *red-setE2*:

assumes $\text{red } B \ p \ q$
obtains $b \text{ where } b \in B \text{ and } b \neq 0 \text{ and } \text{red } \{b\} \ p \ q$
 $\langle \text{proof} \rangle$

lemma *red-monomial-keys*:
assumes $\text{is-monomial } r \text{ and } \text{red } \{r\} \ p \ q$
shows $\text{card } (\text{keys } p) = \text{Suc } (\text{card } (\text{keys } q))$
 $\langle \text{proof} \rangle$

lemma *red-monomial-monomial-setD*:
assumes $\text{is-monomial } p \text{ and } \text{is-monomial-set } B \text{ and } \text{red } B \ p \ q$
shows $q = 0$
 $\langle \text{proof} \rangle$

corollary *is-red-monomial-monomial-setD*:
assumes $\text{is-monomial } p \text{ and } \text{is-monomial-set } B \text{ and } \text{is-red } B \ p$
shows $\text{red } B \ p \ 0$
 $\langle \text{proof} \rangle$

corollary *is-red-monomial-monomial-set-in-pmdl*:
 $\text{is-monomial } p \implies \text{is-monomial-set } B \implies \text{is-red } B \ p \implies p \in \text{pmdl } B$
 $\langle \text{proof} \rangle$

corollary *red-rtrancl-monomial-monomial-set-cases*:
assumes $\text{is-monomial } p \text{ and } \text{is-monomial-set } B \text{ and } (\text{red } B)^{**} \ p \ q$
obtains $q = p \mid q = 0$
 $\langle \text{proof} \rangle$

lemma *is-red-monomial-lt*:
assumes $0 \notin B$
shows $\text{is-red } (\text{monomial } (1::'b::\text{field}) \ ' \ \text{lt} \ ' \ B) = \text{is-red } B$
 $\langle \text{proof} \rangle$

end

7.4 Gröbner Bases

context *gd-term*
begin

lemma *monomial-set-is-GB*:
assumes $\text{is-monomial-set } G$
shows $\text{is-Groebner-basis } G$
 $\langle \text{proof} \rangle$

context
fixes d
assumes $d\text{grad}: \text{dickson-grading } (d::'a \Rightarrow \text{nat})$
begin

```

context
  fixes  $F m$ 
  assumes fin-comps: finite (component-of-term ‘ Keys F)
    and F-sub:  $F \subseteq \text{dgrad-p-set } d m$ 
    and F-monom: is-monomial-set (F::(-  $\Rightarrow_0$  'b::field) set)
begin

```

The proof of the following lemma could be simplified, analogous to homogeneous ideals.

```

lemma reduced-GB-subset-monic-dgrad-p-set: reduced-GB F  $\subseteq$  monic ‘ F
  <proof>

```

```

corollary reduced-GB-is-monomial-set-dgrad-p-set: is-monomial-set (reduced-GB F)
  <proof>

```

end

```

lemma is-red-reduced-GB-monomial-dgrad-set:
  assumes finite (component-of-term ‘ S) and pp-of-term ‘ S  $\subseteq$  dgrad-set d m
  shows is-red (reduced-GB (monomial 1 ‘ S)) = is-red (monomial (1::'b::field) ‘ S)
  <proof>

```

```

corollary is-red-reduced-GB-monomial-lt-GB-dgrad-p-set:
  assumes finite (component-of-term ‘ Keys G) and  $G \subseteq \text{dgrad-p-set } d m$  and  $0 \notin G$ 
  shows is-red (reduced-GB (monomial (1::'b::field) ‘ lt ‘ G)) = is-red G
  <proof>

```

```

lemma reduced-GB-monomial-lt-reduced-GB-dgrad-p-set:
  assumes finite (component-of-term ‘ Keys F) and  $F \subseteq \text{dgrad-p-set } d m$ 
  shows reduced-GB (monomial 1 ‘ lt ‘ reduced-GB F) = monomial (1::'b::field) ‘ lt ‘ reduced-GB F
  <proof>

```

end

end

end

8 Preliminaries

```

theory Dube-Prelims
  imports Groebner-Bases.General
begin

```

8.1 Sets

lemma *card-geq-ex-subset*:
assumes $\text{card } A \geq n$
obtains B where $\text{card } B = n$ and $B \subseteq A$
<proof>

lemma *card-2-E-1*:
assumes $\text{card } A = 2$ and $x \in A$
obtains y where $x \neq y$ and $A = \{x, y\}$
<proof>

lemma *card-2-E*:
assumes $\text{card } A = 2$
obtains $x y$ where $x \neq y$ and $A = \{x, y\}$
<proof>

8.2 Sums

lemma *sum-tail-nat*: $0 < b \implies a \leq (b::\text{nat}) \implies \text{sum } f \{a..b\} = f b + \text{sum } f \{a..b - 1\}$
<proof>

lemma *sum-atLeast-Suc-shift*: $0 < b \implies a \leq b \implies \text{sum } f \{\text{Suc } a..b\} = (\sum i=a..b - 1. f (\text{Suc } i))$
<proof>

lemma *sum-split-nat-ivl*:
 $a \leq \text{Suc } j \implies j \leq b \implies \text{sum } f \{a..j\} + \text{sum } f \{\text{Suc } j..b\} = \text{sum } f \{a..b\}$
<proof>

8.3 count-list

lemma *count-list-gr-1-E*:
assumes $1 < \text{count-list } xs \ x$
obtains $i j$ where $i < j$ and $j < \text{length } xs$ and $xs ! i = x$ and $xs ! j = x$
<proof>

8.4 listset

lemma *listset-Cons*: $\text{listset } (x \# xs) = (\bigcup y \in x. (\#) y \text{ ' listset } xs)$
<proof>

lemma *listset-ConsI*: $y \in x \implies ys' \in \text{listset } xs \implies ys = y \# ys' \implies ys \in \text{listset } (x \# xs)$
<proof>

lemma *listset-ConsE*:
assumes $ys \in \text{listset } (x \# xs)$
obtains $y ys'$ where $y \in x$ and $ys' \in \text{listset } xs$ and $ys = y \# ys'$

<proof>

lemma *listsetI*:

length ys = length xs \implies ($\bigwedge i. i < \text{length } xs \implies ys ! i \in xs ! i$) $\implies ys \in \text{listset } xs$

<proof>

lemma *listsetD*:

assumes *ys \in listset xs*

shows *length ys = length xs and $\bigwedge i. i < \text{length } xs \implies ys ! i \in xs ! i$*

<proof>

lemma *listset-singletonI*: *a \in A $\implies ys = [a] \implies ys \in \text{listset } [A]$*

<proof>

lemma *listset-singletonE*:

assumes *ys \in listset [A]*

obtains *a where a \in A and ys = [a]*

<proof>

lemma *listset-doubletonI*: *a \in A $\implies b \in B \implies ys = [a, b] \implies ys \in \text{listset } [A, B]$*

<proof>

lemma *listset-doubletonE*:

assumes *ys \in listset [A, B]*

obtains *a b where a \in A and b \in B and ys = [a, b]*

<proof>

lemma *listset-appendI*:

ys1 \in listset xs1 \implies ys2 \in listset xs2 \implies ys = ys1 @ ys2 \implies ys \in listset (xs1 @ xs2)

<proof>

lemma *listset-appendE*:

assumes *ys \in listset (xs1 @ xs2)*

obtains *ys1 ys2 where ys1 \in listset xs1 and ys2 \in listset xs2 and ys = ys1 @ ys2*

<proof>

lemma *listset-map-imageI*: *ys' \in listset xs \implies ys = map f ys' \implies ys \in listset (map ((\cdot) f) xs)*

<proof>

lemma *listset-map-imageE*:

assumes *ys \in listset (map ((\cdot) f) xs)*

obtains *ys' where ys' \in listset xs and ys = map f ys'*

<proof>

lemma *listset-permE*:
assumes $ys \in \text{listset } xs$ **and** *bij-betw* $f \{..<\text{length } xs\} \{..<\text{length } xs'\}$
and $\bigwedge i. i < \text{length } xs \implies xs' ! i = xs ! f i$
obtains ys' **where** $ys' \in \text{listset } xs'$ **and** $\text{length } ys' = \text{length } ys$
and $\bigwedge i. i < \text{length } ys \implies ys' ! i = ys ! f i$
<proof>

lemma *listset-closed-map*:
assumes $ys \in \text{listset } xs$ **and** $\bigwedge x y. x \in \text{set } xs \implies y \in x \implies f y \in x$
shows $\text{map } f \text{ } ys \in \text{listset } xs$
<proof>

lemma *listset-closed-map2*:
assumes $ys1 \in \text{listset } xs$ **and** $ys2 \in \text{listset } xs$
and $\bigwedge x y1 y2. x \in \text{set } xs \implies y1 \in x \implies y2 \in x \implies f y1 y2 \in x$
shows $\text{map2 } f \text{ } ys1 \text{ } ys2 \in \text{listset } xs$
<proof>

lemma *listset-empty-iff*: $\text{listset } xs = \{\}$ $\longleftrightarrow \{\} \in \text{set } xs$
<proof>

lemma *listset-mono*:
assumes $\text{length } xs = \text{length } ys$ **and** $\bigwedge i. i < \text{length } ys \implies xs ! i \subseteq ys ! i$
shows $\text{listset } xs \subseteq \text{listset } ys$
<proof>

end

9 Direct Decompositions and Hilbert Functions

theory *Hilbert-Function*
imports
HOL-Combinatorics.Permutations
Dube-Prelims
Degree-Section
begin

9.1 Direct Decompositions

The main reason for defining *direct-decomp* in terms of lists rather than sets is that lemma *direct-decomp-direct-decomp* can be proved easier. At some point one could invest the time to re-define *direct-decomp* in terms of sets (possibly adding a couple of further assumptions to *direct-decomp-direct-decomp*).

definition *direct-decomp* :: $'a \text{ set} \Rightarrow 'a::\text{comm-monoid-add set list} \Rightarrow \text{bool}$
where $\text{direct-decomp } A \text{ } ss \longleftrightarrow \text{bij-betw } \text{sum-list } (\text{listset } ss) \text{ } A$

lemma *direct-decompI*:
 $\text{inj-on } \text{sum-list } (\text{listset } ss) \implies \text{sum-list } ' \text{listset } ss = A \implies \text{direct-decomp } A \text{ } ss$

<proof>

lemma *direct-decompI-alt:*

$(\bigwedge qs. qs \in \text{listset } ss \implies \text{sum-list } qs \in A) \implies (\bigwedge a. a \in A \implies \exists ! qs \in \text{listset } ss. a = \text{sum-list } qs) \implies$
direct-decomp A ss
<proof>

lemma *direct-decompD:*

assumes *direct-decomp A ss*

shows $qs \in \text{listset } ss \implies \text{sum-list } qs \in A$ **and** *inj-on sum-list (listset ss)*

and *sum-list ' listset ss = A*

<proof>

lemma *direct-decompE:*

assumes *direct-decomp A ss* **and** $a \in A$

obtains *qs* **where** $qs \in \text{listset } ss$ **and** $a = \text{sum-list } qs$

<proof>

lemma *direct-decomp-unique:*

direct-decomp A ss $\implies qs \in \text{listset } ss \implies qs' \in \text{listset } ss \implies \text{sum-list } qs = \text{sum-list } qs' \implies$

$qs = qs'$

<proof>

lemma *direct-decomp-singleton:* *direct-decomp A [A]*

<proof>

lemma *mset-bij:*

assumes *bij-betw f {..*length xs*} {..*length ys*}* **and** $\bigwedge i. i < \text{length } xs \implies xs ! i = ys ! f i$

shows $mset xs = mset ys$

<proof>

lemma *direct-decomp-perm:*

assumes *direct-decomp A ss1* **and** $mset ss1 = mset ss2$

shows *direct-decomp A ss2*

<proof>

lemma *direct-decomp-split-map:*

direct-decomp A (map f ss) \implies direct-decomp A (map f (filter P ss) @ map f (filter (~ P) ss))

<proof>

lemmas *direct-decomp-split = direct-decomp-split-map[where f=id, simplified]*

lemma *direct-decomp-direct-decomp:*

assumes *direct-decomp A (s # ss)* **and** *direct-decomp s rs*

shows *direct-decomp* A (ss @ rs) (**is** *direct-decomp* A ? ss)
 ⟨*proof*⟩

lemma *sum-list-map-times*: *sum-list* (*map* (($*$) x) xs) = ($x::'a::\text{semiring-0}$) * *sum-list* xs
 ⟨*proof*⟩

lemma *direct-decomp-image-times*:
assumes *direct-decomp* ($A::'a::\text{semiring-0}$ *set*) ss **and** $\bigwedge a b. x * a = x * b \implies x \neq 0 \implies a = b$
shows *direct-decomp* (($*$) x ' A) (*map* ((') (($*$) x)) ss) (**is** *direct-decomp* ? A ? ss)
 ⟨*proof*⟩

lemma *direct-decomp-appendD*:
assumes *direct-decomp* A ($ss1$ @ $ss2$)
shows $\{\} \notin \text{set } ss2 \implies \text{direct-decomp } (\text{sum-list ' listset } ss1) ss1$ (**is** - \implies ?*thesis1*)
and $\{\} \notin \text{set } ss1 \implies \text{direct-decomp } (\text{sum-list ' listset } ss2) ss2$ (**is** - \implies ?*thesis2*)
and *direct-decomp* A [*sum-list* ' *listset* $ss1$, *sum-list* ' *listset* $ss2$] (**is** *direct-decomp* - ? ss)
 ⟨*proof*⟩

lemma *direct-decomp-Cons-zeroI*:
assumes *direct-decomp* A ss
shows *direct-decomp* A ($\{0\}$ # ss)
 ⟨*proof*⟩

lemma *direct-decomp-Cons-zeroD*:
assumes *direct-decomp* A ($\{0\}$ # ss)
shows *direct-decomp* A ss
 ⟨*proof*⟩

lemma *direct-decomp-Cons-subsetI*:
assumes *direct-decomp* A (s # ss) **and** $\bigwedge s0. s0 \in \text{set } ss \implies 0 \in s0$
shows $s \subseteq A$
 ⟨*proof*⟩

lemma *direct-decomp-Int-zero*:
assumes *direct-decomp* A ss **and** $i < j$ **and** $j < \text{length } ss$ **and** $\bigwedge s. s \in \text{set } ss \implies 0 \in s$
shows $ss ! i \cap ss ! j = \{0\}$
 ⟨*proof*⟩

corollary *direct-decomp-pairwise-zero*:
assumes *direct-decomp* A ss **and** $\bigwedge s. s \in \text{set } ss \implies 0 \in s$
shows *pairwise* ($\lambda s1 s2. s1 \cap s2 = \{0\}$) (*set* ss)
 ⟨*proof*⟩

corollary *direct-decomp-repeated-eq-zero*:

assumes *direct-decomp* A ss **and** $1 < \text{count-list } ss \ X$ **and** $\bigwedge s. s \in \text{set } ss \implies 0 \in s$
shows $X = \{0\}$
 $\langle \text{proof} \rangle$

corollary *direct-decomp-map-Int-zero*:

assumes *direct-decomp* A $(\text{map } f \ ss)$ **and** $s1 \in \text{set } ss$ **and** $s2 \in \text{set } ss$ **and** $s1 \neq s2$
and $\bigwedge s. s \in \text{set } ss \implies 0 \in f \ s$
shows $f \ s1 \cap f \ s2 = \{0\}$
 $\langle \text{proof} \rangle$

9.2 Direct Decompositions and Vector Spaces

definition (*in vector-space*) *is-basis* :: 'b set \Rightarrow 'b set \Rightarrow bool

where *is-basis* $V \ B \longleftrightarrow (B \subseteq V \wedge \text{independent } B \wedge V \subseteq \text{span } B \wedge \text{card } B = \text{dim } V)$

definition (*in vector-space*) *some-basis* :: 'b set \Rightarrow 'b set

where *some-basis* $V = \text{Eps } (\text{local.is-basis } V)$

hide-const (**open**) *real-vector.is-basis* *real-vector.some-basis*

context *vector-space*

begin

lemma *dim-empty* [*simp*]: $\text{dim } \{\} = 0$
 $\langle \text{proof} \rangle$

lemma *dim-zero* [*simp*]: $\text{dim } \{0\} = 0$
 $\langle \text{proof} \rangle$

lemma *independent-UnI*:

assumes *independent* A **and** *independent* B **and** $\text{span } A \cap \text{span } B = \{0\}$

shows *independent* $(A \cup B)$

$\langle \text{proof} \rangle$

lemma *subspace-direct-decomp*:

assumes *direct-decomp* A ss **and** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$

shows *subspace* A

$\langle \text{proof} \rangle$

lemma *is-basis-alt*: $\text{subspace } V \implies \text{is-basis } V \ B \longleftrightarrow (\text{independent } B \wedge \text{span } B = V)$

$\langle \text{proof} \rangle$

lemma *is-basis-finite*: $\text{is-basis } V \ A \implies \text{is-basis } V \ B \implies \text{finite } A \longleftrightarrow \text{finite } B$

