Gröbner Bases, Macaulay Matrices and Dubé's Degree Bounds

Alexander Maletzky*

May 26, 2024

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

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

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 $deg\text{-}sect\ X\ d = .[X] \cap \{t.\ deg\text{-}pm\ t = d\}$

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

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

lemma *deg-sectD*:
  assumes $t \in deg\text{-}sect\ X\ d$
  shows $t \in .[X]$ **and** $deg\text{-}pm\ t = d$
  ⟨*proof*⟩

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

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

⟨*proof*⟩

lemma *deg-le-sectD*:
assumes $t \in$ *deg-le-sect X d*
shows $t \in .[X]$ **and** *deg-pm t* $\leq d$
⟨*proof*⟩

lemma *deg-sect-zero* [*simp*]: *deg-sect X 0* = {*0*}
⟨*proof*⟩

lemma *deg-sect-empty*: *deg-sect* {} *d* = (*if d* = *0 then* {*0*} *else* {})
⟨*proof*⟩

lemma *deg-sect-singleton* [*simp*]: *deg-sect* {*x*} *d* = {*Poly-Mapping.single x d*}
⟨*proof*⟩

lemma *deg-le-sect-zero* [*simp*]: *deg-le-sect X 0* = {*0*}
⟨*proof*⟩

lemma *deg-le-sect-empty* [*simp*]: *deg-le-sect* {} *d* = {*0*}
⟨*proof*⟩

lemma *deg-le-sect-singleton*: *deg-le-sect* {*x*} *d* = *Poly-Mapping.single x* ‘ {..*d*}
⟨*proof*⟩

lemma *deg-sect-mono*: $X \subseteq Y \Longrightarrow$ *deg-sect X d* $\subseteq$ *deg-sect Y d*
⟨*proof*⟩

lemma *deg-le-sect-mono-1*: $X \subseteq Y \Longrightarrow$ *deg-le-sect X d* $\subseteq$ *deg-le-sect Y d*
⟨*proof*⟩

lemma *deg-le-sect-mono-2*: *d1* $\leq$ *d2* $\Longrightarrow$ *deg-le-sect X d1* $\subseteq$ *deg-le-sect X d2*
⟨*proof*⟩

lemma *zero-in-deg-le-sect*: *0* $\in$ *deg-le-sect n d*
⟨*proof*⟩

lemma *deg-sect-disjoint*: *d1* $\neq$ *d2* $\Longrightarrow$ *deg-sect X d1* $\cap$ *deg-sect Y d2* = {}
⟨*proof*⟩

lemma *deg-le-sect-deg-sect-disjoint*: *d1* < *d2* $\Longrightarrow$ *deg-le-sect Y d1* $\cap$ *deg-sect X d2* = {}
⟨*proof*⟩

lemma *deg-sect-Suc*:
deg-sect X (*Suc d*) = ($\bigcup x \in X$. (+) (*Poly-Mapping.single x 1*) ‘ *deg-sect X d*) (**is** *?A* = *?B*)
⟨*proof*⟩

lemma *deg-sect-insert*:
deg-sect (*insert x X*) *d* = ($\bigcup$*d0*≤*d*. (+) (*Poly-Mapping.single x* (*d* − *d0*)) ‘ *deg-sect X d0*)
(**is** *?A* = *?B*)
⟨*proof*⟩

lemma *deg-le-sect-Suc*: *deg-le-sect X* (*Suc d*) = *deg-le-sect X d* ∪ *deg-sect X* (*Suc d*)
⟨*proof*⟩

lemma *deg-le-sect-Suc-2*:
deg-le-sect X (*Suc d*) = *insert 0* ($\bigcup$*x*∈*X*. (+) (*Poly-Mapping.single x 1*) ‘ *deg-le-sect X d*)
(**is** *?A* = *?B*)
⟨*proof*⟩

lemma *finite-deg-sect*:
assumes *finite X*
shows *finite* ((*deg-sect X d*)::(*'x*::*countable* $\Rightarrow_0$ *nat*) *set*)
⟨*proof*⟩

corollary *finite-deg-le-sect*: *finite X* ⟹ *finite* ((*deg-le-sect X d*)::(*'x*::*countable* $\Rightarrow_0$ *nat*) *set*)
⟨*proof*⟩

lemma *keys-subset-deg-le-sectI*:
assumes *p* ∈ *P*[*X*] **and** *poly-deg p* ≤ *d*
shows *keys p* ⊆ *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* ≠ {}
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* :: $(('x \Rightarrow_0 nat) \Rightarrow_0 'b::field)$ *set* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where *is-GB-cofactor-bound F b* $\longleftrightarrow$
$(\exists\, G.$ *punit.is-Groebner-basis G* $\wedge$ *ideal G* $=$ *ideal F* $\wedge$ $($*UN g*:*G. indets g*$)$ $\subseteq$ $($*UN f*:*F. indets f*$)$ $\wedge$
$(\forall\, g{\in}G.\ \exists\, F'\ q.$ *finite* $F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\, f * f) \wedge (\forall f{\in}F'.$ *poly-deg* $(q\, f * f) \leq b)))$

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

lemma *is-GB-cofactor-boundI*:
assumes *punit.is-Groebner-basis G* **and** *ideal G* $=$ *ideal F* **and** $\bigcup($*indets* ‘ $G)$ $\subseteq \bigcup($*indets* ‘ $F)$
and $\bigwedge g.\ g \in G \Longrightarrow \exists\, F'\ q.$ *finite* $F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\ f * f) \wedge (\forall f{\in}F'.$ *poly-deg* $(q\ f * f) \leq b)$
shows *is-GB-cofactor-bound F b*
⟨*proof*⟩

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

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

lemma *is-GB-cofactor-boundE-finite-Polys*:
fixes F :: (('$x \Rightarrow_0$ *nat*) $\Rightarrow_0$ 'b::*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 = *ideal* F **and** $G \subseteq P[X]$
and $\bigwedge g.\ g \in G \Longrightarrow \exists q.\ g = (\sum f{\in}F.\ q\ f * f) \wedge (\forall f.\ q\ f \in P[X] \wedge$ *poly-deg* $(q\ f * f) \leq b)$
⟨*proof*⟩

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

lemma *is-hom-GB-boundI*:
$(\bigwedge g.\ (\bigwedge f.\ f \in F \Longrightarrow$ *homogeneous* $f) \Longrightarrow g \in$ *punit.reduced-GB* $F \Longrightarrow$ *poly-deg* $g \leq b) \Longrightarrow$ *is-hom-GB-bound* F b
⟨*proof*⟩

lemma *is-hom-GB-boundD*:
is-hom-GB-bound F $b \Longrightarrow (\bigwedge f.\ f \in F \Longrightarrow$ *homogeneous* $f) \Longrightarrow g \in$ *punit.reduced-GB* $F \Longrightarrow$ *poly-deg* $g \leq b$
⟨*proof*⟩

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 None* ‘ *extend-indets* ‘ F) b
shows *is-GB-cofactor-bound* F b
⟨*proof*⟩

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** *pmdl* (*set ps*) = *pmdl G* **and** *G* ⊆ *phull* (*set ps*)
shows *is-Groebner-basis* (*set* (*Macaulay-list ps*))
⟨*proof*⟩

4.2 Bounds

context *pm-powerprod*
begin

context
fixes *X* :: *'x set*
assumes *fin-X*: *finite X*
begin

definition *deg-shifts* :: $nat \Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0 'b)$ *list* $\Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0 'b::semiring\text{-}1)$ *list*
where *deg-shifts d fs* = *concat* (*map* (λ*f*. (*map* (λ*t*. *punit.monom-mult 1 t f*) (*punit.pps-to-list* (*deg-le-sect X* (*d* − *poly-deg f*))))) *fs*)

lemma *set-deg-shifts*:
set (*deg-shifts d fs*) = (⋃*f*∈*set fs*. (λ*t*. *punit.monom-mult 1 t f*) ‘ (*deg-le-sect X* (*d* − *poly-deg f*)))
⟨*proof*⟩

corollary *set-deg-shifts-singleton*:
set (*deg-shifts d* [*f*]) = (λ*t*. *punit.monom-mult 1 t f*) ‘ (*deg-le-sect X* (*d* − *poly-deg f*))
⟨*proof*⟩

lemma *deg-shifts-superset*: *set fs* ⊆ *set* (*deg-shifts d fs*)
⟨*proof*⟩

lemma *deg-shifts-mono*:
assumes *set fs* ⊆ *set gs*
shows *set* (*deg-shifts d fs*) ⊆ *set* (*deg-shifts d gs*)
⟨*proof*⟩

lemma *ideal-deg-shifts* [*simp*]: *ideal* (*set* (*deg-shifts d fs*)) = *ideal* (*set fs*)
⟨*proof*⟩

lemma *thm-2-3-6*:
assumes *set fs* ⊆ *P*[*X*] **and** *is-GB-cofactor-bound* (*set fs*) *b*
shows *punit.is-Groebner-basis* (*set* (*punit.Macaulay-list* (*deg-shifts b fs*)))
⟨*proof*⟩

lemma *thm-2-3-7*:
assumes *set fs* ⊆ *P*[*X*] **and** *is-GB-cofactor-bound* (*set fs*) *b*
shows *1* ∈ *ideal* (*set fs*) ⟷ *1* ∈ *set* (*punit.Macaulay-list* (*deg-shifts b fs*)) (**is** *?L* ⟷ *?R*)
⟨*proof*⟩

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* (⋃(*indets* ' (*set fs*))) *b fs*)))
⟨*proof*⟩

lemma *thm-2-3-7-indets*:
assumes *is-GB-cofactor-bound* (*set fs*) *b*
shows *1* ∈ *ideal* (*set fs*) ⟷ *1* ∈ *set* (*punit.Macaulay-list* (*deg-shifts* (⋃(*indets* ' (*set fs*))) *b fs*))
⟨*proof*⟩

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* ≤ *n choose k*
⟨*proof*⟩

Restore original sort constraints:

⟨*ML*⟩

lemma *gbinomial-0-left*: *0 gchoose k* = (*if k* = *0 then 1 else 0*)
⟨*proof*⟩

lemma *gbinomial-eq-0-int*:
assumes *n* < *k*
shows (*int n*) *gchoose k* = *0*
⟨*proof*⟩

corollary *gbinomial-eq-0*: *0* ≤ *a* ⟹ *a* < *int k* ⟹ *a gchoose k* = *0*
⟨*proof*⟩

lemma *int-binomial*: *int* (*n choose k*) = (*int n*) *gchoose k*
⟨*proof*⟩

lemma *falling-fact-pochhammer*: *prod* (*λi. a* − *int i*) {*0..<k*} = (− *1*) ^ *k* ∗ *pochhammer* (− *a*) *k*
⟨*proof*⟩

lemma *falling-fact-pochhammer′*: *prod* (*λi. a* − *int i*) {*0..<k*} = *pochhammer* (*a* − *int k* + *1*) *k*
⟨*proof*⟩

lemma *gbinomial-int-pochhammer*: (*a::int*) *gchoose k* = (− *1*) ^ *k* ∗ *pochhammer* (− *a*) *k div fact k*
⟨*proof*⟩

lemma *gbinomial-int-pochhammer′*: *a gchoose k* = *pochhammer* (*a* − *int k* + *1*) *k div fact k*
⟨*proof*⟩

lemma *fact-dvd-pochhammer*: *fact k dvd pochhammer* (*a::int*) *k*
⟨*proof*⟩

lemma *gbinomial-int-negated-upper*: (*a gchoose k*) = (−*1*) ^ *k* ∗ ((*int k* − *a* − *1*) *gchoose k*)
⟨*proof*⟩

lemma *gbinomial-int-mult-fact*: *fact k* ∗ (*a gchoose k*) = (∏ *i* = *0..<k. a* − *int i*)
⟨*proof*⟩

corollary *gbinomial-int-mult-fact′*: (*a gchoose k*) ∗ *fact k* = (∏ *i* = *0..<k. a* − *int i*)
⟨*proof*⟩

lemma *gbinomial-int-binomial*:
a gchoose k = (*if 0* ≤ *a then int* ((*nat a*) *choose k*) *else* (−*1::int*)^*k* ∗ *int* ((*k* + (*nat* (− *a*)) − *1*) *choose k*))
⟨*proof*⟩

corollary *gbinomial-nneg*: *0* ≤ *a* ⟹ *a gchoose k* = *int* ((*nat a*) *choose k*)
⟨*proof*⟩

corollary *gbinomial-neg*: *a* < *0* ⟹ *a gchoose k* = (−*1::int*)^*k* ∗ *int* ((*k* + (*nat* (− *a*)) − *1*) *choose k*)
⟨*proof*⟩

lemma *of-int-gbinomial*: *of-int* (*a gchoose k*) = (*of-int a* :: *′a::field-char-0*) *gchoose k*
⟨*proof*⟩

lemma *uminus-one-gbinomial* [*simp*]: (− *1*::*int*) *gchoose k* = (− *1*) ^ *k*
⟨*proof*⟩

lemma *gbinomial-int-Suc-Suc*: (*x* + *1*::*int*) *gchoose* (*Suc k*) = (*x gchoose k*) + (*x gchoose* (*Suc k*))
⟨*proof*⟩

corollary *plus-Suc-gbinomial*:
(*x* + (*1* + *int k*)) *gchoose* (*Suc k*) = ((*x* + *int k*) *gchoose k*) + ((*x* + *int k*) *gchoose* (*Suc k*))
(**is** *?l* = *?r*)
⟨*proof*⟩

lemma *gbinomial-int-n-n* [*simp*]: (*int n*) *gchoose n* = *1*
⟨*proof*⟩

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

lemma *zbinomial-eq-0-iff* [*simp*]: *a gchoose k* = *0* ⟷ (*0* ≤ *a* ∧ *a* < *int k*)
⟨*proof*⟩

5.1 Sums

lemma *gchoose-rising-sum-nat*: (∑*j*≤*n*. *int j* + *int k gchoose k*) = (*int n* + *int k* + *1*) *gchoose* (*Suc k*)
⟨*proof*⟩

lemma *gchoose-rising-sum*:
assumes *0* ≤ *n* — Necessary condition.
shows (∑*j*=*0*..*n*. *j* + *int k gchoose k*) = (*n* + *int k* + *1*) *gchoose* (*Suc k*)
⟨*proof*⟩

