Gröbner Bases, Macaulay Matrices and Dubé's Degree Bounds

Alexander Maletzky*

March 17, 2025

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 $deg\text{-}le\text{-}sect\ X\ d = (\bigcup d0{\le}d.\ deg\text{-}sect\ X\ d0)$

lemma *deg-sectI*: $t \in .[X] \Longrightarrow deg\text{-}pm\ t = d \Longrightarrow t \in deg\text{-}sect\ X\ d$
  by (*simp add*: *deg-sect-def*)

lemma *deg-sectD*:
  assumes $t \in deg\text{-}sect\ X\ d$
  shows $t \in .[X]$ **and** $deg\text{-}pm\ t = d$
  using *assms* **by** (*simp-all add*: *deg-sect-def*)

lemma *deg-le-sect-alt*: $deg\text{-}le\text{-}sect\ X\ d = .[X] \cap \{t.\ deg\text{-}pm\ t \le d\}$
  by (*auto simp*: *deg-le-sect-def deg-sect-def*)

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

by (*simp add*: *deg-le-sect-alt*)

lemma *deg-le-sectD*:
assumes $t \in$ *deg-le-sect X d*
shows $t \in .[X]$ **and** *deg-pm* $t \leq d$
using *assms* **by** (*simp-all add*: *deg-le-sect-alt*)

lemma *deg-sect-zero* [*simp*]: *deg-sect X 0* = {*0*}
by (*auto simp*: *deg-sect-def zero-in-PPs*)

lemma *deg-sect-empty*: *deg-sect* {} *d* = (*if d* = *0 then* {*0*} *else* {})
by (*auto simp*: *deg-sect-def*)

lemma *deg-sect-singleton* [*simp*]: *deg-sect* {*x*} *d* = {*Poly-Mapping.single x d*}
by (*auto simp*: *deg-sect-def deg-pm-single PPs-singleton*)

lemma *deg-le-sect-zero* [*simp*]: *deg-le-sect X 0* = {*0*}
by (*auto simp*: *deg-le-sect-def*)

lemma *deg-le-sect-empty* [*simp*]: *deg-le-sect* {} *d* = {*0*}
by (*auto simp*: *deg-le-sect-alt varnum-eq-zero-iff*)

lemma *deg-le-sect-singleton*: *deg-le-sect* {*x*} *d* = *Poly-Mapping.single x* ‘ {..*d*}
by (*auto simp*: *deg-le-sect-alt deg-pm-single PPs-singleton*)

lemma *deg-sect-mono*: $X \subseteq Y \Longrightarrow$ *deg-sect X d* $\subseteq$ *deg-sect Y d*
by (*auto simp*: *deg-sect-def dest*: *PPs-mono*)

lemma *deg-le-sect-mono-1*: $X \subseteq Y \Longrightarrow$ *deg-le-sect X d* $\subseteq$ *deg-le-sect Y d*
by (*auto simp*: *deg-le-sect-alt dest*: *PPs-mono*)

lemma *deg-le-sect-mono-2*: $d1 \leq d2 \Longrightarrow$ *deg-le-sect X d1* $\subseteq$ *deg-le-sect X d2*
by (*auto simp*: *deg-le-sect-alt*)

lemma *zero-in-deg-le-sect*: $0 \in$ *deg-le-sect n d*
by (*simp add*: *deg-le-sect-alt zero-in-PPs*)

lemma *deg-sect-disjoint*: $d1 \neq d2 \Longrightarrow$ *deg-sect X d1* $\cap$ *deg-sect Y d2* = {}
by (*auto simp*: *deg-sect-def*)

lemma *deg-le-sect-deg-sect-disjoint*: $d1 < d2 \Longrightarrow$ *deg-le-sect Y d1* $\cap$ *deg-sect X d2* = {}
by (*auto simp*: *deg-sect-def deg-le-sect-alt*)

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 (*rule set-eqI*)
fix *t*

```
  show t ∈ ?A ⟷ t ∈ ?B
  proof
    assume t ∈ ?A
    hence t ∈ .[X] and deg-pm t = Suc d by (rule deg-sectD)+
    from this(2) have keys t ≠ {} by auto
    then obtain x where x ∈ keys t by blast
    hence 1 ≤ lookup t x by (simp add: in-keys-iff)
    from ‹t ∈ .[X]› have keys t ⊆ X by (rule PPsD)
    with ‹x ∈ keys t› have x ∈ X ..
    let ?s = Poly-Mapping.single x (1::nat)
    from ‹1 ≤ lookup t x› have ?s adds t
      by (auto simp: lookup-single when-def intro!: adds-poly-mappingI le-funI)
    hence t: ?s + (t − ?s) = t by (metis add.commute adds-minus)
    have t − ?s ∈ deg-sect X d
    proof (rule deg-sectI)
      from ‹t ∈ .[X]› show t − ?s ∈ .[X] by (rule PPs-closed-minus)
    next
      from deg-pm-plus[of ?s t − ?s] have deg-pm t = Suc (deg-pm (t − ?s))
        by (simp only: t deg-pm-single)
      thus deg-pm (t − ?s) = d by (simp add: ‹deg-pm t = Suc d›)
    qed
    hence ?s + (t − ?s) ∈ (+) ?s ‘ deg-sect X d by (rule imageI)
    hence t ∈ (+) ?s ‘ deg-sect X d by (simp only: t)
    with ‹x ∈ X› show t ∈ ?B ..
  next
    assume t ∈ ?B
    then obtain x where x ∈ X and t ∈ (+) (Poly-Mapping.single x 1) ‘ deg-sect X d ..
    from this(2) obtain s where s: s ∈ deg-sect X d
      and t: t = Poly-Mapping.single x 1 + s (is t = ?s + s) ..
    show t ∈ ?A
    proof (rule deg-sectI)
      from ‹x ∈ X› have ?s ∈ .[X] by (rule PPs-closed-single)
      moreover from s have s ∈ .[X] by (rule deg-sectD)
      ultimately show t ∈ .[X] unfolding t by (rule PPs-closed-plus)
    next
      from s have deg-pm s = d by (rule deg-sectD)
      thus deg-pm t = Suc d by (simp add: t deg-pm-single deg-pm-plus)
    qed
  qed
qed

lemma deg-sect-insert:
  deg-sect (insert x X) d = (⋃d0≤d. (+) (Poly-Mapping.single x (d − d0)) ‘ deg-sect X d0)
    (is ?A = ?B)
proof (rule set-eqI)
  fix t
  show t ∈ ?A ⟷ t ∈ ?B
```

proof
  assume *t* ∈ *?A*
  hence *t* ∈ .[*insert x X*] **and** *deg-pm t* = *d* **by** (*rule deg-sectD*)+
  from *this*(*1*) **obtain** *e tx* **where** *tx* ∈ .[*X*] **and** *t*: *t* = *Poly-Mapping.single x e* + *tx*
    by (*rule PPs-insertE*)
  have *e* + *deg-pm tx* = *deg-pm t* **by** (*simp add*: *t deg-pm-plus deg-pm-single*)
  hence *e* + *deg-pm tx* = *d* **by** (*simp only*: ‹*deg-pm t* = *d*›)
  hence *deg-pm tx* ∈ {..*d*} **and** *e*: *e* = *d* − *deg-pm tx* **by** *simp-all*
  from ‹*tx* ∈ .[*X*]› *refl* **have** *tx* ∈ *deg-sect X* (*deg-pm tx*) **by** (*rule deg-sectI*)
  hence *t* ∈ (+) (*Poly-Mapping.single x* (*d* − *deg-pm tx*)) ‘ *deg-sect X* (*deg-pm tx*)
    unfolding *t e* **by** (*rule imageI*)
  with ‹*deg-pm tx* ∈ {..*d*}› **show** *t* ∈ *?B* **..**
next
  assume *t* ∈ *?B*
  then obtain *d0* **where** *d0* ∈ {..*d*} **and** *t* ∈ (+) (*Poly-Mapping.single x* (*d* − *d0*)) ‘ *deg-sect X d0* **..**
  from *this*(*2*) **obtain** *s* **where** *s*: *s* ∈ *deg-sect X d0*
    and *t*: *t* = *Poly-Mapping.single x* (*d* − *d0*) + *s* (**is** *t* = *?s* + *s*) **..**
  show *t* ∈ *?A*
  proof (*rule deg-sectI*)
    have *?s* ∈ .[*insert x X*] **by** (*rule PPs-closed-single*, *simp*)
    from *s* **have** *s* ∈ .[*X*] **by** (*rule deg-sectD*)
    also have ... ⊆ .[*insert x X*] **by** (*rule PPs-mono*, *blast*)
    finally have *s* ∈ .[*insert x X*] **.**
      with ‹*?s* ∈ .[*insert x X*]› **show** *t* ∈ .[*insert x X*] **unfolding** *t* **by** (*rule PPs-closed-plus*)
  next
    from *s* **have** *deg-pm s* = *d0* **by** (*rule deg-sectD*)
    moreover from ‹*d0* ∈ {..*d*}› **have** *d0* ≤ *d* **by** *simp*
    ultimately show *deg-pm t* = *d* **by** (*simp add*: *t deg-pm-single deg-pm-plus*)
  qed
qed
qed

lemma *deg-le-sect-Suc*: *deg-le-sect X* (*Suc d*) = *deg-le-sect X d* ∪ *deg-sect X* (*Suc d*)
  by (*simp add*: *deg-le-sect-def atMost-Suc Un-commute*)

lemma *deg-le-sect-Suc-2*:
  deg-le-sect X (*Suc d*) = *insert 0* (⋃*x*∈*X*. (+) (*Poly-Mapping.single x 1*) ‘ *deg-le-sect X d*)
    (**is** *?A* = *?B*)
proof −
  have *eq1*: {*Suc 0*..*Suc d*} = *Suc* ‘ {..*d*} **by** (*simp add*: *image-Suc-atMost*)
  have *insert 0* {*1*..*Suc d*} = {..*Suc d*} **by** *fastforce*
  hence *?A* = (⋃ *d0*∈*insert 0* {*1*..*Suc d*}. *deg-sect X d0*) **by** (*simp add*: *deg-le-sect-def*)
  also have ... = *insert 0* (⋃ *d0*≤*d*. *deg-sect X* (*Suc d0*)) **by** (*simp add*: *eq1*)

also have ... = *insert 0* ($\bigcup$*d0*≤*d*. ($\bigcup$*x*∈*X*. (+) (*Poly-Mapping.single x 1*) ‘ *deg-sect X d0*))
  by (*simp only*: *deg-sect-Suc*)
 also have ... = *insert 0* ($\bigcup$*x*∈*X*. (+) (*Poly-Mapping.single x 1*) ‘ ($\bigcup$*d0*≤*d*. *deg-sect X d0*))
  by *fastforce*
 also have ... = *?B* **by** (*simp only*: *deg-le-sect-def*)
 finally show *?thesis* **.**
qed

lemma *finite-deg-sect*:
 assumes *finite X*
 shows *finite* ((*deg-sect X d*)::(′*x*::*countable* $\Rightarrow_0$ *nat*) *set*)
proof (*induct d*)
 case *0*
 show *?case* **by** *simp*
next
 case (*Suc d*)
 with *assms* **show** *?case* **by** (*simp add*: *deg-sect-Suc*)
qed

corollary *finite-deg-le-sect*: *finite X* ⟹ *finite* ((*deg-le-sect X d*)::(′*x*::*countable* $\Rightarrow_0$ *nat*) *set*)
 by (*simp add*: *deg-le-sect-def finite-deg-sect*)

lemma *keys-subset-deg-le-sectI*:
 assumes *p* ∈ *P*[*X*] **and** *poly-deg p* ≤ *d*
 shows *keys p* ⊆ *deg-le-sect X d*
proof
 fix *t*
 assume *t* ∈ *keys p*
 also from *assms*(*1*) **have** ... ⊆ .[*X*] **by** (*rule PolysD*)
 finally have *t* ∈ .[*X*] **.**
 from ‹*t* ∈ *keys p*› **have** *deg-pm t* ≤ *poly-deg p* **by** (*rule poly-deg-max-keys*)
 from *this assms*(*2*) **have** *deg-pm t* ≤ *d* **by** (*rule le-trans*)
 with ‹*t* ∈ .[*X*]› **show** *t* ∈ *deg-le-sect X d* **by** (*rule deg-le-sectI*)
qed

lemma *binomial-symmetric-plus*: (*n* + *k*) *choose n* = (*n* + *k*) *choose k*
 by (*metis add-diff-cancel-left′ binomial-symmetric le-add1*)

lemma *card-deg-sect*:
 assumes *finite X* **and** *X* ≠ {}
 shows *card* (*deg-sect X d*) = (*d* + (*card X* − *1*)) *choose* (*card X* − *1*)
 using *assms*
proof (*induct X arbitrary*: *d*)
 case *empty*
 thus *?case* **by** *simp*
next

```
  case (insert x X)
  from insert(1, 2) have eq1: card (insert x X) = Suc (card X) by simp
  show ?case
  proof (cases X = {})
    case True
    thus ?thesis by simp
  next
    case False
    with insert.hyps(1) have 0 < card X by (simp add: card-gt-0-iff)
    let ?f = λd0. Poly-Mapping.single x (d − d0)
    from False have eq2: card (deg-sect X d0) = d0 + (card X − 1) choose (card
X − 1) for d0
      by (rule insert.hyps)
    have finite {..d} by simp
    moreover from insert.hyps(1) have ∀ d0∈{..d}. finite ((+) (?f d0) ‘ deg-sect
X d0)
      by (simp add: finite-deg-sect)
    moreover have ∀ d0∈{..d}. ∀ d1∈{..d}. d0 ≠ d1 ⟶
                    ((+) (?f d0) ‘ deg-sect X d0) ∩ ((+) (?f d1) ‘ deg-sect X d1)
= {}
    proof (intro ballI impI, rule ccontr)
      fix d1 d2 :: nat
      assume d1 ≠ d2
      assume ((+) (?f d1) ‘ deg-sect X d1) ∩ ((+) (?f d2) ‘ deg-sect X d2) ≠ {}
       then obtain t where t ∈ ((+) (?f d1) ‘ deg-sect X d1) ∩ ((+) (?f d2) ‘
deg-sect X d2)
        by blast
      hence t1: t ∈ (+) (?f d1) ‘ deg-sect X d1 and t2: t ∈ (+) (?f d2) ‘ deg-sect
X d2 by simp-all
      from t1 obtain s1 where s1 ∈ deg-sect X d1 and s1: t = ?f d1 + s1 ..
      from this(1) have s1 ∈ .[X] by (rule deg-sectD)
      hence keys s1 ⊆ X by (rule PPsD)
      with insert.hyps(2) have eq3: lookup s1 x = 0 by (auto simp: in-keys-iff)
      from t2 obtain s2 where s2 ∈ deg-sect X d2 and s2: t = ?f d2 + s2 ..
      from this(1) have s2 ∈ .[X] by (rule deg-sectD)
      hence keys s2 ⊆ X by (rule PPsD)
      with insert.hyps(2) have eq4: lookup s2 x = 0 by (auto simp: in-keys-iff)
     from s2 have lookup (?f d1 + s1) x = lookup (?f d2 + s2) x by (simp only:
s1)
      hence d − d1 = d − d2 by (simp add: lookup-add eq3 eq4)
      moreover assume d1 ∈ {..d} and d2 ∈ {..d}
      ultimately have d1 = d2 by simp
      with ‹d1 ≠ d2› show False ..
    qed
    ultimately have card (deg-sect (insert x X) d) =
                  (∑ d0≤d. card ((+) (monomial (d − d0) x) ‘ deg-sect X d0))
      unfolding deg-sect-insert by (rule card-UN-disjoint)
    also from refl have ... = (∑ d0≤d. card (deg-sect X d0))
    proof (rule sum.cong)
```

fix *d0*
have *inj-on* ((+) (*monomial* (*d* − *d0*) *x*)) (*deg-sect X d0*) **by** (*rule*, *rule add-left-imp-eq*)
thus *card* ((+) (*monomial* (*d* − *d0*) *x*) ' *deg-sect X d0*) = *card* (*deg-sect X d0*)
by (*rule card-image*)
qed
also have ... = ($\sum$ *d0*≤*d*. (*card X* − *1*) + *d0 choose* (*card X* − *1*)) **by** (*simp only*: *eq2 add.commute*)
also have ... = ($\sum$ *d0*≤*d*. (*card X* − *1*) + *d0 choose d0*) **by** (*simp only*: *binomial-symmetric-plus*)
also have ... = *Suc* ((*card X* − *1*) + *d*) *choose d* **by** (*rule sum-choose-lower*)
also from ‹*0* < *card X*› **have** ... = *d* + (*card* (*insert x X*) − *1*) *choose d*
by (*simp add*: *eq1 add.commute*)
also have ... = *d* + (*card* (*insert x X*) − *1*) *choose* (*card* (*insert x X*) − *1*)
by (*fact binomial-symmetric-plus*)
finally show *?thesis* **.**
qed
qed