$\langle \text{proof} \rangle$

lemma *some-basis-is-basis*: *is-basis* V (*some-basis* V)
<proof>

corollary

shows *some-basis-subset*: *some-basis* $V \subseteq V$
and *independent-some-basis*: *independent* (*some-basis* V)
and *span-some-basis-supset*: $V \subseteq \text{span}$ (*some-basis* V)
and *card-some-basis*: card (*some-basis* V) = dim V
<proof>

lemma *some-basis-not-zero*: $0 \notin$ *some-basis* V
<proof>

lemma *span-some-basis*: *subspace* $V \implies \text{span}$ (*some-basis* V) = V
<proof>

lemma *direct-decomp-some-basis-pairwise-disjnt*:
assumes *direct-decomp* A *ss* **and** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$
shows *pairwise* ($\lambda s1 s2. \text{disjnt}$ (*some-basis* $s1$) (*some-basis* $s2$)) (*set* *ss*)
<proof>

lemma *direct-decomp-span-some-basis*:
assumes *direct-decomp* A *ss* **and** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$
shows span (\bigcup (*some-basis* ' *set* *ss*)) = A
<proof>

lemma *direct-decomp-independent-some-basis*:
assumes *direct-decomp* A *ss* **and** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$
shows *independent* (\bigcup (*some-basis* ' *set* *ss*))
<proof>

corollary *direct-decomp-is-basis*:
assumes *direct-decomp* A *ss* **and** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$
shows *is-basis* A (\bigcup (*some-basis* ' *set* *ss*))
<proof>

lemma *dim-direct-decomp*:
assumes *direct-decomp* A *ss* **and** *finite* B **and** $A \subseteq \text{span } B$ **and** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$
shows $\text{dim } A = (\sum_{s \in \text{set } ss. \text{dim } s}$
<proof>

end

9.3 Homogeneous Sets of Polynomials with Fixed Degree

lemma *homogeneous-set-direct-decomp*:
assumes *direct-decomp* A *ss* **and** $\bigwedge s. s \in \text{set } ss \implies \text{homogeneous-set } s$
shows *homogeneous-set* A

<proof>

definition *hom-deg-set* :: $\text{nat} \Rightarrow ((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \text{ set} \Rightarrow ((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{zero}) \text{ set}$
where *hom-deg-set* $z A = (\lambda a. \text{hom-component } a z) \text{ ` } A$

lemma *hom-deg-setD*:

assumes $p \in \text{hom-deg-set } z A$

shows *homogeneous* p **and** $p \neq 0 \implies \text{poly-deg } p = z$

<proof>

lemma *zero-in-hom-deg-set*:

assumes $0 \in A$

shows $0 \in \text{hom-deg-set } z A$

<proof>

lemma *hom-deg-set-closed-uminus*:

assumes $\bigwedge a. a \in A \implies -a \in A$ **and** $p \in \text{hom-deg-set } z A$

shows $-p \in \text{hom-deg-set } z A$

<proof>

lemma *hom-deg-set-closed-plus*:

assumes $\bigwedge a1 a2. a1 \in A \implies a2 \in A \implies a1 + a2 \in A$

and $p \in \text{hom-deg-set } z A$ **and** $q \in \text{hom-deg-set } z A$

shows $p + q \in \text{hom-deg-set } z A$

<proof>

lemma *hom-deg-set-closed-minus*:

assumes $\bigwedge a1 a2. a1 \in A \implies a2 \in A \implies a1 - a2 \in A$

and $p \in \text{hom-deg-set } z A$ **and** $q \in \text{hom-deg-set } z A$

shows $p - q \in \text{hom-deg-set } z A$

<proof>

lemma *hom-deg-set-closed-scalar*:

assumes $\bigwedge a. a \in A \implies c \cdot a \in A$ **and** $p \in \text{hom-deg-set } z A$

shows $(c::'a::\text{semiring-0}) \cdot p \in \text{hom-deg-set } z A$

<proof>

lemma *hom-deg-set-closed-sum*:

assumes $0 \in A$ **and** $\bigwedge a1 a2. a1 \in A \implies a2 \in A \implies a1 + a2 \in A$

and $\bigwedge i. i \in I \implies f i \in \text{hom-deg-set } z A$

shows $\text{sum } f I \in \text{hom-deg-set } z A$

<proof>

lemma *hom-deg-set-subset*: *homogeneous-set* $A \implies \text{hom-deg-set } z A \subseteq A$

<proof>

lemma *Polys-closed-hom-deg-set*:

assumes $A \subseteq P[X]$

shows $\text{hom-deg-set } z \ A \subseteq P[X]$
 $\langle \text{proof} \rangle$

lemma *hom-deg-set-alt-homogeneous-set:*

assumes *homogeneous-set* A
shows $\text{hom-deg-set } z \ A = \{p \in A. \text{homogeneous } p \wedge (p = 0 \vee \text{poly-deg } p = z)\}$
(is $?A = ?B$)
 $\langle \text{proof} \rangle$

lemma *hom-deg-set-sum-list-listset:*

assumes $A = \text{sum-list } ' \text{ listset } ss$
shows $\text{hom-deg-set } z \ A = \text{sum-list } ' \text{ listset } (\text{map } (\text{hom-deg-set } z) \ ss)$ **(is** $?A = ?B$)
 $\langle \text{proof} \rangle$

lemma *direct-decomp-hom-deg-set:*

assumes *direct-decomp* $A \ ss$ **and** $\bigwedge s. s \in \text{set } ss \implies \text{homogeneous-set } s$
shows $\text{direct-decomp } (\text{hom-deg-set } z \ A) \ (\text{map } (\text{hom-deg-set } z) \ ss)$
 $\langle \text{proof} \rangle$

9.4 Interpreting Polynomial Rings as Vector Spaces over the Coefficient Field

There is no need to set up any further interpretation, since interpretation *phull* is exactly what we need.

lemma *subspace-ideal:* $\text{phull.subspace } (\text{ideal } (F::('b::\text{comm-powerprod} \Rightarrow_0 'a::\text{field}) \text{ set}))$
 $\langle \text{proof} \rangle$

lemma *subspace-Polys:* $\text{phull.subspace } (P[X>::(('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{field}) \text{ set})$
 $\langle \text{proof} \rangle$

lemma *subspace-hom-deg-set:*

assumes $\text{phull.subspace } A$
shows $\text{phull.subspace } (\text{hom-deg-set } z \ A)$ **(is** $\text{phull.subspace } ?A$)
 $\langle \text{proof} \rangle$

lemma *hom-deg-set-Polys-eq-span:*

$\text{hom-deg-set } z \ P[X] = \text{phull.span } (\text{monomial } (1::'a::\text{field}) \ ' \ \text{deg-sect } X \ z)$ **(is** $?A = ?B$)
 $\langle \text{proof} \rangle$

9.5 (Projective) Hilbert Function

interpretation *phull:* *vector-space map-scale*
 $\langle \text{proof} \rangle$

definition *Hilbert-fun* $:: (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{field}) \ \text{set} \Rightarrow \text{nat} \Rightarrow \text{nat}$
where $\text{Hilbert-fun } A \ z = \text{phull.dim } (\text{hom-deg-set } z \ A)$

lemma *Hilbert-fun-empty* [simp]: *Hilbert-fun* {} = 0
 ⟨proof⟩

lemma *Hilbert-fun-zero* [simp]: *Hilbert-fun* {0} = 0
 ⟨proof⟩

lemma *Hilbert-fun-direct-decomp*:
 assumes *finite X* and $A \subseteq P[X]$ and *direct-decomp* ($A::('x::countable \Rightarrow_0 \text{nat})$
 $\Rightarrow_0 'a::\text{field}) \text{ set} ps$
 and $\bigwedge s. s \in \text{set } ps \Rightarrow \text{homogeneous-set } s$ and $\bigwedge s. s \in \text{set } ps \Rightarrow \text{phull.subspace}$
 s
 shows *Hilbert-fun* $A z = (\sum p \in \text{set } ps. \text{Hilbert-fun } p z)$
 ⟨proof⟩

context *pm-powerprod*
begin

lemma *image-lt-hom-deg-set*:
 assumes *homogeneous-set A*
 shows $\text{lpp } \text{' } (\text{hom-deg-set } z A - \{0\}) = \{t \in \text{lpp } \text{' } (A - \{0\}). \text{deg-pm } t = z\}$ (**is**
 $?B = ?A$)
 ⟨proof⟩

lemma *Hilbert-fun-alt*:
 assumes *finite X* and $A \subseteq P[X]$ and *phull.subspace A*
 shows *Hilbert-fun* $A z = \text{card } (\text{lpp } \text{' } (\text{hom-deg-set } z A - \{0\}))$ (**is** $- = \text{card } ?A$)
 ⟨proof⟩

end

end

10 Cone Decompositions

theory *Cone-Decomposition*
imports *Groebner-Bases.Groebner-PM Monomial-Module Hilbert-Function*
begin

10.1 More Properties of Reduced Gröbner Bases

context *pm-powerprod*
begin

lemmas *reduced-GB-subset-monic-Polys =*
punit.reduced-GB-subset-monic-dgrad-p-set[simplified, OF dickson-grading-varnum,
where $m=0$, *simplified dgrad-p-set-varnum]*
lemmas *reduced-GB-is-monomial-set-Polys =*
punit.reduced-GB-is-monomial-set-dgrad-p-set[simplified, OF dickson-grading-varnum,

where $m=0$, *simplified dgrad-p-set-varnum*
lemmas *is-red-reduced-GB-monomial-lt-GB-Polys* =
punit.is-red-reduced-GB-monomial-lt-GB-dgrad-p-set[*simplified, OF dickson-grading-varnum*,
where $m=0$, *simplified dgrad-p-set-varnum*]
lemmas *reduced-GB-monomial-lt-reduced-GB-Polys* =
punit.reduced-GB-monomial-lt-reduced-GB-dgrad-p-set[*simplified, OF dickson-grading-varnum*,
where $m=0$, *simplified dgrad-p-set-varnum*]
end

10.2 Quotient Ideals

definition *quot-set* :: 'a set \Rightarrow 'a \Rightarrow 'a::semigroup-mult set (**infixl** \div 55)
where *quot-set* A x = (*) x \div A

lemma *quot-set-iff*: $a \in A \div x \iff x * a \in A$
<proof>

lemma *quot-setI*: $x * a \in A \implies a \in A \div x$
<proof>

lemma *quot-setD*: $a \in A \div x \implies x * a \in A$
<proof>

lemma *quot-set-quot-set* [*simp*]: $A \div x \div y = A \div x * y$
<proof>

lemma *quot-set-one* [*simp*]: $A \div (1::\text{monoid-mult}) = A$
<proof>

lemma *ideal-quot-set-ideal* [*simp*]: *ideal* (*ideal* B \div x) = (*ideal* B) \div (x::comm-ring)
<proof>

lemma *quot-set-image-times*: *inj* ((*) x) \implies ((*) x \div A) \div x = A
<proof>

10.3 Direct Decompositions of Polynomial Rings

context *pm-powerprod*
begin

definition *normal-form* :: (('x \Rightarrow_0 nat) \Rightarrow_0 'a) set \Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0 'a::field)
 \Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0 'a::field)

where *normal-form* F p = (SOME q. (punit.red (punit.reduced-GB F))** p q \wedge
 \neg punit.is-red (punit.reduced-GB F) q)

Of course, *normal-form* could be defined in a much more general context.

context
fixes X :: 'x set

```

assumes fin-X: finite X
begin

context
  fixes F :: ('x  $\Rightarrow_0$  nat)  $\Rightarrow_0$  'a::field) set
  assumes F-sub:  $F \subseteq P[X]$ 
begin

lemma normal-form:
  shows (punit.red (punit.reduced-GB F))** p (normal-form F p) (is ?thesis1)
  and  $\neg$  punit.is-red (punit.reduced-GB F) (normal-form F p) (is ?thesis2)
  <proof>

lemma normal-form-unique:
  assumes (punit.red (punit.reduced-GB F))** p q and  $\neg$  punit.is-red (punit.reduced-GB F) q
  shows normal-form F p = q
  <proof>

lemma normal-form-id-iff: normal-form F p = p  $\longleftrightarrow$  ( $\neg$  punit.is-red (punit.reduced-GB F) p)
  <proof>

lemma normal-form-normal-form: normal-form F (normal-form F p) = normal-form F p
  <proof>

lemma normal-form-zero: normal-form F 0 = 0
  <proof>

lemma normal-form-map-scale: normal-form F (c  $\cdot$  p) = c  $\cdot$  (normal-form F p)
  <proof>

lemma normal-form-uminus: normal-form F ( $-$  p) =  $-$  normal-form F p
  <proof>

lemma normal-form-plus-normal-form:
  normal-form F (normal-form F p + normal-form F q) = normal-form F p +
  normal-form F q
  <proof>

lemma normal-form-minus-normal-form:
  normal-form F (normal-form F p - normal-form F q) = normal-form F p -
  normal-form F q
  <proof>

lemma normal-form-ideal-Polys: normal-form (ideal F  $\cap$  P[X]) = normal-form F
  <proof>

```

lemma *normal-form-diff-in-ideal*: $p - \text{normal-form } F p \in \text{ideal } F$
 ⟨proof⟩

lemma *normal-form-zero-iff*: $\text{normal-form } F p = 0 \iff p \in \text{ideal } F$
 ⟨proof⟩

lemma *normal-form-eq-iff*: $\text{normal-form } F p = \text{normal-form } F q \iff p - q \in \text{ideal } F$
 ⟨proof⟩

lemma *Polys-closed-normal-form*:
 assumes $p \in P[X]$
 shows $\text{normal-form } F p \in P[X]$
 ⟨proof⟩

lemma *image-normal-form-iff*:
 $p \in \text{normal-form } F \text{ ' } P[X] \iff (p \in P[X] \wedge \neg \text{punit.is-red } (\text{punit.reduced-GB } F) p)$
 ⟨proof⟩

end

lemma *direct-decomp-ideal-insert*:
 fixes F and f
 defines $I \equiv \text{ideal } (\text{insert } f F)$
 defines $L \equiv (\text{ideal } F \div f) \cap P[X]$
 assumes $F \subseteq P[X]$ and $f \in P[X]$
 shows $\text{direct-decomp } (I \cap P[X]) [\text{ideal } F \cap P[X], (*) f \text{ ' normal-form } L \text{ ' } P[X]]$
 (is *direct-decomp - ?ss*)
 ⟨proof⟩

corollary *direct-decomp-ideal-normal-form*:
 assumes $F \subseteq P[X]$
 shows $\text{direct-decomp } P[X] [\text{ideal } F \cap P[X], \text{normal-form } F \text{ ' } P[X]]$
 ⟨proof⟩

end

10.4 Basic Cone Decompositions

definition *cone* :: $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \Rightarrow (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{comm-semiring-0}) \text{ set}$

where $\text{cone } hU = (*) (\text{fst } hU) \text{ ' } P[\text{snd } hU]$

lemma *coneI*: $p = a * h \implies a \in P[U] \implies p \in \text{cone } (h, U)$
 ⟨proof⟩

lemma *coneE*:
 assumes $p \in \text{cone } (h, U)$

obtains *a* **where** $a \in P[U]$ **and** $p = a * h$
<proof>

lemma *cone-empty*: $\text{cone } (h, \{\}) = \text{range } (\lambda c. c \cdot h)$
<proof>

lemma *cone-zero* [*simp*]: $\text{cone } (0, U) = \{0\}$
<proof>

lemma *cone-one* [*simp*]: $\text{cone } (1::-\Rightarrow_0 'a::\text{comm-semiring-1}, U) = P[U]$
<proof>

lemma *zero-in-cone*: $0 \in \text{cone } hU$
<proof>

corollary *empty-not-in-map-cone*: $\{\} \notin \text{set } (\text{map } \text{cone } ps)$
<proof>

lemma *tip-in-cone*: $h \in \text{cone } (h::-\Rightarrow_0 -::\text{comm-semiring-1}, U)$
<proof>

lemma *cone-closed-plus*:
assumes $a \in \text{cone } hU$ **and** $b \in \text{cone } hU$
shows $a + b \in \text{cone } hU$
<proof>

lemma *cone-closed-uminus*:
assumes $(a::-\Rightarrow_0 -::\text{comm-ring}) \in \text{cone } hU$
shows $- a \in \text{cone } hU$
<proof>

lemma *cone-closed-minus*:
assumes $(a::-\Rightarrow_0 -::\text{comm-ring}) \in \text{cone } hU$ **and** $b \in \text{cone } hU$
shows $a - b \in \text{cone } hU$
<proof>

lemma *cone-closed-times*:
assumes $a \in \text{cone } (h, U)$ **and** $q \in P[U]$
shows $q * a \in \text{cone } (h, U)$
<proof>

corollary *cone-closed-monom-mult*:
assumes $a \in \text{cone } (h, U)$ **and** $t \in \cdot[U]$
shows $\text{punit.monom-mult } c t a \in \text{cone } (h, U)$
<proof>

lemma *coneD*:
assumes $p \in \text{cone } (h, U)$ **and** $p \neq 0$
shows $\text{lpp } h \text{ adds } \text{lpp } (p::-\Rightarrow_0 -::\{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\})$