5.2 Inequalities

lemma *binomial-mono*:
assumes *m* ≤ *n*
shows *m choose k* ≤ *n choose k*
⟨*proof*⟩

lemma *binomial-plus-le*:
assumes *0* < *k*
shows (*m choose k*) + (*n choose k*) ≤ (*m* + *n*) *choose k*
⟨*proof*⟩

lemma *binomial-ineq-1*: *2* ∗ ((*n* + *i*) *choose k*) ≤ *n choose k* + ((*n* + *2* ∗ *i*) *choose k*)
⟨*proof*⟩

lemma *gbinomial-int-nonneg*:

assumes $0 \le (x::int)$
shows $0 \le x$ *gchoose* k
⟨*proof*⟩

lemma *gbinomial-int-mono*:
assumes $0 \le x$ **and** $x \le (y::int)$
shows x *gchoose* $k \le y$ *gchoose* k
⟨*proof*⟩

lemma *gbinomial-int-plus-le*:
assumes $0 < k$ **and** $0 \le x$ **and** $0 \le (y::int)$
shows $(x$ *gchoose* $k) + (y$ *gchoose* $k) \le (x + y)$ *gchoose* k
⟨*proof*⟩

lemma *binomial-int-ineq-1*:
assumes $0 \le x$ **and** $0 \le (y::int)$
shows $2 * (x + y$ *gchoose* $k) \le x$ *gchoose* $k + ((x + 2 * y)$ *gchoose* $k)$
⟨*proof*⟩

corollary *binomial-int-ineq-2*:
assumes $0 \le y$ **and** $y \le (x::int)$
shows $2 * (x$ *gchoose* $k) \le x - y$ *gchoose* $k + (x + y$ *gchoose* $k)$
⟨*proof*⟩

corollary *binomial-int-ineq-3*:
assumes $0 \le y$ **and** $y \le 2 * (x::int)$
shows $2 * (x$ *gchoose* $k) \le y$ *gchoose* $k + (2 * x - y$ *gchoose* $k)$
⟨*proof*⟩

5.3 Backward Difference Operator

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

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

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

lemma *bw-diff-plus* [*simp*]: *bw-diff* $(\lambda x.\ f\ x + g\ x) = (\lambda x.$ *bw-diff* $f\ x +$ *bw-diff* $g\ x)$
⟨*proof*⟩

lemma *bw-diff-uminus* [*simp*]: *bw-diff* $(\lambda x.\ -\ f\ x) = (\lambda x.\ -$ *bw-diff* $f\ x)$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *bw-diff-const-pow*: $(bw\text{-}diff \ \hat{}\hat{}\ k)\ (\lambda\text{-}.\ c) = (if\ k = 0\ then\ \lambda\text{-}.\ c\ else\ (\lambda\text{-}.\ 0))$
⟨*proof*⟩

lemma *bw-diff-id-pow*:
$(bw\text{-}diff \ \hat{}\hat{}\ k)\ (\lambda x.\ x) = (if\ k = 0\ then\ (\lambda x.\ x)\ else\ if\ k = 1\ then\ (\lambda\text{-}.\ 1)\ else\ (\lambda\text{-}.\ 0))$
⟨*proof*⟩

lemma *bw-diff-plus-pow* [*simp*]:
$(bw\text{-}diff \ \hat{}\hat{}\ k)\ (\lambda x.\ f\ x + g\ x) = (\lambda x.\ (bw\text{-}diff \ \hat{}\hat{}\ k)\ f\ x + (bw\text{-}diff \ \hat{}\hat{}\ k)\ g\ x)$
⟨*proof*⟩

lemma *bw-diff-uminus-pow* [*simp*]: $(bw\text{-}diff \ \hat{}\hat{}\ k)\ (\lambda x.\ -\ f\ x) = (\lambda x.\ -\ (bw\text{-}diff \ \hat{}\hat{}\ k)\ f\ x)$
⟨*proof*⟩

lemma *bw-diff-minus-pow* [*simp*]:
$(bw\text{-}diff \ \hat{}\hat{}\ k)\ (\lambda x.\ f\ x - g\ x) = (\lambda x.\ (bw\text{-}diff \ \hat{}\hat{}\ k)\ f\ x - (bw\text{-}diff \ \hat{}\hat{}\ k)\ g\ x)$
⟨*proof*⟩

lemma *bw-diff-sum-pow* [*simp*]:
$(bw\text{-}diff \ \hat{}\hat{}\ k)\ (\lambda x.\ (\sum i{\in}I.\ f\ i\ x)) = (\lambda x.\ (\sum i{\in}I.\ (bw\text{-}diff \ \hat{}\hat{}\ k)\ (f\ i)\ x))$
⟨*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*:
$(bw\text{-}diff \ \hat{}\hat{}\ l)\ (\lambda x{::}int.\ (x + n)\ gchoose\ k) =$
$(if\ l \le k\ then\ (\lambda x.\ (x + n - int\ l)\ gchoose\ (k - l))\ else\ (\lambda\text{-}.\ 0))$
⟨*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 *poly-fun f* $\longleftrightarrow$ ($\exists$ *p*::*rat poly*. $\forall$ *a*. *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*))

lemma *poly-funI*: ($\bigwedge$*a*. *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)) $\Longrightarrow$ *poly-fun f*
⟨*proof*⟩

lemma *poly-funE*:
assumes *poly-fun f*
obtains *p* **where** $\bigwedge$*a*. *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
⟨*proof*⟩

lemma *poly-fun-eqI*:
assumes *poly-fun f* **and** *poly-fun g* **and** *infinite* {*a*. *f a* = *g a*}
shows *f* = *g*
⟨*proof*⟩

corollary *poly-fun-eqI-ge*:
assumes *poly-fun f* **and** *poly-fun g* **and** $\bigwedge$*a*. $b \le a \Longrightarrow f\ a = g\ a$
shows *f* = *g*
⟨*proof*⟩

corollary *poly-fun-eqI-gr*:
assumes *poly-fun f* **and** *poly-fun g* **and** $\bigwedge$*a*. $b < a \Longrightarrow f\ a = g\ a$
shows *f* = *g*
⟨*proof*⟩

6.2 Closure Properties

lemma *poly-fun-const* [*simp*]: *poly-fun* (λ-. *c*)
⟨*proof*⟩

lemma *poly-fun-id* [*simp*]: *poly-fun* (λx. *x*) *poly-fun id*
⟨*proof*⟩

lemma *poly-fun-uminus*:
assumes *poly-fun f*
shows *poly-fun* (λx. − *f x*) **and** *poly-fun* (− *f*)
⟨*proof*⟩

lemma *poly-fun-uminus-iff* [*simp*]:
poly-fun (λx. − *f x*) $\longleftrightarrow$ *poly-fun f* *poly-fun* (− *f*) $\longleftrightarrow$ *poly-fun f*
⟨*proof*⟩

lemma *poly-fun-plus* [*simp*]:
assumes *poly-fun f* **and** *poly-fun g*
shows *poly-fun* (λx. *f x* + *g x*)
⟨*proof*⟩

lemma *poly-fun-minus* [*simp*]:
assumes *poly-fun f* **and** *poly-fun g*

shows *poly-fun* (λ*x*. *f x* − *g x*)
⟨*proof*⟩

lemma *poly-fun-times* [*simp*]:
assumes *poly-fun f* **and** *poly-fun g*
shows *poly-fun* (λ*x*. *f x* ∗ *g x*)
⟨*proof*⟩

lemma *poly-fun-divide*:
assumes *poly-fun f* **and** ⋀*a*. *c dvd f a*
shows *poly-fun* (λ*x*. *f x div c*)
⟨*proof*⟩

lemma *poly-fun-pow* [*simp*]:
assumes *poly-fun f*
shows *poly-fun* (λ*x*. *f x* ^ *k*)
⟨*proof*⟩

lemma *poly-fun-comp*:
assumes *poly-fun f* **and** *poly-fun g*
shows *poly-fun* (λ*x*. *f* (*g x*)) **and** *poly-fun* (*f* ∘ *g*)
⟨*proof*⟩

lemma *poly-fun-sum* [*simp*]: (⋀*i*. *i* ∈ *I* ⟹ *poly-fun* (*f i*)) ⟹ *poly-fun* (λ*x*. (∑ *i*∈*I*. *f i x*))
⟨*proof*⟩

lemma *poly-fun-prod* [*simp*]: (⋀*i*. *i* ∈ *I* ⟹ *poly-fun* (*f i*)) ⟹ *poly-fun* (λ*x*. (∏ *i*∈*I*. *f i x*))
⟨*proof*⟩

lemma *poly-fun-pochhammer* [*simp*]: *poly-fun f* ⟹ *poly-fun* (λ*x*. *pochhammer* (*f x*) *k*)
⟨*proof*⟩

lemma *poly-fun-gbinomial* [*simp*]: *poly-fun f* ⟹ *poly-fun* (λ*x*. *f x 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* $A \longleftrightarrow (\forall p \in A.$ *is-monomial* $p)$

lemma *is-monomial-setI*: $(\bigwedge p.\ p \in A \Longrightarrow$ *is-monomial* $p) \Longrightarrow$ *is-monomial-set* A
⟨*proof*⟩

lemma *is-monomial-setD*: *is-monomial-set* $A \Longrightarrow p \in A \Longrightarrow$ *is-monomial* p
⟨*proof*⟩

lemma *is-monomial-set-subset*: *is-monomial-set* $B \Longrightarrow A \subseteq B \Longrightarrow$ *is-monomial-set* A
⟨*proof*⟩

lemma *is-monomial-set-Un*: *is-monomial-set* $(A \cup B) \longleftrightarrow$ (*is-monomial-set* $A \wedge$ *is-monomial-set* B)
⟨*proof*⟩

7.2 Modules

context *term-powerprod*
begin

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

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

end

context *ordered-term*
begin

lemma *keys-monomial-pmdl*:
assumes *is-monomial-set* F **and** $p \in$ *pmdl* F **and** $t \in$ *keys* p
obtains f **where** $f \in F$ **and** $f \neq 0$ **and** *lt* f $adds_t$ t
⟨*proof*⟩

lemma *image-lt-monomial-lt*: *lt* ‘ *monomial* (*1*::'b::*zero-neq-one*) ‘ *lt* ‘ F = *lt* ‘ F
⟨*proof*⟩

7.3 Reduction

lemma *red-setE2*:

assumes *red B p q*
obtains *b* **where** $b \in B$ **and** $b \neq 0$ **and** *red* {*b*} *p q*
⟨*proof*⟩

lemma *red-monomial-keys*:
assumes *is-monomial r* **and** *red* {*r*} *p q*
shows *card* (*keys p*) = *Suc* (*card* (*keys q*))
⟨*proof*⟩

lemma *red-monomial-monomial-setD*:
assumes *is-monomial p* **and** *is-monomial-set B* **and** *red B p q*
shows *q* = *0*
⟨*proof*⟩

corollary *is-red-monomial-monomial-setD*:
assumes *is-monomial p* **and** *is-monomial-set B* **and** *is-red B p*
shows *red B p 0*
⟨*proof*⟩

corollary *is-red-monomial-monomial-set-in-pmdl*:
is-monomial p ⟹ *is-monomial-set B* ⟹ *is-red B p* ⟹ *p* ∈ *pmdl B*
⟨*proof*⟩

corollary *red-rtrancl-monomial-monomial-set-cases*:
assumes *is-monomial p* **and** *is-monomial-set B* **and** (*red B*)** *p q*
obtains *q* = *p* | *q* = *0*
⟨*proof*⟩

lemma *is-red-monomial-lt*:
assumes *0* ∉ *B*
shows *is-red* (*monomial* (*1*::′*b*::*field*) ‘ *lt* ‘ *B*) = *is-red B*
⟨*proof*⟩

end

7.4 Gröbner Bases

context *gd-term*
begin

lemma *monomial-set-is-GB*:
assumes *is-monomial-set G*
shows *is-Groebner-basis G*
⟨*proof*⟩

context
fixes *d*
assumes *dgrad*: *dickson-grading* (*d*::′*a* ⇒ *nat*)
begin

context
  fixes *F m*
  assumes *fin-comps*: *finite* (*component-of-term* ‘ *Keys F*)
    and *F-sub*: *F* ⊆ *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* ⊆ *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* ⊆ *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* ⊆ *dgrad-p-set d m* **and** *0* ∉ *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* ⊆ *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 *card* $A \geq n$
  obtains B **where** *card* $B = n$ **and** $B \subseteq A$
  ⟨*proof*⟩

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

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

8.2 Sums

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

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

lemma *sum-split-nat-ivl*:
  $a \leq Suc\ j \Longrightarrow j \leq b \Longrightarrow sum\ f\ \{a..j\} + sum\ f\ \{Suc\ j..b\} = sum\ f\ \{a..b\}$
  ⟨*proof*⟩

8.3 *count-list*

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

8.4 *listset*

lemma *listset-Cons*: $listset\ (x\ \#\ xs) = (\bigcup y{\in}x.\ (\#)\ y\ `\ listset\ xs)$
  ⟨*proof*⟩

lemma *listset-ConsI*: $y \in x \Longrightarrow ys' \in listset\ xs \Longrightarrow ys = y\ \#\ ys' \Longrightarrow ys \in listset\ (x\ \#\ xs)$
  ⟨*proof*⟩

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

⟨proof⟩

lemma *listsetI*:
$length\ ys = length\ xs \Longrightarrow (\bigwedge i.\ i < length\ xs \Longrightarrow ys\ !\ i \in xs\ !\ i) \Longrightarrow ys \in listset\ xs$
⟨proof⟩

lemma *listsetD*:
assumes $ys \in listset\ xs$
shows $length\ ys = length\ xs$ **and** $\bigwedge i.\ i < length\ xs \Longrightarrow ys\ !\ i \in xs\ !\ i$
⟨proof⟩

lemma *listset-singletonI*: $a \in A \Longrightarrow ys = [a] \Longrightarrow ys \in listset\ [A]$
⟨proof⟩

lemma *listset-singletonE*:
assumes $ys \in listset\ [A]$
obtains a **where** $a \in A$ **and** $ys = [a]$
⟨proof⟩

lemma *listset-doubletonI*: $a \in A \Longrightarrow b \in B \Longrightarrow ys = [a, b] \Longrightarrow ys \in listset\ [A, B]$
⟨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*:
$ys1 \in listset\ xs1 \Longrightarrow ys2 \in listset\ xs2 \Longrightarrow ys = ys1\ @\ ys2 \Longrightarrow ys \in listset\ (xs1\ @\ xs2)$
⟨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*: $ys' \in listset\ xs \Longrightarrow ys = map\ f\ ys' \Longrightarrow ys \in listset\ (map\ ((`)\ f)\ xs)$
⟨proof⟩

lemma *listset-map-imageE*:
assumes $ys \in listset\ (map\ ((`)\ f)\ xs)$
obtains ys' **where** $ys' \in listset\ xs$ **and** $ys = map\ f\ ys'$
⟨proof⟩