corollary *card-deg-sect-Suc*:
assumes *finite X*
shows *card* (*deg-sect X* (*Suc d*)) = (*d* + *card X*) *choose* (*Suc d*)
proof (*cases X* = {})
case *True*
thus *?thesis* **by** (*simp add*: *deg-sect-empty*)
next
case *False*
with *assms* **have** *0* < *card X* **by** (*simp add*: *card-gt-0-iff*)
from *assms False* **have** *card* (*deg-sect X* (*Suc d*)) = (*Suc d* + (*card X* − *1*)) *choose* (*card X* − *1*)
by (*rule card-deg-sect*)
also have ... = (*Suc d* + (*card X* − *1*)) *choose* (*Suc d*) **by** (*rule sym*, *rule binomial-symmetric-plus*)
also from ‹*0* < *card X*› **have** ... = (*d* + *card X*) *choose* (*Suc d*) **by** *simp*
finally show *?thesis* **.**
qed

corollary *card-deg-le-sect*:
assumes *finite X*
shows *card* (*deg-le-sect X d*) = (*d* + *card X*) *choose card X*
proof (*induct d*)
case *0*
show *?case* **by** *simp*
next
case (*Suc d*)
from *assms* **have** *finite* (*deg-le-sect X d*) **by** (*rule finite-deg-le-sect*)
moreover from *assms* **have** *finite* (*deg-sect X* (*Suc d*)) **by** (*rule finite-deg-sect*)
moreover from *lessI* **have** *deg-le-sect X d* ∩ *deg-sect X* (*Suc d*) = {}

by (*rule deg-le-sect-deg-sect-disjoint*)
ultimately have *card* (*deg-le-sect* X (*Suc* d)) = *card* (*deg-le-sect* X d) + *card* (*deg-sect* X (*Suc* d))
unfolding *deg-le-sect-Suc* **by** (*rule card-Un-disjoint*)
also from *assms* **have** ... = (*Suc* d + *card* X) *choose Suc* d
by (*simp add*: *Suc.hyps card-deg-sect-Suc binomial-symmetric-plus*[*of* d])
also have ... = (*Suc* d + *card* X) *choose card* X **by** (*rule binomial-symmetric-plus*)
finally show *?case* **.**
qed

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
unfolding *is-GB-cofactor-bound-def* **using** *assms* **by** *blast*

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 $-$
  let $?X = \bigcup(indets\ `\ F)$
  from *assms* **obtain** G **where** *punit.is-Groebner-basis* G **and** $ideal\ G = ideal\ F$ **and** $\bigcup(indets\ `\ G) \subseteq ?X$
    and 1: $\bigwedge g.\ g \in G \Longrightarrow \exists F'\ q.\ finite\ F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\ f * f) \wedge (\forall f{\in}F'.\ poly\text{-}deg\ (q\ f * f) \leq b)$
    by (*auto simp*: *is-GB-cofactor-bound-def*)
  from *this*(*1*, *2*, *3*) **show** *?thesis*
  proof
    fix g
    assume $g \in G$
    show $\exists F'\ q.\ finite\ F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\ f * f) \wedge$
        $(\forall f.\ indets\ (q\ f) \subseteq ?X \wedge poly\text{-}deg\ (q\ f * f) \leq b \wedge (f \notin F' \longrightarrow q\ f = 0))$
    proof (*cases* $g = 0$)
      case *True*
      define q **where** $q = (\lambda f{::}('x \Rightarrow_0 nat) \Rightarrow_0 'b.\ 0{::}('x \Rightarrow_0 nat) \Rightarrow_0 'b)$
      show *?thesis*
      proof (*intro exI conjI allI*)
        show $g = (\sum f{\in}\{\}.\ q\ f * f)$ **by** (*simp add*: *True q-def*)
      qed (*simp-all add*: *q-def*)
    next
      case *False*
      let $?X = \bigcup(indets\ `\ F)$
      from ‹$g \in G$› **have** $\exists F'\ q.\ finite\ F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\ f * f) \wedge (\forall f{\in}F'.\ poly\text{-}deg\ (q\ f * f) \leq b)$
        by (*rule 1*)
      then obtain F' $q0$ **where** *finite* F' **and** $F' \subseteq F$ **and** g: $g = (\sum f{\in}F'.\ q0\ f * f)$
        and $q0$: $\bigwedge f.\ f \in F' \Longrightarrow poly\text{-}deg\ (q0\ f * f) \leq b$ **by** *blast*
      define *sub* **where** $sub = (\lambda x{::}'x.\ if\ x \in ?X\ then$
                    $monomial\ (1{::}'b)\ (Poly\text{-}Mapping.single\ x\ (1{::}nat))$
                    $else\ 1)$
      have 1: $sub\ x = monomial\ 1\ (monomial\ 1\ x)$ **if** $x \in indets\ g$ **for** x
      proof (*simp add*: *sub-def*, *rule*)
        from *that* ‹$g \in G$› **have** $x \in \bigcup(indets\ `\ G)$ **by** *blast*
        also have $\ldots \subseteq ?X$ **by** *fact*
        finally obtain f **where** $f \in F$ **and** $x \in indets\ f$ **..**
        assume $\forall f{\in}F.\ x \notin indets\ f$
        hence $x \notin indets\ f$ **using** ‹$f \in F$› **..**
        thus $monomial\ 1\ (monomial\ (Suc\ 0)\ x) = 1$ **using** ‹$x \in indets\ f$› **..**
      qed
      have 2: $sub\ x = monomial\ 1\ (monomial\ 1\ x)$ **if** $f \in F'$ **and** $x \in indets\ f$ **for** $f\ x$
      proof (*simp add*: *sub-def*, *rule*)
        assume $\forall f{\in}F.\ x \notin indets\ f$
        moreover from *that*(*1*) ‹$F' \subseteq F$› **have** $f \in F$ **..**
        ultimately have $x \notin indets\ f$ **..**

thus *monomial 1* (*monomial* (*Suc 0*) *x*) = *1* **using** *that*(*2*) **..**
qed
have *3*: *poly-subst sub f* = *f* **if** *f* ∈ *F′* **for** *f* **by** (*rule poly-subst-id*, *rule 2*, *rule that*)
define *q* **where** *q* = (λ*f*. *if f* ∈ *F′ then poly-subst sub* (*q0 f*) *else 0*)
show *?thesis*
proof (*intro exI allI conjI impI*)
from *1* **have** *g* = *poly-subst sub g* **by** (*rule poly-subst-id*[*symmetric*])
also have ... = (∑*f*∈*F′*. *q f* ∗ (*poly-subst sub f*))
by (*simp add*: *g poly-subst-sum poly-subst-times q-def cong*: *sum.cong*)
also from *refl* **have** ... = (∑*f*∈*F′*. *q f* ∗ *f*)
proof (*rule sum.cong*)
fix *f*
assume *f* ∈ *F′*
hence *poly-subst sub f* = *f* **by** (*rule 3*)
thus *q f* ∗ *poly-subst sub f* = *q f* ∗ *f* **by** *simp*
qed
finally show *g* = (∑*f*∈*F′*. *q f* ∗ *f*) **.**
next
fix *f*
have *indets* (*q f*) ⊆ *?X* ∧ *poly-deg* (*q f* ∗ *f*) ≤ *b*
proof (*cases f* ∈ *F′*)
case *True*
hence *qf*: *q f* = *poly-subst sub* (*q0 f*) **by** (*simp add*: *q-def*)
show *?thesis*
proof
show *indets* (*q f*) ⊆ *?X*
proof
fix *x*
assume *x* ∈ *indets* (*q f*)
then obtain *y* **where** *x* ∈ *indets* (*sub y*) **unfolding** *qf* **by** (*rule in-indets-poly-substE*)
hence *y*: *y* ∈ *?X* **and** *x* ∈ *indets* (*monomial* (*1*::′*b*) (*monomial* (*1*::*nat*) *y*))
by (*simp-all add*: *sub-def split*: *if-splits*)
from *this*(*2*) **have** *x* = *y* **by** (*simp add*: *indets-monomial*)
with *y* **show** *x* ∈ *?X* **by** (*simp only*:)
qed
next
from ‹*f* ∈ *F′*› **have** *poly-subst sub f* = *f* **by** (*rule 3*)
hence *poly-deg* (*q f* ∗ *f*) = *poly-deg* (*q f* ∗ *poly-subst sub f*) **by** (*simp only*:)
also have ... = *poly-deg* (*poly-subst sub* (*q0 f* ∗ *f*)) **by** (*simp only*: *qf poly-subst-times*)
also have ... ≤ *poly-deg* (*q0 f* ∗ *f*)
proof (*rule poly-deg-poly-subst-le*)
fix *x*
show *poly-deg* (*sub x*) ≤ *1* **by** (*simp add*: *sub-def poly-deg-monomial deg-pm-single*)

qed
also from ‹$f \in F'$› **have** $\ldots \leq b$ **by** (*rule q0*)
finally show *poly-deg* $(q\ f * f) \leq b$ **.**
qed
next
case *False*
thus *?thesis* **by** (*simp add*: *q-def*)
qed
thus *indets* $(q\ f) \subseteq$ *?X* **and** *poly-deg* $(q\ f * f) \leq b$ **by** *simp-all*

assume $f \notin F'$
thus $q\ f = 0$ **by** (*simp add*: *q-def*)
qed *fact+*
qed
qed
qed

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 −
let *?X* $= \bigcup($*indets* ‘ $F)$
have *?X* $\subseteq X$
proof
fix x
assume $x \in$ *?X*
then obtain f **where** $f \in F$ **and** $x \in$ *indets* f **..**
from *this*(1) *assms*(2) **have** $f \in P[X]$ **..**
hence *indets* $f \subseteq X$ **by** (*rule PolysD*)
with ‹$x \in$ *indets* f› **show** $x \in X$ **..**
qed
from *assms*(1) **obtain** G **where** *punit.is-Groebner-basis G* **and** *ideal* $G =$ *ideal* F
and 1: $\bigcup($*indets* ‘ $G) \subseteq$ *?X*
and 2: $\bigwedge g.\ g \in G \Longrightarrow \exists F'\ q.$ *finite* $F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\ f * f) \wedge$
$(\forall f.$ *indets* $(q\ f) \subseteq$ *?X* $\wedge$ *poly-deg* $(q\ f * f) \leq b \wedge (f \notin F' \longrightarrow q\ f = 0))$
by (*rule is-GB-cofactor-boundE*) *blast*
from *this*(1, 2) **show** *?thesis*
proof
show $G \subseteq P[X]$
proof
fix g
assume $g \in G$

hence *indets* $g \subseteq \bigcup($*indets* ‘ $G)$ **by** *blast*
also have ... $\subseteq$ *?X* **by** *fact*
also have ... $\subseteq X$ **by** *fact*
finally show $g \in P[X]$ **by** (*rule PolysI-alt*)
qed
next
fix g
assume $g \in G$
hence $\exists F'$ q. *finite* $F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\ f * f) \wedge$
$(\forall f.$ *indets* $(q\ f) \subseteq$ *?X* $\wedge$ *poly-deg* $(q\ f * f) \leq b \wedge (f \notin F' \longrightarrow q\ f$
$= 0))$
by (*rule 2*)
then obtain F' q **where** *finite* F' **and** $F' \subseteq F$ **and** $g = (\sum f{\in}F'.\ q\ f * f)$
and $\bigwedge f$. *indets* $(q\ f) \subseteq$ *?X* **and** $\bigwedge f$. *poly-deg* $(q\ f * f) \leq b$ **and** $\bigwedge f.\ f \notin F'$
$\Longrightarrow q\ f = 0$
by *blast*
show $\exists F'$ 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 (*intro exI allI conjI impI*)
fix f
from ‹*indets* $(q\ f) \subseteq$ *?X*› ‹*?X* $\subseteq X$› **have** *indets* $(q\ f) \subseteq X$ **by** (*rule*
subset-trans)
thus $q\ f \in P[X]$ **by** (*rule PolysI-alt*)
qed *fact+*
qed
qed

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 −
from *assms*(*1*, *3*) **obtain** G **where** *punit.is-Groebner-basis G* **and** *ideal G =*
ideal F **and** $G \subseteq P[X]$
and *1*: $\bigwedge g.\ g \in G \Longrightarrow \exists F'$ 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* $(q\ f * f) \leq b \wedge (f \notin F' \longrightarrow q\ f$
$= 0))$
by (*rule is-GB-cofactor-boundE-Polys*) *blast*
from *this*(*1*, *2*, *3*) **show** *?thesis*
proof
fix g
assume $g \in G$
hence $\exists F'$ 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* $(q\ f * f) \leq b \wedge (f \notin F' \longrightarrow q\ f$
$= 0))$
by (*rule 1*)

then obtain F' *q* **where** $F' \subseteq F$ **and** *g*: $g = (\sum f{\in}F'.\ q\ f * f)$
and $\bigwedge f.\ q\ f \in P[X]$ **and** $\bigwedge f.$ *poly-deg* $(q\ f * f) \leq b$ **and** *2*: $\bigwedge f.\ f \notin F' \Longrightarrow q\ f = 0$ **by** *blast*
show $\exists\, q.\ g = (\sum f{\in}F.\ q\ f * f) \wedge (\forall f.\ q\ f \in P[X] \wedge$ *poly-deg* $(q\ f * f) \leq b)$
proof (*intro exI conjI impI allI*)
from *assms*(*2*) ‹$F' \subseteq F$› **have** $(\sum f{\in}F'.\ q\ f * f) = (\sum f{\in}F.\ q\ f * f)$
proof (*intro sum.mono-neutral-left ballI*)
fix *f*
assume $f \in F - F'$
hence $f \notin F'$ **by** *simp*
hence $q\ f = 0$ **by** (*rule 2*)
thus $q\ f * f = 0$ **by** *simp*
qed
thus $g = (\sum f{\in}F.\ q\ f * f)$ **by** (*simp only*: *g*)
qed *fact*+
qed
qed

lemma *is-GB-cofactor-boundI-subset-zero*:
assumes $F \subseteq \{0\}$
shows *is-GB-cofactor-bound F b*
using *punit.is-Groebner-basis-empty*
proof (*rule is-GB-cofactor-boundI*)
from *assms* **show** *ideal* {} = *ideal F* **by** (*metis ideal.span-empty ideal-eq-zero-iff*)
qed *simp-all*

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*
unfolding *is-hom-GB-bound-def* **by** *blast*

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* $g \leq b$
unfolding *is-hom-GB-bound-def* **by** *blast*