<proof>

lemma *cone-mono-1*:

assumes $h' \in P[U]$

shows $\text{cone}(h' * h, U) \subseteq \text{cone}(h, U)$

<proof>

lemma *cone-mono-2*:

assumes $U1 \subseteq U2$

shows $\text{cone}(h, U1) \subseteq \text{cone}(h, U2)$

<proof>

lemma *cone-subsetD*:

assumes $\text{cone}(h1, U1) \subseteq \text{cone}(h2 :: - \Rightarrow_0 'a :: \{\text{comm-ring-1}, \text{ring-no-zero-divisors}\}, U2)$

shows $h2 \text{ dvd } h1 \text{ and } h1 \neq 0 \implies U1 \subseteq U2$

<proof>

lemma *cone-subset-PolysD*:

assumes $\text{cone}(h :: - \Rightarrow_0 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}, U) \subseteq P[X]$

shows $h \in P[X] \text{ and } h \neq 0 \implies U \subseteq X$

<proof>

lemma *cone-subset-PolysI*:

assumes $h \in P[X] \text{ and } h \neq 0 \implies U \subseteq X$

shows $\text{cone}(h, U) \subseteq P[X]$

<proof>

lemma *cone-image-times*: $(*) a \text{ ' cone}(h, U) = \text{cone}(a * h, U)$

<proof>

lemma *cone-image-times'*: $(*) a \text{ ' cone } hU = \text{cone}(\text{apfst } ((*) a) hU)$

<proof>

lemma *homogeneous-set-coneI*:

assumes *homogeneous* h

shows *homogeneous-set* $(\text{cone}(h, U))$

<proof>

lemma *subspace-cone*: $\text{phull.subspace}(\text{cone } hU)$

<proof>

lemma *direct-decomp-cone-insert*:

fixes $h :: - \Rightarrow_0 'a :: \{\text{comm-ring-1}, \text{ring-no-zero-divisors}\}$

assumes $x \notin U$

shows *direct-decomp* $(\text{cone}(h, \text{insert } x U))$

$[\text{cone}(h, U), \text{cone}(\text{monomial } 1 (\text{Poly-Mapping.single } x (\text{Suc } 0)) * h, \text{insert } x U)]$

<proof>

definition *valid-decomp* :: ('x \Rightarrow_0 nat) \Rightarrow_0 'a::zero \times 'x set) list \Rightarrow bool

where *valid-decomp* X ps \longleftrightarrow ((\forall (h, U) \in set ps. h \in P[X] \wedge h \neq 0 \wedge U \subseteq X))

definition *monomial-decomp* :: ((('x \Rightarrow_0 nat) \Rightarrow_0 'a::{one,zero}) \times 'x set) list \Rightarrow bool

where *monomial-decomp* ps \longleftrightarrow (\forall hU \in set ps. is-monomial (fst hU) \wedge punit.lc (fst hU) = 1)

definition *hom-decomp* :: ((('x \Rightarrow_0 nat) \Rightarrow_0 'a::{one,zero}) \times 'x set) list \Rightarrow bool

where *hom-decomp* ps \longleftrightarrow (\forall hU \in set ps. homogeneous (fst hU))

definition *cone-decomp* :: (('x \Rightarrow_0 nat) \Rightarrow_0 'a) set \Rightarrow

((('x \Rightarrow_0 nat) \Rightarrow_0 'a::comm-semiring-0) \times 'x set) list \Rightarrow bool

where *cone-decomp* T ps \longleftrightarrow direct-decomp T (map cone ps)

lemma *valid-decompI*:

(\bigwedge h U. (h, U) \in set ps \implies h \in P[X]) \implies (\bigwedge h U. (h, U) \in set ps \implies h \neq 0)

\implies

(\bigwedge h U. (h, U) \in set ps \implies U \subseteq X) \implies *valid-decomp* X ps

<proof>

lemma *valid-decompD*:

assumes *valid-decomp* X ps **and** (h, U) \in set ps

shows h \in P[X] **and** h \neq 0 **and** U \subseteq X

<proof>

lemma *valid-decompD-finite*:

assumes finite X **and** *valid-decomp* X ps **and** (h, U) \in set ps

shows finite U

<proof>

lemma *valid-decomp-Nil*: *valid-decomp* X []

<proof>

lemma *valid-decomp-concat*:

assumes \bigwedge ps. ps \in set pss \implies *valid-decomp* X ps

shows *valid-decomp* X (concat pss)

<proof>

corollary *valid-decomp-append*:

assumes *valid-decomp* X ps **and** *valid-decomp* X qs

shows *valid-decomp* X (ps @ qs)

<proof>

lemma *valid-decomp-map-times*:

assumes *valid-decomp* X ps **and** s \in P[X] **and** s \neq (0:: \Rightarrow_0 \Rightarrow_0 ::semiring-no-zero-divisors)

shows *valid-decomp* X ($\text{map } (\text{apfst } ((*)) s)$) ps
<proof>

lemma *monomial-decompI*:

$(\bigwedge h U. (h, U) \in \text{set } ps \implies \text{is-monomial } h) \implies (\bigwedge h U. (h, U) \in \text{set } ps \implies \text{punit.lc } h = 1) \implies$
monomial-decomp ps
<proof>

lemma *monomial-decompD*:

assumes *monomial-decomp* ps **and** $(h, U) \in \text{set } ps$
shows *is-monomial* h **and** $\text{punit.lc } h = 1$
<proof>

lemma *monomial-decomp-append-iff*:

monomial-decomp $(ps @ qs) \iff \text{monomial-decomp } ps \wedge \text{monomial-decomp } qs$
<proof>

lemma *monomial-decomp-concat*:

$(\bigwedge ps. ps \in \text{set } pss \implies \text{monomial-decomp } ps) \implies \text{monomial-decomp } (\text{concat } pss)$
<proof>

lemma *monomial-decomp-map-times*:

assumes *monomial-decomp* ps **and** *is-monomial* f **and** $\text{punit.lc } f = (1::'a::\text{semiring-1})$
shows *monomial-decomp* ($\text{map } (\text{apfst } ((*)) f$) ps)
<proof>

lemma *monomial-decomp-monomial-in-cone*:

assumes *monomial-decomp* ps **and** $hU \in \text{set } ps$ **and** $a \in \text{cone } hU$
shows *monomial* ($\text{lookup } a t$) $t \in \text{cone } hU$
<proof>

lemma *monomial-decomp-sum-list-monomial-in-cone*:

assumes *monomial-decomp* ps **and** $a \in \text{sum-list ' listset } (\text{map } \text{cone } ps)$ **and** $t \in \text{keys } a$
obtains $c h U$ **where** $(h, U) \in \text{set } ps$ **and** $c \neq 0$ **and** *monomial* $c t \in \text{cone } (h, U)$
<proof>

lemma *hom-decompI*: $(\bigwedge h U. (h, U) \in \text{set } ps \implies \text{homogeneous } h) \implies \text{hom-decomp } ps$
<proof>

lemma *hom-decompD*: $\text{hom-decomp } ps \implies (h, U) \in \text{set } ps \implies \text{homogeneous } h$
<proof>

lemma *hom-decomp-append-iff*: $\text{hom-decomp } (ps @ qs) \iff \text{hom-decomp } ps \wedge \text{hom-decomp } qs$
<proof>

lemma *hom-decomp-concat*: $(\bigwedge ps. ps \in set\ pss \implies hom-decomp\ ps) \implies hom-decomp\ (concat\ pss)$
 ⟨proof⟩

lemma *hom-decomp-map-times*:
 assumes *hom-decomp ps* and *homogeneous f*
 shows *hom-decomp (map (apfst ((* f))) ps)*
 ⟨proof⟩

lemma *monomial-decomp-imp-hom-decomp*:
 assumes *monomial-decomp ps*
 shows *hom-decomp ps*
 ⟨proof⟩

lemma *cone-decompI*: *direct-decomp T (map cone ps) \implies cone-decomp T ps*
 ⟨proof⟩

lemma *cone-decompD*: *cone-decomp T ps \implies direct-decomp T (map cone ps)*
 ⟨proof⟩

lemma *cone-decomp-cone-subset*:
 assumes *cone-decomp T ps* and $hU \in set\ ps$
 shows *cone hU \subseteq T*
 ⟨proof⟩

lemma *cone-decomp-indets*:
 assumes *cone-decomp T ps* and $T \subseteq P[X]$ and $(h, U) \in set\ ps$
 shows $h \in P[X]$ and $h \neq (0 :: \Rightarrow_0 :: \{comm-semiring-1, semiring-no-zero-divisors\})$
 $\implies U \subseteq X$
 ⟨proof⟩

lemma *cone-decomp-closed-plus*:
 assumes *cone-decomp T ps* and $a \in T$ and $b \in T$
 shows $a + b \in T$
 ⟨proof⟩

lemma *cone-decomp-closed-uminus*:
 assumes *cone-decomp T ps* and $(a :: \Rightarrow_0 :: comm-ring) \in T$
 shows $- a \in T$
 ⟨proof⟩

corollary *cone-decomp-closed-minus*:
 assumes *cone-decomp T ps* and $(a :: \Rightarrow_0 :: comm-ring) \in T$ and $b \in T$
 shows $a - b \in T$
 ⟨proof⟩

lemma *cone-decomp-Nil*: *cone-decomp {0} []*
 ⟨proof⟩

lemma *cone-decomp-singleton*: *cone-decomp* (*cone* (*t*, *U*)) [(*t*, *U*)]
 ⟨*proof*⟩

lemma *cone-decomp-append*:
assumes *direct-decomp* *T* [*S1*, *S2*] **and** *cone-decomp* *S1* *ps* **and** *cone-decomp* *S2* *qs*
shows *cone-decomp* *T* (*ps* @ *qs*)
 ⟨*proof*⟩

lemma *cone-decomp-concat*:
assumes *direct-decomp* *T* *ss* **and** *length* *pss* = *length* *ss*
and $\bigwedge i. i < \text{length } ss \implies \text{cone-decomp } (ss ! i) (pss ! i)$
shows *cone-decomp* *T* (*concat* *pss*)
 ⟨*proof*⟩

lemma *cone-decomp-map-times*:
assumes *cone-decomp* *T* *ps*
shows *cone-decomp* ((*) *s* ' *T*) (*map* (*apfst* ((*) (*s*::- \Rightarrow_0 -::{*comm-ring-1*, *ring-no-zero-divisors*})))
ps)
 ⟨*proof*⟩

lemma *cone-decomp-perm*:
assumes *cone-decomp* *T* *ps* **and** *mset* *ps* = *mset* *qs*
shows *cone-decomp* *T* *qs*
 ⟨*proof*⟩

lemma *valid-cone-decomp-subset-Polys*:
assumes *valid-decomp* *X* *ps* **and** *cone-decomp* *T* *ps*
shows *T* \subseteq *P*[*X*]
 ⟨*proof*⟩

lemma *homogeneous-set-cone-decomp*:
assumes *cone-decomp* *T* *ps* **and** *hom-decomp* *ps*
shows *homogeneous-set* *T*
 ⟨*proof*⟩

lemma *subspace-cone-decomp*:
assumes *cone-decomp* *T* *ps*
shows *phull.subspace* (*T*::(- \Rightarrow_0 -::*field*) *set*)
 ⟨*proof*⟩

definition *pos-decomp* :: (((*'x* \Rightarrow_0 *nat*) \Rightarrow_0 *'a*) \times *'x set*) *list* \Rightarrow (((*'x* \Rightarrow_0 *nat*) \Rightarrow_0 *'a*) \times *'x set*) *list*
 ((-₊) [1000] 999)
where *pos-decomp* *ps* = *filter* ($\lambda p. \text{snd } p \neq \{\}$) *ps*

definition *standard-decomp* :: *nat* \Rightarrow (((*'x* \Rightarrow_0 *nat*) \Rightarrow_0 *'a*::*zero*) \times *'x set*) *list* \Rightarrow *bool*

where *standard-decomp* k $ps \iff (\forall (h, U) \in \text{set } (ps_+). k \leq \text{poly-deg } h \wedge$
 $(\forall d. k \leq d \implies d \leq \text{poly-deg } h \implies$
 $(\exists (h', U') \in \text{set } ps. \text{poly-deg } h' = d \wedge \text{card } U \leq$
 $\text{card } U'))$

lemma *pos-decomp-Nil* [*simp*]: $[\]_+ = [\]$
 $\langle \text{proof} \rangle$

lemma *pos-decomp-subset*: $\text{set } (ps_+) \subseteq \text{set } ps$
 $\langle \text{proof} \rangle$

lemma *pos-decomp-append*: $(ps \text{ @ } qs)_+ = ps_+ \text{ @ } qs_+$
 $\langle \text{proof} \rangle$

lemma *pos-decomp-concat*: $(\text{concat } pss)_+ = \text{concat } (\text{map } \text{pos-decomp } pss)$
 $\langle \text{proof} \rangle$

lemma *pos-decomp-map*: $(\text{map } (\text{apfst } f) ps)_+ = \text{map } (\text{apfst } f) (ps_+)$
 $\langle \text{proof} \rangle$

lemma *card-Diff-pos-decomp*: $\text{card } \{(h, U) \in \text{set } qs - \text{set } (qs_+). P h\} = \text{card } \{h.$
 $(h, \{\}) \in \text{set } qs \wedge P h\}$
 $\langle \text{proof} \rangle$

lemma *standard-decompI*:
assumes $\bigwedge h U. (h, U) \in \text{set } (ps_+) \implies k \leq \text{poly-deg } h$
and $\bigwedge h U d. (h, U) \in \text{set } (ps_+) \implies k \leq d \implies d \leq \text{poly-deg } h \implies$
 $(\exists h' U'. (h', U') \in \text{set } ps \wedge \text{poly-deg } h' = d \wedge \text{card } U \leq \text{card } U')$
shows *standard-decomp* k ps
 $\langle \text{proof} \rangle$

lemma *standard-decompD*: *standard-decomp* k $ps \implies (h, U) \in \text{set } (ps_+) \implies k \leq$
 $\text{poly-deg } h$
 $\langle \text{proof} \rangle$

lemma *standard-decompE*:
assumes *standard-decomp* k ps **and** $(h, U) \in \text{set } (ps_+)$ **and** $k \leq d$ **and** $d \leq$
 $\text{poly-deg } h$
obtains $h' U'$ **where** $(h', U') \in \text{set } ps$ **and** $\text{poly-deg } h' = d$ **and** $\text{card } U \leq \text{card}$
 U'
 $\langle \text{proof} \rangle$

lemma *standard-decomp-Nil*: $ps_+ = [\] \implies \text{standard-decomp } k ps$
 $\langle \text{proof} \rangle$

lemma *standard-decomp-singleton*: *standard-decomp* $(\text{poly-deg } h) [(h, U)]$
 $\langle \text{proof} \rangle$

lemma *standard-decomp-concat*:

assumes $\bigwedge ps. ps \in \text{set } pss \implies \text{standard-decomp } k \ ps$
shows $\text{standard-decomp } k \ (\text{concat } pss)$
 <proof>

corollary *standard-decomp-append*:
assumes $\text{standard-decomp } k \ ps$ **and** $\text{standard-decomp } k \ qs$
shows $\text{standard-decomp } k \ (ps \ @ \ qs)$
 <proof>

lemma *standard-decomp-map-times*:
assumes $\text{standard-decomp } k \ ps$ **and** $\text{valid-decomp } X \ ps$ **and** $s \neq (0::-\Rightarrow_0 'a::\text{semiring-no-zero-divisors})$
shows $\text{standard-decomp } (k + \text{poly-deg } s) \ (\text{map } (\text{apfst } ((*)) \ s) \ ps)$
 <proof>

lemma *standard-decomp-nonempty-unique*:
assumes $\text{finite } X$ **and** $\text{valid-decomp } X \ ps$ **and** $\text{standard-decomp } k \ ps$ **and** $ps_+ \neq []$
shows $k = \text{Min } (\text{poly-deg } 'fst \ 'set \ (ps_+))$
 <proof>

lemma *standard-decomp-SucE*:
assumes $\text{finite } X$ **and** $U \subseteq X$ **and** $h \in P[X]$ **and** $h \neq (0::-\Rightarrow_0 'a::\{\text{comm-ring-1}, \text{ring-no-zero-divisors}\})$
obtains ps **where** $\text{valid-decomp } X \ ps$ **and** $\text{cone-decomp } (\text{cone } (h, U)) \ ps$
and $\text{standard-decomp } (\text{Suc } (\text{poly-deg } h)) \ ps$
and $\text{is-monomial } h \implies \text{punit.lc } h = 1 \implies \text{monomial-decomp } ps$ **and** $\text{homogeneous } h \implies \text{hom-decomp } ps$
 <proof>

lemma *standard-decomp-geE*:
assumes $\text{finite } X$ **and** $\text{valid-decomp } X \ ps$
and $\text{cone-decomp } (T::('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\{\text{comm-ring-1}, \text{ring-no-zero-divisors}\}) \ ps$
and $\text{standard-decomp } k \ ps$ **and** $k \leq d$
obtains qs **where** $\text{valid-decomp } X \ qs$ **and** $\text{cone-decomp } T \ qs$ **and** $\text{standard-decomp } d \ qs$
and $\text{monomial-decomp } ps \implies \text{monomial-decomp } qs$ **and** $\text{hom-decomp } ps \implies \text{hom-decomp } qs$
 <proof>