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

lemma *listset-closed-map*:
assumes $ys \in listset\ xs$ **and** $\bigwedge x\ y.\ x \in set\ xs \Longrightarrow y \in x \Longrightarrow f\ y \in x$
shows $map\ f\ ys \in listset\ xs$
⟨*proof*⟩

lemma *listset-closed-map2*:
assumes $ys1 \in listset\ xs$ **and** $ys2 \in listset\ xs$
and $\bigwedge x\ y1\ y2.\ x \in set\ xs \Longrightarrow y1 \in x \Longrightarrow y2 \in x \Longrightarrow f\ y1\ y2 \in x$
shows $map2\ f\ ys1\ ys2 \in listset\ xs$
⟨*proof*⟩

lemma *listset-empty-iff*: $listset\ xs = \{\} \longleftrightarrow \{\} \in set\ xs$
⟨*proof*⟩

lemma *listset-mono*:
assumes $length\ xs = length\ ys$ **and** $\bigwedge i.\ i < length\ ys \Longrightarrow xs\ !\ i \subseteq ys\ !\ i$
shows $listset\ xs \subseteq listset\ ys$
⟨*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 set ⇒ 'a::comm-monoid-add set list ⇒ bool*
where $direct\text{-}decomp\ A\ ss \longleftrightarrow bij\text{-}betw\ sum\text{-}list\ (listset\ ss)\ A$

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

⟨proof⟩

lemma *direct-decompI-alt*:
$(\bigwedge qs.\ qs \in listset\ ss \Longrightarrow sum\text{-}list\ qs \in A) \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow \exists!qs{\in}listset\ ss.\ a = sum\text{-}list\ qs) \Longrightarrow$
$direct\text{-}decomp\ A\ ss$
⟨proof⟩

lemma *direct-decompD*:
assumes *direct-decomp A ss*
shows $qs \in listset\ ss \Longrightarrow sum\text{-}list\ qs \in A$ **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** $qs \in listset\ ss$ **and** $a = sum\text{-}list\ qs$
⟨proof⟩

lemma *direct-decomp-unique*:
$direct\text{-}decomp\ A\ ss \Longrightarrow qs \in listset\ ss \Longrightarrow qs' \in listset\ ss \Longrightarrow sum\text{-}list\ qs = sum\text{-}list\ qs' \Longrightarrow$
$qs = qs'$
⟨proof⟩

lemma *direct-decomp-singleton*: *direct-decomp A* [*A*]
⟨proof⟩

lemma *mset-bij*:
assumes *bij-betw f* {..<*length xs*} {..<*length ys*} **and** $\bigwedge i.\ i < length\ xs \Longrightarrow xs\ !\ i = ys\ !\ f\ i$
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*:
$direct\text{-}decomp\ A\ (map\ f\ ss) \Longrightarrow direct\text{-}decomp\ A\ (map\ f\ (filter\ P\ ss)\ @\ map\ f\ (filter\ (-\ P)\ ss))$
⟨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*::*semiring-0*) ∗ *sum-list xs*
⟨*proof*⟩

lemma *direct-decomp-image-times*:
assumes *direct-decomp* (*A*::′*a*::*semiring-0 set*) *ss* **and** ⋀*a b*. *x* ∗ *a* = *x* ∗ *b* ⟹ *x* ≠ *0* ⟹ *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 {} ∉ *set ss2* ⟹ *direct-decomp* (*sum-list* ‘ *listset ss1*) *ss1* (**is** - ⟹ *?thesis1*)
and {} ∉ *set ss1* ⟹ *direct-decomp* (*sum-list* ‘ *listset ss2*) *ss2* (**is** - ⟹ *?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** ⋀*s0*. *s0* ∈ *set ss* ⟹ *0* ∈ *s0*
shows *s* ⊆ *A*
⟨*proof*⟩

lemma *direct-decomp-Int-zero*:
assumes *direct-decomp A ss* **and** *i* < *j* **and** *j* < *length ss* **and** ⋀*s*. *s* ∈ *set ss* ⟹ *0* ∈ *s*
shows *ss* ! *i* ∩ *ss* ! *j* = {*0*}
⟨*proof*⟩

corollary *direct-decomp-pairwise-zero*:
assumes *direct-decomp A ss* **and** ⋀*s*. *s* ∈ *set ss* ⟹ *0* ∈ *s*
shows *pairwise* (λ*s1 s2*. *s1* ∩ *s2* = {*0*}) (*set ss*)
⟨*proof*⟩

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

assumes *direct-decomp A ss* **and** *1 < count-list ss X* **and** ⋀*s. s ∈ set ss ⟹ 0 ∈ s*
shows *X = {0}*
⟨*proof*⟩

corollary *direct-decomp-map-Int-zero*:
assumes *direct-decomp A (map f ss)* **and** *s1 ∈ set ss* **and** *s2 ∈ set ss* **and** *s1 ≠ s2*
and ⋀*s. s ∈ set ss ⟹ 0 ∈ f s*
shows *f s1 ∩ f s2 = {0}*
⟨*proof*⟩

9.2 Direct Decompositions and Vector Spaces

definition (**in** *vector-space*) *is-basis* :: *'b set ⇒ 'b set ⇒ bool*
where *is-basis V B ⟷ (B ⊆ V ∧ independent B ∧ V ⊆ span B ∧ card B = dim V)*

definition (**in** *vector-space*) *some-basis* :: *'b set ⇒ 'b set*
where *some-basis V = Eps (local.is-basis V)*

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

context *vector-space*
begin

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

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

lemma *independent-UnI*:
assumes *independent A* **and** *independent B* **and** *span A ∩ span B = {0}*
shows *independent (A ∪ B)*
⟨*proof*⟩

lemma *subspace-direct-decomp*:
assumes *direct-decomp A ss* **and** ⋀*s. s ∈ set ss ⟹ subspace s*
shows *subspace A*
⟨*proof*⟩

lemma *is-basis-alt*: *subspace V ⟹ is-basis V B ⟷ (independent B ∧ span B = V)*
⟨*proof*⟩

lemma *is-basis-finite*: *is-basis V A ⟹ is-basis V B ⟹ finite A ⟷ finite B*
⟨*proof*⟩

lemma *some-basis-is-basis*: *is-basis V (some-basis V)*
⟨*proof*⟩

corollary
shows *some-basis-subset*: *some-basis V* ⊆ *V*
and *independent-some-basis*: *independent (some-basis V)*
and *span-some-basis-supset*: *V* ⊆ *span (some-basis V)*
and *card-some-basis*: *card (some-basis V)* = *dim V*
⟨*proof*⟩

lemma *some-basis-not-zero*: *0* ∉ *some-basis V*
⟨*proof*⟩

lemma *span-some-basis*: *subspace V* ⟹ *span (some-basis V)* = *V*
⟨*proof*⟩

lemma *direct-decomp-some-basis-pairwise-disjnt*:
assumes *direct-decomp A ss* **and** ⋀*s. s* ∈ *set ss* ⟹ *subspace s*
shows *pairwise* (λ*s1 s2. disjnt (some-basis s1) (some-basis s2)) (set ss)*
⟨*proof*⟩

lemma *direct-decomp-span-some-basis*:
assumes *direct-decomp A ss* **and** ⋀*s. s* ∈ *set ss* ⟹ *subspace s*
shows *span* (⋃(*some-basis* ‘ *set ss*)) = *A*
⟨*proof*⟩

lemma *direct-decomp-independent-some-basis*:
assumes *direct-decomp A ss* **and** ⋀*s. s* ∈ *set ss* ⟹ *subspace s*
shows *independent* (⋃(*some-basis* ‘ *set ss*))
⟨*proof*⟩

corollary *direct-decomp-is-basis*:
assumes *direct-decomp A ss* **and** ⋀*s. s* ∈ *set ss* ⟹ *subspace s*
shows *is-basis A* (⋃(*some-basis* ‘ *set ss*))
⟨*proof*⟩

lemma *dim-direct-decomp*:
assumes *direct-decomp A ss* **and** *finite B* **and** *A* ⊆ *span B* **and** ⋀*s. s* ∈ *set ss* ⟹ *subspace s*
shows *dim A* = (∑ *s*∈*set ss. dim s*)
⟨*proof*⟩

end

9.3 Homogeneous Sets of Polynomials with Fixed Degree

lemma *homogeneous-set-direct-decomp*:
assumes *direct-decomp A ss* **and** ⋀*s. s* ∈ *set ss* ⟹ *homogeneous-set s*
shows *homogeneous-set A*

⟨*proof*⟩

definition *hom-deg-set* :: *nat* ⇒ (($'x$ ⇒$_0$ *nat*) ⇒$_0$ $'a$) *set* ⇒ (($'x$ ⇒$_0$ *nat*) ⇒$_0$ $'a$::*zero*) *set*
  where *hom-deg-set z A* = (λ*a*. *hom-component a z*) ' *A*

lemma *hom-deg-setD*:
  assumes *p* ∈ *hom-deg-set z A*
  shows *homogeneous p* **and** *p* ≠ *0* ⟹ *poly-deg p* = *z*
⟨*proof*⟩

lemma *zero-in-hom-deg-set*:
  assumes *0* ∈ *A*
  shows *0* ∈ *hom-deg-set z A*
⟨*proof*⟩

lemma *hom-deg-set-closed-uminus*:
  assumes ⋀*a*. *a* ∈ *A* ⟹ − *a* ∈ *A* **and** *p* ∈ *hom-deg-set z A*
  shows − *p* ∈ *hom-deg-set z A*
⟨*proof*⟩

lemma *hom-deg-set-closed-plus*:
  assumes ⋀*a1 a2*. *a1* ∈ *A* ⟹ *a2* ∈ *A* ⟹ *a1* + *a2* ∈ *A*
    and *p* ∈ *hom-deg-set z A* **and** *q* ∈ *hom-deg-set z A*
  shows *p* + *q* ∈ *hom-deg-set z A*
⟨*proof*⟩

lemma *hom-deg-set-closed-minus*:
  assumes ⋀*a1 a2*. *a1* ∈ *A* ⟹ *a2* ∈ *A* ⟹ *a1* − *a2* ∈ *A*
    and *p* ∈ *hom-deg-set z A* **and** *q* ∈ *hom-deg-set z A*
  shows *p* − *q* ∈ *hom-deg-set z A*
⟨*proof*⟩

lemma *hom-deg-set-closed-scalar*:
  assumes ⋀*a*. *a* ∈ *A* ⟹ *c* · *a* ∈ *A* **and** *p* ∈ *hom-deg-set z A*
  shows (*c*::$'a$::*semiring-0*) · *p* ∈ *hom-deg-set z A*
⟨*proof*⟩

lemma *hom-deg-set-closed-sum*:
  assumes *0* ∈ *A* **and** ⋀*a1 a2*. *a1* ∈ *A* ⟹ *a2* ∈ *A* ⟹ *a1* + *a2* ∈ *A*
    and ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *hom-deg-set z A*
  shows *sum f I* ∈ *hom-deg-set z A*
  ⟨*proof*⟩

lemma *hom-deg-set-subset*: *homogeneous-set A* ⟹ *hom-deg-set z A* ⊆ *A*
  ⟨*proof*⟩

lemma *Polys-closed-hom-deg-set*:
  assumes *A* ⊆ *P*[*X*]

shows *hom-deg-set z A* $\subseteq$ *P*[*X*]
⟨*proof*⟩

lemma *hom-deg-set-alt-homogeneous-set*:
assumes *homogeneous-set A*
shows *hom-deg-set z A* = {*p* ∈ *A*. *homogeneous p* ∧ (*p* = *0* ∨ *poly-deg p* = *z*)}
(**is** *?A* = *?B*)
⟨*proof*⟩

lemma *hom-deg-set-sum-list-listset*:
assumes *A* = *sum-list* ‘ *listset ss*
shows *hom-deg-set z A* = *sum-list* ‘ *listset* (*map* (*hom-deg-set z*) *ss*) (**is** *?A* = *?B*)
⟨*proof*⟩

lemma *direct-decomp-hom-deg-set*:
assumes *direct-decomp A ss* **and** ⋀*s*. *s* ∈ *set ss* ⟹ *homogeneous-set s*
shows *direct-decomp* (*hom-deg-set z A*) (*map* (*hom-deg-set z*) *ss*)
⟨*proof*⟩

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*: *phull.subspace* (*ideal* (*F*::('*b*::*comm-powerprod* $\Rightarrow_0$ '*a*::*field*) *set*))
⟨*proof*⟩

lemma *subspace-Polys*: *phull.subspace* (*P*[*X*]::(('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*::*field*) *set*)
⟨*proof*⟩

lemma *subspace-hom-deg-set*:
assumes *phull.subspace A*
shows *phull.subspace* (*hom-deg-set z A*) (**is** *phull.subspace ?A*)
⟨*proof*⟩

lemma *hom-deg-set-Polys-eq-span*:
hom-deg-set z P[*X*] = *phull.span* (*monomial* (*1*::'*a*::*field*) ‘ *deg-sect X z*) (**is** *?A* = *?B*)
⟨*proof*⟩

9.5 (Projective) Hilbert Function

interpretation *phull*: *vector-space map-scale*
⟨*proof*⟩

definition *Hilbert-fun* :: (('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*::*field*) *set* ⇒ *nat* ⇒ *nat*
where *Hilbert-fun A z* = *phull.dim* (*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* ⊆ *P*[*X*] **and** *direct-decomp* (*A*::((′*x*::*countable* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::*field*) *set*) *ps*
and ⋀*s. s* ∈ *set ps* ⟹ *homogeneous-set s* **and** ⋀*s. s* ∈ *set ps* ⟹ *phull.subspace s*
shows *Hilbert-fun A z* = (∑ *p*∈*set ps. Hilbert-fun p z*)
⟨*proof*⟩

context *pm-powerprod*
begin

lemma *image-lt-hom-deg-set*:
assumes *homogeneous-set A*
shows *lpp* ‘ (*hom-deg-set z A* − {*0*}) = {*t* ∈ *lpp* ‘ (*A* − {*0*}). *deg-pm t* = *z*} (**is** *?B* = *?A*)
⟨*proof*⟩

lemma *Hilbert-fun-alt*:
assumes *finite X* **and** *A* ⊆ *P*[*X*] **and** *phull.subspace A*
shows *Hilbert-fun A z* = *card* (*lpp* ‘ (*hom-deg-set z A* − {*0*})) (**is** - = *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* ⇒ *'a* ⇒ *'a::semigroup-mult set* (**infixl** ÷ *55*)
where *quot-set A x* = (∗) *x* −' *A*

lemma *quot-set-iff*: $a \in A \div x \longleftrightarrow x * a \in A$
⟨*proof*⟩

lemma *quot-setI*: $x * a \in A \Longrightarrow a \in A \div x$
⟨*proof*⟩

lemma *quot-setD*: $a \in A \div x \Longrightarrow x * a \in A$
⟨*proof*⟩

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

lemma *quot-set-one* [*simp*]: *A* ÷ (*1*::-::*monoid-mult*) = *A*
⟨*proof*⟩

lemma *ideal-quot-set-ideal* [*simp*]: *ideal* (*ideal B* ÷ *x*) = (*ideal B*) ÷ (*x*::-::*comm-ring*)
⟨*proof*⟩

lemma *quot-set-image-times*: *inj* ((∗) *x*) ⟹ ((∗) *x* ' *A*) ÷ *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* ⇒ (($'x \Rightarrow_0 nat$) $\Rightarrow_0$ *'a::field*)
⇒ (($'x \Rightarrow_0 nat$) $\Rightarrow_0$ *'a::field*)
where *normal-form F p* = (*SOME q*. (*punit.red* (*punit.reduced-GB F*))** *p q* ∧
¬ *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 ¬ *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** ¬ *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* ⟷ (¬ *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* · *p*) = *c* · (*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* ∩ $P[X]$) = *normal-form F*
⟨*proof*⟩