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 −
let *?F* = *homogenize None ' extend-indets ' F*
define *Y* **where** $Y = \bigcup$ (*indets ' F*)
define *G* **where** *G* = *restrict-indets ' (extended-ord.punit.reduced-GB ?F)*
have $Y \subseteq X$
proof

fix x
assume $x \in Y$
then obtain f **where** $f \in F$ **and** $x \in$ *indets* f **unfolding** *Y-def* **..**
from *this*(*1*) *assms*(*2*) **have** $f \in P[X]$ **..**
hence *indets* $f \subseteq X$ **by** (*rule PolysD*)
with ‹$x \in$ *indets* f› **show** $x \in X$ **..**
qed
hence *finite* Y **using** *assms*(*1*) **by** (*rule finite-subset*)
moreover have $F \subseteq P[Y]$ **by** (*auto simp*: *Y-def Polys-alt*)
ultimately have *punit.is-Groebner-basis* G **and** *ideal* G = *ideal* F **and** $G \subseteq P[Y]$
unfolding *G-def* **by** (*rule restrict-indets-reduced-GB*)+
from *this*(*1*, *2*) **show** *?thesis*
proof (*rule is-GB-cofactor-boundI*)
from ‹$G \subseteq P[Y]$› **show** $\bigcup$ (*indets* ‘ G) $\subseteq \bigcup$ (*indets* ‘ F) **by** (*auto simp*: *Y-def Polys-alt*)
next
fix g
assume $g \in G$
then obtain g' **where** g': $g' \in$ *extended-ord.punit.reduced-GB ?F*
and g: g = *restrict-indets* g' **unfolding** *G-def* **..**
have $f \in$ *?F* $\Longrightarrow$ *homogeneous* f **for** f **by** (*auto simp*: *homogeneous-homogenize*)
with *assms*(*3*) **have** *poly-deg* $g' \leq b$ **using** g' **by** (*rule extended-ord.is-hom-GB-boundD*)
from g' **have** $g' \in$ *ideal* (*extended-ord.punit.reduced-GB ?F*) **by** (*rule ideal.span-base*)
also have ... = *ideal ?F*
proof (*rule extended-ord.reduced-GB-ideal-Polys*)
from ‹*finite* Y› **show** *finite* (*insert None* (*Some* ‘ Y)) **by** *simp*
next
show *?F* $\subseteq P$[*insert None* (*Some* ‘ Y)]
proof
fix *f0*
assume *f0* $\in$ *?F*
then obtain f **where** $f \in F$ **and** *f0*: *f0* = *homogenize None* (*extend-indets* f) **by** *blast*
from *this*(*1*) ‹$F \subseteq P[Y]$› **have** $f \in P[Y]$ **..**
hence *extend-indets* $f \in P$[*Some* ‘ Y] **by** (*auto simp*: *indets-extend-indets Polys-alt*)
thus *f0* $\in P$[*insert None* (*Some* ‘ Y)] **unfolding** *f0* **by** (*rule homogenize-in-Polys*)
qed
qed
finally have $g' \in$ *ideal ?F* **.**
with ‹$\bigwedge f$. $f \in$ *?F* $\Longrightarrow$ *homogeneous* f› **obtain** *F0* q **where** *finite F0* **and** *F0* $\subseteq$ *?F*
and g': $g' = (\sum f \in F0.\ q\ f * f)$ **and** *deg-le*: $\bigwedge f$. *poly-deg* $(q\ f * f) \leq$ *poly-deg* g'
by (*rule homogeneous-idealE*) *blast*+
from *this*(*2*) **obtain** F' **where** $F' \subseteq F$ **and** *F0*: *F0* = *homogenize None* ‘ *extend-indets* ‘ F'

and *inj-on*: *inj-on* (*homogenize None* ∘ *extend-indets*) F'
unfolding *image-comp* **by** (*rule subset-imageE-inj*)
show $\exists F'$ *q*. *finite* $F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\ f * f) \wedge (\forall f{\in}F'.$ *poly-deg* $(q\ f * f) \leq b)$
proof (*intro exI conjI ballI*)
from *inj-on* ‹*finite F0*› **show** *finite* F' **by** (*simp only*: *finite-image-iff F0 image-comp*)
next
from *inj-on* **show** $g = (\sum f{\in}F'.$ (*restrict-indets* ∘ *q* ∘ *homogenize None* ∘ *extend-indets*) $f * f)$
by (*simp add*: *g g′ F0 restrict-indets-sum restrict-indets-times sum.reindex image-comp o-def*)
next
fix *f*
assume $f \in F'$
have *poly-deg* ((*restrict-indets* ∘ *q* ∘ *homogenize None* ∘ *extend-indets*) $f * f$) =
poly-deg (*restrict-indets* (*q* (*homogenize None* (*extend-indets f*)) ∗ *homogenize None* (*extend-indets f*)))
by (*simp add*: *restrict-indets-times*)
also have ... ≤ *poly-deg* (*q* (*homogenize None* (*extend-indets f*)) ∗ *homogenize None* (*extend-indets f*))
by (*rule poly-deg-restrict-indets-le*)
also have ... ≤ *poly-deg g′* **by** (*rule deg-le*)
also have ... ≤ *b* **by** *fact*
finally show *poly-deg* ((*restrict-indets* ∘ *q* ∘ *homogenize None* ∘ *extend-indets*) $f * f) \leq b$ **.**
qed *fact*
qed
qed

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** $G \subseteq$ *phull* (*set ps*)
shows *is-Groebner-basis* (*set* (*Macaulay-list ps*))
proof (*simp only*: *GB-alt-3-finite*[*OF finite-set*] *pmdl-Macaulay-list*, *intro ballI impI*)
fix *f*
assume $f \in$ *pmdl* (*set ps*)
also from *assms*(*2*) **have** ... = *pmdl G* **.**
finally have $f \in$ *pmdl G* **.**
assume $f \neq 0$
with *assms*(*1*) ‹$f \in$ *pmdl G*› **obtain** *g* **where** $g \in G$ **and** $g \neq 0$ **and** *lt g* $adds_t$ *lt f*
by (*rule GB-adds-lt*)
from *assms*(*3*) ‹$g \in G$› **have** $g \in$ *phull* (*set ps*) **..**
from *this* ‹$g \neq 0$› **obtain** g' **where** $g' \in$ *set* (*Macaulay-list ps*) **and** $g' \neq 0$ **and** *lt g* = *lt* g'
by (*rule Macaulay-list-lt*)
show $\exists$ *g*∈*set* (*Macaulay-list ps*). $g \neq 0$ ∧ *lt g* $adds_t$ *lt f*
proof (*rule*, *rule*)
from ‹*lt g* $adds_t$ *lt f*› **show** *lt* g' $adds_t$ *lt f* **by** (*simp only*: ‹*lt g* = *lt* g'›)
qed *fact+*
qed

4.2 Bounds

context *pm-powerprod*
begin

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

definition *deg-shifts* :: *nat* ⇒ ((*'x* ⇒$_0$ *nat*) ⇒$_0$ *'b*) *list* ⇒ ((*'x* ⇒$_0$ *nat*) ⇒$_0$ *'b*::*semiring-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 −
from *fin-X* **have** *finite* (*deg-le-sect X d0*) **for** *d0* **by** (*rule finite-deg-le-sect*)
thus *?thesis* **by** (*simp add*: *deg-shifts-def punit.set-pps-to-list*)
qed

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))
  by (*simp add*: *set-deg-shifts*)

lemma *deg-shifts-superset*: *set fs* ⊆ *set* (*deg-shifts d fs*)
proof −
  have *set fs* = (⋃*f*∈*set fs*. {*punit.monom-mult 1 0 f*}) **by** *simp*
  also have … ⊆ *set* (*deg-shifts d fs*) **unfolding** *set-deg-shifts* **using** *subset-refl*
  proof (*rule UN-mono*)
    fix *f*
    assume *f* ∈ *set fs*
    have *punit.monom-mult 1 0 f* ∈ (λ*t*. *punit.monom-mult 1 t f*) ‘ *deg-le-sect X* (*d* − *poly-deg f*)
      using *zero-in-deg-le-sect* **by** (*rule imageI*)
    thus {*punit.monom-mult 1 0 f*} ⊆ (λ*t*. *punit.monom-mult 1 t f*) ‘ *deg-le-sect X* (*d* − *poly-deg f*)
      by *simp*
  qed
  finally show *?thesis* **.**
qed

lemma *deg-shifts-mono*:
  assumes *set fs* ⊆ *set gs*
  shows *set* (*deg-shifts d fs*) ⊆ *set* (*deg-shifts d gs*)
  using *assms* **by** (*auto simp add*: *set-deg-shifts*)

lemma *ideal-deg-shifts* [*simp*]: *ideal* (*set* (*deg-shifts d fs*)) = *ideal* (*set fs*)
proof
  show *ideal* (*set* (*deg-shifts d fs*)) ⊆ *ideal* (*set fs*)
    by (*rule ideal.span-subset-spanI*, *simp add*: *set-deg-shifts UN-subset-iff*,
      intro ballI image-subsetI) (*metis ideal.span-scale times-monomial-left ideal.span-base*)
next
  from *deg-shifts-superset* **show** *ideal* (*set fs*) ⊆ *ideal* (*set* (*deg-shifts d fs*))
    by (*rule ideal.span-mono*)
qed

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 −
  from *assms*(*2*) *finite-set assms*(*1*) **obtain** *G* **where** *punit.is-Groebner-basis G*
    and *ideal-G*: *ideal G* = *ideal* (*set fs*) **and** *G-sub*: *G* ⊆ *P*[*X*]
    and *1*: ⋀*g*. *g* ∈ *G* ⟹ ∃ *q*. *g* = (∑*f*∈*set fs*. *q f* ∗ *f*) ∧ (∀*f*. *q f* ∈ *P*[*X*] ∧ *poly-deg* (*q f* ∗ *f*) ≤ *b*)
    by (*rule is-GB-cofactor-boundE-finite-Polys*) *blast*
  from *this*(*1*) **show** *?thesis*
  proof (*rule punit.Macaulay-list-is-GB*)
    show *G* ⊆ *phull* (*set* (*deg-shifts b fs*)) (**is** - ⊆ *?H*)
    proof
      fix *g*

assume $g \in G$
hence $\exists\, q.\ g = (\sum f{\in}set\ fs.\ q\ f * f) \wedge (\forall f.\ q\ f \in P[X] \wedge poly\text{-}deg\ (q\ f * f) \leq b)$ **by** (*rule 1*)
then obtain q **where** g: $g = (\sum f{\in}set\ fs.\ q\ f * f)$ **and** $\bigwedge f.\ q\ f \in P[X]$
and $\bigwedge f.\ poly\text{-}deg\ (q\ f * f) \leq b$ **by** *blast*
show $g \in ?H$ **unfolding** g
proof (*rule phull.span-sum*)
fix f
assume $f \in set\ fs$
have $1 \neq (0{::}'a)$ **by** *simp*
show $q\ f * f \in ?H$
proof (*cases* $f = 0 \vee q\ f = 0$)
case *True*
thus *?thesis* **by** (*auto simp add*: *phull.span-zero*)
next
case *False*
hence $q\ f \neq 0$ **and** $f \neq 0$ **by** *simp-all*
with ‹$poly\text{-}deg\ (q\ f * f) \leq b$› **have** $poly\text{-}deg\ (q\ f) \leq b - poly\text{-}deg\ f$
by (*simp add*: *poly-deg-times*)
with ‹$q\ f \in P[X]$› **have** $keys\ (q\ f) \subseteq deg\text{-}le\text{-}sect\ X\ (b - poly\text{-}deg\ f)$
by (*rule keys-subset-deg-le-sectI*)
with *finite-deg-le-sect*[*OF fin-X*]
have $q\ f * f = (\sum t{\in}deg\text{-}le\text{-}sect\ X\ (b - poly\text{-}deg\ f).\ punit.monom\text{-}mult\ (lookup\ (q\ f)\ t)\ t\ f)$
unfolding *punit.mult-scalar-sum-monomials*[*simplified*]
by (*rule sum.mono-neutral-left*) (*simp add*: *in-keys-iff*)
also have $\ldots = (\sum t{\in}deg\text{-}le\text{-}sect\ X\ (b - poly\text{-}deg\ f).$
$(lookup\ (q\ f)\ t) \cdot (punit.monom\text{-}mult\ 1\ t\ f))$
by (*simp add*: *punit.monom-mult-assoc punit.map-scale-eq-monom-mult*)
also have $\ldots = (\sum t{\in}deg\text{-}le\text{-}sect\ X\ (b - poly\text{-}deg\ f).$
$((\lambda f0.\ (lookup\ (q\ f)\ (punit.lp\ f0 - punit.lp\ f)) \cdot f0) \circ$
$(\lambda t.\ punit.monom\text{-}mult\ 1\ t\ f))\ t)$
using *refl* **by** (*rule sum.cong*) (*simp add*: *punit.lt-monom-mult*[*OF* ‹$1 \neq 0$› ‹$f \neq 0$›])
also have $\ldots = (\sum f0{\in}set\ (deg\text{-}shifts\ b\ [f]).\ (lookup\ (q\ f)\ (punit.lp\ f0 - punit.lp\ f)) \cdot f0)$
unfolding *set-deg-shifts-singleton*
proof (*intro sum.reindex*[*symmetric*] *inj-onI*)
fix $s\ t$
assume $punit.monom\text{-}mult\ 1\ s\ f = punit.monom\text{-}mult\ 1\ t\ f$
thus $s = t$ **using** ‹$1 \neq 0$› ‹$f \neq 0$› **by** (*rule punit.monom-mult-inj-2*)
qed
finally have $q\ f * f \in phull\ (set\ (deg\text{-}shifts\ b\ [f]))$
by (*simp add*: *phull.sum-in-spanI*)
also have $\ldots \subseteq ?H$ **by** (*rule phull.span-mono*, *rule deg-shifts-mono*, *simp add*: ‹$f \in set\ fs$›)
finally show *?thesis* **.**
qed
qed

qed
qed (*simp add*: *ideal-G*)
qed

lemma *thm-2-3-7*:
assumes *set fs* $\subseteq$ *P*[*X*] **and** *is-GB-cofactor-bound* (*set fs*) *b*
shows *1* $\in$ *ideal* (*set fs*) $\longleftrightarrow$ *1* $\in$ *set* (*punit.Macaulay-list* (*deg-shifts b fs*)) (**is** *?L* $\longleftrightarrow$ *?R*)
proof
assume *?L*
let *?G* = *set* (*punit.Macaulay-list* (*deg-shifts b fs*))
from *assms* **have** *punit.is-Groebner-basis ?G* **by** (*rule thm-2-3-6*)
moreover from ‹*?L*› **have** *1* $\in$ *ideal ?G* **by** (*simp add*: *punit.pmdl-Macaulay-list*[*simplified*])
moreover have *1* $\neq$ (*0*::- $\Rightarrow_0$ *'a*) **by** *simp*
ultimately obtain *g* **where** *g* $\in$ *?G* **and** *g* $\neq$ *0* **and** *punit.lt g adds punit.lt* (*1*::- $\Rightarrow_0$ *'a*)
by (*rule punit.GB-adds-lt*[*simplified*])
from *this*(*3*) **have** *lp-g*: *punit.lt g* = *0* **by** (*simp add*: *punit.lt-monomial adds-zero flip*: *single-one*)
from *punit.Macaulay-list-is-monic-set* ‹*g* $\in$ *?G*› ‹*g* $\neq$ *0*› **have** *lc-g*: *punit.lc g* = *1*
by (*rule punit.is-monic-setD*)
have *g* = *1*
proof (*rule poly-mapping-eqI*)
fix *t*
show *lookup g t* = *lookup 1 t*
proof (*cases t* = *0*)
case *True*
thus *?thesis* **using** *lc-g* **by** (*simp add*: *lookup-one punit.lc-def lp-g*)
next
case *False*
with *zero-min*[*of t*] **have** $\neg$ *t* $\preceq$ *punit.lt g* **by** (*simp add*: *lp-g*)
with *punit.lt-max-keys* **have** *t* $\notin$ *keys g* **by** *blast*
with *False* **show** *?thesis* **by** (*simp add*: *lookup-one in-keys-iff*)
qed
qed
with ‹*g* $\in$ *?G*› **show** *1* $\in$ *?G* **by** *simp*
next
assume *?R*
also have ... $\subseteq$ *phull* (*set* (*punit.Macaulay-list* (*deg-shifts b fs*)))
by (*rule phull.span-superset*)
also have ... = *phull* (*set* (*deg-shifts b fs*)) **by** (*fact punit.phull-Macaulay-list*)
also have ... $\subseteq$ *ideal* (*set* (*deg-shifts b fs*)) **using** *punit.phull-subset-module* **by** *force*
finally show *?L* **by** *simp*
qed

end

lemma *thm-2-3-6-indets*:
assumes *is-GB-cofactor-bound* (*set fs*) *b*
shows *punit.is-Groebner-basis* (*set* (*punit.Macaulay-list* (*deg-shifts* ($\bigcup$(*indets* ‘ (*set fs*))) *b fs*)))
using - - *assms*
proof (*rule thm-2-3-6*)
from *finite-set* **show** *finite* ($\bigcup$(*indets* ‘ (*set fs*))) **by** (*simp add*: *finite-indets*)
next
show *set fs* $\subseteq$ *P*[$\bigcup$(*indets* ‘ (*set fs*))] **by** (*auto simp*: *Polys-alt*)
qed

lemma *thm-2-3-7-indets*:
assumes *is-GB-cofactor-bound* (*set fs*) *b*
shows *1* $\in$ *ideal* (*set fs*) $\longleftrightarrow$ *1* $\in$ *set* (*punit.Macaulay-list* (*deg-shifts* ($\bigcup$(*indets* ‘ (*set fs*))) *b fs*))
using - - *assms*
proof (*rule thm-2-3-7*)
from *finite-set* **show** *finite* ($\bigcup$(*indets* ‘ (*set fs*))) **by** (*simp add*: *finite-indets*)
next
show *set fs* $\subseteq$ *P*[$\bigcup$(*indets* ‘ (*set fs*))] **by** (*auto simp*: *Polys-alt*)
qed

end

end

5 Integer Binomial Coefficients

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

Restore original sort constraints:

setup ‹*Sign.add-const-constraint* (@{*const-name gbinomial*}, *SOME* @{*typ* ′*a*::{*semidom-divide*,*semiring-char*
$\Rightarrow$ *nat* $\Rightarrow$ ′*a*})›