10.5 Splitting w.r.t. Ideals

context
fixes $X :: 'x \ \text{set}$
begin

definition *splits-wrt* :: $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \ \text{set}) \ \text{list} \times ((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \ \text{set}) \ \text{list} \Rightarrow ((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{comm-ring-1}) \ \text{set} \Rightarrow (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \ \text{set} \Rightarrow \text{bool})$

where $\text{splits-wrt } pqs \ T \ F \iff \text{cone-decomp } T \ (\text{fst } pqs \ @ \ \text{snd } pqs) \wedge$
 $(\forall hU \in \text{set } (\text{fst } pqs). \text{cone } hU \subseteq \text{ideal } F \cap P[X]) \wedge$
 $(\forall (h, U) \in \text{set } (\text{snd } pqs). \text{cone } (h, U) \subseteq P[X] \wedge \text{cone } (h,$
 $U) \cap \text{ideal } F = \{0\})$

lemma *splits-wrtI*:

assumes $\text{cone-decomp } T \ (ps \ @ \ qs)$
and $\bigwedge h \ U. (h, U) \in \text{set } ps \implies \text{cone } (h, U) \subseteq P[X]$ **and** $\bigwedge h \ U. (h, U) \in \text{set } ps \implies h \in \text{ideal } F$
and $\bigwedge h \ U. (h, U) \in \text{set } qs \implies \text{cone } (h, U) \subseteq P[X]$
and $\bigwedge h \ U \ a. (h, U) \in \text{set } qs \implies a \in \text{cone } (h, U) \implies a \in \text{ideal } F \implies a = 0$
shows $\text{splits-wrt } (ps, qs) \ T \ F$
 $\langle \text{proof} \rangle$

lemma *splits-wrtI-valid-decomp*:

assumes $\text{valid-decomp } X \ ps$ **and** $\text{valid-decomp } X \ qs$ **and** $\text{cone-decomp } T \ (ps \ @ \ qs)$
and $\bigwedge h \ U. (h, U) \in \text{set } ps \implies h \in \text{ideal } F$
and $\bigwedge h \ U \ a. (h, U) \in \text{set } qs \implies a \in \text{cone } (h, U) \implies a \in \text{ideal } F \implies a = 0$
shows $\text{splits-wrt } (ps, qs) \ T \ F$
 $\langle \text{proof} \rangle$

lemma *splits-wrtD*:

assumes $\text{splits-wrt } (ps, qs) \ T \ F$
shows $\text{cone-decomp } T \ (ps \ @ \ qs)$ **and** $hU \in \text{set } ps \implies \text{cone } hU \subseteq \text{ideal } F \cap P[X]$
and $hU \in \text{set } qs \implies \text{cone } hU \subseteq P[X]$ **and** $hU \in \text{set } qs \implies \text{cone } hU \cap \text{ideal } F = \{0\}$
 $\langle \text{proof} \rangle$

lemma *splits-wrt-image-sum-list-fst-subset*:

assumes $\text{splits-wrt } (ps, qs) \ T \ F$
shows $\text{sum-list 'listset } (\text{map } \text{cone } ps) \subseteq \text{ideal } F \cap P[X]$
 $\langle \text{proof} \rangle$

lemma *splits-wrt-image-sum-list-snd-subset*:

assumes $\text{splits-wrt } (ps, qs) \ T \ F$
shows $\text{sum-list 'listset } (\text{map } \text{cone } qs) \subseteq P[X]$
 $\langle \text{proof} \rangle$

lemma *splits-wrt-cone-decomp-1*:

assumes $\text{splits-wrt } (ps, qs) \ T \ F$ **and** $\text{monomial-decomp } qs$ **and** $\text{is-monomial-set } (F::(- \implies_0 'a::\text{field}) \text{ set})$

— The last two assumptions are missing in the paper.

shows $\text{cone-decomp } (T \cap \text{ideal } F) \ ps$
 $\langle \text{proof} \rangle$

Together, Theorems *splits-wrt-image-sum-list-fst-subset* and *splits-wrt-cone-decomp-1* imply that ps is also a cone decomposition of $T \cap \text{ideal } F \cap P[X]$.

lemma *splits-wrt-cone-decomp-2*:

assumes *finite X and splits-wrt (ps, qs) T F and monomial-decomp qs and is-monomial-set F*
and $F \subseteq P[X]$
shows *cone-decomp (T \cap normal-form F ' P[X]) qs*
<proof>

lemma *quot-monomial-ideal-monomial*:

ideal (monomial 1 ' S) \div monomial 1 (Poly-Mapping.single (x::'x) (1::nat)) =
ideal (monomial (1::'a::comm-ring-1) ' ($\lambda s. s - \text{Poly-Mapping.single } x \ 1$) ' S)
<proof>

lemma *lem-4-2-1*:

assumes *ideal F \div monomial 1 t = ideal (monomial (1::'a::comm-ring-1) ' S)*
shows *cone (monomial 1 t, U) \subseteq ideal F \longleftrightarrow $0 \in S$*
<proof>

lemma *lem-4-2-2*:

assumes *ideal F \div monomial 1 t = ideal (monomial (1::'a::comm-ring-1) ' S)*
shows *cone (monomial 1 t, U) \cap ideal F = $\{0\}$ \longleftrightarrow $S \cap .[U] = \{\}$*
<proof>

10.6 Function *split*

definition *max-subset :: 'a set \Rightarrow ('a set \Rightarrow bool) \Rightarrow 'a set*
where *max-subset A P = (ARG-MAX card B. B \subseteq A \wedge P B)*

lemma *max-subset*:

assumes *finite A and B \subseteq A and P B*
shows *max-subset A P \subseteq A (is ?thesis1)*
and *P (max-subset A P) (is ?thesis2)*
and *card B \leq card (max-subset A P) (is ?thesis3)*
<proof>

function (*domintros*) *split :: ('x \Rightarrow_0 nat) \Rightarrow 'x set \Rightarrow ('x \Rightarrow_0 nat) set \Rightarrow*
(((((('x \Rightarrow_0 nat) \Rightarrow_0 'a) \times ('x set)) list) \times
(((((('x \Rightarrow_0 nat) \Rightarrow_0 'a::{zero,one}) \times ('x set)) list)))

where

split t U S =
(if $0 \in S$ then
([(monomial 1 t, U)], []])
else if $S \cap .[U] = \{\}$ then
([], [(monomial 1 t, U)])
else
let x = SOME x'. x' \in U - (max-subset U ($\lambda V. S \cap .[V] = \{\}$));
(ps0, qs0) = split t (U - {x}) S;
(ps1, qs1) = split (Poly-Mapping.single x 1 + t) U (($\lambda f. f -$
Poly-Mapping.single x 1) ' S) in
(ps0 @ ps1, qs0 @ qs1))

<proof>

Function *split* is not executable, because this is not necessary. With some effort, it could be made executable, though.

lemma *split-domI'*:

assumes *finite X and fst (snd args) ⊆ X and finite (snd (snd args))*

shows *split-dom TYPE('a::{zero,one}) args*

<proof>

corollary *split-domI: finite X ⇒ U ⊆ X ⇒ finite S ⇒ split-dom TYPE('a::{zero,one}) (t, U, S)*

<proof>

lemma *split-empty*:

assumes *finite X and U ⊆ X*

shows *split t U {} = ([], [(monomial (1::'a::{zero,one}) t, U)])*

<proof>

lemma *split-induct [consumes 3, case-names base1 base2 step]*:

fixes *P :: ('x ⇒₀ nat) ⇒ -*

assumes *finite X and U ⊆ X and finite S*

assumes $\bigwedge t U S. U \subseteq X \Rightarrow \text{finite } S \Rightarrow 0 \in S \Rightarrow P t U S$ $([(\text{monomial } (1::'a::\{\text{zero,one}\}) t, U)], [])$

assumes $\bigwedge t U S. U \subseteq X \Rightarrow \text{finite } S \Rightarrow 0 \notin S \Rightarrow S \cap \cdot[U] = \{\} \Rightarrow P t U S$ $([], [(\text{monomial } 1 t, U)])$

assumes $\bigwedge t U S V x ps0 ps1 qs0 qs1. U \subseteq X \Rightarrow \text{finite } S \Rightarrow 0 \notin S \Rightarrow S \cap \cdot[U] \neq \{\} \Rightarrow V \subseteq U \Rightarrow$
 $S \cap \cdot[V] = \{\} \Rightarrow (\bigwedge V'. V' \subseteq U \Rightarrow S \cap \cdot[V'] = \{\} \Rightarrow \text{card } V' \leq \text{card } V) \Rightarrow$

$x \in U \Rightarrow x \notin V \Rightarrow V = \text{max-subset } U (\lambda V'. S \cap \cdot[V'] = \{\}) \Rightarrow x = (\text{SOME } x'. x' \in U - V) \Rightarrow$

$(ps0, qs0) = \text{split } t (U - \{x\}) S \Rightarrow$

$(ps1, qs1) = \text{split } (\text{Poly-Mapping.single } x 1 + t) U ((\lambda f. f - \text{Poly-Mapping.single } x 1) ' S) \Rightarrow$

$\text{split } t U S = (ps0 @ ps1, qs0 @ qs1) \Rightarrow$

$P t (U - \{x\}) S (ps0, qs0) \Rightarrow$

$P (\text{Poly-Mapping.single } x 1 + t) U ((\lambda f. f - \text{Poly-Mapping.single } x 1) ' S) (ps1, qs1) \Rightarrow$

$P t U S (ps0 @ ps1, qs0 @ qs1)$

shows $P t U S (\text{split } t U S)$

<proof>

lemma *valid-decomp-split*:

assumes *finite X and U ⊆ X and finite S and t ∈ ·[X]*

shows *valid-decomp X (fst ((split t U S)::(- × (((- ⇒₀ 'a::zero-neq-one) × -) list))))*

and *valid-decomp X (snd ((split t U S)::(- × (((- ⇒₀ 'a::zero-neq-one) × -) list))))*

(is valid-decomp - (snd ?s))

<proof>

lemma *monomial-decomp-split*:

assumes *finite X and $U \subseteq X$ and finite S*

shows *monomial-decomp (fst ((split t U S)::(- × (((- \Rightarrow_0 'a::zero-neg-one) × -) list))))*

and *monomial-decomp (snd ((split t U S)::(- × (((- \Rightarrow_0 'a::zero-neg-one) × -) list))))*

(is monomial-decomp (snd ?s))

<proof>

lemma *split-splits-wrt*:

assumes *finite X and $U \subseteq X$ and finite S and $t \in .[X]$*

and *ideal $F \div$ monomial 1 t = ideal (monomial 1 ' S)*

shows *splits-wrt (split t U S) (cone (monomial (1::'a::{comm-ring-1,ring-no-zero-divisors}) t, U)) F*

<proof>

lemma *lem-4-5*:

assumes *finite X and $U \subseteq X$ and $t \in .[X]$ and $F \subseteq P[X]$*

and *ideal $F \div$ monomial 1 t = ideal (monomial (1::'a) ' S)*

and *cone (monomial (1::'a::field) t', V) \subseteq cone (monomial 1 t, U) \cap normal-form F ' P[X]*

shows *$V \subseteq U$ and $S \cap .[V] = \{\}$*

<proof>

lemma *lem-4-6*:

assumes *finite X and $U \subseteq X$ and finite S and $t \in .[X]$ and $F \subseteq P[X]$*

and *ideal $F \div$ monomial 1 t = ideal (monomial 1 ' S)*

assumes *cone (monomial 1 t', V) \subseteq cone (monomial 1 t, U) \cap normal-form F ' P[X]*

obtains *V' where (monomial 1 t, V') \in set (snd (split t U S)) and card V \leq card V'*

<proof>

lemma *lem-4-7*:

assumes *finite X and $S \subseteq .[X]$ and $g \in$ punit.reduced-GB (monomial (1::'a) ' S)*

and *cone-decomp (P[X] \cap ideal (monomial (1::'a::field) ' S)) ps*

and *monomial-decomp ps*

obtains *U where (g, U) \in set ps*

<proof>

lemma *snd-splitI*:

assumes *finite X and $U \subseteq X$ and finite S and $0 \notin S$*

obtains *V where $V \subseteq U$ and (monomial 1 t, V) \in set (snd (split t U S))*

<proof>

lemma *fst-splitE*:

assumes *finite X and $U \subseteq X$ and finite S and $0 \notin S$*
and (*monomial $(1::'a)$ s, V*) \in *set (fst (split t U S))*
obtains *t' x where $t' \in .[X]$ and $x \in X$ and $V \subseteq U$ and $0 \notin (\lambda s. s - t')$ ' S*
and *s = t' + t + Poly-Mapping.single x 1*
and (*monomial $(1::'a::zero-neq-one)$ s, V*) \in *set (fst (split (t' + t) V ((\lambda s. s - t') ' S)))*
and *set (snd (split (t' + t) V ((\lambda s. s - t') ' S))) \subseteq (set (snd (split t U S))) ::*
((- \Rightarrow_0 'a) \times -) set)
<proof>

lemma *lem-4-8:*

assumes *finite X and finite S and $S \subseteq .[X]$ and $0 \notin S$*
and *g \in punit.reduced-GB (monomial $(1::'a)$ ' S)*
obtains *t U where $U \subseteq X$ and (monomial $(1::'a::field)$ t, U) \in set (snd (split 0 X S))*
and *poly-deg g = Suc (deg-pm t)*
<proof>

corollary *cor-4-9:*

assumes *finite X and finite S and $S \subseteq .[X]$*
and *g \in punit.reduced-GB (monomial $(1::'a::field)$ ' S)*
shows *poly-deg g \leq Suc (Max (poly-deg ' fst ' (set (snd (split 0 X S))) :: ((- \Rightarrow_0 'a) \times -) set)))*
(is - \leq Suc (Max (poly-deg ' fst ' ?S)))
<proof>

lemma *standard-decomp-snd-split:*

assumes *finite X and $U \subseteq X$ and finite S and $S \subseteq .[X]$ and $t \in .[X]$*
shows *standard-decomp (deg-pm t) (snd (split t U S)) :: ((- \Rightarrow_0 'a::field) \times -) list*
<proof>

theorem *standard-cone-decomp-snd-split:*

fixes *F*
defines *G \equiv punit.reduced-GB F*
defines *ss \equiv (split 0 X (lpp ' G)) :: ((- \Rightarrow_0 'a::field) \times -) list \times -*
defines *d \equiv Suc (Max (poly-deg ' fst ' set (snd ss)))*
assumes *finite X and $F \subseteq P[X]$*
shows *standard-decomp 0 (snd ss) (is ?thesis1)*
and *cone-decomp (normal-form F ' P[X]) (snd ss) (is ?thesis2)*
and *($\bigwedge f. f \in F \Rightarrow$ homogeneous f) \Rightarrow g \in G \Rightarrow poly-deg g \leq d*
<proof>

10.7 Splitting Ideals

qualified definition *ideal-decomp-aux :: ((x \Rightarrow_0 nat) \Rightarrow_0 'a) set \Rightarrow ((x \Rightarrow_0 nat) \Rightarrow_0 'a) \Rightarrow*
((x \Rightarrow_0 nat) \Rightarrow_0 'a::field) set \times (((x \Rightarrow_0 nat) \Rightarrow_0 'a) \times 'x set) list)
where *ideal-decomp-aux F f =*

(let $J = \text{ideal } F$; $L = (J \div f) \cap P[X]$; $L' = \text{lpp } \text{'punit.reduced-GB } L \text{ in}$
 $((*) f \text{' normal-form } L \text{' } P[X], \text{map } (\text{apfst } ((*) f)) (\text{snd } (\text{split } 0 \ X$
 $L'))))$

context

assumes $\text{fin-}X$: $\text{finite } X$

begin

lemma ideal-decomp-aux :

assumes $\text{finite } F$ **and** $F \subseteq P[X]$ **and** $f \in P[X]$

shows $\text{fst } (\text{ideal-decomp-aux } F f) \subseteq \text{ideal } \{f\}$ (**is** $?thesis1$)

and $\text{ideal } F \cap \text{fst } (\text{ideal-decomp-aux } F f) = \{0\}$ (**is** $?thesis2$)

and $\text{direct-decomp } (\text{ideal } (\text{insert } f F) \cap P[X]) [\text{fst } (\text{ideal-decomp-aux } F f), \text{ideal}$
 $F \cap P[X]]$ (**is** $?thesis3$)

and $\text{cone-decomp } (\text{fst } (\text{ideal-decomp-aux } F f)) (\text{snd } (\text{ideal-decomp-aux } F f))$ (**is**
 $?thesis4$)

and $f \neq 0 \implies \text{valid-decomp } X (\text{snd } (\text{ideal-decomp-aux } F f))$ (**is** $- \implies ?thesis5$)

and $f \neq 0 \implies \text{standard-decomp } (\text{poly-deg } f) (\text{snd } (\text{ideal-decomp-aux } F f))$ (**is**
 $- \implies ?thesis6$)

and $\text{homogeneous } f \implies \text{hom-decomp } (\text{snd } (\text{ideal-decomp-aux } F f))$ (**is** $- \implies$
 $?thesis7$)