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

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

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

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

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

end

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

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

end

10.4 Basic Cone Decompositions

definition *cone* :: $((('x \Rightarrow_0 nat) \Rightarrow_0 'a) \times 'x\ set) \Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0 'a{::}comm\text{-}semiring\text{-}0)\ set$
where $\textit{cone } hU = (*)\ (\textit{fst } hU)\ \text{‘}\ P[\textit{snd } hU]$

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

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

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

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

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

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

lemma *zero-in-cone*: $0 \in cone\ hU$
⟨*proof*⟩

corollary *empty-not-in-map-cone*: $\{\} \notin set\ (map\ cone\ ps)$
⟨*proof*⟩

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

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

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

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

lemma *cone-closed-times*:
assumes $a \in cone\ (h,\ U)$ **and** $q \in P[U]$
shows $q * a \in cone\ (h,\ U)$
⟨*proof*⟩

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

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

⟨proof⟩

lemma *cone-mono-1*:
  assumes $h' \in P[U]$
  shows $cone\ (h' * h,\ U) \subseteq cone\ (h,\ U)$
⟨proof⟩

lemma *cone-mono-2*:
  assumes $U1 \subseteq U2$
  shows $cone\ (h,\ U1) \subseteq cone\ (h,\ U2)$
⟨proof⟩

lemma *cone-subsetD*:
  assumes $cone\ (h1,\ U1) \subseteq cone\ (h2{::}\text{-} \Rightarrow_0 {'a}{::}\{comm\text{-}ring\text{-}1, ring\text{-}no\text{-}zero\text{-}divisors\},$ $U2)$
  shows $h2\ dvd\ h1$ **and** $h1 \neq 0 \Longrightarrow U1 \subseteq U2$
⟨proof⟩

lemma *cone-subset-PolysD*:
  assumes $cone\ (h{::}\text{-} \Rightarrow_0 {'a}{::}\{comm\text{-}semiring\text{-}1, semiring\text{-}no\text{-}zero\text{-}divisors\},\ U) \subseteq$ $P[X]$
  shows $h \in P[X]$ **and** $h \neq 0 \Longrightarrow U \subseteq X$
⟨proof⟩

lemma *cone-subset-PolysI*:
  assumes $h \in P[X]$ **and** $h \neq 0 \Longrightarrow U \subseteq X$
  shows $cone\ (h,\ U) \subseteq P[X]$
⟨proof⟩

lemma *cone-image-times*: $(*)\ a\ `\ cone\ (h,\ U) = cone\ (a * h,\ U)$
  ⟨proof⟩

lemma *cone-image-times'*: $(*)\ a\ `\ cone\ hU = cone\ (apfst\ ((*)\ a)\ hU)$
⟨proof⟩

lemma *homogeneous-set-coneI*:
  assumes *homogeneous h*
  shows *homogeneous-set* (*cone* (*h*, *U*))
⟨proof⟩

lemma *subspace-cone*: *phull.subspace* (*cone hU*)
  ⟨proof⟩

lemma *direct-decomp-cone-insert*:
  fixes $h :: \text{-} \Rightarrow_0 {'a}{::}\{comm\text{-}ring\text{-}1, ring\text{-}no\text{-}zero\text{-}divisors\}$
  assumes $x \notin U$
  shows *direct-decomp* (*cone* (*h*, *insert x U*))
    [*cone* (*h*, *U*), *cone* (*monomial 1* (*Poly-Mapping.single x* (*Suc 0*)) $*$ *h*, *insert x U*)]

⟨*proof*⟩

definition *valid-decomp* :: $'x$ *set* ⇒ (((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::*zero*) × ′*x set*) *list* ⇒ *bool*
where *valid-decomp X ps* ⟷ ((∀(*h*, *U*)∈*set ps*. *h* ∈ *P*[*X*] ∧ *h* ≠ *0* ∧ *U* ⊆ *X*))

definition *monomial-decomp* :: (((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::{*one*,*zero*}) × ′*x set*) *list* ⇒ *bool*
where *monomial-decomp ps* ⟷ (∀ *hU*∈*set ps*. *is-monomial* (*fst hU*) ∧ *punit.lc* (*fst hU*) = *1*)

definition *hom-decomp* :: (((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::{*one*,*zero*}) × ′*x set*) *list* ⇒ *bool*
where *hom-decomp ps* ⟷ (∀ *hU*∈*set ps*. *homogeneous* (*fst hU*))

definition *cone-decomp* :: ((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*) *set* ⇒
(((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::*comm-semiring-0*) × ′*x set*) *list* ⇒ *bool*
where *cone-decomp T ps* ⟷ *direct-decomp T* (*map cone ps*)

lemma *valid-decompI*:
(⋀*h U*. (*h*, *U*) ∈ *set ps* ⟹ *h* ∈ *P*[*X*]) ⟹ (⋀*h U*. (*h*, *U*) ∈ *set ps* ⟹ *h* ≠ *0*) ⟹
(⋀*h U*. (*h*, *U*) ∈ *set ps* ⟹ *U* ⊆ *X*) ⟹ *valid-decomp X ps*
⟨*proof*⟩

lemma *valid-decompD*:
assumes *valid-decomp X ps* **and** (*h*, *U*) ∈ *set ps*
shows *h* ∈ *P*[*X*] **and** *h* ≠ *0* **and** *U* ⊆ *X*
⟨*proof*⟩

lemma *valid-decompD-finite*:
assumes *finite X* **and** *valid-decomp X ps* **and** (*h*, *U*) ∈ *set ps*
shows *finite U*
⟨*proof*⟩

lemma *valid-decomp-Nil*: *valid-decomp X* []
⟨*proof*⟩

lemma *valid-decomp-concat*:
assumes ⋀*ps*. *ps* ∈ *set pss* ⟹ *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* ∈ *P*[*X*] **and** *s* ≠ (*0*::- $\Rightarrow_0$ -::*semiring-no-zero-divisors*)

shows *valid-decomp X* (*map* (*apfst* ((∗) *s*)) *ps*)
⟨*proof*⟩

lemma *monomial-decompI*:
(⋀*h U*. (*h*, *U*) ∈ *set ps* ⟹ *is-monomial h*) ⟹ (⋀*h U*. (*h*, *U*) ∈ *set ps* ⟹ *punit.lc h* = *1*) ⟹
monomial-decomp ps
⟨*proof*⟩

lemma *monomial-decompD*:
assumes *monomial-decomp ps* **and** (*h*, *U*) ∈ *set ps*
shows *is-monomial h* **and** *punit.lc h* = *1*
⟨*proof*⟩

lemma *monomial-decomp-append-iff*:
monomial-decomp (*ps* @ *qs*) ⟷ *monomial-decomp ps* ∧ *monomial-decomp qs*
⟨*proof*⟩

lemma *monomial-decomp-concat*:
(⋀*ps*. *ps* ∈ *set pss* ⟹ *monomial-decomp ps*) ⟹ *monomial-decomp* (*concat pss*)
⟨*proof*⟩

lemma *monomial-decomp-map-times*:
assumes *monomial-decomp ps* **and** *is-monomial f* **and** *punit.lc f* = (*1*::'*a*::*semiring-1*)
shows *monomial-decomp* (*map* (*apfst* ((∗) *f*)) *ps*)
⟨*proof*⟩

lemma *monomial-decomp-monomial-in-cone*:
assumes *monomial-decomp ps* **and** *hU* ∈ *set ps* **and** *a* ∈ *cone hU*
shows *monomial* (*lookup a t*) *t* ∈ *cone hU*
⟨*proof*⟩

lemma *monomial-decomp-sum-list-monomial-in-cone*:
assumes *monomial-decomp ps* **and** *a* ∈ *sum-list* ‘ *listset* (*map cone ps*) **and** *t* ∈ *keys a*
obtains *c h U* **where** (*h*, *U*) ∈ *set ps* **and** *c* ≠ *0* **and** *monomial c t* ∈ *cone* (*h*, *U*)
⟨*proof*⟩

lemma *hom-decompI*: (⋀*h U*. (*h*, *U*) ∈ *set ps* ⟹ *homogeneous h*) ⟹ *hom-decomp ps*
⟨*proof*⟩

lemma *hom-decompD*: *hom-decomp ps* ⟹ (*h*, *U*) ∈ *set ps* ⟹ *homogeneous h*
⟨*proof*⟩

lemma *hom-decomp-append-iff*: *hom-decomp* (*ps* @ *qs*) ⟷ *hom-decomp ps* ∧ *hom-decomp qs*
⟨*proof*⟩

lemma *hom-decomp-concat*: $(\bigwedge ps.\ ps \in set\ pss \Longrightarrow hom\text{-}decomp\ ps) \Longrightarrow hom\text{-}decomp$ $(concat\ pss)$
⟨*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*) $\Longrightarrow$ *cone-decomp T ps*
⟨*proof*⟩

lemma *cone-decompD*: *cone-decomp T ps* $\Longrightarrow$ *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*}) $\Longrightarrow U \subseteq X$
⟨*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 $\bigwedge i.\ i < length\ ss \Longrightarrow cone\text{-}decomp\ (ss\ !\ i)\ (pss\ !\ i)$
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*) × '*x set*) *list* ⇒ ((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*) × '*x set*) *list*
((-$_+$) [*1000*] *999*)
where *pos-decomp ps* = *filter* (λ*p*. *snd p* ≠ {}) *ps*

definition *standard-decomp* :: *nat* ⇒ ((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*::*zero*) × '*x set*) *list* ⇒ *bool*

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

lemma *pos-decomp-Nil* [*simp*]: $[]_+ = []$
⟨*proof*⟩

lemma *pos-decomp-subset*: $set\ (ps_+) \subseteq set\ ps$
⟨*proof*⟩

lemma *pos-decomp-append*: $(ps\ @\ qs)_+ = ps_+\ @\ qs_+$
⟨*proof*⟩

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

lemma *pos-decomp-map*: $(map\ (apfst\ f)\ ps)_+ = map\ (apfst\ f)\ (ps_+)$
⟨*proof*⟩

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

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

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

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

lemma *standard-decomp-Nil*: $ps_+ = [] \Longrightarrow standard\text{-}decomp\ k\ ps$
⟨*proof*⟩

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

lemma *standard-decomp-concat*:

assumes $\bigwedge ps.\ ps \in set\ pss \Longrightarrow standard\text{-}decomp\ k\ ps$
shows *standard-decomp k (concat pss)*
⟨*proof*⟩

corollary *standard-decomp-append*:
assumes *standard-decomp k ps* **and** *standard-decomp k qs*
shows *standard-decomp k (ps @ qs)*
⟨*proof*⟩

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

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

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

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

10.5 Splitting w.r.t. Ideals

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

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

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

lemma *splits-wrtI*:
assumes *cone-decomp T* (*ps* @ *qs*)
and $\bigwedge$*h U*. (*h*, *U*) $\in$ *set ps* $\Longrightarrow$ *cone* (*h*, *U*) $\subseteq$ *P*[*X*] **and** $\bigwedge$*h U*. (*h*, *U*) $\in$ *set ps* $\Longrightarrow$ *h* $\in$ *ideal F*
and $\bigwedge$*h U*. (*h*, *U*) $\in$ *set qs* $\Longrightarrow$ *cone* (*h*, *U*) $\subseteq$ *P*[*X*]
and $\bigwedge$*h U a*. (*h*, *U*) $\in$ *set qs* $\Longrightarrow$ *a* $\in$ *cone* (*h*, *U*) $\Longrightarrow$ *a* $\in$ *ideal F* $\Longrightarrow$ *a* = *0*
shows *splits-wrt* (*ps*, *qs*) *T F*
⟨*proof*⟩

lemma *splits-wrtI-valid-decomp*:
assumes *valid-decomp X ps* **and** *valid-decomp X qs* **and** *cone-decomp T* (*ps* @ *qs*)
and $\bigwedge$*h U*. (*h*, *U*) $\in$ *set ps* $\Longrightarrow$ *h* $\in$ *ideal F*
and $\bigwedge$*h U a*. (*h*, *U*) $\in$ *set qs* $\Longrightarrow$ *a* $\in$ *cone* (*h*, *U*) $\Longrightarrow$ *a* $\in$ *ideal F* $\Longrightarrow$ *a* = *0*
shows *splits-wrt* (*ps*, *qs*) *T F*
⟨*proof*⟩

lemma *splits-wrtD*:
assumes *splits-wrt* (*ps*, *qs*) *T F*
shows *cone-decomp T* (*ps* @ *qs*) **and** *hU* $\in$ *set ps* $\Longrightarrow$ *cone hU* $\subseteq$ *ideal F* $\cap$ *P*[*X*]
and *hU* $\in$ *set qs* $\Longrightarrow$ *cone hU* $\subseteq$ *P*[*X*] **and** *hU* $\in$ *set qs* $\Longrightarrow$ *cone hU* $\cap$ *ideal F* = {*0*}
⟨*proof*⟩

lemma *splits-wrt-image-sum-list-fst-subset*:
assumes *splits-wrt* (*ps*, *qs*) *T F*
shows *sum-list* ‘ *listset* (*map cone ps*) $\subseteq$ *ideal F* $\cap$ *P*[*X*]
⟨*proof*⟩

lemma *splits-wrt-image-sum-list-snd-subset*:
assumes *splits-wrt* (*ps*, *qs*) *T F*
shows *sum-list* ‘ *listset* (*map cone qs*) $\subseteq$ *P*[*X*]
⟨*proof*⟩

lemma *splits-wrt-cone-decomp-1*:
assumes *splits-wrt* (*ps*, *qs*) *T F* **and** *monomial-decomp qs* **and** *is-monomial-set* (*F*::(- $\Rightarrow_0$ '*a*::*field*) *set*)
— The last two assumptions are missing in the paper.
shows *cone-decomp* (*T* $\cap$ *ideal F*) *ps*
⟨*proof*⟩

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* $\cap$ *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* $(T \cap \textit{normal-form } F \text{ ‘ } P[X])$ *qs*
⟨*proof*⟩

lemma *quot-monomial-ideal-monomial*:
$\textit{ideal } (\textit{monomial } 1 \text{ ‘ } S) \div \textit{monomial } 1 \ (\textit{Poly-Mapping.single } (x::'x) \ (1::nat)) = \textit{ideal } (\textit{monomial } (1::'a::\textit{comm-ring-1}) \text{ ‘ } (\lambda s.\ s - \textit{Poly-Mapping.single } x\ 1) \text{ ‘ } S)$
⟨*proof*⟩

lemma *lem-4-2-1*:
assumes $\textit{ideal } F \div \textit{monomial } 1\ t = \textit{ideal } (\textit{monomial } (1::'a::\textit{comm-ring-1}) \text{ ‘ } S)$
shows $\textit{cone } (\textit{monomial } 1\ t,\ U) \subseteq \textit{ideal } F \longleftrightarrow 0 \in S$
⟨*proof*⟩

lemma *lem-4-2-2*:
assumes $\textit{ideal } F \div \textit{monomial } 1\ t = \textit{ideal } (\textit{monomial } (1::'a::\textit{comm-ring-1}) \text{ ‘ } S)$
shows $\textit{cone } (\textit{monomial } 1\ t,\ U) \cap \textit{ideal } F = \{0\} \longleftrightarrow S \cap .[U] = \{\}$
⟨*proof*⟩