5.1 Inequalities

lemma *binomial-mono*:
assumes *m* $\leq$ *n*
shows *m choose k* $\leq$ *n choose k*
by (*simp add*: *assms binomial-right-mono*)

lemma *binomial-plus-le*:
assumes *0* < *k*
shows (*m choose k*) + (*n choose k*) $\leq$ (*m* + *n*) *choose k*
proof −
define *k0* **where** *k0* = *k* − *1*
with *assms* **have** *k*: *k* = *Suc k0* **by** *simp*

show *?thesis* **unfolding** *k*
proof (*induct n*)
case *0*
show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
by (*simp add*: *add.left-commute add-le-mono binomial-right-mono*)
qed
qed

lemma *binomial-ineq-1*: *2* ∗ ((*n* + *i*) *choose k*) ≤ (*n choose k*) + ((*n* + *2* ∗ *i*) *choose k*)
proof (*cases k*)
case *0*
thus *?thesis* **by** *simp*
next
case *k*: (*Suc k0*)
show *?thesis* **unfolding** *k*
proof (*induct i*)
case *0*
thus *?case* **by** *simp*
next
case (*Suc i*)
have *2* ∗ (*n* + *Suc i choose Suc k0*) = *2* ∗ (*n* + *i choose k0*) + *2* ∗ (*n* + *i choose Suc k0*)
by *simp*
also have ... ≤ ((*n* + *2* ∗ *i choose k0*) + (*Suc* (*n* + *2* ∗ *i*) *choose k0*)) + ((*n choose Suc k0*) + (*n* + *2* ∗ *i choose Suc k0*))
proof (*rule add-mono*)
have *n* + *i choose k0* ≤ *n* + *2* ∗ *i choose k0*
by (*rule binomial-mono*) *simp*
moreover have *n* + *2* ∗ *i choose k0* ≤ *Suc* (*n* + *2* ∗ *i*) *choose k0*
by (*rule binomial-mono*) *simp*
ultimately show *2* ∗ (*n* + *i choose k0*) ≤ (*n* + *2* ∗ *i choose k0*) + (*Suc* (*n* + *2* ∗ *i*) *choose k0*)
by *simp*
qed (*fact Suc*)
also have ... = (*n choose Suc k0*) + (*n* + *2* ∗ *Suc i choose Suc k0*) **by** *simp*
finally show *?case* **.**
qed
qed

lemma *gbinomial-int-mono*:
assumes *0* ≤ *x* **and** *x* ≤ (*y*::*int*)
shows *x gchoose k* ≤ *y gchoose k*
proof −
from *assms* **have** *nat x* ≤ *nat y* **by** *simp*
hence *nat x choose k* ≤ *nat y choose k* **by** (*rule binomial-mono*)

hence *int* (*nat x choose k*) ≤ *int* (*nat y choose k*) **by** (*simp only*: *zle-int*)
hence *int* (*nat x*) *gchoose k* ≤ *int* (*nat y*) *gchoose k* **by** (*simp only*: *int-binomial*)
with *assms* **show** *?thesis* **by** *simp*
qed

lemma *gbinomial-int-plus-le*:
assumes *0* < *k* **and** *0* ≤ *x* **and** *0* ≤ (*y*::*int*)
shows (*x gchoose k*) + (*y gchoose k*) ≤ (*x* + *y*) *gchoose k*
proof −
from *assms*(*1*) **have** (*nat x choose k*) + (*nat y choose k*) ≤ *nat x* + *nat y choose k*
by (*rule binomial-plus-le*)
hence *int* ((*nat x choose k*) + (*nat y choose k*)) ≤ *int* (*nat x* + *nat y choose k*)
by (*simp only*: *zle-int*)
hence (*int* (*nat x*) *gchoose k*) + (*int* (*nat y*) *gchoose k*) ≤ *int* (*nat x*) + *int* (*nat y*) *gchoose k*
by (*simp only*: *int-plus int-binomial*)
with *assms*(*2*, *3*) **show** *?thesis* **by** *simp*
qed

lemma *binomial-int-ineq-1*:
assumes *0* ≤ *x* **and** *0* ≤ (*y*::*int*)
shows *2* ∗ (*x* + *y gchoose k*) ≤ (*x gchoose k*) + ((*x* + *2* ∗ *y*) *gchoose k*)
proof −
from *binomial-ineq-1*[*of nat x nat y k*]
have *int* (*2* ∗ (*nat x* + *nat y choose k*)) ≤ *int* ((*nat x choose k*) + (*nat x* + *2* ∗ *nat y choose k*))
by (*simp only*: *zle-int*)
hence *2* ∗ (*int* (*nat x*) + *int* (*nat y*) *gchoose k*) ≤ (*int* (*nat x*) *gchoose k*) + (*int* (*nat x*) + *2* ∗ *int* (*nat y*) *gchoose k*)
by (*simp only*: *int-binomial int-plus int-ops*(*7*)) *simp*
with *assms* **show** *?thesis* **by** *simp*
qed

corollary *binomial-int-ineq-2*:
assumes *0* ≤ *y* **and** *y* ≤ (*x*::*int*)
shows *2* ∗ (*x gchoose k*) ≤ (*x* − *y gchoose k*) + (*x* + *y gchoose k*)
proof −
from *assms*(*2*) **have** *0* ≤ *x* − *y* **by** *simp*
hence *2* ∗ ((*x* − *y*) + *y gchoose k*) ≤ (*x* − *y gchoose k*) + ((*x* − *y* + *2* ∗ *y*) *gchoose k*)
using *assms*(*1*) **by** (*rule binomial-int-ineq-1*)
thus *?thesis* **by** *smt*
qed

corollary *binomial-int-ineq-3*:
assumes *0* ≤ *y* **and** *y* ≤ *2* ∗ (*x*::*int*)
shows *2* ∗ (*x gchoose k*) ≤ (*y gchoose k*) + (*2* ∗ *x* − *y gchoose k*)
proof (*cases y* ≤ *x*)

case *True*
hence $0 \leq x - y$ **by** *simp*
moreover from *assms*(*1*) **have** $x - y \leq x$ **by** *simp*
ultimately have $2 * (x\ gchoose\ k) \leq (x - (x - y)\ gchoose\ k) + (x + (x - y)\ gchoose\ k)$
by (*rule binomial-int-ineq-2*)
thus *?thesis* **by** *simp*
next
case *False*
hence $0 \leq y - x$ **by** *simp*
moreover from *assms*(*2*) **have** $y - x \leq x$ **by** *simp*
ultimately have $2 * (x\ gchoose\ k) \leq (x - (y - x)\ gchoose\ k) + (x + (y - x)\ gchoose\ k)$
by (*rule binomial-int-ineq-2*)
thus *?thesis* **by** *simp*
qed

5.2 Backward Difference Operator

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

lemma *bw-diff-const* [*simp*]: *bw-diff* (λ-. *c*) = (λ-. *0*)
by (*rule ext*) (*simp add*: *bw-diff-def*)

lemma *bw-diff-id* [*simp*]: *bw-diff* (λ*x*. *x*) = (λ-. *1*)
by (*rule ext*) (*simp add*: *bw-diff-def*)

lemma *bw-diff-plus* [*simp*]: *bw-diff* (λ*x*. *f x* + *g x*) = (λ*x*. *bw-diff f x* + *bw-diff g x*)
by (*rule ext*) (*simp add*: *bw-diff-def*)

lemma *bw-diff-uminus* [*simp*]: *bw-diff* (λ*x*. − *f x*) = (λ*x*. − *bw-diff f x*)
by (*rule ext*) (*simp add*: *bw-diff-def*)

lemma *bw-diff-minus* [*simp*]: *bw-diff* (λ*x*. *f x* − *g x*) = (λ*x*. *bw-diff f x* − *bw-diff g x*)
by (*rule ext*) (*simp add*: *bw-diff-def*)

lemma *bw-diff-const-pow*: (*bw-diff* ^^ *k*) (λ-. *c*) = (*if k* = *0 then* λ-. *c else* (λ-. *0*))
by (*induct k*, *simp-all*)

lemma *bw-diff-id-pow*:
(*bw-diff* ^^ *k*) (λ*x*. *x*) = (*if k* = *0 then* (λ*x*. *x*) *else if k* = *1 then* (λ-. *1*) *else* (λ-. *0*))
by (*induct k*, *simp-all*)

lemma *bw-diff-plus-pow* [*simp*]:

$(bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ (\lambda x.\ f\ x + g\ x) = (\lambda x.\ (bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ f\ x + (bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ g\ x)$
by (*induct k*, *simp-all*)

lemma *bw-diff-uminus-pow* [*simp*]: $(bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ (\lambda x.\ -\ f\ x) = (\lambda x.\ -\ (bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ f\ x)$
by (*induct k*, *simp-all*)

lemma *bw-diff-minus-pow* [*simp*]:
$(bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ (\lambda x.\ f\ x - g\ x) = (\lambda x.\ (bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ f\ x - (bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ g\ x)$
by (*induct k*, *simp-all*)

lemma *bw-diff-sum-pow* [*simp*]:
$(bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ (\lambda x.\ (\sum i{\in}I.\ f\ i\ x)) = (\lambda x.\ (\sum i{\in}I.\ (bw\text{-}diff\ \widehat{\ }\widehat{\ }\ k)\ (f\ i)\ x))$
by (*induct I rule*: *infinite-finite-induct*, *simp-all add*: *bw-diff-const-pow*)

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 (*rule ext*)
fix $x{::}int$
from *assms* **have** *eq*: $Suc\ (k - Suc\ 0) = k$ **by** *simp*
have $x + n\ gchoose\ k = (x + n - 1) + 1\ gchoose\ (Suc\ (k - 1))$ **by** (*simp add*: *eq*)
also have $\ldots = (x + n - 1\ gchoose\ k - 1) + ((x + n - 1)\ gchoose\ (Suc\ (k - 1)))$
by (*fact gbinomial-int-Suc-Suc*)
finally show $bw\text{-}diff\ (\lambda x.\ x + n\ gchoose\ k)\ x = x + n - 1\ gchoose\ (k - 1)$
by (*simp add*: *eq bw-diff-def algebra-simps*)
qed

lemma *bw-diff-gbinomial-pow*:
$(bw\text{-}diff\ \widehat{\ }\widehat{\ }\ l)\ (\lambda x{::}int.\ (x + n)\ gchoose\ k) =$
$(if\ l \le k\ then\ (\lambda x.\ (x + n - int\ l)\ gchoose\ (k - l))\ else\ (\lambda\text{-}.\ 0))$
proof −
have $*$: $l0 \le k \Longrightarrow (bw\text{-}diff\ \widehat{\ }\widehat{\ }\ l0)\ (\lambda x{::}int.\ (x + n)\ gchoose\ k) = (\lambda x.\ (x + n - int\ l0)\ gchoose\ (k - l0))$
for *l0*
proof (*induct l0*)
case *0*
show *?case* **by** *simp*
next
case (*Suc l0*)
from *Suc.prems* **have** $0 < k - l0$ **and** $l0 \le k$ **by** *simp-all*
from *this*(*2*) **have** *eq*: $(bw\text{-}diff\ \widehat{\ }\widehat{\ }\ l0)\ (\lambda x.\ x + n\ gchoose\ k) = (\lambda x.\ x + n - int\ l0\ gchoose\ (k - l0))$
by (*rule Suc.hyps*)
have $(bw\text{-}diff\ \widehat{\ }\widehat{\ }\ Suc\ l0)\ (\lambda x.\ x + n\ gchoose\ k) = bw\text{-}diff\ (\lambda x.\ x + (n - int\ l0)\ gchoose\ (k - l0))$

by (*simp add*: *eq algebra-simps*)
also from ‹$0 < k - l0$› **have** … = ($\lambda x.$ (x + (n − *int* $l0$) − 1) *gchoose* (k − $l0$ − 1))
by (*rule bw-diff-gbinomial*)
also have … = ($\lambda x.$ $x + n$ − *int* (*Suc* $l0$) *gchoose* (k − *Suc* $l0$)) **by** (*simp add*: *algebra-simps*)
finally show *?case* **.**
qed
show *?thesis*
proof (*simp add*: ∗ *split*: *if-split*, *intro impI*)
assume ¬ $l \le k$
hence $(l - k) + k = l$ **and** $l - k \neq 0$ **by** *simp-all*
hence (*bw-diff* ^^ l) ($\lambda x.$ $x + n$ *gchoose* k) = (*bw-diff* ^^ (($l - k$) + k)) ($\lambda x.$ $x + n$ *gchoose* k)
by (*simp only*:)
also have … = (*bw-diff* ^^ ($l - k$)) (λ-. 1) **by** (*simp add*: ∗ *funpow-add*)
also from ‹$l - k \neq 0$› **have** … = (λ-. 0) **by** (*simp add*: *bw-diff-const-pow*)
finally show (*bw-diff* ^^ l) ($\lambda x.$ $x + n$ *gchoose* k) = (λ-. 0) **.**
qed
qed

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* ⇒ *int*) ⇒ *bool*
where *poly-fun f* ⟷ (∃ *p*::*rat poly*. ∀ *a*. *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*))

lemma *poly-funI*: (⋀*a*. *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)) ⟹ *poly-fun f*
by (*auto simp*: *poly-fun-def*)

lemma *poly-funE*:
assumes *poly-fun f*
obtains *p* **where** ⋀*a*. *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
using *assms* **by** (*auto simp*: *poly-fun-def*)

lemma *poly-fun-eqI*:
assumes *poly-fun f* **and** *poly-fun g* **and** *infinite* {*a*. *f a* = *g a*}
shows *f* = *g*
proof (*rule ext*)
fix *a*
from *assms*(*1*) **obtain** *p* **where** *p*: ⋀*a*. *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)

from *assms*(*2*) **obtain** *q* **where** *q*: $\bigwedge$*a. rat-of-int* (*g a*) = *poly q* (*rat-of-int a*)
  by (*rule poly-funE*, *blast*)
have *p* = *q*
proof (*rule ccontr*)
  let *?A* = {*a. poly p* (*rat-of-int a*) = *poly q* (*rat-of-int a*)}
  assume *p* ≠ *q*
  hence *p* − *q* ≠ *0* **by** *simp*
  hence *fin*: *finite* {*x. poly* (*p* − *q*) *x* = *0*} **by** (*rule poly-roots-finite*)
  have *rat-of-int* ‘ *?A* ⊆ {*x. poly* (*p* − *q*) *x* = *0*} **by** (*simp add*: *image-Collect-subsetI*)
  hence *finite* (*rat-of-int* ‘ *?A*) **using** *fin* **by** (*rule finite-subset*)
  moreover have *inj-on rat-of-int ?A* **by** (*simp add*: *inj-on-def*)
  ultimately have *finite ?A* **by** (*simp only*: *finite-image-iff*)
  also have *?A* = {*a. f a* = *g a*} **by** (*simp flip*: *p q*)
  finally show *False* **using** *assms*(*3*) **by** *simp*
qed
hence *rat-of-int* (*f a*) = *rat-of-int* (*g a*) **by** (*simp add*: *p q*)
thus *f a* = *g a* **by** *simp*
qed

corollary *poly-fun-eqI-ge*:
  assumes *poly-fun f* **and** *poly-fun g* **and** $\bigwedge$*a. b* ≤ *a* ⟹ *f a* = *g a*
  shows *f* = *g*
  using *assms*(*1*, *2*)
proof (*rule poly-fun-eqI*)
  have {*b..*} ⊆ {*a. f a* = *g a*} **by** (*auto intro*: *assms*(*3*))
  thus *infinite* {*a. f a* = *g a*} **using** *infinite-Ici* **by** (*rule infinite-super*)
qed

corollary *poly-fun-eqI-gr*:
  assumes *poly-fun f* **and** *poly-fun g* **and** $\bigwedge$*a. b* < *a* ⟹ *f a* = *g a*
  shows *f* = *g*
  using *assms*(*1*, *2*)
proof (*rule poly-fun-eqI*)
  have {*b*<..} ⊆ {*a. f a* = *g a*} **by** (*auto intro*: *assms*(*3*))
  thus *infinite* {*a. f a* = *g a*} **using** *infinite-Ioi* **by** (*rule infinite-super*)
qed