$\langle \text{proof} \rangle$

lemma ideal-decompE :

fixes $f0 :: - \Rightarrow_0 'a::\text{field}$

assumes $\text{finite } F$ **and** $F \subseteq P[X]$ **and** $f0 \in P[X]$ **and** $\bigwedge f. f \in F \implies \text{poly-deg } f$
 $\leq \text{poly-deg } f0$

obtains $T \text{ ps}$ **where** $\text{valid-decomp } X \text{ ps}$ **and** $\text{standard-decomp } (\text{poly-deg } f0) \text{ ps}$
and $\text{cone-decomp } T \text{ ps}$

and $(\bigwedge f. f \in F \implies \text{homogeneous } f) \implies \text{hom-decomp } \text{ps}$

and $\text{direct-decomp } (\text{ideal } (\text{insert } f0 F) \cap P[X]) [\text{ideal } \{f0\} \cap P[X], T]$

$\langle \text{proof} \rangle$

10.8 Exact Cone Decompositions

definition $\text{exact-decomp} :: \text{nat} \Rightarrow (((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{zero}) \times 'x \text{ set}) \text{ list} \Rightarrow \text{bool}$

where $\text{exact-decomp } m \text{ ps} \iff (\forall (h, U) \in \text{set } \text{ps}. h \in P[X] \wedge U \subseteq X) \wedge$

$(\forall (h, U) \in \text{set } \text{ps}. \forall (h', U') \in \text{set } \text{ps}. \text{poly-deg } h = \text{poly-deg}$

$h' \implies$

$m < \text{card } U \implies m < \text{card } U' \implies (h, U) = (h',$

$U')$)

lemma exact-decompI :

$(\bigwedge h U. (h, U) \in \text{set } \text{ps} \implies h \in P[X]) \implies (\bigwedge h U. (h, U) \in \text{set } \text{ps} \implies U \subseteq X)$

\implies

$(\bigwedge h h' U U'. (h, U) \in \text{set } \text{ps} \implies (h', U') \in \text{set } \text{ps} \implies \text{poly-deg } h = \text{poly-deg}$
 $h' \implies$

$m < \text{card } U \implies m < \text{card } U' \implies (h, U) = (h', U')) \implies$

$\text{exact-decomp } m \text{ ps}$

<proof>

lemma *exact-decompD*:

assumes *exact-decomp m ps* **and** $(h, U) \in \text{set } ps$

shows $h \in P[X]$ **and** $U \subseteq X$

and $(h', U') \in \text{set } ps \implies \text{poly-deg } h = \text{poly-deg } h' \implies m < \text{card } U \implies m < \text{card } U' \implies$

$(h, U) = (h', U')$

<proof>

lemma *exact-decompI-zero*:

assumes $\bigwedge h U. (h, U) \in \text{set } ps \implies h \in P[X]$ **and** $\bigwedge h U. (h, U) \in \text{set } ps \implies U \subseteq X$

and $\bigwedge h h' U U'. (h, U) \in \text{set } (ps_+) \implies (h', U') \in \text{set } (ps_+) \implies \text{poly-deg } h = \text{poly-deg } h' \implies$

$(h, U) = (h', U')$

shows *exact-decomp 0 ps*

<proof>

lemma *exact-decompD-zero*:

assumes *exact-decomp 0 ps* **and** $(h, U) \in \text{set } (ps_+)$ **and** $(h', U') \in \text{set } (ps_+)$

and $\text{poly-deg } h = \text{poly-deg } h'$

shows $(h, U) = (h', U')$

<proof>

lemma *exact-decomp-imp-valid-decomp*:

assumes *exact-decomp m ps* **and** $\bigwedge h U. (h, U) \in \text{set } ps \implies h \neq 0$

shows *valid-decomp X ps*

<proof>

lemma *exact-decomp-card-X*:

assumes *valid-decomp X ps* **and** $\text{card } X \leq m$

shows *exact-decomp m ps*

<proof>

definition $a :: (((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{zero}) \times 'x \text{ set}) \text{ list} \Rightarrow \text{nat}$

where $a \text{ ps} = (\text{LEAST } k. \text{standard-decomp } k \text{ ps})$

definition $b :: (((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{zero}) \times 'x \text{ set}) \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat}$

where $b \text{ ps } i = (\text{LEAST } d. a \text{ ps} \leq d \wedge (\forall (h, U) \in \text{set } ps. i \leq \text{card } U \longrightarrow \text{poly-deg } h < d))$

lemma $a: \text{standard-decomp } k \text{ ps} \implies \text{standard-decomp } (a \text{ ps}) \text{ ps}$

<proof>

lemma *a-Nil*:

assumes $ps_+ = []$

shows $a \text{ ps} = 0$

<proof>

lemma a-nonempty:

assumes *valid-decomp X ps and standard-decomp k ps and ps₊ ≠ []*
shows $a\ ps = \text{Min}(\text{poly-deg } 'fst \ 'set \ (ps_+))$
<proof>

lemma a-nonempty-unique:

assumes *valid-decomp X ps and standard-decomp k ps and ps₊ ≠ []*
shows $a\ ps = k$
<proof>

lemma b:

shows $a\ ps \leq b\ ps\ i$ **and** $(h, U) \in \text{set } ps \implies i \leq \text{card } U \implies \text{poly-deg } h < b\ ps\ i$
<proof>

lemma b-le:

$a\ ps \leq d \implies (\bigwedge h' U'. (h', U') \in \text{set } ps \implies i \leq \text{card } U' \implies \text{poly-deg } h' < d)$
 $\implies b\ ps\ i \leq d$
<proof>

lemma b-decreasing:

assumes $i \leq j$
shows $b\ ps\ j \leq b\ ps\ i$
<proof>

lemma b-Nil:

assumes $ps_+ = []$ **and** $\text{Suc } 0 \leq i$
shows $b\ ps\ i = 0$
<proof>

lemma b-zero:

assumes $ps \neq []$
shows $\text{Suc} (\text{Max} (\text{poly-deg } 'fst \ 'set \ ps)) \leq b\ ps\ 0$
<proof>

corollary b-zero-gr:

assumes $(h, U) \in \text{set } ps$
shows $\text{poly-deg } h < b\ ps\ 0$
<proof>

lemma b-one:

assumes *valid-decomp X ps and standard-decomp k ps*
shows $b\ ps (\text{Suc } 0) = (\text{if } ps_+ = [] \text{ then } 0 \text{ else } \text{Suc} (\text{Max} (\text{poly-deg } 'fst \ 'set \ (ps_+))))$
<proof>

corollary b-one-gr:

assumes *valid-decomp X ps and standard-decomp k ps and (h, U) ∈ set (ps₊)*

shows $\text{poly-deg } h < \text{b } ps \text{ (Suc } 0)$
 <proof>

lemma *b-card-X*:

assumes $\text{exact-decomp } m \text{ ps}$ **and** $\text{Suc } (\text{card } X) \leq i$
shows $\text{b } ps \ i = \text{a } ps$
 <proof>

lemma *lem-6-1-1*:

assumes $\text{standard-decomp } k \text{ ps}$ **and** $\text{exact-decomp } m \text{ ps}$ **and** $\text{Suc } 0 \leq i$
and $i \leq \text{card } X$ **and** $\text{b } ps \text{ (Suc } i) \leq d$ **and** $d < \text{b } ps \ i$
obtains $h \ U$ **where** $(h, U) \in \text{set } (ps_+)$ **and** $\text{poly-deg } h = d$ **and** $\text{card } U = i$
 <proof>

corollary *lem-6-1-2*:

assumes $\text{standard-decomp } k \text{ ps}$ **and** $\text{exact-decomp } 0 \text{ ps}$ **and** $\text{Suc } 0 \leq i$
and $i \leq \text{card } X$ **and** $\text{b } ps \text{ (Suc } i) \leq d$ **and** $d < \text{b } ps \ i$
obtains $h \ U$ **where** $\{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d\} = \{(h, U)\}$ **and**
 $\text{card } U = i$
 <proof>

corollary *lem-6-1-2'*:

assumes $\text{standard-decomp } k \text{ ps}$ **and** $\text{exact-decomp } 0 \text{ ps}$ **and** $\text{Suc } 0 \leq i$
and $i \leq \text{card } X$ **and** $\text{b } ps \text{ (Suc } i) \leq d$ **and** $d < \text{b } ps \ i$
shows $\text{card } \{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d\} = 1$ **(is** $\text{card } ?A = -)$
and $\{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d \wedge \text{card } U' = i\} = \{(h', U') \in \text{set}$
 $(ps_+). \text{poly-deg } h' = d\}$
(is $?B = -)$
and $\text{card } \{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d \wedge \text{card } U' = i\} = 1$
 <proof>

corollary *lem-6-1-3*:

assumes $\text{standard-decomp } k \text{ ps}$ **and** $\text{exact-decomp } 0 \text{ ps}$ **and** $\text{Suc } 0 \leq i$
and $i \leq \text{card } X$ **and** $(h, U) \in \text{set } (ps_+)$ **and** $\text{card } U = i$
shows $\text{b } ps \text{ (Suc } i) \leq \text{poly-deg } h$
 <proof> **fun** *shift-list* :: $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\{\text{comm-ring-1}, \text{ring-no-zero-divisors}\})$
 $\times 'x \text{ set}) \Rightarrow$

$'x \Rightarrow - \text{list} \Rightarrow - \text{list}$ **where**

$\text{shift-list } (h, U) \ x \ ps =$
 $((\text{punit.monom-mult } 1 \ (\text{Poly-Mapping.single } x \ 1) \ h, U) \# (h, U - \{x\}) \#$
 $\text{removeAll } (h, U) \ ps)$

declare *shift-list.simps*[*simp del*]

lemma *monomial-decomp-shift-list*:

assumes $\text{monomial-decomp } ps$ **and** $hU \in \text{set } ps$
shows $\text{monomial-decomp } (\text{shift-list } hU \ x \ ps)$
 <proof>

lemma *hom-decomp-shift-list*:
assumes *hom-decomp ps* **and** $hU \in \text{set } ps$
shows *hom-decomp (shift-list hU x ps)*
 $\langle \text{proof} \rangle$

lemma *valid-decomp-shift-list*:
assumes *valid-decomp X ps* **and** $(h, U) \in \text{set } ps$ **and** $x \in U$
shows *valid-decomp X (shift-list (h, U) x ps)*
 $\langle \text{proof} \rangle$

lemma *standard-decomp-shift-list*:
assumes *standard-decomp k ps* **and** $(h1, U1) \in \text{set } ps$ **and** $(h2, U2) \in \text{set } ps$
and *poly-deg h1 = poly-deg h2* **and** $\text{card } U2 \leq \text{card } U1$ **and** $(h1, U1) \neq (h2, U2)$ **and** $x \in U2$
shows *standard-decomp k (shift-list (h2, U2) x ps)*
 $\langle \text{proof} \rangle$

lemma *cone-decomp-shift-list*:
assumes *valid-decomp X ps* **and** *cone-decomp T ps* **and** $(h, U) \in \text{set } ps$ **and** $x \in U$
shows *cone-decomp T (shift-list (h, U) x ps)*
 $\langle \text{proof} \rangle$

10.9 Functions *shift* and *exact*

context
fixes $k \ m :: \text{nat}$
begin

context
fixes $d :: \text{nat}$
begin

definition *shift2-inv* :: $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \text{zero}) \times 'x \text{ set}) \text{ list} \Rightarrow \text{bool}$ **where**
shift2-inv qs $\longleftrightarrow \text{valid-decomp } X \text{ qs} \wedge \text{standard-decomp } k \text{ qs} \wedge \text{exact-decomp } (\text{Suc } m) \text{ qs} \wedge$
 $(\forall d0 < d. \text{card } \{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d0 \wedge m < \text{card } (\text{snd } q)\} \leq 1)$

fun *shift1-inv* :: $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \text{ list} \times ((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \text{zero}) \times 'x \text{ set}) \text{ set} \Rightarrow \text{bool}$
where *shift1-inv (qs, B)* $\longleftrightarrow B = \{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d \wedge m < \text{card } (\text{snd } q)\} \wedge \text{shift2-inv } qs$

lemma *shift2-invI*:
 $\text{valid-decomp } X \text{ qs} \Longrightarrow \text{standard-decomp } k \text{ qs} \Longrightarrow \text{exact-decomp } (\text{Suc } m) \text{ qs} \Longrightarrow$
 $(\wedge d0. d0 < d \Longrightarrow \text{card } \{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d0 \wedge m < \text{card } (\text{snd } q)\} \leq 1) \Longrightarrow$
 $\text{shift2-inv } qs$

<proof>

lemma *shift2-invD*:

assumes *shift2-inv qs*

shows *valid-decomp X qs and standard-decomp k qs and exact-decomp (Suc m) qs*

and $d0 < d \implies \text{card } \{q \in \text{set } qs. \text{poly-deg } (fst\ q) = d0 \wedge m < \text{card } (snd\ q)\} \leq 1$

<proof>

lemma *shift1-invI*:

$B = \{q \in \text{set } qs. \text{poly-deg } (fst\ q) = d \wedge m < \text{card } (snd\ q)\} \implies \text{shift2-inv } qs \implies \text{shift1-inv } (qs, B)$

<proof>

lemma *shift1-invD*:

assumes *shift1-inv (qs, B)*

shows $B = \{q \in \text{set } qs. \text{poly-deg } (fst\ q) = d \wedge m < \text{card } (snd\ q)\}$ **and** *shift2-inv qs*

<proof>

declare *shift1-inv.simps[simp del]*

lemma *shift1-inv-finite-snd*:

assumes *shift1-inv (qs, B)*

shows *finite B*

<proof>

lemma *shift1-inv-some-snd*:

assumes *shift1-inv (qs, B)* **and** $1 < \text{card } B$ **and** $(h, U) = (\text{SOME } b. b \in B \wedge \text{card } (snd\ b) = \text{Suc } m)$

shows $(h, U) \in B$ **and** $(h, U) \in \text{set } qs$ **and** *poly-deg h = d* **and** $\text{card } U = \text{Suc } m$

<proof>

lemma *shift1-inv-preserved*:

assumes *shift1-inv (qs, B)* **and** $1 < \text{card } B$ **and** $(h, U) = (\text{SOME } b. b \in B \wedge \text{card } (snd\ b) = \text{Suc } m)$

and $x = (\text{SOME } y. y \in U)$

shows *shift1-inv (shift-list (h, U) x qs, B - {(h, U)})*

<proof>

function (*domintros*) *shift1* :: $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \text{ list} \times (((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \text{ set}) \Rightarrow$

$((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \text{ list} \times$
 $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a)::\{\text{comm-ring-1, ring-no-zero-divisors}\})$

$\times 'x \text{ set}) \text{ set})$

where

shift1 (qs, B) =

(if $1 < \text{card } B$ then
 let $(h, U) = \text{SOME } b. b \in B \wedge \text{card } (\text{snd } b) = \text{Suc } m; x = \text{SOME } y. y \in U$
 in
 $\text{shift1 } (\text{shift-list } (h, U) \ x \ qs, B - \{(h, U)\})$
 $\text{else } (qs, B)$
 <proof>

lemma *shift1-domI*:
 assumes *shift1-inv args*
 shows *shift1-dom args*
 <proof>

lemma *shift1-induct* [consumes 1, case-names base step]:
 assumes *shift1-inv args*
 assumes $\bigwedge qs \ B. \text{shift1-inv } (qs, B) \implies \text{card } B \leq 1 \implies P \ (qs, B) \ (qs, B)$
 assumes $\bigwedge qs \ B \ h \ U \ x. \text{shift1-inv } (qs, B) \implies 1 < \text{card } B \implies$
 $(h, U) = (\text{SOME } b. b \in B \wedge \text{card } (\text{snd } b) = \text{Suc } m) \implies x = (\text{SOME } y.$
 $y \in U) \implies$
 $\text{finite } U \implies x \in U \implies \text{card } (U - \{x\}) = m \implies$
 $P \ (\text{shift-list } (h, U) \ x \ qs, B - \{(h, U)\}) \ (\text{shift1 } (\text{shift-list } (h, U) \ x \ qs, B$
 $- \{(h, U)\})) \implies$
 $P \ (qs, B) \ (\text{shift1 } (\text{shift-list } (h, U) \ x \ qs, B - \{(h, U)\}))$
 shows $P \ \text{args } (\text{shift1 } \text{args})$
 <proof>

lemma *shift1-1*:
 assumes *shift1-inv args* and $d0 \leq d$
 shows $\text{card } \{q \in \text{set } (\text{fst } (\text{shift1 } \text{args})). \text{poly-deg } (\text{fst } q) = d0 \wedge m < \text{card } (\text{snd } q)\} \leq 1$
 <proof>

lemma *shift1-2*:
shift1-inv args \implies
 $\text{card } \{q \in \text{set } (\text{fst } (\text{shift1 } \text{args})). m < \text{card } (\text{snd } q)\} \leq \text{card } \{q \in \text{set } (\text{fst } \text{args}).$
 $m < \text{card } (\text{snd } q)\}$
 <proof>

lemma *shift1-3*: *shift1-inv args* $\implies \text{cone-decomp } T \ (\text{fst } \text{args}) \implies \text{cone-decomp } T$
 $(\text{fst } (\text{shift1 } \text{args}))$
 <proof>

lemma *shift1-4*:
shift1-inv args \implies
 $\text{Max } (\text{poly-deg } \text{'fst ' set } (\text{fst } \text{args})) \leq \text{Max } (\text{poly-deg } \text{'fst ' set } (\text{fst } (\text{shift1 } \text{args})))$
 <proof>

lemma *shift1-5*: *shift1-inv args* $\implies \text{fst } (\text{shift1 } \text{args}) = [] \iff \text{fst } \text{args} = []$
 <proof>

lemma *shift1-6*: $\text{shift1-inv args} \implies \text{monomial-decomp (fst args)} \implies \text{monomial-decomp (fst (shift1 args))}$
 ⟨proof⟩

lemma *shift1-7*: $\text{shift1-inv args} \implies \text{hom-decomp (fst args)} \implies \text{hom-decomp (fst (shift1 args))}$
 ⟨proof⟩

end

lemma *shift2-inv-preserved*:

assumes *shift2-inv d qs*

shows $\text{shift2-inv (Suc d) (fst (shift1 (qs, \{q \in \text{set } qs. \text{poly-deg (fst } q) = d \wedge m < \text{card (snd } q)\})))}$