10.6 Function *split*

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

lemma *max-subset*:
assumes *finite A* **and** $B \subseteq A$ **and** *P B*
shows $\textit{max-subset } A\ P \subseteq A$ (**is** *?thesis1*)
and *P (max-subset A P)* (**is** *?thesis2*)
and $\textit{card } B \le \textit{card } (\textit{max-subset } A\ P)$ (**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 $x = \textit{SOME } x'.\ x' \in U - (\textit{max-subset } U\ (\lambda V.\ S \cap .[V] = \{\}))$;
$(ps0,\ qs0) = \textit{split } t\ (U - \{x\})\ S$;
$(ps1,\ qs1) = \textit{split } (\textit{Poly-Mapping.single } x\ 1 + t)\ U\ ((\lambda f.\ f - \textit{Poly-Mapping.single } x\ 1) \text{ ‘ } S)$ *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* $\Rightarrow_0$ *nat*) ⇒ -
assumes *finite X* **and** *U* ⊆ *X* **and** *finite S*
assumes ⋀*t U S*. *U* ⊆ *X* ⟹ *finite S* ⟹ *0* ∈ *S* ⟹ *P t U S* ([(*monomial* (*1*::′*a*::{*zero*,*one*}) *t*, *U*)], [])
assumes ⋀*t U S*. *U* ⊆ *X* ⟹ *finite S* ⟹ *0* ∉ *S* ⟹ *S* ∩ .[*U*] = {} ⟹ *P t U S* ([], [(*monomial 1 t*, *U*)])
assumes ⋀*t U S V x ps0 ps1 qs0 qs1*. *U* ⊆ *X* ⟹ *finite S* ⟹ *0* ∉ *S* ⟹ *S* ∩ .[*U*] ≠ {} ⟹ *V* ⊆ *U* ⟹
S ∩ .[*V*] = {} ⟹ (⋀*V*′. *V*′ ⊆ *U* ⟹ *S* ∩ .[*V*′] = {} ⟹ *card V*′ ≤ *card V*) ⟹
x ∈ *U* ⟹ *x* ∉ *V* ⟹ *V* = *max-subset U* (λ*V*′. *S* ∩ .[*V*′] = {}) ⟹ *x* = (*SOME x*′. *x*′ ∈ *U* − *V*) ⟹
(*ps0*, *qs0*) = *split t* (*U* − {*x*}) *S* ⟹
(*ps1*, *qs1*) = *split* (*Poly-Mapping.single x 1* + *t*) *U* ((λ*f*. *f* − *Poly-Mapping.single x 1*) ‘ *S*) ⟹
split t U S = (*ps0* @ *ps1*, *qs0* @ *qs1*) ⟹
P t (*U* − {*x*}) *S* (*ps0*, *qs0*) ⟹
P (*Poly-Mapping.single x 1* + *t*) *U* ((λ*f*. *f* − *Poly-Mapping.single x 1*) ‘ *S*) (*ps1*, *qs1*) ⟹
P t U S (*ps0* @ *ps1*, *qs0* @ *qs1*)
shows *P t U S* (*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*)::(- × (((- $\Rightarrow_0$ ′*a*::*zero-neq-one*) × -) *list*))))
and *valid-decomp X* (*snd* ((*split t U S*)::(- × (((- $\Rightarrow_0$ ′*a*::*zero-neq-one*) × -) *list*))))
(**is** *valid-decomp* - (*snd ?s*))

⟨*proof*⟩

lemma *monomial-decomp-split*:
assumes *finite X* **and** *U* ⊆ *X* **and** *finite S*
shows *monomial-decomp* (*fst* ((*split t U S*)::(- × (((- ⇒$_0$ 'a::zero-neq-one) × -) *list*))))
and *monomial-decomp* (*snd* ((*split t U S*)::(- × (((- ⇒$_0$ 'a::zero-neq-one) × -) *list*))))
(**is** *monomial-decomp* (*snd ?s*))
⟨*proof*⟩

lemma *split-splits-wrt*:
assumes *finite X* **and** *U* ⊆ *X* **and** *finite S* **and** *t* ∈ .[*X*]
and *ideal F* ÷ *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* ⊆ *X* **and** *t* ∈ .[*X*] **and** *F* ⊆ *P*[*X*]
and *ideal F* ÷ *monomial 1 t* = *ideal* (*monomial* (*1*::'*a*) ' *S*)
and *cone* (*monomial* (*1*::'*a*::*field*) *t*', *V*) ⊆ *cone* (*monomial 1 t*, *U*) ∩ *normal-form F* ' *P*[*X*]
shows *V* ⊆ *U* **and** *S* ∩ .[*V*] = {}
⟨*proof*⟩

lemma *lem-4-6*:
assumes *finite X* **and** *U* ⊆ *X* **and** *finite S* **and** *t* ∈ .[*X*] **and** *F* ⊆ *P*[*X*]
and *ideal F* ÷ *monomial 1 t* = *ideal* (*monomial 1* ' *S*)
assumes *cone* (*monomial 1 t*', *V*) ⊆ *cone* (*monomial 1 t*, *U*) ∩ *normal-form F* ' *P*[*X*]
obtains *V*' **where** (*monomial 1 t*, *V*') ∈ *set* (*snd* (*split t U S*)) **and** *card V* ≤ *card V*'
⟨*proof*⟩

lemma *lem-4-7*:
assumes *finite X* **and** *S* ⊆ .[*X*] **and** *g* ∈ *punit.reduced-GB* (*monomial* (*1*::'*a*) ' *S*)
and *cone-decomp* (*P*[*X*] ∩ *ideal* (*monomial* (*1*::'*a*::*field*) ' *S*)) *ps*
and *monomial-decomp ps*
obtains *U* **where** (*g*, *U*) ∈ *set ps*
⟨*proof*⟩

lemma *snd-splitI*:
assumes *finite X* **and** *U* ⊆ *X* **and** *finite S* **and** *0* ∉ *S*
obtains *V* **where** *V* ⊆ *U* **and** (*monomial 1 t*, *V*) ∈ *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*) ∈ *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*) ∈ *set* (*fst* (*split* ($t' + t$) *V* ((λ*s*. *s* − *t'*) ‘ *S*)))
and *set* (*snd* (*split* ($t' + t$) *V* ((λ*s*. *s* − *t'*) ‘ *S*))) ⊆ (*set* (*snd* (*split t U S*)) :: ((- $\Rightarrow_0$ *'a*) × -) *set*)
⟨*proof*⟩

lemma *lem-4-8*:
assumes *finite X* **and** *finite S* **and** $S \subseteq .[X]$ **and** $0 \notin S$
and *g* ∈ *punit.reduced-GB* (*monomial* (*1*::*'a*) ‘ *S*)
obtains *t U* **where** $U \subseteq X$ **and** (*monomial* (*1*::*'a*::*field*) *t*, *U*) ∈ *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* ∈ *punit.reduced-GB* (*monomial* (*1*::*'a*::*field*) ‘ *S*)
shows *poly-deg g* ≤ *Suc* (*Max* (*poly-deg* ‘ *fst* ‘ (*set* (*snd* (*split 0 X S*)) :: ((- $\Rightarrow_0$ *'a*) × -) *set*)))
(**is** - ≤ *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*) × -) *list*)
⟨*proof*⟩

theorem *standard-cone-decomp-snd-split*:
fixes *F*
defines *G* ≡ *punit.reduced-GB F*
defines *ss* ≡ (*split 0 X* (*lpp* ‘ *G*)) :: ((- $\Rightarrow_0$ *'a*::*field*) × -) *list* × -
defines *d* ≡ *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 (⋀*f*. *f* ∈ *F* ⟹ *homogeneous f*) ⟹ *g* ∈ *G* ⟹ *poly-deg g* ≤ *d*
⟨*proof*⟩

10.7 Splitting Ideals

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

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

context
assumes *fin-X*: *finite* X
begin

lemma *ideal-decomp-aux*:
assumes *finite* F **and** $F \subseteq P[X]$ **and** $f \in P[X]$
shows *fst* (*ideal-decomp-aux* F f) $\subseteq$ *ideal* $\{f\}$ (**is** *?thesis1*)
and *ideal* $F \cap$ *fst* (*ideal-decomp-aux* F f) = $\{0\}$ (**is** *?thesis2*)
and *direct-decomp* (*ideal* (*insert* f F) $\cap P[X]$) [*fst* (*ideal-decomp-aux* F f), *ideal* $F \cap P[X]$] (**is** *?thesis3*)
and *cone-decomp* (*fst* (*ideal-decomp-aux* F f)) (*snd* (*ideal-decomp-aux* F f)) (**is** *?thesis4*)
and $f \neq 0 \Longrightarrow$ *valid-decomp* X (*snd* (*ideal-decomp-aux* F f)) (**is** - $\Longrightarrow$ *?thesis5*)
and $f \neq 0 \Longrightarrow$ *standard-decomp* (*poly-deg* f) (*snd* (*ideal-decomp-aux* F f)) (**is** - $\Longrightarrow$ *?thesis6*)
and *homogeneous* $f \Longrightarrow$ *hom-decomp* (*snd* (*ideal-decomp-aux* F f)) (**is** - $\Longrightarrow$ *?thesis7*)
⟨*proof*⟩

lemma *ideal-decompE*:
fixes *f0* :: - $\Rightarrow_0$ *'a::field*
assumes *finite* F **and** $F \subseteq P[X]$ **and** *f0* $\in P[X]$ **and** $\bigwedge f.\ f \in F \Longrightarrow$ *poly-deg* f $\leq$ *poly-deg* *f0*
obtains T *ps* **where** *valid-decomp* X *ps* **and** *standard-decomp* (*poly-deg* *f0*) *ps* **and** *cone-decomp* T *ps*
and ($\bigwedge f.\ f \in F \Longrightarrow$ *homogeneous* f) $\Longrightarrow$ *hom-decomp* *ps*
and *direct-decomp* (*ideal* (*insert* *f0* F) $\cap P[X]$) [*ideal* {*f0*} $\cap P[X]$, T]
⟨*proof*⟩

10.8 Exact Cone Decompositions

definition *exact-decomp* :: *nat* ⇒ (((*'x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'a::zero*) × *'x set*) *list* ⇒ *bool*
where *exact-decomp* m *ps* ⟷ (∀ (h, U)∈*set ps*. $h \in P[X] \wedge U \subseteq X$) ∧
(∀ (h, U)∈*set ps*. ∀ (h', U')∈*set ps*. *poly-deg* h = *poly-deg* h' ⟶
m < *card* U ⟶ m < *card* U' ⟶ $(h, U) = (h', U')$)

lemma *exact-decompI*:
($\bigwedge h\ U.\ (h, U) \in$ *set ps* $\Longrightarrow h \in P[X]$) $\Longrightarrow$ ($\bigwedge h\ U.\ (h, U) \in$ *set ps* $\Longrightarrow U \subseteq X$) $\Longrightarrow$
($\bigwedge h\ h'\ U\ U'.\ (h, U) \in$ *set ps* $\Longrightarrow (h', U') \in$ *set ps* $\Longrightarrow$ *poly-deg* h = *poly-deg* h' $\Longrightarrow$
m < *card* $U \Longrightarrow m$ < *card* $U' \Longrightarrow (h, U) = (h', U')$) $\Longrightarrow$
exact-decomp m *ps*

⟨*proof*⟩

lemma *exact-decompD*:
assumes *exact-decomp m ps* **and** $(h,\ U) \in$ *set ps*
shows $h \in P[X]$ **and** $U \subseteq X$
and $(h',\ U') \in$ *set ps* $\Longrightarrow$ *poly-deg* $h =$ *poly-deg* $h' \Longrightarrow m <$ *card* $U \Longrightarrow m <$ *card* $U' \Longrightarrow$
$(h,\ U) = (h',\ U')$
⟨*proof*⟩

lemma *exact-decompI-zero*:
assumes $\bigwedge h\ U.\ (h,\ U) \in$ *set ps* $\Longrightarrow h \in P[X]$ **and** $\bigwedge h\ U.\ (h,\ U) \in$ *set ps* $\Longrightarrow U \subseteq X$
and $\bigwedge h\ h'\ U\ U'.\ (h,\ U) \in$ *set* $(ps_+) \Longrightarrow (h',\ U') \in$ *set* $(ps_+) \Longrightarrow$ *poly-deg* $h =$ *poly-deg* $h' \Longrightarrow$
$(h,\ U) = (h',\ U')$
shows *exact-decomp 0 ps*
⟨*proof*⟩

lemma *exact-decompD-zero*:
assumes *exact-decomp 0 ps* **and** $(h,\ U) \in$ *set* (ps_+) **and** $(h',\ U') \in$ *set* (ps_+)
and *poly-deg* $h =$ *poly-deg* h'
shows $(h,\ U) = (h',\ U')$
⟨*proof*⟩

lemma *exact-decomp-imp-valid-decomp*:
assumes *exact-decomp m ps* **and** $\bigwedge h\ U.\ (h,\ U) \in$ *set ps* $\Longrightarrow h \neq 0$
shows *valid-decomp X ps*
⟨*proof*⟩

lemma *exact-decomp-card-X*:
assumes *valid-decomp X ps* **and** *card* $X \leq m$
shows *exact-decomp m ps*
⟨*proof*⟩

definition a :: $((('x \Rightarrow_0 nat) \Rightarrow_0 'a::zero) \times 'x\ set)\ list \Rightarrow nat$
where a *ps* = (*LEAST k. standard-decomp k ps*)

definition b :: $((('x \Rightarrow_0 nat) \Rightarrow_0 'a::zero) \times 'x\ set)\ list \Rightarrow nat \Rightarrow nat$
where b *ps i* = (*LEAST d.* a *ps* $\leq d \wedge (\forall (h,\ U) \in set\ ps.\ i \leq card\ U \longrightarrow$ *poly-deg* $h < d$))

lemma a: *standard-decomp k ps* $\Longrightarrow$ *standard-decomp* (a *ps*) *ps*
⟨*proof*⟩

lemma a-*Nil*:
assumes $ps_+ = []$
shows a *ps* = *0*
⟨*proof*⟩

lemma a-*nonempty*:
assumes *valid-decomp X ps* **and** *standard-decomp k ps* **and** $ps_+ \neq []$
shows a $ps = Min\ (poly\text{-}deg\ `\ fst\ `\ set\ (ps_+))$
⟨*proof*⟩