6.2 Closure Properties

lemma *poly-fun-const* [*simp*]: *poly-fun* (λ-. *c*)
  by (*rule poly-funI*[**where** *p*=[:*rat-of-int c*:]]) *simp*

lemma *poly-fun-id* [*simp*]: *poly-fun* (λ*x. x*) *poly-fun id*
proof −
  show *poly-fun* (λ*x. x*) **by** (*rule poly-funI*[**where** *p*=[:*0*, *1*:]]) *simp*
  thus *poly-fun id* **by** (*simp only*: *id-def*)
qed

lemma *poly-fun-uminus*:

assumes *poly-fun f*
shows *poly-fun* ($\lambda x.\ -\ f\ x$) **and** *poly-fun* ($-\ f$)
proof $-$
from *assms* **obtain** *p* **where** *p*: $\bigwedge a.$ *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
show *poly-fun* ($\lambda x.\ -\ f\ x$) **by** (*rule poly-funI*[**where** *p*=$-$ *p*]) (*simp add*: *p*)
thus *poly-fun* ($-\ f$) **by** (*simp only*: *fun-Compl-def*)
qed

lemma *poly-fun-uminus-iff* [*simp*]:
poly-fun ($\lambda x.\ -\ f\ x$) $\longleftrightarrow$ *poly-fun f poly-fun* ($-\ f$) $\longleftrightarrow$ *poly-fun f*
proof $-$
show *poly-fun* ($\lambda x.\ -\ f\ x$) $\longleftrightarrow$ *poly-fun f*
proof
assume *poly-fun* ($\lambda x.\ -\ f\ x$)
hence *poly-fun* ($\lambda x.\ -\ (-\ f\ x)$) **by** (*rule poly-fun-uminus*)
thus *poly-fun f* **by** *simp*
qed (*rule poly-fun-uminus*)
thus *poly-fun* ($-\ f$) $\longleftrightarrow$ *poly-fun f* **by** (*simp only*: *fun-Compl-def*)
qed

lemma *poly-fun-plus* [*simp*]:
assumes *poly-fun f* **and** *poly-fun g*
shows *poly-fun* ($\lambda x.\ f\ x + g\ x$)
proof $-$
from *assms*(*1*) **obtain** *p* **where** *p*: $\bigwedge a.$ *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
from *assms*(*2*) **obtain** *q* **where** *q*: $\bigwedge a.$ *rat-of-int* (*g a*) = *poly q* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
show *?thesis* **by** (*rule poly-funI*[**where** *p*=*p* + *q*]) (*simp add*: *p q*)
qed

lemma *poly-fun-minus* [*simp*]:
assumes *poly-fun f* **and** *poly-fun g*
shows *poly-fun* ($\lambda x.\ f\ x - g\ x$)
proof $-$
from *assms*(*1*) **obtain** *p* **where** *p*: $\bigwedge a.$ *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
from *assms*(*2*) **obtain** *q* **where** *q*: $\bigwedge a.$ *rat-of-int* (*g a*) = *poly q* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
show *?thesis* **by** (*rule poly-funI*[**where** *p*=*p* $-$ *q*]) (*simp add*: *p q*)
qed

lemma *poly-fun-times* [*simp*]:
assumes *poly-fun f* **and** *poly-fun g*
shows *poly-fun* ($\lambda x.\ f\ x * g\ x$)
proof $-$
from *assms*(*1*) **obtain** *p* **where** *p*: $\bigwedge a.$ *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)

from *assms*(*2*) **obtain** *q* **where** *q*: $\bigwedge a.$ *rat-of-int* (*g a*) = *poly q* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
show *?thesis* **by** (*rule poly-funI*[**where** *p*=*p* ∗ *q*]) (*simp add*: *p q*)
qed

lemma *poly-fun-divide*:
assumes *poly-fun f* **and** $\bigwedge a.$ *c dvd f a*
shows *poly-fun* (λ*x*. *f x div c*)
proof −
from *assms*(*1*) **obtain** *p* **where** *p*: $\bigwedge a.$ *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
let *?p* = *p* ∗ [:*1* / *rat-of-int c*:]
show *?thesis*
proof (*rule poly-funI*)
fix *a*
have *c dvd f a* **by** *fact*
hence *rat-of-int* (*f a div c*) = *rat-of-int* (*f a*) / *rat-of-int c* **by** *auto*
also have ... = *poly ?p* (*rat-of-int a*) **by** (*simp add*: *p*)
finally show *rat-of-int* (*f a div c*) = *poly ?p* (*rat-of-int a*) **.**
qed
qed

lemma *poly-fun-pow* [*simp*]:
assumes *poly-fun f*
shows *poly-fun* (λ*x*. *f x* ^ *k*)
proof −
from *assms*(*1*) **obtain** *p* **where** *p*: $\bigwedge a.$ *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
show *?thesis* **by** (*rule poly-funI*[**where** *p*=*p* ^ *k*]) (*simp add*: *p*)
qed

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 −
from *assms*(*1*) **obtain** *p* **where** *p*: $\bigwedge a.$ *rat-of-int* (*f a*) = *poly p* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
from *assms*(*2*) **obtain** *q* **where** *q*: $\bigwedge a.$ *rat-of-int* (*g a*) = *poly q* (*rat-of-int a*)
by (*rule poly-funE*, *blast*)
show *poly-fun* (λ*x*. *f* (*g x*)) **by** (*rule poly-funI*[**where** *p*=*p* $\circ_p$ *q*]) (*simp add*: *p q poly-pcompose*)
thus *poly-fun* (*f* ∘ *g*) **by** (*simp only*: *comp-def*)
qed

lemma *poly-fun-sum* [*simp*]: ($\bigwedge i.$ *i* ∈ *I* ⟹ *poly-fun* (*f i*)) ⟹ *poly-fun* (λ*x*. ($\sum$ *i*∈*I*. *f i x*))
proof (*induct I rule*: *infinite-finite-induct*)
case (*infinite I*)
from *infinite*(*1*) **show** *?case* **by** *simp*

next
  case *empty*
  show *?case* **by** *simp*
next
  case (*insert i I*)
  have $i \in$ *insert i I* **by** *simp*
  hence *poly-fun* (*f i*) **by** (*rule insert.prems*)
  moreover have *poly-fun* ($\lambda x.$ $\sum i{\in}I.$ *f i x*)
  proof (*rule insert.hyps*)
    fix *j*
    assume $j \in I$
    hence $j \in$ *insert i I* **by** *simp*
    thus *poly-fun* (*f j*) **by** (*rule insert.prems*)
  qed
  ultimately have *poly-fun* ($\lambda x.$ *f i x* + ($\sum i{\in}I.$ *f i x*)) **by** (*rule poly-fun-plus*)
  with *insert.hyps*(*1*, *2*) **show** *?case* **by** *simp*
qed

lemma *poly-fun-prod* [*simp*]: ($\bigwedge i.$ $i \in I \Longrightarrow$ *poly-fun* (*f i*)) $\Longrightarrow$ *poly-fun* ($\lambda x.$ ($\prod i{\in}I.$ *f i x*))
proof (*induct I rule*: *infinite-finite-induct*)
  case (*infinite I*)
  from *infinite*(*1*) **show** *?case* **by** *simp*
next
  case *empty*
  show *?case* **by** *simp*
next
  case (*insert i I*)
  have $i \in$ *insert i I* **by** *simp*
  hence *poly-fun* (*f i*) **by** (*rule insert.prems*)
  moreover have *poly-fun* ($\lambda x.$ $\prod i{\in}I.$ *f i x*)
  proof (*rule insert.hyps*)
    fix *j*
    assume $j \in I$
    hence $j \in$ *insert i I* **by** *simp*
    thus *poly-fun* (*f j*) **by** (*rule insert.prems*)
  qed
  ultimately have *poly-fun* ($\lambda x.$ *f i x* $*$ ($\prod i{\in}I.$ *f i x*)) **by** (*rule poly-fun-times*)
  with *insert.hyps*(*1*, *2*) **show** *?case* **by** *simp*
qed

lemma *poly-fun-pochhammer* [*simp*]: *poly-fun f* $\Longrightarrow$ *poly-fun* ($\lambda x.$ *pochhammer* (*f x*) *k*)
  by (*simp add*: *pochhammer-prod*)

lemma *poly-fun-gbinomial* [*simp*]: *poly-fun f* $\Longrightarrow$ *poly-fun* ($\lambda x.$ *f x gchoose k*)
  by (*simp add*: *gbinomial-int-pochhammer′ poly-fun-divide fact-dvd-pochhammer*)

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*
  by (*simp add*: *is-monomial-set-def*)

lemma *is-monomial-setD*: *is-monomial-set A* $\Longrightarrow p \in A \Longrightarrow$ *is-monomial p*
  by (*simp add*: *is-monomial-set-def*)

lemma *is-monomial-set-subset*: *is-monomial-set B* $\Longrightarrow A \subseteq B \Longrightarrow$ *is-monomial-set A*
  by (*auto simp*: *is-monomial-set-def*)

lemma *is-monomial-set-Un*: *is-monomial-set* $(A \cup B) \longleftrightarrow$ (*is-monomial-set A* $\wedge$ *is-monomial-set B*)
  by (*auto simp*: *is-monomial-set-def*)

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*
  using *assms*(*2*)
proof (*induct p rule*: *pmdl-induct*)
  case *base*: *module-0*
  show *?case* **by** (*simp add*: *pmdl.span-zero*)
next
  case *step*: (*module-plus p b c t*)
  have *eq*: *monomial* (*lookup* (*p* + *monom-mult c t b*) *v*) *v* =
      monomial (*lookup p v*) *v* + *monomial* (*lookup* (*monom-mult c t b*) *v*) *v*
    by (*simp only*: *single-add lookup-add*)
  from *assms*(*1*) *step.hyps*(*3*) **have** *is-monomial b* **by** (*rule is-monomial-setD*)
  then obtain *d u* **where** *b*: *b* = *monomial d u* **by** (*rule is-monomial-monomial*)
  have *monomial* (*lookup* (*monom-mult c t b*) *v*) *v* $\in$ *pmdl B*
  proof (*simp add*: *b monom-mult-monomial lookup-single when-def pmdl.span-zero*, *intro impI*)

assume $t \oplus u = v$
hence *monomial* $(c * d)$ v = *monom-mult c t b* **by** (*simp add*: *b monom-mult-monomial*)
also from *step.hyps*(*3*) **have** ... ∈ *pmdl B* **by** (*rule monom-mult-in-pmdl*)
finally show *monomial* $(c * d)$ v ∈ *pmdl B* **.**
qed
with *step.hyps*(*2*) **show** *?case* **unfolding** *eq* **by** (*rule pmdl.span-add*)
qed

lemma *monomial-pmdl-field*:
assumes *is-monomial-set B* **and** p ∈ *pmdl B* **and** v ∈ *keys* (p::- $\Rightarrow_0$ $'b$::*field*)
shows *monomial c v* ∈ *pmdl B*
proof −
from *assms*(*1*, *2*) **have** *monomial* (*lookup p v*) v ∈ *pmdl B* **by** (*rule monomial-pmdl*)
hence *monom-mult* (*c* / *lookup p v*) *0* (*monomial* (*lookup p v*) *v*) ∈ *pmdl B*
by (*rule pmdl-closed-monom-mult*)
with *assms*(*3*) **show** *?thesis* **by** (*simp add*: *monom-mult-monomial splus-zero in-keys-iff*)
qed

end

context *ordered-term*
begin

lemma *keys-monomial-pmdl*:
assumes *is-monomial-set F* **and** p ∈ *pmdl F* **and** t ∈ *keys p*
obtains f **where** f ∈ F **and** $f \neq 0$ **and** *lt f* $adds_t$ t
using *assms*(*2*) *assms*(*3*)
proof (*induct arbitrary*: *thesis rule*: *pmdl-induct*)
case *module-0*
from *this*(*2*) **show** *?case* **by** *simp*
next
case *step*: (*module-plus p f0 c s*)
from *assms*(*1*) *step*(*3*) **have** *is-monomial f0* **unfolding** *is-monomial-set-def* **..**
hence *keys f0* = {*lt f0*} **and** *f0* ≠ *0* **by** (*rule keys-monomial*, *rule monomial-not-0*)
from *Poly-Mapping.keys-add step*(*6*) **have** t ∈ *keys p* ∪ *keys* (*monom-mult c s f0*) **..**
thus *?case*
proof
assume t ∈ *keys p*
from *step*(*2*)[*OF - this*] **obtain** f **where** f ∈ F **and** $f \neq 0$ **and** *lt f* $adds_t$ t **by** *blast*
thus *?thesis* **by** (*rule step*(*5*))
next
assume t ∈ *keys* (*monom-mult c s f0*)
with *keys-monom-mult-subset* **have** t ∈ (⊕) *s* ‘ *keys f0* **..**
hence $t = s \oplus$ *lt f0* **by** (*simp add*: ‹*keys f0* = {*lt f0*}›)

hence *lt f0 adds*$_t$ *t* **by** (*simp add*: *term-simps*)
with ‹*f0* ∈ *F*› ‹*f0* ≠ *0*› **show** *?thesis* **by** (*rule step*(*5*))
qed
qed

lemma *image-lt-monomial-lt*: *lt* ‘ *monomial* (*1*::'*b*::*zero-neq-one*) ‘ *lt* ‘ *F* = *lt* ‘ *F*
by (*auto simp*: *lt-monomial intro*!: *image-eqI*)

7.3 Reduction

lemma *red-setE2*:
assumes *red B p q*
obtains *b* **where** *b* ∈ *B* **and** *b* ≠ *0* **and** *red* {*b*} *p q*
proof −
from *assms* **obtain** *b t* **where** *b* ∈ *B* **and** *red-single p q b t* **by** (*rule red-setE*)
from *this*(*2*) **have** *b* ≠ *0* **by** (*simp add*: *red-single-def*)
have *red* {*b*} *p q* **by** (*rule red-setI*, *simp*, *fact*)
show *?thesis* **by** (*rule*, *fact*+)
qed

lemma *red-monomial-keys*:
assumes *is-monomial r* **and** *red* {*r*} *p q*
shows *card* (*keys p*) = *Suc* (*card* (*keys q*))
proof −
from *assms*(*2*) **obtain** *s* **where** *rs*: *red-single p q r s* **unfolding** *red-singleton* **..**
hence *cp0*: *lookup p* (*s* ⊕ *lt r*) ≠ *0* **and** *q-def0*: *q* = *p* − *monom-mult* (*lookup p* (*s* ⊕ *lt r*) / *lc r*) *s r*
unfolding *red-single-def* **by** *simp-all*
from *assms*(*1*) **obtain** *c t* **where** *c* ≠ *0* **and** *r-def*: *r* = *monomial c t* **by** (*rule is-monomial-monomial*)
have *ltr*: *lt r* = *t* **unfolding** *r-def* **by** (*rule lt-monomial*, *fact*)
have *lcr*: *lc r* = *c* **unfolding** *r-def* **by** (*rule lc-monomial*)
define *u* **where** *u* = *s* ⊕ *t*
from *q-def0* **have** *q* = *p* − *monom-mult* (*lookup p u* / *c*) *s r* **unfolding** *u-def ltr lcr* **.**
also have ... = *p* − *monomial* ((*lookup p u* / *c*) ∗ *c*) *u* **unfolding** *u-def r-def monom-mult-monomial* **..**
finally have *q-def*: *q* = *p* − *monomial* (*lookup p u*) *u* **using** ‹*c* ≠ *0*› **by** *simp*
from *cp0* **have** *lookup p u* ≠ *0* **unfolding** *u-def ltr* **.**
hence *u* ∈ *keys p* **by** (*simp add*: *in-keys-iff*)

have *keys q* = *keys p* − {*u*} **unfolding** *q-def*
proof (*rule*, *rule*)
fix *x*
assume *x* ∈ *keys* (*p* − *monomial* (*lookup p u*) *u*)
hence *lookup* (*p* − *monomial* (*lookup p u*) *u*) *x* ≠ *0* **by** (*simp add*: *in-keys-iff*)
hence *a*: *lookup p x* − *lookup* (*monomial* (*lookup p u*) *u*) *x* ≠ *0* **unfolding** *lookup-minus* **.**
hence *x* ≠ *u* **unfolding** *lookup-single* **by** *auto*

with *a* **have** *lookup p x* $\neq$ *0* **unfolding** *lookup-single* **by** *auto*
show $x \in$ *keys p* $- \{u\}$
proof
from ‹*lookup p x* $\neq$ *0*› **show** $x \in$ *keys p* **by** (*simp add*: *in-keys-iff*)
next
from ‹$x \neq u$› **show** $x \notin \{u\}$ **by** *simp*
qed
next
show *keys p* $- \{u\} \subseteq$ *keys* (*p* $-$ *monomial* (*lookup p u*) *u*)
proof
fix *x*
assume $x \in$ *keys p* $- \{u\}$
hence $x \in$ *keys p* **and** $x \neq u$ **by** *auto*
from ‹$x \in$ *keys p*› **have** *lookup p x* $\neq$ *0* **by** (*simp add*: *in-keys-iff*)
with ‹$x \neq u$› **have** *lookup* (*p* $-$ *monomial* (*lookup p u*) *u*) *x* $\neq$ *0* **by** (*simp add*: *lookup-minus lookup-single*)
thus $x \in$ *keys* (*p* $-$ *monomial* (*lookup p u*) *u*) **by** (*simp add*: *in-keys-iff*)
qed
qed

have *Suc* (*card* (*keys q*)) = *card* (*keys p*) **unfolding** ‹*keys q* = *keys p* $- \{u\}$›
by (*rule card-Suc-Diff1*, *rule finite-keys*, *fact*)
thus *?thesis* **by** *simp*
qed

lemma *red-monomial-monomial-setD*:
assumes *is-monomial p* **and** *is-monomial-set B* **and** *red B p q*
shows *q* = *0*
proof −
from *assms*(*3*) **obtain** *b* **where** $b \in B$ **and** $b \neq$ *0* **and** $*$: *red* $\{b\}$ *p q* **by** (*rule red-setE2*)
from *assms*(*2*) *this*(*1*) **have** *is-monomial b* **by** (*rule is-monomial-setD*)
hence *card* (*keys p*) = *Suc* (*card* (*keys q*)) **using** $*$ **by** (*rule red-monomial-keys*)
with *assms*(*1*) **show** *?thesis* **by** (*simp add*: *is-monomial-def*)
qed