⟨proof⟩

function *shift2* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow (((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \text{ list} \Rightarrow$
 $((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\{\text{comm-ring-1, ring-no-zero-divisors}\}) \times 'x$
 $\text{set}) \text{ list}$ **where**

shift2 c d qs =

(if c ≤ d then qs

else shift2 c (Suc d) (fst (shift1 (qs, \{q \in \text{set } qs. \text{poly-deg (fst } q) = d \wedge m < \text{card (snd } q)\})))

⟨proof⟩

termination ⟨proof⟩

lemma *shift2-1*: $\text{shift2-inv d qs} \implies \text{shift2-inv c (shift2 c d qs)}$
 ⟨proof⟩

lemma *shift2-2*:

$\text{shift2-inv d qs} \implies$

$\text{card } \{q \in \text{set (shift2 c d qs). } m < \text{card (snd } q)\} \leq \text{card } \{q \in \text{set } qs. m < \text{card (snd } q)\}$

⟨proof⟩

lemma *shift2-3*: $\text{shift2-inv d qs} \implies \text{cone-decomp T qs} \implies \text{cone-decomp T (shift2 c d qs)}$

⟨proof⟩

lemma *shift2-4*:

$\text{shift2-inv d qs} \implies \text{Max (poly-deg 'fst ' set } qs) \leq \text{Max (poly-deg 'fst ' set (shift2 c d qs))}$

⟨proof⟩

lemma *shift2-5*:

$\text{shift2-inv d qs} \implies \text{shift2 c d qs} = [] \iff qs = []$
 ⟨proof⟩

lemma *shift2-6*:

$shift2\text{-inv } d \text{ } qs \implies monomial\text{-decomp } qs \implies monomial\text{-decomp } (shift2 \text{ } c \text{ } d \text{ } qs)$
 ⟨proof⟩

lemma *shift2-7*:

$shift2\text{-inv } d \text{ } qs \implies hom\text{-decomp } qs \implies hom\text{-decomp } (shift2 \text{ } c \text{ } d \text{ } qs)$
 ⟨proof⟩

definition *shift* :: ((($'x \Rightarrow_0 \text{ } nat$) $\Rightarrow_0 'a$) $\times 'x \text{ } set$) *list* \Rightarrow
 ((($'x \Rightarrow_0 \text{ } nat$) $\Rightarrow_0 'a::\{comm\text{-ring-1}, ring\text{-no-zero-divisors}\}$) \times

$'x \text{ } set$) *list*

where $shift \text{ } qs = shift2 \text{ } (k + card \{q \in set \text{ } qs. m < card \text{ } (snd \text{ } q)\}) \text{ } k \text{ } qs$

lemma *shift2-inv-init*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $(Suc \text{ } m) \text{ } qs$

shows *shift2-inv* $k \text{ } qs$

⟨proof⟩

lemma *shift*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $(Suc \text{ } m) \text{ } qs$

shows *valid-decomp* $X \text{ } (shift \text{ } qs)$ **and** *standard-decomp* $k \text{ } (shift \text{ } qs)$ **and** *exact-decomp* $m \text{ } (shift \text{ } qs)$

⟨proof⟩

lemma *monomial-decomp-shift*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $(Suc \text{ } m) \text{ } qs$

and *monomial-decomp* qs

shows *monomial-decomp* $(shift \text{ } qs)$

⟨proof⟩

lemma *hom-decomp-shift*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $(Suc \text{ } m) \text{ } qs$

and *hom-decomp* qs

shows *hom-decomp* $(shift \text{ } qs)$

⟨proof⟩

lemma *cone-decomp-shift*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $(Suc \text{ } m) \text{ } qs$

and *cone-decomp* $T \text{ } qs$

shows *cone-decomp* $T \text{ } (shift \text{ } qs)$

⟨proof⟩

lemma *Max-shift-ge*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $(Suc \text{ } m) \text{ } qs$

shows $Max (poly-deg \text{ 'fst ' set } qs) \leq Max (poly-deg \text{ 'fst ' set } (shift \text{ } qs))$
 ⟨proof⟩

lemma *shift-Nil-iff*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $(Suc \text{ } m) \text{ } qs$

shows $shift \text{ } qs = [] \longleftrightarrow qs = []$
 ⟨proof⟩

end

primrec *exact-aux* :: $nat \Rightarrow nat \Rightarrow (((x \Rightarrow_0 \text{ } nat) \Rightarrow_0 \text{ 'a}) \times \text{ 'x set}) \text{ list} \Rightarrow$
 $((x \Rightarrow_0 \text{ } nat) \Rightarrow_0 \text{ 'a}::\{comm-ring-1,ring-no-zero-divisors\}) \times \text{ 'x}$
set) list **where**

exact-aux $k \text{ } 0 \text{ } qs = qs \mid$
exact-aux $k \text{ } (Suc \text{ } m) \text{ } qs = exact-aux \text{ } k \text{ } m \text{ } (shift \text{ } k \text{ } m \text{ } qs)$

lemma *exact-aux*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $m \text{ } qs$

shows *valid-decomp* $X \text{ } (exact-aux \text{ } k \text{ } m \text{ } qs)$ **(is** *?thesis1*)
and *standard-decomp* $k \text{ } (exact-aux \text{ } k \text{ } m \text{ } qs)$ **(is** *?thesis2*)
and *exact-decomp* $0 \text{ } (exact-aux \text{ } k \text{ } m \text{ } qs)$ **(is** *?thesis3*)

⟨proof⟩

lemma *monomial-decomp-exact-aux*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $m \text{ } qs$
and *monomial-decomp* qs

shows *monomial-decomp* $(exact-aux \text{ } k \text{ } m \text{ } qs)$
 ⟨proof⟩

lemma *hom-decomp-exact-aux*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $m \text{ } qs$
and *hom-decomp* qs

shows *hom-decomp* $(exact-aux \text{ } k \text{ } m \text{ } qs)$
 ⟨proof⟩

lemma *cone-decomp-exact-aux*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $m \text{ } qs$
and *cone-decomp* $T \text{ } qs$

shows *cone-decomp* $T \text{ } (exact-aux \text{ } k \text{ } m \text{ } qs)$
 ⟨proof⟩

lemma *Max-exact-aux-ge*:

assumes *valid-decomp* $X \text{ } qs$ **and** *standard-decomp* $k \text{ } qs$ **and** *exact-decomp* $m \text{ } qs$

shows $Max (poly-deg \text{ 'fst ' set } qs) \leq Max (poly-deg \text{ 'fst ' set } (exact-aux \text{ } k \text{ } m \text{ } qs))$
 ⟨proof⟩

lemma *exact-aux-Nil-iff*:

assumes *valid-decomp* X qs **and** *standard-decomp* k qs **and** *exact-decomp* m qs
shows *exact-aux* k m $qs = [] \longleftrightarrow qs = []$
 $\langle proof \rangle$

definition *exact* :: $nat \Rightarrow ((('x \Rightarrow_0 nat) \Rightarrow_0 'a) \times 'x set) list \Rightarrow$
 $((('x \Rightarrow_0 nat) \Rightarrow_0 'a :: \{comm-ring-1, ring-no-zero-divisors\}) \times$
 $'x set) list$
where *exact* k $qs = exact-aux$ k (*card* X) qs

lemma *exact*:
assumes *valid-decomp* X qs **and** *standard-decomp* k qs
shows *valid-decomp* X (*exact* k qs) (**is** *?thesis1*)
and *standard-decomp* k (*exact* k qs) (**is** *?thesis2*)
and *exact-decomp* 0 (*exact* k qs) (**is** *?thesis3*)
 $\langle proof \rangle$

lemma *monomial-decomp-exact*:
assumes *valid-decomp* X qs **and** *standard-decomp* k qs **and** *monomial-decomp* qs
shows *monomial-decomp* (*exact* k qs)
 $\langle proof \rangle$

lemma *hom-decomp-exact*:
assumes *valid-decomp* X qs **and** *standard-decomp* k qs **and** *hom-decomp* qs
shows *hom-decomp* (*exact* k qs)
 $\langle proof \rangle$

lemma *cone-decomp-exact*:
assumes *valid-decomp* X qs **and** *standard-decomp* k qs **and** *cone-decomp* T qs
shows *cone-decomp* T (*exact* k qs)
 $\langle proof \rangle$

lemma *Max-exact-ge*:
assumes *valid-decomp* X qs **and** *standard-decomp* k qs
shows *Max* (*poly-deg* 'fst 'set qs) \leq *Max* (*poly-deg* 'fst 'set (*exact* k qs))
 $\langle proof \rangle$

lemma *exact-Nil-iff*:
assumes *valid-decomp* X qs **and** *standard-decomp* k qs
shows *exact* k $qs = [] \longleftrightarrow qs = []$
 $\langle proof \rangle$

corollary *b-zero-exact*:
assumes *valid-decomp* X qs **and** *standard-decomp* k qs **and** $qs \neq []$
shows *Suc* (*Max* (*poly-deg* 'fst 'set qs)) \leq b (*exact* k qs) 0
 $\langle proof \rangle$

lemma *normal-form-exact-decompE*:
assumes $F \subseteq P[X]$
obtains qs **where** *valid-decomp* X qs **and** *standard-decomp* 0 qs **and** *mono-*


```

mial-decomp qs
  and cone-decomp (normal-form  $F \in P[X]$ ) qs and exact-decomp  $0 \in qs$ 
  and  $\bigwedge g. (\bigwedge f. f \in F \implies \text{homogeneous } f) \implies g \in \text{punit.reduced-GB } F \implies$ 
poly-deg  $g \leq b \text{ } qs \ 0$ 
  <proof>

end

end

end

end

```

11 Dubé's Degree-Bound for Homogeneous Gröbner Bases

```

theory Dube-Bound
  imports Poly-Fun Cone-Decomposition Degree-Bound-Utils
begin

  context fixes  $n \ d :: \text{nat}$ 
begin

  function Dube-aux ::  $\text{nat} \Rightarrow \text{nat}$  where
    Dube-aux  $j = (\text{if } j + 2 < n \text{ then}$ 
       $2 + ((\text{Dube-aux } (j + 1)) \text{ choose } 2) + (\sum_{i=j+3..n-1}. (\text{Dube-aux } i) \text{ choose } (\text{Suc } (i - j)))$ 
       $\text{else if } j + 2 = n \text{ then } d^2 + 2 * d \text{ else } 2 * d)$ 
    <proof>
  termination <proof>

  definition Dube ::  $\text{nat}$  where Dube = (if  $n \leq 1 \vee d = 0$  then  $d$  else Dube-aux  $1$ )

  lemma Dube-aux-ge-d:  $d \leq \text{Dube-aux } j$ 
  <proof>

  corollary Dube-ge-d:  $d \leq \text{Dube}$ 
  <proof>

```

Dubé in [1] proves the following theorem, to obtain a short closed form for the degree bound. However, the proof he gives is wrong: In the last-but-one proof step of Lemma 8.1 the sum on the right-hand-side of the inequality can be greater than $1/2$ (e.g. for $n = 7$, $d = 2$ and $j = (1::'a)$), rendering the value inside the big brackets negative. This is also true without the additional summand 2 we had to introduce in function *local.Dube-aux* to correct another mistake found in [1]. Nonetheless, experiments carried out in Mathematica still suggest that the short closed form is a valid upper bound

for *local.Dube*, even with the additional summand 2. So, with some effort it might be possible to prove the theorem below; but in fact function *local.Dube* gives typically much better (i.e. smaller) values for concrete values of n and d , so it is better to stick to *local.Dube* instead of the closed form anyway. Asymptotically, as n tends to infinity, *local.Dube* grows double exponentially, too.

theorem *rat-of-nat Dube* $\leq 2 * ((\text{rat-of-nat } d)^2 / 2 + (\text{rat-of-nat } d)) \wedge (2 \wedge (n - 2))$

<proof>

end

11.1 Hilbert Function and Hilbert Polynomial

context *pm-powerprod*

begin

context

fixes $X :: 'x \text{ set}$

assumes *fin-X: finite X*

begin

lemma *Hilbert-fun-cone-aux:*

assumes $h \in P[X]$ **and** $h \neq 0$ **and** $U \subseteq X$ **and** *homogeneous* ($h :: - \Rightarrow_0 'a :: \text{field}$)

shows *Hilbert-fun (cone (h, U)) z = card {t \in .[U]. deg-pm t + poly-deg h = z}*

<proof>

lemma *Hilbert-fun-cone-empty:*

assumes $h \in P[X]$ **and** $h \neq 0$ **and** *homogeneous* ($h :: - \Rightarrow_0 'a :: \text{field}$)

shows *Hilbert-fun (cone (h, {})) z = (if poly-deg h = z then 1 else 0)*

<proof>

lemma *Hilbert-fun-cone-nonempty:*

assumes $h \in P[X]$ **and** $h \neq 0$ **and** $U \subseteq X$ **and** *homogeneous* ($h :: - \Rightarrow_0 'a :: \text{field}$)

and $U \neq \{\}$

shows *Hilbert-fun (cone (h, U)) z =*

(if poly-deg h \le z then ((z - poly-deg h) + (card U - 1)) choose (card U - 1) else 0)

<proof>

corollary *Hilbert-fun-Polys:*

assumes $X \neq \{\}$

shows *Hilbert-fun (P[X]::(- \Rightarrow_0 'a :: \text{field}) set) z = (z + (card X - 1)) choose (card X - 1)*

<proof>

lemma *Hilbert-fun-cone-decomp:*

assumes *cone-decomp T ps and valid-decomp X ps and hom-decomp ps*

shows *Hilbert-fun* $T z = (\sum hU \in \text{set } ps. \text{Hilbert-fun } (\text{cone } hU) z)$
 ⟨*proof*⟩

definition *Hilbert-poly* :: $(\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{int} \Rightarrow \text{int}$

where *Hilbert-poly* $b =$
 $(\lambda z :: \text{int}. \text{let } n = \text{card } X \text{ in}$
 $((z - b (\text{Suc } n) + n) \text{gchoose } n) - 1 - (\sum i=1..n. (z - b i + i -$
 $1) \text{gchoose } i))$

lemma *poly-fun-Hilbert-poly*: *poly-fun* (*Hilbert-poly* b)
 ⟨*proof*⟩

lemma *Hilbert-fun-eq-Hilbert-poly-plus-card*:

assumes $X \neq \{\}$ **and** *valid-decomp* $X ps$ **and** *hom-decomp* ps **and** *cone-decomp*
 $T ps$
and *standard-decomp* $k ps$ **and** *exact-decomp* $X 0 ps$ **and** $b ps (\text{Suc } 0) \leq d$
shows $\text{int } (\text{Hilbert-fun } T d) = \text{card } \{h :: - \Rightarrow_0 'a :: \text{field}. (h, \{\}) \in \text{set } ps \wedge \text{poly-deg}$
 $h = d\} + \text{Hilbert-poly } (b ps) d$
 ⟨*proof*⟩

corollary *Hilbert-fun-eq-Hilbert-poly*:

assumes $X \neq \{\}$ **and** *valid-decomp* $X ps$ **and** *hom-decomp* ps **and** *cone-decomp*
 $T ps$
and *standard-decomp* $k ps$ **and** *exact-decomp* $X 0 ps$ **and** $b ps 0 \leq d$
shows $\text{int } (\text{Hilbert-fun } (T :: (- \Rightarrow_0 'a :: \text{field}) \text{set}) d) = \text{Hilbert-poly } (b ps) d$
 ⟨*proof*⟩