lemma a-*nonempty-unique*:
assumes *valid-decomp X ps* **and** *standard-decomp k ps* **and** $ps_+ \neq []$
shows a $ps = k$
⟨*proof*⟩

lemma b:
shows a $ps \leq$ b $ps\ i$ **and** $(h,\ U) \in set\ ps \Longrightarrow i \leq card\ U \Longrightarrow poly\text{-}deg\ h <$ b $ps\ i$
⟨*proof*⟩

lemma b-*le*:
a $ps \leq d \Longrightarrow (\bigwedge h'\ U'.\ (h',\ U') \in set\ ps \Longrightarrow i \leq card\ U' \Longrightarrow poly\text{-}deg\ h' < d) \Longrightarrow$ 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** $Suc\ 0 \leq i$
shows b $ps\ i = 0$
⟨*proof*⟩

lemma b-*zero*:
assumes $ps \neq []$
shows $Suc\ (Max\ (poly\text{-}deg\ `\ fst\ `\ set\ ps)) \leq$ b $ps\ 0$
⟨*proof*⟩

corollary b-*zero-gr*:
assumes $(h,\ U) \in set\ ps$
shows $poly\text{-}deg\ h <$ b $ps\ 0$
⟨*proof*⟩

lemma b-*one*:
assumes *valid-decomp X ps* **and** *standard-decomp k ps*
shows b $ps\ (Suc\ 0) = (if\ ps_+ = []\ then\ 0\ else\ Suc\ (Max\ (poly\text{-}deg\ `\ fst\ `\ set\ (ps_+))))$
⟨*proof*⟩

corollary b-*one-gr*:
assumes *valid-decomp X ps* **and** *standard-decomp k ps* **and** $(h,\ U) \in set\ (ps_+)$

shows *poly-deg h* < b *ps* (*Suc 0*)
⟨*proof*⟩

lemma b-*card-X*:
assumes *exact-decomp m ps* **and** *Suc* (*card X*) ≤ *i*
shows b *ps i* = a *ps*
⟨*proof*⟩

lemma *lem-6-1-1*:
assumes *standard-decomp k ps* **and** *exact-decomp m ps* **and** *Suc 0* ≤ *i*
and *i* ≤ *card X* **and** b *ps* (*Suc i*) ≤ *d* **and** *d* < b *ps i*
obtains *h U* **where** (*h*, *U*) ∈ *set* (ps_+) **and** *poly-deg h* = *d* **and** *card U* = *i*
⟨*proof*⟩

corollary *lem-6-1-2*:
assumes *standard-decomp k ps* **and** *exact-decomp 0 ps* **and** *Suc 0* ≤ *i*
and *i* ≤ *card X* **and** b *ps* (*Suc i*) ≤ *d* **and** *d* < b *ps i*
obtains *h U* **where** {(*h*′, *U*′) ∈ *set* (ps_+). *poly-deg h*′ = *d*} = {(*h*, *U*)} **and** *card U* = *i*
⟨*proof*⟩

corollary *lem-6-1-2*′:
assumes *standard-decomp k ps* **and** *exact-decomp 0 ps* **and** *Suc 0* ≤ *i*
and *i* ≤ *card X* **and** b *ps* (*Suc i*) ≤ *d* **and** *d* < b *ps i*
shows *card* {(*h*′, *U*′) ∈ *set* (ps_+). *poly-deg h*′ = *d*} = *1* (**is** *card ?A* = -)
and {(*h*′, *U*′) ∈ *set* (ps_+). *poly-deg h*′ = *d* ∧ *card U*′ = *i*} = {(*h*′, *U*′) ∈ *set* (ps_+). *poly-deg h*′ = *d*}
(**is** *?B* = -)
and *card* {(*h*′, *U*′) ∈ *set* (ps_+). *poly-deg h*′ = *d* ∧ *card U*′ = *i*} = *1*
⟨*proof*⟩

corollary *lem-6-1-3*:
assumes *standard-decomp k ps* **and** *exact-decomp 0 ps* **and** *Suc 0* ≤ *i*
and *i* ≤ *card X* **and** (*h*, *U*) ∈ *set* (ps_+) **and** *card U* = *i*
shows b *ps* (*Suc i*) ≤ *poly-deg h*
⟨*proof*⟩ **fun** *shift-list* :: (((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::{*comm-ring-1*,*ring-no-zero-divisors*}) × ′*x set*) ⇒
′*x* ⇒ - *list* ⇒ - *list* **where**
shift-list (*h*, *U*) *x ps* =
((*punit.monom-mult 1* (*Poly-Mapping.single x 1*) *h*, *U*) # (*h*, *U* − {*x*}) # *removeAll* (*h*, *U*) *ps*)

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

lemma *monomial-decomp-shift-list*:
assumes *monomial-decomp ps* **and** *hU* ∈ *set ps*
shows *monomial-decomp* (*shift-list hU x ps*)
⟨*proof*⟩

lemma *hom-decomp-shift-list*:
assumes *hom-decomp ps* **and** *hU* ∈ *set ps*
shows *hom-decomp* (*shift-list hU x ps*)
⟨*proof*⟩

lemma *valid-decomp-shift-list*:
assumes *valid-decomp X ps* **and** (*h*, *U*) ∈ *set ps* **and** *x* ∈ *U*
shows *valid-decomp X* (*shift-list* (*h*, *U*) *x ps*)
⟨*proof*⟩

lemma *standard-decomp-shift-list*:
assumes *standard-decomp k ps* **and** (*h1*, *U1*) ∈ *set ps* **and** (*h2*, *U2*) ∈ *set ps*
and *poly-deg h1* = *poly-deg h2* **and** *card U2* ≤ *card U1* **and** (*h1*, *U1*) ≠ (*h2*, *U2*) **and** *x* ∈ *U2*
shows *standard-decomp k* (*shift-list* (*h2*, *U2*) *x ps*)
⟨*proof*⟩

lemma *cone-decomp-shift-list*:
assumes *valid-decomp X ps* **and** *cone-decomp T ps* **and** (*h*, *U*) ∈ *set ps* **and** *x* ∈ *U*
shows *cone-decomp T* (*shift-list* (*h*, *U*) *x ps*)
⟨*proof*⟩

10.9 Functions *shift* and *exact*

context
fixes *k m* :: *nat*
begin

context
fixes *d* :: *nat*
begin

definition *shift2-inv* :: ((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*::*zero*) × '*x set*) *list* ⇒ *bool* **where**
shift2-inv qs ⟷ *valid-decomp X qs* ∧ *standard-decomp k qs* ∧ *exact-decomp* (*Suc m*) *qs* ∧
(∀ *d0*<*d*. *card* {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d0* ∧ *m* < *card* (*snd q*)} ≤ *1*)

fun *shift1-inv* :: (((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*) × '*x set*) *list* × ((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*::*zero*) × '*x set*) *set*) ⇒ *bool*
where *shift1-inv* (*qs*, *B*) ⟷ *B* = {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d* ∧ *m* < *card* (*snd q*)} ∧ *shift2-inv qs*

lemma *shift2-invI*:
valid-decomp X qs ⟹ *standard-decomp k qs* ⟹ *exact-decomp* (*Suc m*) *qs* ⟹
(⋀*d0*. *d0* < *d* ⟹ *card* {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d0* ∧ *m* < *card* (*snd q*)} ≤ *1*) ⟹
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 $d0 < d \Longrightarrow$ *card* $\{q \in$ *set qs. poly-deg* (*fst q*) $= d0 \wedge m <$ *card* (*snd q*)$\} \leq 1$
⟨*proof*⟩

lemma *shift1-invI*:
$B = \{q \in$ *set qs. poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\} \Longrightarrow$ *shift2-inv qs* $\Longrightarrow$ *shift1-inv* (*qs*, *B*)
⟨*proof*⟩

lemma *shift1-invD*:
assumes *shift1-inv* (*qs*, *B*)
shows $B = \{q \in$ *set qs. poly-deg* (*fst q*) $= d \wedge m <$ *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** $1 <$ *card B* **and** (*h*, *U*) = (*SOME b*. $b \in B \wedge$ *card* (*snd b*) = *Suc m*)
shows (*h*, *U*) $\in B$ **and** (*h*, *U*) $\in$ *set qs* **and** *poly-deg h* = *d* **and** *card U* = *Suc m*
⟨*proof*⟩

lemma *shift1-inv-preserved*:
assumes *shift1-inv* (*qs*, *B*) **and** $1 <$ *card B* **and** (*h*, *U*) = (*SOME b*. $b \in B \wedge$ *card* (*snd b*) = *Suc m*)
and *x* = (*SOME y*. $y \in U$)
shows *shift1-inv* (*shift-list* (*h*, *U*) *x qs*, $B - \{(h, U)\}$)
⟨*proof*⟩

function (*domintros*) *shift1* :: $(((('x \Rightarrow_0 nat) \Rightarrow_0 'a) \times 'x\ set)\ list \times ((('x \Rightarrow_0 nat) \Rightarrow_0 'a) \times 'x\ set)\ set) \Rightarrow$
$(((('x \Rightarrow_0 nat) \Rightarrow_0 'a) \times 'x\ set)\ list \times$
$((('x \Rightarrow_0 nat) \Rightarrow_0 'a::\{comm\text{-}ring\text{-}1, ring\text{-}no\text{-}zero\text{-}divisors\}) \times 'x\ set)\ set)$
where
shift1 (*qs*, *B*) =

(*if* *1* < *card B then*
let (*h*, *U*) = *SOME b*. *b* ∈ *B* ∧ *card* (*snd b*) = *Suc m*; *x* = *SOME y*. *y* ∈ *U*
in
shift1 (*shift-list* (*h*, *U*) *x qs*, *B* − {(*h*, *U*)})
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 ⋀*qs B*. *shift1-inv* (*qs*, *B*) ⟹ *card B* ≤ *1* ⟹ *P* (*qs*, *B*) (*qs*, *B*)
assumes ⋀*qs B h U x*. *shift1-inv* (*qs*, *B*) ⟹ *1* < *card B* ⟹
(*h*, *U*) = (*SOME b*. *b* ∈ *B* ∧ *card* (*snd b*) = *Suc m*) ⟹ *x* = (*SOME y*.
y ∈ *U*) ⟹
finite U ⟹ *x* ∈ *U* ⟹ *card* (*U* − {*x*}) = *m* ⟹
P (*shift-list* (*h*, *U*) *x qs*, *B* − {(*h*, *U*)}) (*shift1* (*shift-list* (*h*, *U*) *x qs*, *B*
− {(*h*, *U*)})) ⟹
P (*qs*, *B*) (*shift1* (*shift-list* (*h*, *U*) *x qs*, *B* − {(*h*, *U*)}))
shows *P args* (*shift1 args*)
⟨*proof*⟩

lemma *shift1-1*:
assumes *shift1-inv args* **and** *d0* ≤ *d*
shows *card* {*q* ∈ *set* (*fst* (*shift1 args*)). *poly-deg* (*fst q*) = *d0* ∧ *m* < *card* (*snd*
q)} ≤ *1*
⟨*proof*⟩

lemma *shift1-2*:
shift1-inv args ⟹
card {*q* ∈ *set* (*fst* (*shift1 args*)). *m* < *card* (*snd q*)} ≤ *card* {*q* ∈ *set* (*fst args*).
m < *card* (*snd q*)}
⟨*proof*⟩

lemma *shift1-3*: *shift1-inv args* ⟹ *cone-decomp T* (*fst args*) ⟹ *cone-decomp T*
(*fst* (*shift1 args*))
⟨*proof*⟩

lemma *shift1-4*:
shift1-inv args ⟹
Max (*poly-deg* ‘ *fst* ‘ *set* (*fst args*)) ≤ *Max* (*poly-deg* ‘ *fst* ‘ *set* (*fst* (*shift1 args*)))
⟨*proof*⟩

lemma *shift1-5*: *shift1-inv args* ⟹ *fst* (*shift1 args*) = [] ⟷ *fst args* = []
⟨*proof*⟩

lemma *shift1-6*: *shift1-inv args* $\Longrightarrow$ *monomial-decomp* (*fst args*) $\Longrightarrow$ *monomial-decomp* (*fst* (*shift1 args*))
⟨*proof*⟩

lemma *shift1-7*: *shift1-inv args* $\Longrightarrow$ *hom-decomp* (*fst args*) $\Longrightarrow$ *hom-decomp* (*fst* (*shift1 args*))
⟨*proof*⟩

end

lemma *shift2-inv-preserved*:
  assumes *shift2-inv d qs*
  shows *shift2-inv* (*Suc d*) (*fst* (*shift1* (*qs*, {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d* ∧ *m* < *card* (*snd q*)})))
⟨*proof*⟩

function *shift2* :: *nat* ⇒ *nat* ⇒ (((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*) × ′*x set*) *list* ⇒
   (((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::{*comm-ring-1*,*ring-no-zero-divisors*}) × ′*x set*) *list* **where**
  shift2 c d qs =
    (*if c* ≤ *d then qs*
    else shift2 c (*Suc d*) (*fst* (*shift1* (*qs*, {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d* ∧ *m* < *card* (*snd q*)}))))
  ⟨*proof*⟩
termination ⟨*proof*⟩

lemma *shift2-1*: *shift2-inv d qs* $\Longrightarrow$ *shift2-inv c* (*shift2 c d qs*)
⟨*proof*⟩

lemma *shift2-2*:
  shift2-inv d qs $\Longrightarrow$
    card {*q* ∈ *set* (*shift2 c d qs*). *m* < *card* (*snd q*)} ≤ *card* {*q* ∈ *set qs*. *m* < *card* (*snd q*)}
⟨*proof*⟩

lemma *shift2-3*: *shift2-inv d qs* $\Longrightarrow$ *cone-decomp T qs* $\Longrightarrow$ *cone-decomp T* (*shift2 c d qs*)
⟨*proof*⟩

lemma *shift2-4*:
  shift2-inv d qs $\Longrightarrow$ *Max* (*poly-deg* ‘ *fst* ‘ *set qs*) ≤ *Max* (*poly-deg* ‘ *fst* ‘ *set* (*shift2 c d qs*))
⟨*proof*⟩

lemma *shift2-5*:
  shift2-inv d qs $\Longrightarrow$ *shift2 c d qs* = [] ⟷ *qs* = []
⟨*proof*⟩

lemma *shift2-6*:

shift2-inv d qs ⟹ *monomial-decomp qs* ⟹ *monomial-decomp* (*shift2 c d qs*)
⟨*proof*⟩

lemma *shift2-7*:
shift2-inv d qs ⟹ *hom-decomp qs* ⟹ *hom-decomp* (*shift2 c d qs*)
⟨*proof*⟩