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 −
from *assms*(*3*) **obtain** *q* **where** *red B p q* **by** (*rule is-redE*)
moreover from *assms*(*1*, *2*) *this* **have** *q* = *0* **by** (*rule red-monomial-monomial-setD*)
ultimately show *?thesis* **by** *simp*
qed

corollary *is-red-monomial-monomial-set-in-pmdl*:
is-monomial p $\Longrightarrow$ *is-monomial-set B* $\Longrightarrow$ *is-red B p* $\Longrightarrow$ $p \in$ *pmdl B*
by (*intro red-rtranclp-0-in-pmdl r-into-rtranclp is-red-monomial-monomial-setD*)

corollary *red-rtrancl-monomial-monomial-set-cases*:
  assumes *is-monomial p* **and** *is-monomial-set B* **and** $(red\ B)^{**}\ p\ q$
  obtains $q = p \mid q = 0$
  using *assms*(*3*)
proof (*induct q arbitrary*: *thesis rule*: *rtranclp-induct*)
  case *base*
  from *refl* **show** *?case* **by** (*rule base*)
next
  case (*step y z*)
  show *?case*
  proof (*rule step.hyps*)
    assume $y = p$
    with *step.hyps*(*2*) **have** *red B p z* **by** *simp*
    with *assms*(*1*, *2*) **have** $z = 0$ **by** (*rule red-monomial-monomial-setD*)
    thus *?thesis* **by** (*rule step.prems*)
  next
    assume $y = 0$
    from *step.hyps*(*2*) **have** *is-red B 0* **unfolding** ‹$y = 0$› **by** (*rule is-redI*)
    with *irred-0* **show** *?thesis* **..**
  qed
qed

lemma *is-red-monomial-lt*:
  assumes $0 \notin B$
  shows *is-red* (*monomial* (*1*::′b::*field*) ‘ *lt* ‘ *B*) = *is-red B*
proof
  fix *p*
  let *?B* = *monomial* (*1*::′b) ‘ *lt* ‘ *B*
  show *is-red ?B p* ⟷ *is-red B p*
  proof
    assume *is-red ?B p*
    then obtain *f v* **where** *f* ∈ *?B* **and** *v* ∈ *keys p* **and** *adds*: *lt f* $adds_t$ *v* **by** (*rule is-red-addsE*)
    from *this*(*1*) **have** *lt f* ∈ *lt* ‘ *?B* **by** (*rule imageI*)
    also have . . . = *lt* ‘ *B* **by** (*fact image-lt-monomial-lt*)
    finally obtain *b* **where** *b* ∈ *B* **and** *eq*: *lt f* = *lt b* **..**
    note *this*(*1*)
    moreover from *this assms* **have** $b \neq 0$ **by** *blast*
    moreover note ‹*v* ∈ *keys p*›
    moreover from *adds* **have** *lt b* $adds_t$ *v* **by** (*simp only*: *eq*)
    ultimately show *is-red B p* **by** (*rule is-red-addsI*)
  next
    assume *is-red B p*
    then obtain *b v* **where** *b* ∈ *B* **and** *v* ∈ *keys p* **and** *adds*: *lt b* $adds_t$ *v* **by** (*rule is-red-addsE*)
    from *this*(*1*) **have** *lt b* ∈ *lt* ‘ *B* **by** (*rule imageI*)
    also from *image-lt-monomial-lt* **have** . . . = *lt* ‘ *?B* **by** (*rule sym*)
    finally obtain *f* **where** *f* ∈ *?B* **and** *eq*: *lt b* = *lt f* **..**
    note *this*(*1*)

```
    moreover from this have f ≠ 0 by (auto simp: monomial-0-iff)
    moreover note ‹v ∈ keys p›
    moreover from adds have lt f adds_t v by (simp only: eq)
    ultimately show is-red ?B p by (rule is-red-addsI)
  qed
qed

end
```

7.4 Gröbner Bases

```
context gd-term
begin

lemma monomial-set-is-GB:
  assumes is-monomial-set G
  shows is-Groebner-basis G
  unfolding GB-alt-1
proof
  fix f
  assume f ∈ pmdl G
  thus (red G)** f 0
  proof (induct f rule: poly-mapping-plus-induct)
    case 1
    show ?case ..
  next
    case (2 f c t)
    let ?f = monomial c t + f
    from 2(1) have t ∈ keys (monomial c t) by simp
    from this 2(2) have t ∈ keys ?f by (rule in-keys-plusI1)
    with assms ‹?f ∈ pmdl G› obtain g where g ∈ G and g ≠ 0 and lt g adds_t t
      by (rule keys-monomial-pmdl)
    from this(1) have red G ?f f
    proof (rule red-setI)
      from ‹lt g adds_t t› have component-of-term (lt g) = component-of-term t
and lp g adds pp-of-term t
        by (simp-all add: adds-term-def)
      from this have eq: (pp-of-term t − lp g) ⊕ lt g = t
        by (simp add: adds-minus splus-def term-of-pair-pair)
      moreover from 2(2) have lookup ?f t = c by (simp add: lookup-add
in-keys-iff)
      ultimately show red-single (monomial c t + f) f g (pp-of-term t − lp g)
      proof (simp add: red-single-def ‹g ≠ 0› ‹t ∈ keys ?f› 2(1))
        from ‹g ≠ 0› have lc g ≠ 0 by (rule lc-not-0)
        hence monomial c t = monom-mult (c / lc g) (pp-of-term t − lp g)
(monomial (lc g) (lt g))
          by (simp add: monom-mult-monomial eq)
      moreover from assms ‹g ∈ G› have is-monomial g unfolding is-monomial-set-def
..
```

ultimately show *monomial c t* = *monom-mult* (*c* / *lc g*) (*pp-of-term t* − *lp g*) *g*
by (*simp only*: *monomial-eq-itself*)
qed
qed
have *f* ∈ *pmdl G* **by** (*rule pmdl-closed-red*, *fact subset-refl*, *fact+*)
hence (*red G*)** *f 0* **by** (*rule 2*(*3*))
with ‹*red G ?f f*› **show** *?case* **by** *simp*
qed
qed

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 −
from *subset-refl* **obtain** *F′* **where** *F′* ⊆ *F* − {*0*} **and** *lt* ‘ (*F* − {*0*}) = *lt* ‘ *F′* **and** *inj-on lt F′*
by (*rule subset-imageE-inj*)
define *G* **where** *G* = {*f* ∈ *F′*. ∀ *f′*∈*F′*. *lt f′* $adds_t$ *lt f* ⟶ *f′* = *f*}
have *G* ⊆ *F′* **by** (*simp add*: *G-def*)
hence *G* ⊆ *F* − {*0*} **using** ‹*F′* ⊆ *F* − {*0*}› **by** (*rule subset-trans*)
also have ... ⊆ *F* **by** *blast*
finally have *G* ⊆ *F* **.**
have *1*: *thesis* **if** *f* ∈ *F* **and** *f* ≠ *0* **and** ⋀*g*. *g* ∈ *G* ⟹ *lt g* $adds_t$ *lt f* ⟹ *thesis* **for** *thesis f*
proof −
let *?K* = *component-of-term* ‘ *Keys F*
let *?A* = {*t*. *pp-of-term t* ∈ *dgrad-set d m* ∧ *component-of-term t* ∈ *?K*}
let *?Q* = {*f′* ∈ *F′*. *lt f′* $adds_t$ *lt f*}
from *dgrad fin-comps* **have** *almost-full-on* ($adds_t$) *?A* **by** (*rule Dickson-term*)
moreover have *transp-on ?A* ($adds_t$) **by** (*auto intro*: *transp-onI dest*: *adds-term-trans*)
ultimately have *wfp-on* (*strict* ($adds_t$)) *?A* **by** (*rule af-trans-imp-wf*)
moreover have *lt f* ∈ *lt* ‘ *?Q*
proof −
from *that*(*1*, *2*) **have** *f* ∈ *F* − {*0*} **by** *simp*
hence *lt f* ∈ *lt* ‘ (*F* − {*0*}) **by** (*rule imageI*)
also have ... = *lt* ‘ *F′* **by** *fact*

finally have *lt f* ∈ *lt* ‘ *F′* **.**
with *adds-term-refl* **show** *?thesis* **by** *fastforce*
qed
moreover have *lt* ‘ *?Q* ⊆ *?A*
proof
fix *s*
assume *s* ∈ *lt* ‘ *?Q*
then obtain *q* **where** *q* ∈ *?Q* **and** *s*: *s* = *lt q* **..**
from *this*(*1*) **have** *q* ∈ *F′* **by** *simp*
hence *q* ∈ *F* − {*0*} **using** ‹*F′* ⊆ *F* − {*0*}› **..**
hence *q* ∈ *F* **and** *q* ≠ *0* **by** *simp-all*
from *this*(*1*) *F-sub* **have** *q* ∈ *dgrad-p-set d m* **..**
from ‹*q* ≠ *0*› **have** *lt q* ∈ *keys q* **by** (*rule lt-in-keys*)
hence *pp-of-term* (*lt q*) ∈ *pp-of-term* ‘ *keys q* **by** (*rule imageI*)
also from ‹*q* ∈ *dgrad-p-set d m*› **have** . . . ⊆ *dgrad-set d m* **by** (*simp add*: *dgrad-p-set-def*)
finally have *1*: *pp-of-term s* ∈ *dgrad-set d m* **by** (*simp only*: *s*)
from ‹*lt q* ∈ *keys q*› ‹*q* ∈ *F*› **have** *lt q* ∈ *Keys F* **by** (*rule in-KeysI*)
hence *component-of-term s* ∈ *?K* **unfolding** *s* **by** (*rule imageI*)
with *1* **show** *s* ∈ *?A* **by** *simp*
qed
ultimately obtain *t* **where** *t* ∈ *lt* ‘ *?Q* **and** *t-min*: ⋀*s*. *strict* (*adds*$_t$) *s t* ⟹ *s* ∉ *lt* ‘ *?Q*
by (*rule wfp-onE-min*) *blast*
from *this*(*1*) **obtain** *g* **where** *g* ∈ *?Q* **and** *t*: *t* = *lt g* **..**
from *this*(*1*) **have** *g* ∈ *F′* **and** *adds*: *lt g adds*$_t$ *lt f* **by** *simp-all*
show *?thesis*
proof (*rule that*)
{
fix *f′*
assume *f′* ∈ *F′*
assume *lt f′ adds*$_t$ *lt g*
hence *lt f′ adds*$_t$ *lt f* **using** *adds* **by** (*rule adds-term-trans*)
with ‹*f′* ∈ *F′*› **have** *f′* ∈ *?Q* **by** *simp*
hence *lt f′* ∈ *lt* ‘ *?Q* **by** (*rule imageI*)
with *t-min* **have** ¬ *strict* (*adds*$_t$) (*lt f′*) (*lt g*) **unfolding** *t* **by** *blast*
with ‹*lt f′ adds*$_t$ *lt g*› **have** *lt g adds*$_t$ *lt f′* **by** *blast*
with ‹*lt f′ adds*$_t$ *lt g*› **have** *lt f′* = *lt g* **by** (*rule adds-term-antisym*)
with ‹*inj-on lt F′*› **have** *f′* = *g* **using** ‹*f′* ∈ *F′*› ‹*g* ∈ *F′*› **by** (*rule inj-onD*)
}
with ‹*g* ∈ *F′*› **show** *g* ∈ *G* **by** (*simp add*: *G-def*)
qed *fact*
qed
have *2*: *is-red G q* **if** *q* ∈ *pmdl F* **and** *q* ≠ *0* **for** *q*
proof −
from *that*(*2*) **have** *keys q* ≠ {} **by** *simp*
then obtain *t* **where** *t* ∈ *keys q* **by** *blast*
with *F-monom that*(*1*) **obtain** *f* **where** *f* ∈ *F* **and** *f* ≠ *0* **and** ∗: *lt f adds*$_t$ *t*
by (*rule keys-monomial-pmdl*)

from *this*(*1*, *2*) **obtain** *g* **where** $g \in G$ **and** *lt g adds*$_t$ *lt f* **by** (*rule 1*)
from *this*(*2*) **have** $**$: *lt g adds*$_t$ *t* **using** $*$ **by** (*rule adds-term-trans*)
from ‹$g \in G$› ‹$G \subseteq F - \{0\}$› **have** $g \in F - \{0\}$ **..**
hence $g \neq 0$ **by** *simp*
with ‹$g \in G$› **show** *?thesis* **using** ‹$t \in$ *keys q*› $**$ **by** (*rule is-red-addsI*)
qed
from ‹$G \subseteq F - \{0\}$› **have** $G \subseteq F$ **by** *blast*
hence *pmdl G* $\subseteq$ *pmdl F* **by** (*rule pmdl.span-mono*)
note *dgrad fin-comps F-sub*
moreover have *is-reduced-GB* (*monic ‘ G*) **unfolding** *is-reduced-GB-def GB-image-monic*
proof (*intro conjI image-monic-is-auto-reduced image-monic-is-monic-set*)
from *dgrad* **show** *is-Groebner-basis G*
proof (*rule isGB-I-is-red*)
from ‹$G \subseteq F$› *F-sub* **show** $G \subseteq$ *dgrad-p-set d m* **by** (*rule subset-trans*)
next
fix *f*
assume $f \in$ *pmdl G*
hence $f \in$ *pmdl F* **using** ‹*pmdl G* $\subseteq$ *pmdl F*› **..**
moreover assume $f \neq 0$
ultimately show *is-red G f* **by** (*rule 2*)
qed
next
show *is-auto-reduced G* **unfolding** *is-auto-reduced-def*
proof (*intro ballI notI*)
fix *g*
assume $g \in G$
hence $g \in F$ **using** ‹$G \subseteq F$› **..**
with *F-monom* **have** *is-monomial g* **by** (*rule is-monomial-setD*)
hence *keys-g*: *keys g* $= \{$*lt g*$\}$ **by** (*rule keys-monomial*)
assume *is-red* $(G - \{g\})$ *g*
then obtain g' *t* **where** $g' \in G - \{g\}$ **and** $t \in$ *keys g* **and** *adds*: *lt g' adds*$_t$
t **by** (*rule is-red-addsE*)
from *this*(*1*) **have** $g' \in F'$ **and** $g' \neq g$ **by** (*simp-all add*: *G-def*)
from ‹$t \in$ *keys g*› **have** $t =$ *lt g* **by** (*simp add*: *keys-g*)
with ‹$g \in G$› ‹$g' \in F'$› *adds* **have** $g' = g$ **by** (*simp add*: *G-def*)
with ‹$g' \neq g$› **show** *False* **..**
qed
next
show $0 \notin$ *monic ‘ G*
proof
assume $0 \in$ *monic ‘ G*
then obtain *g* **where** $0 =$ *monic g* **and** $g \in G$ **..**
moreover from *this*(*2*) ‹$G \subseteq F - \{0\}$› **have** $g \neq 0$ **by** *blast*
ultimately show *False* **by** (*simp add*: *monic-0-iff*)
qed
qed
moreover have *pmdl* (*monic ‘ G*) $=$ *pmdl F* **unfolding** *pmdl-image-monic*
proof
show *pmdl F* $\subseteq$ *pmdl G*