11.2 Dubé's Bound

context

fixes $f :: ('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \text{field}$
fixes F
assumes *n-gr-1*: $1 < \text{card } X$ **and** *fin-F*: *finite* F **and** *F-sub*: $F \subseteq P[X]$ **and**
f-in: $f \in F$
and *hom-F*: $\bigwedge f'. f' \in F \Rightarrow \text{homogeneous } f'$ **and** *f-max*: $\bigwedge f'. f' \in F \Rightarrow$
 $\text{poly-deg } f' \leq \text{poly-deg } f$
and *d-gr-0*: $0 < \text{poly-deg } f$ **and** *ideal-f-neq*: $\text{ideal } \{f\} \neq \text{ideal } F$
begin

private abbreviation (*input*) $n \equiv \text{card } X$

private abbreviation (*input*) $d \equiv \text{poly-deg } f$

lemma *f-in-Polys*: $f \in P[X]$
 ⟨*proof*⟩

lemma *hom-f*: *homogeneous* f
 ⟨*proof*⟩

lemma *f-not-0*: $f \neq 0$

<proof>

lemma *X-not-empty*: $X \neq \{\}$
<proof>

lemma *n-gr-0*: $0 < n$
<proof>

corollary *int-n-minus-1* [*simp*]: $\text{int } (n - \text{Suc } 0) = \text{int } n - 1$
<proof>

lemma *int-n-minus-2* [*simp*]: $\text{int } (n - \text{Suc } (\text{Suc } 0)) = \text{int } n - 2$
<proof>

lemma *cone-f-X-sub*: $\text{cone } (f, X) \subseteq P[X]$
<proof>

lemma *ideal-Int-Polys-eq-cone*: $\text{ideal } \{f\} \cap P[X] = \text{cone } (f, X)$

<proof> **definition** *P-ps* **where**

$P\text{-ps} = (\text{SOME } x. \text{valid-decomp } X \text{ (snd } x) \wedge \text{standard-decomp } d \text{ (snd } x) \wedge$
 $\text{exact-decomp } X \ 0 \text{ (snd } x) \wedge \text{cone-decomp } (\text{fst } x) \text{ (snd } x) \wedge$
 $\text{hom-decomp } (\text{snd } x) \wedge$
 $\text{direct-decomp } (\text{ideal } F \cap P[X]) [\text{ideal } \{f\} \cap P[X], \text{fst } x])$

private definition *P* **where** $P = \text{fst } P\text{-ps}$

private definition *ps* **where** $ps = \text{snd } P\text{-ps}$

lemma

shows *valid-ps*: $\text{valid-decomp } X \ ps$ (**is** *?thesis1*)

and *std-ps*: $\text{standard-decomp } d \ ps$ (**is** *?thesis2*)

and *ext-ps*: $\text{exact-decomp } X \ 0 \ ps$ (**is** *?thesis3*)

and *cn-ps*: $\text{cone-decomp } P \ ps$ (**is** *?thesis4*)

and *hom-ps*: $\text{hom-decomp } ps$ (**is** *?thesis5*)

and *decomp-F*: $\text{direct-decomp } (\text{ideal } F \cap P[X]) [\text{ideal } \{f\} \cap P[X], P]$ (**is** *?thesis6*)

<proof>

lemma *P-sub*: $P \subseteq P[X]$
<proof>

lemma *ps-not-Nil*: $ps_+ \neq []$

<proof> **definition** *N* **where** $N = \text{normal-form } F \text{ ' } P[X]$

private definition *qs* **where** $qs = (\text{SOME } qs'. \text{valid-decomp } X \ qs' \wedge \text{standard-decomp}$
 $0 \ qs' \wedge$

$\text{monomial-decomp } qs' \wedge \text{cone-decomp } N \ qs' \wedge$

$\text{exact-decomp } X \ 0 \ qs' \wedge$

$(\forall g \in \text{punit.reduced-GB } F. \text{poly-deg } g \leq \text{b } qs' \ 0))$

private definition $aa \equiv b \ ps$
private definition $bb \equiv b \ qs$
private abbreviation $(input) \ cc \equiv (\lambda i. aa \ i + bb \ i)$

lemma

shows $valid\text{-}qs: valid\text{-}decomp \ X \ qs$ (**is** $?thesis1$)
and $std\text{-}qs: standard\text{-}decomp \ 0 \ qs$ (**is** $?thesis2$)
and $mon\text{-}qs: monomial\text{-}decomp \ qs$ (**is** $?thesis3$)
and $hom\text{-}qs: hom\text{-}decomp \ qs$ (**is** $?thesis6$)
and $cn\text{-}qs: cone\text{-}decomp \ N \ qs$ (**is** $?thesis4$)
and $ext\text{-}qs: exact\text{-}decomp \ X \ 0 \ qs$ (**is** $?thesis5$)
and $deg\text{-}RGB: g \in punit.reduced\text{-}GB \ F \implies poly\text{-}deg \ g \leq bb \ 0$
 $\langle proof \rangle$

lemma $N\text{-}sub: N \subseteq P[X]$
 $\langle proof \rangle$

lemma $decomp\text{-}Polys: direct\text{-}decomp \ P[X] \ [ideal \ \{f\} \cap P[X], P, N]$
 $\langle proof \rangle$

lemma $aa\text{-}Suc\text{-}n \ [simp]: aa \ (Suc \ n) = d$
 $\langle proof \rangle$

lemma $bb\text{-}Suc\text{-}n \ [simp]: bb \ (Suc \ n) = 0$
 $\langle proof \rangle$

lemma $Hilbert\text{-}fun\text{-}X:$

assumes $d \leq z$
shows $Hilbert\text{-}fun \ (P[X]::(- \Rightarrow_0 \ 'a) \ set) \ z =$
 $((z - d) + (n - 1)) \ gchoose \ (n - 1) + Hilbert\text{-}fun \ P \ z + Hilbert\text{-}fun \ N \ z$
 $\langle proof \rangle$

lemma $dube\text{-}eq\text{-}0:$

$(\lambda z::int. (z + int \ n - 1) \ gchoose \ (n - 1)) =$
 $(\lambda z::int. ((z - d + n - 1) \ gchoose \ (n - 1)) + Hilbert\text{-}poly \ aa \ z + Hilbert\text{-}poly$
 $bb \ z)$
(is $?f = ?g$)
 $\langle proof \rangle$

corollary $dube\text{-}eq\text{-}1:$

$(\lambda z::int. (z + int \ n - 1) \ gchoose \ (n - 1)) =$
 $(\lambda z::int. ((z - d + n - 1) \ gchoose \ (n - 1)) + ((z - d + n) \ gchoose \ n) + ((z$
 $+ n) \ gchoose \ n) - 2 -$
 $(\sum_{i=1..n. ((z - aa \ i + i - 1) \ gchoose \ i) + ((z - bb \ i + i - 1) \ gchoose$
 $i)))$
 $\langle proof \rangle$

lemma $dube\text{-}eq\text{-}2:$

assumes $j < n$
shows $(\lambda z::int. (z + int\ n - int\ j - 1)\ gchoose\ (n - j - 1)) =$
 $(\lambda z::int. ((z - d + n - int\ j - 1)\ gchoose\ (n - j - 1)) + ((z - d + n$
 $- j)\ gchoose\ (n - j)) +$
 $((z + n - j)\ gchoose\ (n - j)) - 2 -$
 $(\sum\ i=Suc\ j..n. ((z - aa\ i + i - j - 1)\ gchoose\ (i - j)) + ((z -$
 $bb\ i + i - j - 1)\ gchoose\ (i - j))))$
(is ?f = ?g)
 <proof>

lemma *dube-eq-3*:

assumes $j < n$
shows $(1::int) = (-1) \wedge^{n - Suc\ j} * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) +$
 $(-1) \wedge^{n - j} * ((int\ d - 1)\ gchoose\ (n - j)) - 1 -$
 $(\sum\ i=Suc\ j..n. (-1) \wedge^{i - j} * ((int\ (aa\ i)\ gchoose\ (i - j)) +$
 $(int\ (bb\ i)\ gchoose\ (i - j))))$
 <proof>

lemma *dube-aux-1*:

assumes $(h, \{\}) \in set\ ps \cup set\ qs$
shows $poly-deg\ h < max\ (aa\ 1)\ (bb\ 1)$
 <proof>

lemma

shows $aa\ n: aa\ n = d$ **and** $bb\ n: bb\ n = 0$ **and** $bb\ 0: bb\ 0 \leq max\ (aa\ 1)\ (bb\ 1)$
 <proof>

lemma *dube-eq-4*:

assumes $j < n$
shows $(1::int) = 2 * (-1) \wedge^{n - Suc\ j} * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) -$
 $1 -$
 $(\sum\ i=Suc\ j..n-1. (-1) \wedge^{i - j} * ((int\ (aa\ i)\ gchoose\ (i - j)) +$
 $(int\ (bb\ i)\ gchoose\ (i - j))))$
 <proof>

lemma *cc-Suc*:

assumes $j < n - 1$
shows $int\ (cc\ (Suc\ j)) = 2 + 2 * (-1) \wedge^{n - j} * ((int\ d - 1)\ gchoose\ (n -$
 $Suc\ j)) +$
 $(\sum\ i=j+2..n-1. (-1) \wedge^{i - j} * ((int\ (aa\ i)\ gchoose\ (i - j)) +$
 $(int\ (bb\ i)\ gchoose\ (i - j))))$
 <proof>

lemma *cc-n-minus-1*: $cc\ (n - 1) = 2 * d$

<proof>

Since the case $card\ X = 2$ is settled, we can concentrate on $2 < card\ X$ now.

context

assumes $n\text{-gr-2}$: $2 < n$
begin

lemma $cc\text{-}n\text{-minus-2}$: $cc (n - 2) \leq d^2 + 2 * d$
 $\langle proof \rangle$

lemma $cc\text{-}Suc\text{-}le$:

assumes $j < n - 3$
shows $int (cc (Suc j)) \leq 2 + (int (cc (j + 2))) gchoose 2 + (\sum i=j+4..n-1. int (cc i) gchoose (i - j))$
— Could be proved without coercing to int , because everything is non-negative.
 $\langle proof \rangle$

corollary $cc\text{-}le$:

assumes $0 < j$ **and** $j < n - 2$
shows $cc j \leq 2 + (cc (j + 1) choose 2) + (\sum i=j+3..n-1. cc i choose (Suc (i - j)))$
 $\langle proof \rangle$

corollary $cc\text{-}le\text{-}Dube\text{-}aux$: $0 < j \implies j + 1 \leq n \implies cc j \leq Dube\text{-}aux n d j$
 $\langle proof \rangle$

end

lemma $Dube\text{-}aux$:

assumes $g \in punit.reduced\text{-}GB F$
shows $poly\text{-}deg g \leq Dube\text{-}aux n d 1$
 $\langle proof \rangle$

end

theorem $Dube$:

assumes $finite F$ **and** $F \subseteq P[X]$ **and** $\bigwedge f. f \in F \implies homogeneous f$ **and** $g \in punit.reduced\text{-}GB F$
shows $poly\text{-}deg g \leq Dube (card X) (maxdeg F)$
 $\langle proof \rangle$

corollary $Dube\text{-}is\text{-}hom\text{-}GB\text{-}bound$:

$finite F \implies F \subseteq P[X] \implies is\text{-}hom\text{-}GB\text{-}bound F (Dube (card X) (maxdeg F))$
 $\langle proof \rangle$

end

corollary $Dube\text{-}indets$:

assumes $finite F$ **and** $\bigwedge f. f \in F \implies homogeneous f$ **and** $g \in punit.reduced\text{-}GB F$
shows $poly\text{-}deg g \leq Dube (card (\bigcup (indets ' F))) (maxdeg F)$
 $\langle proof \rangle$

```

corollary Dube-is-hom-GB-bound-indets:
  finite F  $\implies$  is-hom-GB-bound F (Dube (card ( $\bigcup$  (indets ' F)))) (maxdeg F)
  <proof>

end

hide-const (open) pm-powerprod.a pm-powerprod.b

context extended-ord-pm-powerprod
begin

lemma Dube-is-GB-cofactor-bound:
  assumes finite X and finite F and F  $\subseteq$  P[X]
  shows is-GB-cofactor-bound F (Dube (Suc (card X))) (maxdeg F)
  <proof>

lemma Dube-is-GB-cofactor-bound-explicit:
  assumes finite X and finite F and F  $\subseteq$  P[X]
  obtains G where punit.is-Groebner-basis G and ideal G = ideal F and G  $\subseteq$ 
P[X]
  and  $\bigwedge g. g \in G \implies \exists q. g = (\sum_{f \in F}. q f * f) \wedge$ 
  ( $\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq \text{Dube } (\text{Suc } (\text{card } X))$ )
  (maxdeg F)  $\wedge$ 
  (f  $\notin$  F  $\longrightarrow$  q f = 0)
  <proof>

corollary Dube-is-GB-cofactor-bound-indets:
  assumes finite F
  shows is-GB-cofactor-bound F (Dube (Suc (card ( $\bigcup$  (indets ' F)))))) (maxdeg F)
  <proof>

end

end

```

12 Sample Computations of Gröbner Bases via Macaulay Matrices

```

theory Groebner-Macaulay-Examples
imports
  Groebner-Macaulay
  Dube-Bound
  Groebner-Bases.Benchmarks
  Jordan-Normal-Form.Gauss-Jordan-IArray-Impl
  Groebner-Bases.Code-Target-Rat
begin

```


12.1 Combining Groebner-Macaulay. Groebner-Macaulay and Groebner-Macaulay. Dube-Bound

context *extended-ord-pm-powerprod*
begin

theorem *thm-2-3-6-Dube:*

assumes *finite X and set fs* $\subseteq P[X]$

shows *punit.is-Groebner-basis* (*set* (*punit.Macaulay-list* (*deg-shifts X* (*Dube* (*Suc* (*card X*)) (*maxdeg* (*set fs*))) *fs*)))
 ⟨*proof*⟩

theorem *thm-2-3-7-Dube:*

assumes *finite X and set fs* $\subseteq P[X]$

shows $1 \in \text{ideal } (\text{set } fs) \iff 1 \in \text{set } (\text{punit.Macaulay-list } (\text{deg-shifts } X \text{ (Dube (Suc (card X)) (maxdeg (set fs))) } fs))$
 ⟨*proof*⟩

theorem *thm-2-3-6-indets-Dube:*

fixes *fs*

defines $X \equiv \bigcup (\text{indets } ' \text{ set } fs)$

shows *punit.is-Groebner-basis* (*set* (*punit.Macaulay-list* (*deg-shifts X* (*Dube* (*Suc* (*card X*)) (*maxdeg* (*set fs*))) *fs*)))
 ⟨*proof*⟩

theorem *thm-2-3-7-indets-Dube:*

fixes *fs*

defines $X \equiv \bigcup (\text{indets } ' \text{ set } fs)$

shows $1 \in \text{ideal } (\text{set } fs) \iff 1 \in \text{set } (\text{punit.Macaulay-list } (\text{deg-shifts } X \text{ (Dube (Suc (card X)) (maxdeg (set fs))) } fs))$
 ⟨*proof*⟩

end

12.2 Preparations

primrec *remdups-wrt-rev* :: $('a \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow 'b \text{ list} \Rightarrow 'a \text{ list}$ **where**

remdups-wrt-rev f [] vs = [] |

remdups-wrt-rev f (x # xs) vs =

(let fx = f x in if List.member vs fx then remdups-wrt-rev f xs vs else x # (remdups-wrt-rev f xs (fx # vs)))

lemma *remdups-wrt-rev-notin*: $v \in \text{set } vs \implies v \notin f ' \text{ set } (\text{remdups-wrt-rev } f \text{ xs } vs)$
 ⟨*proof*⟩

lemma *distinct-remdups-wrt-rev*: *distinct* (*map f* (*remdups-wrt-rev f xs vs*))

<proof>

lemma *map-of-remdups-wrt-rev'*:

map-of (remdups-wrt-rev fst xs vs) k = map-of (filter ($\lambda x. \text{fst } x \notin \text{set } vs$) xs) k
<proof>

corollary *map-of-remdups-wrt-rev*: *map-of (remdups-wrt-rev fst xs []) = map-of xs*

<proof>

lemma (in *term-powerprod*) *compute-list-to-poly* [code]:

list-to-poly ts cs = distr₀ DRLEX (remdups-wrt-rev fst (zip ts cs) [])
<proof>

lemma (in *ordered-term*) *compute-Macaulay-list* [code]:

Macaulay-list ps =
(let ts = Keys-to-list ps in
filter ($\lambda p. p \neq 0$) (mat-to-polys ts (row-echelon (polys-to-mat ts ps)))
)
<proof>

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*

12.2.1 Connection between $'x \Rightarrow_0 'a \Rightarrow_0 'b$ and $'x, 'a \text{ pp} \Rightarrow_0 'b$

definition *keys-pp-to-list* :: $('x::\text{linorder}, 'a::\text{zero}) \text{ pp} \Rightarrow 'x \text{ list}$

where *keys-pp-to-list t = sorted-list-of-set (keys-pp t)*

lemma *inj-PP*: *inj PP*

<proof>

lemma *inj-mapping-of*: *inj mapping-of*

<proof>

lemma *mapping-of-comp-PP* [simp]:

mapping-of \circ PP = ($\lambda x. x$)

PP \circ mapping-of = ($\lambda x. x$)