definition *shift* :: ((('x ⇒$_0$ nat) ⇒$_0$ 'a) × 'x set) list ⇒
((('x ⇒$_0$ nat) ⇒$_0$ 'a::{comm-ring-1,ring-no-zero-divisors}) × 'x set) list
where *shift qs* = *shift2* (*k* + *card* {*q* ∈ *set qs*. *m* < *card* (*snd q*)}) *k qs*

lemma *shift2-inv-init*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp* (*Suc m*) *qs*
shows *shift2-inv k qs*
⟨*proof*⟩

lemma *shift*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp* (*Suc m*) *qs*
shows *valid-decomp X* (*shift qs*) **and** *standard-decomp k* (*shift qs*) **and** *exact-decomp m* (*shift qs*)
⟨*proof*⟩

lemma *monomial-decomp-shift*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp* (*Suc m*) *qs*
and *monomial-decomp qs*
shows *monomial-decomp* (*shift qs*)
⟨*proof*⟩

lemma *hom-decomp-shift*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp* (*Suc m*) *qs*
and *hom-decomp qs*
shows *hom-decomp* (*shift qs*)
⟨*proof*⟩

lemma *cone-decomp-shift*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp* (*Suc m*) *qs*
and *cone-decomp T qs*
shows *cone-decomp T* (*shift qs*)
⟨*proof*⟩

lemma *Max-shift-ge*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp* (*Suc m*) *qs*

shows *Max* (*poly-deg* ' *fst* ' *set qs*) ≤ *Max* (*poly-deg* ' *fst* ' *set* (*shift qs*))
⟨*proof*⟩

lemma *shift-Nil-iff*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp* (*Suc m*) *qs*
shows *shift qs* = [] ⟷ *qs* = []
⟨*proof*⟩

end

primrec *exact-aux* :: *nat* ⇒ *nat* ⇒ (((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*) × ′*x set*) *list* ⇒ (((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::{*comm-ring-1*,*ring-no-zero-divisors*}) × ′*x set*) *list* **where**
exact-aux k 0 qs = *qs* |
exact-aux k (*Suc m*) *qs* = *exact-aux k m* (*shift k m qs*)

lemma *exact-aux*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp m qs*
shows *valid-decomp X* (*exact-aux k m qs*) (**is** *?thesis1*)
and *standard-decomp k* (*exact-aux k m qs*) (**is** *?thesis2*)
and *exact-decomp 0* (*exact-aux k m qs*) (**is** *?thesis3*)
⟨*proof*⟩

lemma *monomial-decomp-exact-aux*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp m qs* **and** *monomial-decomp qs*
shows *monomial-decomp* (*exact-aux k m qs*)
⟨*proof*⟩

lemma *hom-decomp-exact-aux*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp m qs* **and** *hom-decomp qs*
shows *hom-decomp* (*exact-aux k m qs*)
⟨*proof*⟩

lemma *cone-decomp-exact-aux*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp m qs* **and** *cone-decomp T qs*
shows *cone-decomp T* (*exact-aux k m qs*)
⟨*proof*⟩

lemma *Max-exact-aux-ge*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp m qs*
shows *Max* (*poly-deg* ' *fst* ' *set qs*) ≤ *Max* (*poly-deg* ' *fst* ' *set* (*exact-aux k m 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* = [] ⟷ *qs* = []
⟨*proof*⟩

definition *exact* :: *nat* ⇒ (((′*x* ⇒$_0$ *nat*) ⇒$_0$ ′*a*) × ′*x set*) *list* ⇒
(((′*x* ⇒$_0$ *nat*) ⇒$_0$ ′*a*::{*comm-ring-1*,*ring-no-zero-divisors*}) × ′*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*)
⟨*proof*⟩

lemma *monomial-decomp-exact*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *monomial-decomp qs*
shows *monomial-decomp* (*exact k qs*)
⟨*proof*⟩

lemma *hom-decomp-exact*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *hom-decomp qs*
shows *hom-decomp* (*exact k qs*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *Max-exact-ge*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs*
shows *Max* (*poly-deg* ‘ *fst* ‘ *set qs*) ≤ *Max* (*poly-deg* ‘ *fst* ‘ *set* (*exact k qs*))
⟨*proof*⟩

lemma *exact-Nil-iff*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs*
shows *exact k qs* = [] ⟷ *qs* = []
⟨*proof*⟩

corollary b-*zero-exact*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *qs* ≠ []
shows *Suc* (*Max* (*poly-deg* ‘ *fst* ‘ *set qs*)) ≤ b (*exact k qs*) *0*
⟨*proof*⟩

lemma *normal-form-exact-decompE*:
assumes *F* ⊆ *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* ‘ *P*[*X*]) *qs* **and** *exact-decomp 0 qs*
and $\bigwedge$*g*. ($\bigwedge$*f*. *f* ∈ *F* ⟹ *homogeneous f*) ⟹ *g* ∈ *punit.reduced-GB F* ⟹ *poly-deg g* ≤ b *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* :: *nat*
begin

function *Dube-aux* :: *nat* ⇒ *nat* **where**
Dube-aux j = (*if j* + *2* < *n then*
2 + ((*Dube-aux* (*j* + *1*)) *choose 2*) + ($\sum$ *i*=*j*+*3*..*n*−*1*. (*Dube-aux i*) *choose* (*Suc* (*i* − *j*)))
else if j + *2* = *n then* d^2 + *2* ∗ *d else 2* ∗ *d*)
⟨*proof*⟩
termination ⟨*proof*⟩

definition *Dube* :: *nat* **where** *Dube* = (*if n* ≤ *1* ∨ *d* = *0 then d else Dube-aux 1*)

lemma *Dube-aux-ge-d*: *d* ≤ *Dube-aux j*
⟨*proof*⟩

corollary *Dube-ge-d*: *d* ≤ *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* ≤ *2* ∗ ((*rat-of-nat d*)^2 / *2* + (*rat-of-nat d*)) ^ (*2* ^ (*n* − *2*))
⟨*proof*⟩

end

11.1 Hilbert Function and Hilbert Polynomial

context *pm-powerprod*
begin

context
fixes *X* :: *'x set*
assumes *fin-X*: *finite X*
begin

lemma *Hilbert-fun-cone-aux*:
assumes *h* ∈ *P*[*X*] **and** *h* ≠ *0* **and** *U* ⊆ *X* **and** *homogeneous* (*h*::- ⇒$_0$ *'a*::*field*)
shows *Hilbert-fun* (*cone* (*h*, *U*)) *z* = *card* {*t* ∈ .[*U*]. *deg-pm t* + *poly-deg h* = *z*}
⟨*proof*⟩

lemma *Hilbert-fun-cone-empty*:
assumes *h* ∈ *P*[*X*] **and** *h* ≠ *0* **and** *homogeneous* (*h*::- ⇒$_0$ *'a*::*field*)
shows *Hilbert-fun* (*cone* (*h*, {})) *z* = (*if poly-deg h* = *z then 1 else 0*)
⟨*proof*⟩

lemma *Hilbert-fun-cone-nonempty*:
assumes *h* ∈ *P*[*X*] **and** *h* ≠ *0* **and** *U* ⊆ *X* **and** *homogeneous* (*h*::- ⇒$_0$ *'a*::*field*) **and** *U* ≠ {}
shows *Hilbert-fun* (*cone* (*h*, *U*)) *z* =
(*if poly-deg h* ≤ *z then* ((*z* − *poly-deg h*) + (*card U* − *1*)) *choose* (*card U* − *1*) *else 0*)
⟨*proof*⟩

corollary *Hilbert-fun-Polys*:
assumes *X* ≠ {}
shows *Hilbert-fun* (*P*[*X*]::(- ⇒$_0$ *'a*::*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*∈*set ps*. *Hilbert-fun* (*cone hU*) *z*)
⟨*proof*⟩

definition *Hilbert-poly* :: (*nat* ⇒ *nat*) ⇒ *int* ⇒ *int*
where *Hilbert-poly b* =
(λ*z*::*int*. *let n* = *card X in*
((*z* − *b* (*Suc n*) + *n*) *gchoose n*) − *1* − ($\sum$ *i*=*1*..*n*. (*z* − *b i* + *i* − *1*) *gchoose i*))

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

lemma *Hilbert-fun-eq-Hilbert-poly-plus-card*:
assumes *X* ≠ {} **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* (*Suc 0*) ≤ *d*
shows *int* (*Hilbert-fun T d*) = *card* {*h*::- $\Rightarrow_0$ ′*a*::*field*. (*h*, {}) ∈ *set ps* ∧ *poly-deg h* = *d*} + *Hilbert-poly* (b *ps*) *d*
⟨*proof*⟩

corollary *Hilbert-fun-eq-Hilbert-poly*:
assumes *X* ≠ {} **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* ≤ *d*
shows *int* (*Hilbert-fun* (*T*::(- $\Rightarrow_0$ ′*a*::*field*) *set*) *d*) = *Hilbert-poly* (b *ps*) *d*
⟨*proof*⟩

11.2 Dubé's Bound

context
fixes *f* :: (′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::*field*
fixes *F*
assumes *n-gr-1*: *1* < *card X* **and** *fin-F*: *finite F* **and** *F-sub*: *F* ⊆ *P*[*X*] **and** *f-in*: *f* ∈ *F*
and *hom-F*: ⋀*f*′. *f*′ ∈ *F* ⟹ *homogeneous f*′ **and** *f-max*: ⋀*f*′. *f*′ ∈ *F* ⟹ *poly-deg f*′ ≤ *poly-deg f*
and *d-gr-0*: *0* < *poly-deg f* **and** *ideal-f-neq*: *ideal* {*f*} ≠ *ideal F*
begin

private abbreviation (*input*) *n* ≡ *card X*
private abbreviation (*input*) *d* ≡ *poly-deg f*

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

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

lemma *f-not-0*: *f* ≠ *0*

⟨*proof*⟩

lemma *X-not-empty*: $X \neq \{\}$
⟨*proof*⟩

lemma *n-gr-0*: $0 < n$
⟨*proof*⟩

corollary *int-n-minus-1* [*simp*]: *int* (*n* − *Suc 0*) = *int n* − *1*
⟨*proof*⟩

lemma *int-n-minus-2* [*simp*]: *int* (*n* − *Suc* (*Suc 0*)) = *int n* − *2*
⟨*proof*⟩

lemma *cone-f-X-sub*: *cone* (*f*, *X*) ⊆ *P*[*X*]
⟨*proof*⟩

lemma *ideal-Int-Polys-eq-cone*: *ideal* {*f*} ∩ *P*[*X*] = *cone* (*f*, *X*)
⟨*proof*⟩ **definition** *P-ps* **where**
P-ps = (*SOME x*. *valid-decomp X* (*snd x*) ∧ *standard-decomp d* (*snd x*) ∧
exact-decomp X 0 (*snd x*) ∧ *cone-decomp* (*fst x*) (*snd x*) ∧
hom-decomp (*snd x*) ∧
direct-decomp (*ideal F* ∩ *P*[*X*]) [*ideal* {*f*} ∩ *P*[*X*], *fst x*])

private definition *P* **where** *P* = *fst P-ps*

private definition *ps* **where** *ps* = *snd P-ps*

lemma
shows *valid-ps*: *valid-decomp X ps* (**is** *?thesis1*)
and *std-ps*: *standard-decomp d ps* (**is** *?thesis2*)
and *ext-ps*: *exact-decomp X 0 ps* (**is** *?thesis3*)
and *cn-ps*: *cone-decomp P ps* (**is** *?thesis4*)
and *hom-ps*: *hom-decomp ps* (**is** *?thesis5*)
and *decomp-F*: *direct-decomp* (*ideal F* ∩ *P*[*X*]) [*ideal* {*f*} ∩ *P*[*X*], *P*] (**is** *?thesis6*)
⟨*proof*⟩

lemma *P-sub*: *P* ⊆ *P*[*X*]
⟨*proof*⟩

lemma *ps-not-Nil*: $ps_+ \neq []$
⟨*proof*⟩ **definition** *N* **where** *N* = *normal-form F* ‘ *P*[*X*]

private definition *qs* **where** *qs* = (*SOME qs′*. *valid-decomp X qs′* ∧ *standard-decomp 0 qs′* ∧
monomial-decomp qs′ ∧ *cone-decomp N qs′* ∧
exact-decomp X 0 qs′ ∧
(∀ *g*∈*punit.reduced-GB F*. *poly-deg g* ≤ b *qs′ 0*))

private definition *aa* ≡ b *ps*
private definition *bb* ≡ b *qs*
private abbreviation (*input*) *cc* ≡ (λ*i*. *aa i* + *bb i*)

lemma
  shows *valid-qs*: *valid-decomp X qs* (**is** *?thesis1*)
    and *std-qs*: *standard-decomp 0 qs* (**is** *?thesis2*)
    and *mon-qs*: *monomial-decomp qs* (**is** *?thesis3*)
    and *hom-qs*: *hom-decomp qs* (**is** *?thesis6*)
    and *cn-qs*: *cone-decomp N qs* (**is** *?thesis4*)
    and *ext-qs*: *exact-decomp X 0 qs* (**is** *?thesis5*)
    and *deg-RGB*: *g* ∈ *punit.reduced-GB F* ⟹ *poly-deg g* ≤ *bb 0*
⟨*proof*⟩

lemma *N-sub*: *N* ⊆ *P*[*X*]
  ⟨*proof*⟩

lemma *decomp-Polys*: *direct-decomp P*[*X*] [*ideal* {*f*} ∩ *P*[*X*], *P*, *N*]
⟨*proof*⟩

lemma *aa-Suc-n* [*simp*]: *aa* (*Suc n*) = *d*
⟨*proof*⟩

lemma *bb-Suc-n* [*simp*]: *bb* (*Suc n*) = *0*
⟨*proof*⟩

lemma *Hilbert-fun-X*:
  assumes *d* ≤ *z*
  shows *Hilbert-fun* (*P*[*X*]::(- ⇒$_0$ *'a*) *set*) *z* =
          ((*z* − *d*) + (*n* − *1*)) *choose* (*n* − *1*) + *Hilbert-fun P z* + *Hilbert-fun N z*
⟨*proof*⟩

lemma *dube-eq-0*:
  (λ*z*::*int*. (*z* + *int n* − *1*) *gchoose* (*n* − *1*)) =
    (λ*z*::*int*. ((*z* − *d* + *n* − *1*) *gchoose* (*n* − *1*)) + *Hilbert-poly aa z* + *Hilbert-poly bb z*)
    (**is** *?f* = *?g*)
⟨*proof*⟩

corollary *dube-eq-1*:
  (λ*z*::*int*. (*z* + *int n* − *1*) *gchoose* (*n* − *1*)) =
    (λ*z*::*int*. ((*z* − *d* + *n* − *1*) *gchoose* (*n* − *1*)) + ((*z* − *d* + *n*) *gchoose n*) + ((*z* + *n*) *gchoose n*) − *2* −
        (∑ *i*=*1*..*n*. ((*z* − *aa i* + *i* − *1*) *gchoose i*) + ((*z* − *bb i* + *i* − *1*) *gchoose i*)))
  ⟨*proof*⟩