proof (*rule pmdl.span-subset-spanI*, *rule*)
fix *f*
assume $f \in F$
hence $f \in$ *pmdl F* **by** (*rule pmdl.span-base*)
note *dgrad*
moreover from ‹$G \subseteq F$› *F-sub* **have** $G \subseteq$ *dgrad-p-set d m* **by** (*rule subset-trans*)
moreover note ‹*pmdl G* $\subseteq$ *pmdl F*› *2* ‹$f \in$ *pmdl F*›
moreover from ‹$f \in F$› *F-sub* **have** $f \in$ *dgrad-p-set d m* **..**
ultimately have (*red G*)** *f 0* **by** (*rule is-red-implies-0-red-dgrad-p-set*)
thus $f \in$ *pmdl G* **by** (*rule red-rtranclp-0-in-pmdl*)
qed
qed *fact*
ultimately have *reduced-GB F* = *monic* ‘ *G* **by** (*rule reduced-GB-unique-dgrad-p-set*)
also from ‹$G \subseteq F$› **have** ... $\subseteq$ *monic* ‘ *F* **by** (*rule image-mono*)
finally show *?thesis* **.**
qed

corollary *reduced-GB-is-monomial-set-dgrad-p-set*: *is-monomial-set* (*reduced-GB F*)
proof (*rule is-monomial-setI*)
fix *g*
assume $g \in$ *reduced-GB F*
also have ... $\subseteq$ *monic* ‘ *F* **by** (*fact reduced-GB-subset-monic-dgrad-p-set*)
finally obtain *f* **where** $f \in F$ **and** *g*: *g* = *monic f* **..**
from *F-monom this*(*1*) **have** *is-monomial f* **by** (*rule is-monomial-setD*)
hence *card* (*keys f*) = *1* **by** (*simp only*: *is-monomial-def*)
hence $f \neq 0$ **by** *auto*
hence *lc f* $\neq 0$ **by** (*rule lc-not-0*)
hence *1* / *lc f* $\neq 0$ **by** *simp*
hence *keys g* = ($\oplus$) *0* ‘ *keys f* **by** (*simp add*: *keys-monom-mult monic-def g*)
also from *refl* **have** ... = ($\lambda x.\ x$) ‘ *keys f* **by** (*rule image-cong*) (*simp only*: *splus-zero*)
finally show *is-monomial g* **using** ‹*card* (*keys f*) = *1*› **by** (*simp only*: *is-monomial-def image-ident*)
qed

end

lemma *is-red-reduced-GB-monomial-dgrad-set*:
assumes *finite* (*component-of-term* ‘ *S*) **and** *pp-of-term* ‘ $S \subseteq$ *dgrad-set d m*
shows *is-red* (*reduced-GB* (*monomial 1* ‘ *S*)) = *is-red* (*monomial* (*1*::'*b*::*field*) ‘ *S*)
proof
fix *p*
let *?F* = *monomial* (*1*::'*b*) ‘ *S*
from *assms*(*1*) **have** *1*: *finite* (*component-of-term* ‘ *Keys ?F*) **by** (*simp add*: *Keys-def*)
moreover from *assms*(*2*) **have** *2*: *?F* $\subseteq$ *dgrad-p-set d m* **by** (*auto simp*:

dgrad-p-set-def)
moreover have *is-monomial-set ?F* **by** (*auto intro*!: *is-monomial-setI monomial-is-monomial*)
ultimately have *reduced-GB ?F* ⊆ *monic* ‘ *?F* **by** (*rule reduced-GB-subset-monic-dgrad-p-set*)
also have ... = *?F* **by** (*auto simp*: *monic-def intro*!: *image-eqI*)
finally have *3*: *reduced-GB ?F* ⊆ *?F* **.**
show *is-red* (*reduced-GB ?F*) *p* ⟷ *is-red ?F p*
proof
assume *is-red* (*reduced-GB ?F*) *p*
thus *is-red ?F p* **using** *3* **by** (*rule is-red-subset*)
next
assume *is-red ?F p*
then obtain *f v* **where** *f* ∈ *?F* **and** *v* ∈ *keys p* **and** *f* ≠ *0* **and** *adds1*: *lt f* $adds_t$ *v*
by (*rule is-red-addsE*)
from *this*(*1*) **have** *f* ∈ *pmdl ?F* **by** (*rule pmdl.span-base*)
from *dgrad 1 2* **have** *is-Groebner-basis* (*reduced-GB ?F*) **by** (*rule reduced-GB-is-GB-dgrad-p-set*)
moreover from ‹*f* ∈ *pmdl ?F*› *dgrad 1 2* **have** *f* ∈ *pmdl* (*reduced-GB ?F*)
by (*simp only*: *reduced-GB-pmdl-dgrad-p-set*)
ultimately obtain *g* **where** *g* ∈ *reduced-GB ?F* **and** *g* ≠ *0* **and** *lt g* $adds_t$ *lt f*
using ‹*f* ≠ *0*› **by** (*rule GB-adds-lt*)
from *this*(*3*) *adds1* **have** *lt g* $adds_t$ *v* **by** (*rule adds-term-trans*)
with ‹*g* ∈ *reduced-GB ?F*› ‹*g* ≠ *0*› ‹*v* ∈ *keys p*› **show** *is-red* (*reduced-GB ?F*) *p*
by (*rule is-red-addsI*)
qed
qed

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 −
let *?S* = *lt* ‘ *G*
let *?G* = *monomial* (*1*::′*b*) ‘ *?S*
from *assms*(*3*) **have** *?S* ⊆ *Keys G* **by** (*auto intro*: *lt-in-keys in-KeysI*)
hence *component-of-term* ‘ *?S* ⊆ *component-of-term* ‘ *Keys G*
and ∗: *pp-of-term* ‘ *?S* ⊆ *pp-of-term* ‘ *Keys G* **by** (*rule image-mono*)+
from *this*(*1*) **have** *finite* (*component-of-term* ‘ *?S*) **using** *assms*(*1*) **by** (*rule finite-subset*)
moreover from ∗ **have** *pp-of-term* ‘ *?S* ⊆ *dgrad-set d m*
proof (*rule subset-trans*)
from *assms*(*2*) **show** *pp-of-term* ‘ *Keys G* ⊆ *dgrad-set d m* **by** (*auto simp*: *dgrad-p-set-def Keys-def*)
qed
ultimately have *is-red* (*reduced-GB ?G*) = *is-red ?G* **by** (*rule is-red-reduced-GB-monomial-dgrad-set*)
also from *assms*(*3*) **have** ... = *is-red G* **by** (*rule is-red-monomial-lt*)
finally show *?thesis* **.**
qed

lemma *reduced-GB-monomial-lt-reduced-GB-dgrad-p-set*:
assumes *finite* (*component-of-term* ‘ *Keys F*) **and** $F \subseteq$ *dgrad-p-set d m*
shows *reduced-GB* (*monomial 1* ‘ *lt* ‘ *reduced-GB F*) = *monomial* (*1*::′b::*field*) ‘
lt ‘ *reduced-GB F*
proof (*rule reduced-GB-unique*)
let *?G* = *reduced-GB F*
let *?F* = *monomial* (*1*::′b) ‘ *lt* ‘ *?G*

from *dgrad assms* **have** $0 \notin$ *?G* **and** *ar*: *is-auto-reduced ?G* **and** *finite ?G*
by (*rule reduced-GB-nonzero-dgrad-p-set*, *rule reduced-GB-is-auto-reduced-dgrad-p-set*,
rule finite-reduced-GB-dgrad-p-set)
from *this*(*3*) **show** *finite ?F* **by** (*intro finite-imageI*)

show *is-reduced-GB ?F* **unfolding** *is-reduced-GB-def*
proof (*intro conjI monomial-set-is-GB*)
show *is-monomial-set ?F* **by** (*auto intro*!: *is-monomial-setI monomial-is-monomial*)
next
show *is-monic-set ?F* **by** (*simp add*: *is-monic-set-def*)
next
show $0 \notin$ *?F* **by** (*auto simp*: *monomial-0-iff*)
next
show *is-auto-reduced ?F* **unfolding** *is-auto-reduced-def*
proof (*intro ballI notI*)
fix *f*
assume $f \in$ *?F*
then obtain *g* **where** $g \in$ *?G* **and** *f*: *f* = *monomial 1* (*lt g*) **by** *blast*
assume *is-red* (*?F* − {*f*}) *f*
then obtain *f′ v* **where** *f′* ∈ *?F* − {*f*} **and** *v* ∈ *keys f* **and** *f′* ≠ *0* **and**
adds1: *lt f′* $adds_t$ *v*
by (*rule is-red-addsE*)
from *this*(*1*) **have** *f′* ∈ *?F* **and** *f′* ≠ *f* **by** *simp-all*
from *this*(*1*) **obtain** *g′* **where** *g′* ∈ *?G* **and** *f′*: *f′* = *monomial 1* (*lt g′*) **by**
blast
from ‹*v* ∈ *keys f*› **have** *v*: *v* = *lt g* **by** (*simp add*: *f*)
from *ar* ‹*g* ∈ *?G*› **have** ¬ *is-red* (*?G* − {*g*}) *g* **by** (*rule is-auto-reducedD*)
moreover have *is-red* (*?G* − {*g*}) *g*
proof (*rule is-red-addsI*)
from ‹*g′* ∈ *?G*› ‹*f′* ≠ *f*› **show** *g′* ∈ *?G* − {*g*} **by** (*auto simp*: *f f′*)
next
from ‹*g′* ∈ *?G*› ‹*0* ∉ *?G*› **show** *g′* ≠ *0* **by** *blast*
next
from ‹*g* ∈ *?G*› ‹*0* ∉ *?G*› **have** *g* ≠ *0* **by** *blast*
thus *lt g* ∈ *keys g* **by** (*rule lt-in-keys*)
next
from *adds1* **show** *adds2*: *lt g′* $adds_t$ *lt g* **by** (*simp add*: *v f′ lt-monomial*)
qed
ultimately show *False* **..**
qed

```
  qed
qed (fact refl)

end

end

end
```

8 Preliminaries

```
theory Dube-Prelims
  imports Groebner-Bases.General
begin
```

8.1 Sets

```
lemma card-geq-ex-subset:
  assumes card A ≥ n
  obtains B where card B = n and B ⊆ A
  using assms
proof (induct n arbitrary: thesis)
  case base: 0
  show ?case
  proof (rule base(1))
    show card {} = 0 by simp
  next
    show {} ⊆ A ..
  qed
next
  case ind: (Suc n)
  from ind(3) have n < card A by simp
  obtain B where card: card B = n and B ⊆ A
  proof (rule ind(1))
    from ‹n < card A› show n ≤ card A by simp
  qed
  from ‹n < card A› have card A ≠ 0 by simp
  with card.infinite[of A] have finite A by blast
  let ?C = A − B
  have ?C ≠ {}
  proof
    assume A − B = {}
    hence A ⊆ B by simp
    from this ‹B ⊆ A› have A = B ..
    from ‹n < card A› show False unfolding ‹A = B› card by simp
  qed
  then obtain c where c ∈ ?C by auto
  hence c ∉ B by simp
  hence B − {c} = B by simp
```

show *?case*
proof (*rule ind*(*2*))
thm *card.insert-remove*
have *card* $(B \cup \{c\})$ = *card* (*insert c B*) **by** *simp*
also have ... = *Suc* (*card* $(B - \{c\})$)
by (*rule card.insert-remove*, *rule finite-subset*, *fact* ‹$B \subseteq A$›, *fact*)
finally show *card* $(B \cup \{c\})$ = *Suc n* **unfolding** ‹$B - \{c\} = B$› *card* **.**
next
show $B \cup \{c\} \subseteq A$ **unfolding** *Un-subset-iff*
proof (*intro conjI*, *fact*)
from ‹$c \in$ *?C*› **show** $\{c\} \subseteq A$ **by** *auto*
qed
qed
qed

lemma *card-2-E-1*:
assumes *card* A = *2* **and** $x \in A$
obtains y **where** $x \neq y$ **and** $A = \{x, y\}$
proof −
have $A - \{x\} \neq \{\}$
proof
assume $A - \{x\} = \{\}$
with *assms*(*2*) **have** $A = \{x\}$ **by** *auto*
hence *card* A = *1* **by** *simp*
with *assms* **show** *False* **by** *simp*
qed
then obtain y **where** $y \in A - \{x\}$ **by** *auto*
hence $y \in A$ **and** $x \neq y$ **by** *auto*
show *?thesis*
proof
show $A = \{x, y\}$
proof (*rule sym*, *rule card-seteq*)
from *assms*(*1*) **show** *finite A* **using** *card.infinite* **by** *fastforce*
next
from ‹$x \in A$› ‹$y \in A$› **show** $\{x, y\} \subseteq A$ **by** *simp*
next
from ‹$x \neq y$› **show** *card* $A \leq$ *card* $\{x, y\}$ **by** (*simp add*: *assms*(*1*))
qed
qed *fact*
qed

lemma *card-2-E*:
assumes *card* A = *2*
obtains x y **where** $x \neq y$ **and** $A = \{x, y\}$
proof −
from *assms* **have** $A \neq \{\}$ **by** *auto*
then obtain x **where** $x \in A$ **by** *blast*
with *assms* **obtain** y **where** $x \neq y$ **and** $A = \{x, y\}$ **by** (*rule card-2-E-1*)
thus *?thesis* **..**

qed

8.2 Sums

lemma *sum-tail-nat*: $0 < b \Longrightarrow a \leq (b{::}nat) \Longrightarrow sum\, f\, \{a..b\} = f\, b + sum\, f\, \{a..b - 1\}$
by (*metis One-nat-def Suc-pred add.commute not-le sum.cl-ivl-Suc*)

lemma *sum-atLeast-Suc-shift*: $0 < b \Longrightarrow a \leq b \Longrightarrow sum\, f\, \{Suc\ a..b\} = (\sum i{=}a..b - 1.\ f\ (Suc\ i))$
by (*metis Suc-pred′ sum.shift-bounds-cl-Suc-ivl*)

lemma *sum-split-nat-ivl*:
$a \leq Suc\ j \Longrightarrow j \leq b \Longrightarrow sum\ f\ \{a..j\} + sum\ f\ \{Suc\ j..b\} = sum\ f\ \{a..b\}$
by (*metis Suc-eq-plus1 le-Suc-ex sum.ub-add-nat*)

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 −
from *assms* **have** *count-list xs x* $\neq 0$ **by** *simp*
hence $x \in$ *set xs* **by** (*simp only*: *count-list-0-iff not-not*)
then obtain *ys zs* **where** *xs*: *xs* = *ys* @ *x* # *zs* **and** $x \notin$ *set ys* **by** (*meson split-list-first*)
hence *count-list xs x* = *Suc* (*count-list zs x*) **by** (*simp*)
with *assms* **have** *count-list zs x* $\neq 0$ **by** *simp*
hence $x \in$ *set zs* **by** (*simp only*: *count-list-0-iff not-not*)
then obtain *j* **where** $j <$ *length zs* **and** $x =$ *zs* ! *j* **by** (*metis in-set-conv-nth*)
show *?thesis*
proof
show *length ys* < *length ys* + *Suc j* **by** *simp*
next
from ‹$j <$ *length zs*› **show** *length ys* + *Suc j* < *length xs* **by** (*simp add*: *xs*)
next
show *xs* ! *length ys* = *x* **by** (*simp add*: *xs*)
next
show *xs* ! (*length ys* + *Suc j*) = *x*
by (*simp only*: *xs* ‹*x* = *zs* ! *j*› *nth-append-length-plus nth-Cons-Suc*)
qed
qed