<proof>

lemma *map-key-PP-mapping-of* [simp]: *Poly-Mapping.map-key PP (Poly-Mapping.map-key mapping-of p) = p*
 ⟨proof⟩

lemma *map-key-mapping-of-PP* [simp]: *Poly-Mapping.map-key mapping-of (Poly-Mapping.map-key PP p) = p*
 ⟨proof⟩

lemmas *map-key-PP-plus = map-key-plus[OF inj-PP]*
lemmas *map-key-PP-zero [simp] = map-key-zero[OF inj-PP]*

lemma *lookup-map-key-PP*: *lookup (Poly-Mapping.map-key PP p) t = lookup p (PP t)*
 ⟨proof⟩

lemma *keys-map-key-PP*: *keys (Poly-Mapping.map-key PP p) = mapping-of ‘ keys p*
 ⟨proof⟩

lemma *map-key-PP-zero-iff* [iff]: *Poly-Mapping.map-key PP p = 0 \longleftrightarrow p = 0*
 ⟨proof⟩

lemma *map-key-PP-uminus* [simp]: *Poly-Mapping.map-key PP (- p) = - Poly-Mapping.map-key PP p*
 ⟨proof⟩

lemma *map-key-PP-minus*:
Poly-Mapping.map-key PP (p - q) = Poly-Mapping.map-key PP p - Poly-Mapping.map-key PP q
 ⟨proof⟩

lemma *map-key-PP-monomial* [simp]: *Poly-Mapping.map-key PP (monomial c t) = monomial c (mapping-of t)*
 ⟨proof⟩

lemma *map-key-PP-one* [simp]: *Poly-Mapping.map-key PP 1 = 1*
 ⟨proof⟩

lemma *map-key-PP-monom-mult-punit*:
Poly-Mapping.map-key PP (monom-mult-punit c t p) = monom-mult-punit c (mapping-of t) (Poly-Mapping.map-key PP p)
 ⟨proof⟩

lemma *map-key-PP-times*:
*Poly-Mapping.map-key PP (p * q) = Poly-Mapping.map-key PP p * Poly-Mapping.map-key PP (q::(-, -::add-linorder))*
pp \Rightarrow_0 -)
 ⟨proof⟩

lemma *map-key-PP-sum*: $Poly\text{-Mapping.map-key } PP (sum\ f\ A) = (\sum\ a \in A. Poly\text{-Mapping.map-key } PP (f\ a))$
 ⟨proof⟩

lemma *map-key-PP-ideal*:
 $Poly\text{-Mapping.map-key } PP \text{ ' ideal } F = ideal (Poly\text{-Mapping.map-key } PP \text{ ' (F::((- , -::add-linorder) pp \Rightarrow_0 -) set))$
 ⟨proof⟩

12.2.2 Locale *pp-powerprod*

We have to introduce a new locale analogous to *pm-powerprod*, but this time for power-products represented by *pp* rather than *poly-mapping*. This apparently leads to some (more-or-less) duplicate definitions and lemmas, but seems to be the only feasible way to get both

- the convenient representation by *poly-mapping* for theory development, and
- the executable representation by *pp* for code generation.

locale *pp-powerprod* =
 ordered-powerprod ord ord-strict
 for ord::('x::{countable,linorder}, nat) pp \Rightarrow ('x, nat) pp \Rightarrow bool
 and ord-strict
begin

sublocale *gd-powerprod* ⟨proof⟩

sublocale *pp-pm*: extended-ord-pm-powerprod $\lambda s\ t. ord (PP\ s) (PP\ t) \lambda s\ t. ord\text{-strict} (PP\ s) (PP\ t)$
 ⟨proof⟩

definition *poly-deg-pp* :: ('x, nat) pp \Rightarrow_0 'a::zero \Rightarrow nat
 where *poly-deg-pp* p = (if p = 0 then 0 else max-list (map deg-pp (punit.keys-to-list p)))

primrec *deg-le-sect-pp-aux* :: 'x list \Rightarrow nat \Rightarrow ('x, nat) pp \Rightarrow_0 nat **where**
deg-le-sect-pp-aux xs 0 = 1 |
deg-le-sect-pp-aux xs (Suc n) =
 (let p = *deg-le-sect-pp-aux* xs n in p + foldr (\x. (+) (monom-mult-punit 1 (single-pp x 1) p)) xs 0)

definition *deg-le-sect-pp* :: 'x list \Rightarrow nat \Rightarrow ('x, nat) pp list
 where *deg-le-sect-pp* xs d = punit.keys-to-list (*deg-le-sect-pp-aux* xs d)

definition *deg-shifts-pp* :: 'x list \Rightarrow nat \Rightarrow
 (('x, nat) pp \Rightarrow_0 'b) list \Rightarrow (('x, nat) pp \Rightarrow_0 'b::semiring-1)
 list

where $\text{deg-shifts-pp } xs \ d \ fs = \text{concat } (\text{map } (\lambda f. (\text{map } (\lambda t. \text{monom-mult-punit } 1 \ t \ f)) (\text{deg-le-sect-pp } xs \ (d - \text{poly-deg-pp } f)))) \ fs)$

definition $\text{indets-pp} :: (('x, \text{nat}) \text{pp} \Rightarrow_0 'b::\text{zero}) \Rightarrow 'x \ \text{list}$
where $\text{indets-pp } p = \text{remdups } (\text{concat } (\text{map } \text{keys-pp-to-list } (\text{punit.keys-to-list } p)))$

definition $\text{Indets-pp} :: (('x, \text{nat}) \text{pp} \Rightarrow_0 'b::\text{zero}) \ \text{list} \Rightarrow 'x \ \text{list}$
where $\text{Indets-pp } ps = \text{remdups } (\text{concat } (\text{map } \text{indets-pp } ps))$

lemma map-PP-insort :
 $\text{map } PP \ (\text{pp-pm.ordered-powerprod-lin.insort } x \ xs) = \text{ordered-powerprod-lin.insort } (PP \ x) \ (\text{map } PP \ xs)$
 $\langle \text{proof} \rangle$

lemma $\text{map-PP-sorted-list-of-set}$:
 $\text{map } PP \ (\text{pp-pm.ordered-powerprod-lin.sorted-list-of-set } T) = \text{ordered-powerprod-lin.sorted-list-of-set } (PP \ ' T)$
 $\langle \text{proof} \rangle$

lemma $\text{map-PP-pps-to-list}$: $\text{map } PP \ (\text{pp-pm.punit.pps-to-list } T) = \text{punit.pps-to-list } (PP \ ' T)$
 $\langle \text{proof} \rangle$

lemma $\text{map-mapping-of-pps-to-list}$:
 $\text{map } \text{mapping-of } (\text{punit.pps-to-list } T) = \text{pp-pm.punit.pps-to-list } (\text{mapping-of } ' T)$
 $\langle \text{proof} \rangle$

lemma $\text{keys-to-list-map-key-PP}$:
 $\text{pp-pm.punit.keys-to-list } (\text{Poly-Mapping.map-key } PP \ p) = \text{map } \text{mapping-of } (\text{punit.keys-to-list } p)$
 $\langle \text{proof} \rangle$

lemma $\text{Keys-to-list-map-key-PP}$:
 $\text{pp-pm.punit.Keys-to-list } (\text{map } (\text{Poly-Mapping.map-key } PP) \ fs) = \text{map } \text{mapping-of } (\text{punit.Keys-to-list } fs)$
 $\langle \text{proof} \rangle$

lemma $\text{poly-deg-map-key-PP}$: $\text{poly-deg } (\text{Poly-Mapping.map-key } PP \ p) = \text{poly-deg-pp } p$
 $\langle \text{proof} \rangle$

lemma $\text{deg-le-sect-pp-aux-1}$:
assumes $t \in \text{keys } (\text{deg-le-sect-pp-aux } xs \ n)$
shows $\text{deg-pp } t \leq n$ **and** $\text{keys-pp } t \subseteq \text{set } xs$
 $\langle \text{proof} \rangle$

lemma $\text{deg-le-sect-pp-aux-2}$:
assumes $\text{deg-pp } t \leq n$ **and** $\text{keys-pp } t \subseteq \text{set } xs$

shows $t \in \text{keys } (\text{deg-le-sect-pp-aux } xs \ n)$
<proof>

lemma *keys-deg-le-sect-pp-aux:*

$\text{keys } (\text{deg-le-sect-pp-aux } xs \ n) = \{t. \text{deg-pp } t \leq n \wedge \text{keys-pp } t \subseteq \text{set } xs\}$
<proof>

lemma *deg-le-sect-deg-le-sect-pp:*

$\text{map } PP \ (\text{pp-pm.punit.pps-to-list } (\text{deg-le-sect } (\text{set } xs) \ d)) = \text{deg-le-sect-pp } xs \ d$
<proof>

lemma *deg-shifts-deg-shifts-pp:*

$\text{pp-pm.deg-shifts } (\text{set } xs) \ d \ (\text{map } (\text{Poly-Mapping.map-key } PP) \ fs) =$
 $\text{map } (\text{Poly-Mapping.map-key } PP) \ (\text{deg-shifts-pp } xs \ d \ fs)$
<proof>

lemma *ideal-deg-shifts-pp:* $\text{ideal } (\text{set } (\text{deg-shifts-pp } xs \ d \ fs)) = \text{ideal } (\text{set } fs)$

<proof>

lemma *set-indets-pp:* $\text{set } (\text{indets-pp } p) = \text{indets } (\text{Poly-Mapping.map-key } PP \ p)$

<proof>

lemma *poly-to-row-map-key-PP:*

$\text{poly-to-row } (\text{map } \text{pp.mapping-of } xs) \ (\text{Poly-Mapping.map-key } PP \ p) = \text{poly-to-row}$
 $xs \ p$
<proof>

lemma *Macaulay-mat-map-key-PP:*

$\text{pp-pm.punit.Macaulay-mat } (\text{map } (\text{Poly-Mapping.map-key } PP) \ fs) = \text{punit.Macaulay-mat}$
 fs
<proof>

lemma *row-to-poly-mapping-of:*

assumes *distinct ts and dim-vec r = length ts*

shows $\text{row-to-poly } (\text{map } \text{pp.mapping-of } ts) \ r = \text{Poly-Mapping.map-key } PP \ (\text{row-to-poly}$
 $ts \ r)$

<proof>

lemma *mat-to-polys-mapping-of:*

assumes *distinct ts and dim-col m = length ts*

shows $\text{mat-to-polys } (\text{map } \text{pp.mapping-of } ts) \ m = \text{map } (\text{Poly-Mapping.map-key}$
 $PP) \ (\text{mat-to-polys } ts \ m)$

<proof>

lemma *map-key-PP-Macaulay-list:*

$\text{map } (\text{Poly-Mapping.map-key } PP) \ (\text{punit.Macaulay-list } fs) =$
 $\text{pp-pm.punit.Macaulay-list } (\text{map } (\text{Poly-Mapping.map-key } PP) \ fs)$

<proof>

lemma *lpp-map-key-PP*: $pp\text{-}pm.lpp (Poly\text{-}Mapping.map\text{-}key\ PP\ p) = mapping\text{-}of$
 $(lpp\ p)$
 $\langle proof \rangle$

lemma *is-GB-map-key-PP*:
 $finite\ G \implies pp\text{-}pm.punit.is\text{-}Groebner\text{-}basis (Poly\text{-}Mapping.map\text{-}key\ PP\ 'G) \longleftrightarrow$
 $punit.is\text{-}Groebner\text{-}basis\ G$
 $\langle proof \rangle$

lemma *thm-2-3-6-pp*:
assumes *pp-pm.is-GB-cofactor-bound* $(Poly\text{-}Mapping.map\text{-}key\ PP\ 'set\ fs)\ b$
shows *punit.is-Groebner-basis* $(set\ (punit.Macaulay\text{-}list\ (deg\text{-}shifts\text{-}pp\ (Indets\text{-}pp\ fs)\ b\ fs)))$
 $\langle proof \rangle$

lemma *Dube-is-GB-cofactor-bound-pp*:
 $pp\text{-}pm.is\text{-}GB\text{-}cofactor\text{-}bound\ (Poly\text{-}Mapping.map\text{-}key\ PP\ 'set\ fs)$
 $(Dube\ (Suc\ (length\ (Indets\text{-}pp\ fs)))\ (max\text{-}list\ (map\ poly\text{-}deg\text{-}pp\ fs)))$
 $\langle proof \rangle$

definition *GB-Macaulay-Dube* :: $((x, nat)\ pp \Rightarrow_0\ 'a)\ list \Rightarrow ((x, nat)\ pp \Rightarrow_0$
 $'a::field)\ list$
where *GB-Macaulay-Dube* $fs = punit.Macaulay\text{-}list\ (deg\text{-}shifts\text{-}pp\ (Indets\text{-}pp\ fs)$
 $(Dube\ (Suc\ (length\ (Indets\text{-}pp\ fs)))\ (max\text{-}list\ (map\ poly\text{-}deg\text{-}pp$
 $fs)))\ fs)$

lemma *GB-Macaulay-Dube-is-GB*: $punit.is\text{-}Groebner\text{-}basis\ (set\ (GB\text{-}Macaulay\text{-}Dube$
 $fs))$
 $\langle proof \rangle$

lemma *ideal-GB-Macaulay-Dube*: $ideal\ (set\ (GB\text{-}Macaulay\text{-}Dube\ fs)) = ideal\ (set$
 $fs)$
 $\langle proof \rangle$

end

global-interpretation *punit'*: $pp\text{-}powerprod\ ord\text{-}pp\text{-}punit\ cmp\text{-}term\ ord\text{-}pp\text{-}strict\text{-}punit$
 $cmp\text{-}term$

rewrites *punit.adds-term* = $(adds)$
and *punit.pp-of-term* = $(\lambda x. x)$
and *punit.component-of-term* = $(\lambda. ())$
and *punit.monom-mult* = $monom\text{-}mult\text{-}punit$
and *punit.mult-scalar* = $mult\text{-}scalar\text{-}punit$
and *punit'.punit.min-term* = $min\text{-}term\text{-}punit$
and *punit'.punit.lt* = $lt\text{-}punit\ cmp\text{-}term$
and *punit'.punit.lc* = $lc\text{-}punit\ cmp\text{-}term$
and *punit'.punit.tail* = $tail\text{-}punit\ cmp\text{-}term$
and *punit'.punit.ord-p* = $ord\text{-}p\text{-}punit\ cmp\text{-}term$
and *punit'.punit.keys-to-list* = $keys\text{-}to\text{-}list\text{-}punit\ cmp\text{-}term$

```

for cmp-term :: ('a::nat, nat) pp nat-term-order

defines max-punit = punit'.ordered-powerprod-lin.max
and max-list-punit = punit'.ordered-powerprod-lin.max-list
and Keys-to-list-punit = punit'.punit.Keys-to-list
and Macaulay-mat-punit = punit'.punit.Macaulay-mat
and Macaulay-list-punit = punit'.punit.Macaulay-list
and poly-deg-pp-punit = punit'.poly-deg-pp
and deg-le-sect-pp-aux-punit = punit'.deg-le-sect-pp-aux
and deg-le-sect-pp-punit = punit'.deg-le-sect-pp
and deg-shifts-pp-punit = punit'.deg-shifts-pp
and indets-pp-punit = punit'.indets-pp
and Indets-pp-punit = punit'.Indets-pp
and GB-Macaulay-Dube-punit = punit'.GB-Macaulay-Dube

and find-adds-punit = punit'.punit.find-adds
and trd-aux-punit = punit'.punit.trd-aux
and trd-punit = punit'.punit.trd
and comp-min-basis-punit = punit'.punit.comp-min-basis
and comp-red-basis-aux-punit = punit'.punit.comp-red-basis-aux
and comp-red-basis-punit = punit'.punit.comp-red-basis
⟨proof⟩

```

12.3 Computations

```

experiment begin interpretation trivariate0-rat ⟨proof⟩

```

lemma

```

  comp-red-basis-punit DRLEX (GB-Macaulay-Dube-punit DRLEX [X * Y2 + 3
* X2 * Y, Y3 - X3]) =
  [X5, X3 * Y - C0 (1 / 9) * X4, Y3 - X3, X * Y2 + 3 * X2
* Y]
⟨proof⟩

```

end

end

References

- [1] T. W. Dubé. The Structure of Polynomial Ideals and Gröbner Bases. *SIAM Journal on Computing*, 19(4):750–773, 1990.
- [2] A. Maletzky. Formalization of Dubé’s Degree Bounds for Gröbner Bases in Isabelle/HOL. In C. Kaliszyk, E. Brady, A. Kohlhase, and C. Sacerdoti-Coen, editors, *Intelligent Computer Mathematics (Proceedings of CICM 2019, Prague, Czech Republic, July 8-12)*, volume

11617 of *Lecture Notes in Computer Science*. Springer, 2019. to appear; preprint at http://www.risc.jku.at/publications/download/risc_5919/Paper.pdf.

- [3] A. Maletzky. Gröbner Bases and Macaulay Matrices in Isabelle/HOL. Technical report, RISC, Johannes Kepler University Linz, Austria, 2019. http://www.risc.jku.at/publications/download/risc_5929/Paper.pdf; Submitted to Formal Aspects of Computing.
- [4] M. Wiesinger-Widi. *Gröbner Bases and Generalized Sylvester Matrices*. PhD thesis, RISC, Johannes Kepler University Linz, Austria, 2015.