lemma *dube-eq-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 $(1::int) = (- 1)\hat{}(n - Suc\ j) * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) +$
$(- 1)\hat{}(n - j) * ((int\ d - 1)\ gchoose\ (n - j)) - 1 -$
$(\sum i=Suc\ j..n.\ (- 1)\hat{}(i - j) * ((int\ (aa\ i)\ gchoose\ (i - j)) + (int\ (bb\ i)\ gchoose\ (i - j))))$
⟨*proof*⟩

lemma *dube-aux-1*:
assumes $(h, \{\}) \in set\ ps \cup set\ qs$
shows $poly\text{-}deg\ h < max\ (aa\ 1)\ (bb\ 1)$
⟨*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 $(1::int) = 2 * (- 1)\hat{}(n - Suc\ j) * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) - 1 -$
$(\sum i=Suc\ j..n-1.\ (- 1)\hat{}(i - j) * ((int\ (aa\ i)\ gchoose\ (i - j)) + (int\ (bb\ i)\ gchoose\ (i - j))))$
⟨*proof*⟩

lemma *cc-Suc*:
assumes $j < n - 1$
shows $int\ (cc\ (Suc\ j)) = 2 + 2 * (- 1)\hat{}(n - j) * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) +$
$(\sum i=j+2..n-1.\ (- 1)\hat{}(i - j) * ((int\ (aa\ i)\ gchoose\ (i - j)) + (int\ (bb\ i)\ gchoose\ (i - j))))$
⟨*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-gr-2*: $2 < n$
begin

lemma *cc-n-minus-2*: $cc\ (n - 2) \leq d^2 + 2 * d$
⟨*proof*⟩

lemma *cc-Suc-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.
⟨*proof*⟩

corollary *cc-le*:
assumes $0 < j$ **and** $j < n - 2$
shows $cc\ j \leq 2 + (cc\ (j + 1)\ choose\ 2) + (\sum i=j+3..n-1.\ cc\ i\ choose\ (Suc\ (i - j)))$
⟨*proof*⟩

corollary *cc-le-Dube-aux*: $0 < j \Longrightarrow j + 1 \leq n \Longrightarrow cc\ j \leq Dube\text{-}aux\ n\ d\ j$
⟨*proof*⟩

end

lemma *Dube-aux*:
assumes $g \in punit.reduced\text{-}GB\ F$
shows $poly\text{-}deg\ g \leq Dube\text{-}aux\ n\ d\ 1$
⟨*proof*⟩

end

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

corollary *Dube-is-hom-GB-bound*:
$finite\ F \Longrightarrow F \subseteq P[X] \Longrightarrow is\text{-}hom\text{-}GB\text{-}bound\ F\ (Dube\ (card\ X)\ (maxdeg\ F))$
⟨*proof*⟩

end

corollary *Dube-indets*:
assumes *finite F* **and** $\bigwedge f.\ f \in F \Longrightarrow homogeneous\ f$ **and** $g \in punit.reduced\text{-}GB\ F$
shows $poly\text{-}deg\ g \leq Dube\ (card\ (\bigcup (indets\ `\ F)))\ (maxdeg\ F)$
⟨*proof*⟩

corollary *Dube-is-hom-GB-bound-indets*:
finite F $\Longrightarrow$ *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 $\bigwedge g.\ g \in G \Longrightarrow \exists\, q.\ g = (\sum f{\in}F.\ q\ f * f) \wedge$
$(\forall f.\ q\ f \in P[X] \wedge$ *poly-deg* $(q\ f * f) \leq$ *Dube* (*Suc* (*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* ⊆ *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* ⊆ *P*[*X*]
  shows *1* ∈ *ideal* (*set fs*) ⟷
  1 ∈ *set* (*punit.Macaulay-list* (*deg-shifts X* (*Dube* (*Suc* (*card X*)) (*maxdeg* (*set fs*))) *fs*))
  ⟨*proof*⟩

theorem *thm-2-3-6-indets-Dube*:
  fixes *fs*
  defines *X* ≡ ⋃(*indets* ‘ *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* ≡ ⋃(*indets* ‘ *set fs*)
  shows *1* ∈ *ideal* (*set fs*) ⟷
  1 ∈ *set* (*punit.Macaulay-list* (*deg-shifts X* (*Dube* (*Suc* (*card X*)) (*maxdeg* (*set fs*))) *fs*))
  ⟨*proof*⟩

end

12.2 Preparations

primrec *remdups-wrt-rev* :: (*'a* ⇒ *'b*) ⇒ *'a list* ⇒ *'b list* ⇒ *'a 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* ∈ *set vs* ⟹ *v* ∉ *f* ‘ *set* (*remdups-wrt-rev f 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* (λ*x*. *fst x* ∉ *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 = $distr_0$ *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 (λ*p*. *p* ≠ *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 $('x, 'a)\ pp \Rightarrow_0 'b$

definition *keys-pp-to-list* :: (′*x*::*linorder*, ′*a*::*zero*) *pp* ⇒ ′*x 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 ∘ *PP* = (λ*x*. *x*)
PP ∘ *mapping-of* = (λ*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* ⟷ *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-Mapping.map-key PP* (*sum f A*) = ($\sum$ *a*∈*A*. *Poly-Mapping.map-key PP* (*f a*))
⟨*proof*⟩

lemma *map-key-PP-ideal*:
Poly-Mapping.map-key PP ' *ideal F* = *ideal* (*Poly-Mapping.map-key PP* ' (*F*::((-, -::*add-linorder*) *pp* ⇒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* ⇒ ('*x*, *nat*) *pp* ⇒ *bool*
and *ord-strict*
begin

sublocale *gd-powerprod* ⟨*proof*⟩

sublocale *pp-pm*: *extended-ord-pm-powerprod* λ*s t*. *ord* (*PP s*) (*PP t*) λ*s t*. *ord-strict* (*PP s*) (*PP t*)
⟨*proof*⟩

definition *poly-deg-pp* :: (('*x*, *nat*) *pp* ⇒₀ '*a*::*zero*) ⇒ *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* ⇒ *nat* ⇒ ('*x*, *nat*) *pp* ⇒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* (λ*x*. (+) (*monom-mult-punit 1* (*single-pp x 1*) *p*)) *xs 0*)

definition *deg-le-sect-pp* :: '*x list* ⇒ *nat* ⇒ ('*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* ⇒ *nat* ⇒
(('*x*, *nat*) *pp* ⇒₀ '*b*) *list* ⇒ (('*x*, *nat*) *pp* ⇒₀ '*b*::*semiring-1*) *list*

where *deg-shifts-pp xs d fs* = *concat* (*map* (λ*f*. (*map* (λ*t*. *monom-mult-punit 1 t f*)
(*deg-le-sect-pp xs* (*d* − *poly-deg-pp f*)))) *fs*)

definition *indets-pp* :: ((′*x*, *nat*) *pp* ⇒₀ ′*b*::*zero*) ⇒ ′*x list*
where *indets-pp p* = *remdups* (*concat* (*map keys-pp-to-list* (*punit.keys-to-list p*)))

definition *Indets-pp* :: ((′*x*, *nat*) *pp* ⇒₀ ′*b*::*zero*) *list* ⇒ ′*x list*
where *Indets-pp ps* = *remdups* (*concat* (*map indets-pp ps*))

lemma *map-PP-insort*:
map PP (*pp-pm.ordered-powerprod-lin.insort x xs*) = *ordered-powerprod-lin.insort* (*PP x*) (*map PP xs*)
⟨*proof*⟩

lemma *map-PP-sorted-list-of-set*:
map PP (*pp-pm.ordered-powerprod-lin.sorted-list-of-set T*) =
ordered-powerprod-lin.sorted-list-of-set (*PP* ‘ *T*)
⟨*proof*⟩

lemma *map-PP-pps-to-list*: *map PP* (*pp-pm.punit.pps-to-list T*) = *punit.pps-to-list* (*PP* ‘ *T*)
⟨*proof*⟩

lemma *map-mapping-of-pps-to-list*:
map mapping-of (*punit.pps-to-list T*) = *pp-pm.punit.pps-to-list* (*mapping-of* ‘ *T*)
⟨*proof*⟩

lemma *keys-to-list-map-key-PP*:
pp-pm.punit.keys-to-list (*Poly-Mapping.map-key PP p*) = *map mapping-of* (*punit.keys-to-list p*)
⟨*proof*⟩

lemma *Keys-to-list-map-key-PP*:
pp-pm.punit.Keys-to-list (*map* (*Poly-Mapping.map-key PP*) *fs*) = *map mapping-of* (*punit.Keys-to-list fs*)
⟨*proof*⟩

lemma *poly-deg-map-key-PP*: *poly-deg* (*Poly-Mapping.map-key PP p*) = *poly-deg-pp p*
⟨*proof*⟩

lemma *deg-le-sect-pp-aux-1*:
assumes *t* ∈ *keys* (*deg-le-sect-pp-aux xs n*)
shows *deg-pp t* ≤ *n* **and** *keys-pp t* ⊆ *set xs*
⟨*proof*⟩

lemma *deg-le-sect-pp-aux-2*:
assumes *deg-pp t* ≤ *n* **and** *keys-pp t* ⊆ *set xs*

shows *t* ∈ *keys* (*deg-le-sect-pp-aux xs n*)
⟨*proof*⟩

lemma *keys-deg-le-sect-pp-aux*:
keys (*deg-le-sect-pp-aux xs n*) = {*t*. *deg-pp t* ≤ *n* ∧ *keys-pp t* ⊆ *set xs*}
⟨*proof*⟩

lemma *deg-le-sect-deg-le-sect-pp*:
map PP (*pp-pm.punit.pps-to-list* (*deg-le-sect* (*set xs*) *d*)) = *deg-le-sect-pp xs d*
⟨*proof*⟩

lemma *deg-shifts-deg-shifts-pp*:
pp-pm.deg-shifts (*set xs*) *d* (*map* (*Poly-Mapping.map-key PP*) *fs*) =
map (*Poly-Mapping.map-key PP*) (*deg-shifts-pp xs d fs*)
⟨*proof*⟩

lemma *ideal-deg-shifts-pp*: *ideal* (*set* (*deg-shifts-pp xs d fs*)) = *ideal* (*set fs*)
⟨*proof*⟩

lemma *set-indets-pp*: *set* (*indets-pp p*) = *indets* (*Poly-Mapping.map-key PP p*)
⟨*proof*⟩

lemma *poly-to-row-map-key-PP*:
poly-to-row (*map pp.mapping-of xs*) (*Poly-Mapping.map-key PP p*) = *poly-to-row xs p*
⟨*proof*⟩

lemma *Macaulay-mat-map-key-PP*:
pp-pm.punit.Macaulay-mat (*map* (*Poly-Mapping.map-key PP*) *fs*) = *punit.Macaulay-mat fs*
⟨*proof*⟩

lemma *row-to-poly-mapping-of*:
assumes *distinct ts* **and** *dim-vec r* = *length ts*
shows *row-to-poly* (*map pp.mapping-of ts*) *r* = *Poly-Mapping.map-key PP* (*row-to-poly ts r*)
⟨*proof*⟩

lemma *mat-to-polys-mapping-of*:
assumes *distinct ts* **and** *dim-col m* = *length ts*
shows *mat-to-polys* (*map pp.mapping-of ts*) *m* = *map* (*Poly-Mapping.map-key PP*) (*mat-to-polys ts m*)
⟨*proof*⟩

lemma *map-key-PP-Macaulay-list*:
map (*Poly-Mapping.map-key PP*) (*punit.Macaulay-list fs*) =
pp-pm.punit.Macaulay-list (*map* (*Poly-Mapping.map-key PP*) *fs*)
⟨*proof*⟩

lemma *lpp-map-key-PP*: *pp-pm.lpp* (*Poly-Mapping.map-key PP p*) = *mapping-of* (*lpp p*)
⟨*proof*⟩

lemma *is-GB-map-key-PP*:
finite G ⟹ *pp-pm.punit.is-Groebner-basis* (*Poly-Mapping.map-key PP ‘ G*) ⟷ *punit.is-Groebner-basis G*
⟨*proof*⟩

lemma *thm-2-3-6-pp*:
assumes *pp-pm.is-GB-cofactor-bound* (*Poly-Mapping.map-key PP ‘ set fs*) *b*
shows *punit.is-Groebner-basis* (*set* (*punit.Macaulay-list* (*deg-shifts-pp* (*Indets-pp fs*) *b fs*)))
⟨*proof*⟩

lemma *Dube-is-GB-cofactor-bound-pp*:
pp-pm.is-GB-cofactor-bound (*Poly-Mapping.map-key PP ‘ set fs*)
(*Dube* (*Suc* (*length* (*Indets-pp fs*))) (*max-list* (*map poly-deg-pp fs*)))
⟨*proof*⟩

definition *GB-Macaulay-Dube* :: (('*x*, *nat*) *pp* $\Rightarrow_0$ '*a*) *list* ⇒ (('*x*, *nat*) *pp* $\Rightarrow_0$ '*a*::*field*) *list*
where *GB-Macaulay-Dube fs* = *punit.Macaulay-list* (*deg-shifts-pp* (*Indets-pp fs*)
(*Dube* (*Suc* (*length* (*Indets-pp fs*))) (*max-list* (*map poly-deg-pp fs*))) *fs*)

lemma *GB-Macaulay-Dube-is-GB*: *punit.is-Groebner-basis* (*set* (*GB-Macaulay-Dube fs*))
⟨*proof*⟩

lemma *ideal-GB-Macaulay-Dube*: *ideal* (*set* (*GB-Macaulay-Dube fs*)) = *ideal* (*set fs*)
⟨*proof*⟩

end

global-interpretation *punit'*: *pp-powerprod ord-pp-punit cmp-term ord-pp-strict-punit cmp-term*
rewrites *punit.adds-term* = (*adds*)
and *punit.pp-of-term* = (λ*x*. *x*)
and *punit.component-of-term* = (λ-. ())
and *punit.monom-mult* = *monom-mult-punit*
and *punit.mult-scalar* = *mult-scalar-punit*
and *punit'.punit.min-term* = *min-term-punit*
and *punit'.punit.lt* = *lt-punit cmp-term*
and *punit'.punit.lc* = *lc-punit cmp-term*
and *punit'.punit.tail* = *tail-punit cmp-term*
and *punit'.punit.ord-p* = *ord-p-punit cmp-term*
and *punit'.punit.keys-to-list* = *keys-to-list-punit cmp-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 *trivariate*$_0$*-rat* ⟨*proof*⟩

lemma
comp-red-basis-punit DRLEX (*GB-Macaulay-Dube-punit DRLEX* [$X * Y^2 + 3 * X^2 * Y$, Y ^ $3 - X$ ^ 3]) =
[X ^ 5, X ^ $3 * Y - C_0$ (1 / 9) $* X$ ^ 4, Y ^ $3 - X$ ^ 3, $X * Y^2 + 3 * X^2 * Y$]
⟨*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.