8.4 *listset*

lemma *listset-Cons*: *listset* (*x* # *xs*) = ($\bigcup y \in x.$ (#) *y* ‘ *listset xs*)
by (*auto simp*: *set-Cons-def*)

lemma *listset-ConsI*: $y \in x \Longrightarrow ys' \in$ *listset xs* $\Longrightarrow ys = y$ # $ys' \Longrightarrow ys \in$ *listset* (*x* # *xs*) by (*simp add*: *set-Cons-def*)

lemma *listset-ConsE*:
assumes $ys \in listset\ (x\#\ xs)$
obtains $y\ ys'$ **where** $y \in x$ **and** $ys' \in listset\ xs$ **and** $ys = y\ \#\ ys'$
using *assms* **by** (*auto simp*: *set-Cons-def*)

lemma *listsetI*:
$length\ ys = length\ xs \Longrightarrow (\bigwedge i.\ i < length\ xs \Longrightarrow ys\ !\ i \in xs\ !\ i) \Longrightarrow ys \in listset\ xs$
by (*induct ys xs rule*: *list-induct2*)
(*simp-all*, *smt Suc-mono list.sel*(*3*) *mem-Collect-eq nth-Cons-0 nth-tl set-Cons-def zero-less-Suc*)

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 −
from *assms* **have** $length\ ys = length\ xs \land (\forall\, i{<}length\ xs.\ ys\ !\ i \in xs\ !\ i)$
proof (*induct xs arbitrary*: *ys*)
case *Nil*
thus *?case* **by** *simp*
next
case (*Cons x xs*)
from *Cons.prems* **obtain** $y\ ys'$ **where** $y \in x$ **and** $ys' \in listset\ xs$ **and** *ys*: $ys = y\ \#\ ys'$
by (*rule listset-ConsE*)
from *this*(*2*) **have** $length\ ys' = length\ xs \land (\forall\, i{<}length\ xs.\ ys'\ !\ i \in xs\ !\ i)$ **by** (*rule Cons.hyps*)
hence *1*: $length\ ys' = length\ xs$ **and** *2*: $\bigwedge i.\ i < length\ xs \Longrightarrow ys'\ !\ i \in xs\ !\ i$ **by** *simp-all*
show *?case*
proof (*intro conjI allI impI*)
fix *i*
assume $i < length\ (x\ \#\ xs)$
show $ys\ !\ i \in (x\ \#\ xs)\ !\ i$
proof (*cases i*)
case *0*
with ‹$y \in x$› **show** *?thesis* **by** (*simp add*: *ys*)
next
case (*Suc j*)
with ‹$i < length\ (x\ \#\ xs)$› **have** $j < length\ xs$ **by** *simp*
hence $ys'\ !\ j \in xs\ !\ j$ **by** (*rule 2*)
thus *?thesis* **by** (*simp add*: *ys* ‹$i = Suc\ j$›)
qed
qed (*simp add*: *ys 1*)
qed
thus $length\ ys = length\ xs$ **and** $\bigwedge i.\ i < length\ xs \Longrightarrow ys\ !\ i \in xs\ !\ i$ **by** *simp-all*
qed

lemma *listset-singletonI*: $a \in A \Longrightarrow ys = [a] \Longrightarrow ys \in listset\ [A]$
  by *simp*

lemma *listset-singletonE*:
  assumes $ys \in listset\ [A]$
  obtains *a* **where** $a \in A$ **and** $ys = [a]$
  using *assms* **by** *auto*

lemma *listset-doubletonI*: $a \in A \Longrightarrow b \in B \Longrightarrow ys = [a, b] \Longrightarrow ys \in listset\ [A, B]$
  by (*simp add*: *set-Cons-def*)

lemma *listset-doubletonE*:
  assumes $ys \in listset\ [A, B]$
  obtains *a b* **where** $a \in A$ **and** $b \in B$ **and** $ys = [a, b]$
  using *assms* **by** (*auto simp*: *set-Cons-def*)

lemma *listset-appendI*:
  $ys1 \in listset\ xs1 \Longrightarrow ys2 \in listset\ xs2 \Longrightarrow ys = ys1\ @\ ys2 \Longrightarrow ys \in listset\ (xs1\ @\ xs2)$
  by (*induct xs1 arbitrary*: *ys ys1 ys2*)
    (*simp*, *auto simp del*: *listset.simps elim*!: *listset-ConsE intro*!: *listset-ConsI*)

lemma *listset-appendE*:
  assumes $ys \in listset\ (xs1\ @\ xs2)$
  obtains *ys1 ys2* **where** $ys1 \in listset\ xs1$ **and** $ys2 \in listset\ xs2$ **and** $ys = ys1\ @\ ys2$
  using *assms*
proof (*induct xs1 arbitrary*: *thesis ys*)
  case *Nil*
  have $[] \in listset\ []$ **by** *simp*
  moreover from *Nil*(*2*) **have** $ys \in listset\ xs2$ **by** *simp*
  ultimately show *?case* **by** (*rule Nil*) *simp*
next
  case (*Cons x xs1*)
  from *Cons.prems*(*2*) **have** $ys \in listset\ (x\ \#\ (xs1\ @\ xs2))$ **by** *simp*
  then obtain *y ys'* **where** $y \in x$ **and** $ys' \in listset\ (xs1\ @\ xs2)$ **and** *ys*: $ys = y\ \#\ ys'$
    by (*rule listset-ConsE*)
  from - *this*(*2*) **obtain** *ys1 ys2* **where** *ys1*: $ys1 \in listset\ xs1$ **and** $ys2 \in listset\ xs2$
    and *ys'*: $ys' = ys1\ @\ ys2$ **by** (*rule Cons.hyps*)
  show *?case*
  proof (*rule Cons.prems*)
    from ‹$y \in x$› *ys1 refl* **show** $y\ \#\ ys1 \in listset\ (x\ \#\ xs1)$ **by** (*rule listset-ConsI*)
  next
    show $ys = (y\ \#\ ys1)\ @\ ys2$ **by** (*simp add*: *ys ys'*)
  qed *fact*

qed

lemma *listset-map-imageI*: $ys' \in listset\ xs \Longrightarrow ys = map\ f\ ys' \Longrightarrow ys \in listset$ $(map\ ((`)\ f)\ xs)$
  by (*induct xs arbitrary*: *ys ys'*)
    (*simp*, *auto simp del*: *listset.simps elim*!: *listset-ConsE intro*!: *listset-ConsI*)

lemma *listset-map-imageE*:
  assumes $ys \in listset\ (map\ ((`)\ f)\ xs)$
  obtains *ys'* **where** $ys' \in listset\ xs$ **and** $ys = map\ f\ ys'$
  using *assms*
proof (*induct xs arbitrary*: *thesis ys*)
  case *Nil*
  from *Nil*(*2*) **have** $ys = map\ f\ []$ **by** *simp*
  with - **show** *?case* **by** (*rule Nil*) *simp*
next
  case (*Cons x xs*)
  from *Cons.prems*(*2*) **have** $ys \in listset\ (f\ `\ x\ \#\ map\ ((`)\ f)\ xs)$ **by** *simp*
  then obtain *y ys'* **where** $y \in f\ `\ x$ **and** $ys' \in listset\ (map\ ((`)\ f)\ xs)$ **and** *ys*: $ys = y\ \#\ ys'$
    by (*rule listset-ConsE*)
  from - *this*(*2*) **obtain** *ys1* **where** *ys1*: $ys1 \in listset\ xs$ **and** *ys'*: $ys' = map\ f\ ys1$ **by** (*rule Cons.hyps*)
  from ‹$y \in f\ `\ x$› **obtain** *y1* **where** $y1 \in x$ **and** *y*: $y = f\ y1$ **..**
  show *?case*
  proof (*rule Cons.prems*)
   from ‹$y1 \in x$› *ys1 refl* **show** $y1\ \#\ ys1 \in listset\ (x\ \#\ xs)$ **by** (*rule listset-ConsI*)
  qed (*simp add*: *ys ys' y*)
qed

lemma *listset-permE*:
  assumes $ys \in listset\ xs$ **and** *bij-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 −
  from *assms*(*1*) **have** *len-ys*: $length\ ys = length\ xs$ **by** (*rule listsetD*)
  from *assms*(*2*) **have** $card\ \{..<length\ xs\} = card\ \{..<length\ xs'\}$ **by** (*rule bij-betw-same-card*)
  hence *len-xs*: $length\ xs = length\ xs'$ **by** *simp*
  define *ys'* **where** $ys' = map\ (\lambda i.\ ys\ !\ (f\ i))\ [0..<length\ ys]$
  have *1*: $ys'\ !\ i = ys\ !\ f\ i$ **if** $i < length\ ys$ **for** *i* **using** *that* **by** (*simp add*: *ys'-def*)
  show *?thesis*
  proof
    show $ys' \in listset\ xs'$
    proof (*rule listsetI*)
      show $length\ ys' = length\ xs'$ **by** (*simp add*: *ys'-def len-ys len-xs*)

      fix *i*
      assume $i < length\ xs'$

    hence i < *length xs* **by** (*simp only*: *len-xs*)
    hence i < *length ys* **by** (*simp only*: *len-ys*)
    hence *ys'* ! i = *ys* ! (f i) **by** (*rule 1*)
    also from *assms*(*1*) **have** ... ∈ *xs* ! (f i)
    proof (*rule listsetD*)
      from ‹i < *length xs*› **have** i ∈ {..<*length xs*} **by** *simp*
      hence f i ∈ f ‘ {..<*length xs*} **by** (*rule imageI*)
      also from *assms*(*2*) **have** ... = {..<*length xs'*} **by** (*simp add*: *bij-betw-def*)
      finally show f i < *length xs* **by** (*simp add*: *len-xs*)
    qed
    also have ... = *xs'* ! i **by** (*rule sym*) (*rule assms*(*3*), *fact*)
    finally show *ys'* ! i ∈ *xs'* ! i **.**
  qed
next
  show *length ys'* = *length ys* **by** (*simp add*: *ys'-def*)
qed (*rule 1*)
qed

lemma *listset-closed-map*:
  assumes *ys* ∈ *listset xs* **and** ⋀x y. x ∈ *set xs* ⟹ y ∈ x ⟹ f y ∈ x
  shows *map f ys* ∈ *listset xs*
  using *assms*
proof (*induct xs arbitrary*: *ys*)
  case *Nil*
  from *Nil*(*1*) **show** *?case* **by** *simp*
next
  case (*Cons x xs*)
  from *Cons.prems*(*1*) **obtain** *y ys'* **where** y ∈ x **and** *ys'* ∈ *listset xs* **and** *ys*: *ys* = *y* # *ys'*
    by (*rule listset-ConsE*)
  show *?case*
  proof (*rule listset-ConsI*)
    from - ‹y ∈ x› **show** f y ∈ x **by** (*rule Cons.prems*) *simp*
  next
    show *map f ys'* ∈ *listset xs*
    proof (*rule Cons.hyps*)
      fix *x0 y0*
      assume *x0* ∈ *set xs*
      hence *x0* ∈ *set* (*x* # *xs*) **by** *simp*
      moreover assume *y0* ∈ *x0*
      ultimately show f *y0* ∈ *x0* **by** (*rule Cons.prems*)
    qed *fact*
  qed (*simp add*: *ys*)
qed

lemma *listset-closed-map2*:
  assumes *ys1* ∈ *listset xs* **and** *ys2* ∈ *listset xs*
    and ⋀x *y1 y2*. x ∈ *set xs* ⟹ *y1* ∈ x ⟹ *y2* ∈ x ⟹ f *y1 y2* ∈ x
  shows *map2 f ys1 ys2* ∈ *listset xs*

```
  using assms
proof (induct xs arbitrary: ys1 ys2)
  case Nil
  from Nil(1) show ?case by simp
next
  case (Cons x xs)
  from Cons.prems(1) obtain y1 ys1' where y1 ∈ x and ys1' ∈ listset xs and
ys1: ys1 = y1 # ys1'
    by (rule listset-ConsE)
  from Cons.prems(2) obtain y2 ys2' where y2 ∈ x and ys2' ∈ listset xs and
ys2: ys2 = y2 # ys2'
    by (rule listset-ConsE)
  show ?case
  proof (rule listset-ConsI)
    from - ‹y1 ∈ x› ‹y2 ∈ x› show f y1 y2 ∈ x by (rule Cons.prems) simp
  next
    show map2 f ys1' ys2' ∈ listset xs
    proof (rule Cons.hyps)
      fix x' y1' y2'
      assume x' ∈ set xs
      hence x' ∈ set (x # xs) by simp
      moreover assume y1' ∈ x' and y2' ∈ x'
      ultimately show f y1' y2' ∈ x' by (rule Cons.prems)
    qed fact+
  qed (simp add: ys1 ys2)
qed

lemma listset-empty-iff: listset xs = {} ⟷ {} ∈ set xs
  by (induct xs) (auto simp: listset-Cons simp del: listset.simps(2))

lemma listset-mono:
  assumes length xs = length ys and ⋀i. i < length ys ⟹ xs ! i ⊆ ys ! i
  shows listset xs ⊆ listset ys
  using assms
proof (induct xs ys rule: list-induct2)
  case Nil
  show ?case by simp
next
  case (Cons x xs y ys)
  show ?case
  proof
    fix zs'
    assume zs' ∈ listset (x # xs)
    then obtain z zs where z ∈ x and zs: zs ∈ listset xs and zs': zs' = z # zs
      by (rule listset-ConsE)
    have 0 < length (y # ys) by simp
    hence (x # xs) ! 0 ⊆ (y # ys) ! 0 by (rule Cons.prems)
    hence x ⊆ y by simp
    with ‹z ∈ x› have z ∈ y ..
```

moreover from *zs* **have** $zs \in listset\ ys$
proof
show $listset\ xs \subseteq listset\ ys$
proof (*rule Cons.hyps*)
fix i
assume $i < length\ ys$
hence $Suc\ i < length\ (y \# ys)$ **by** *simp*
hence $(x \# xs) ! Suc\ i \subseteq (y \# ys) ! Suc\ i$ **by** (*rule Cons.prems*)
thus $xs ! i \subseteq ys ! i$ **by** *simp*
qed
qed
ultimately show $zs' \in listset\ (y \# ys)$ **using** zs' **by** (*rule listset-ConsI*)
qed
qed

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-decomp A ss* $\longleftrightarrow$ *bij-betw sum-list* (*listset ss*) *A*

lemma *direct-decompI*:
inj-on sum-list (*listset ss*) $\Longrightarrow$ *sum-list* ‘ *listset ss* = *A* $\Longrightarrow$ *direct-decomp A ss*
by (*simp add*: *direct-decomp-def bij-betw-def*)

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-decomp A ss
by (*auto simp*: *direct-decomp-def intro*!: *bij-betwI'*) *blast*

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*
using *assms* **by** (*auto simp*: *direct-decomp-def bij-betw-def*)

lemma *direct-decompE*:
assumes *direct-decomp A ss* **and** $a \in A$
obtains *qs* **where** $qs \in$ *listset ss* **and** $a =$ *sum-list qs*
using *assms* **by** (*auto simp*: *direct-decomp-def bij-betw-def*)

lemma *direct-decomp-unique*:
direct-decomp A ss $\Longrightarrow$ $qs \in$ *listset ss* $\Longrightarrow$ $qs' \in$ *listset ss* $\Longrightarrow$ *sum-list qs* = *sum-list qs'* $\Longrightarrow$
$qs = qs'$
by (*auto dest*: *direct-decompD simp*: *inj-on-def*)

lemma *direct-decomp-singleton*: *direct-decomp A* [*A*]
proof (*rule direct-decompI-alt*)
fix *qs*
assume $qs \in$ *listset* [*A*]
then obtain *q* **where** $q \in A$ **and** $qs = [q]$ **by** (*rule listset-singletonE*)
thus *sum-list qs* $\in A$ **by** *simp*
next
fix *a*
assume $a \in A$
show ∃!*qs*∈*listset* [*A*]. $a =$ *sum-list qs*
proof (*intro ex1I conjI allI impI*)
from ‹$a \in A$› *refl* **show** $[a] \in$ *listset* [*A*] **by** (*rule listset-singletonI*)
next
fix *qs*
assume $qs \in$ *listset* [*A*] $\wedge$ $a =$ *sum-list qs*
hence *a*: $a =$ *sum-list qs* **and** $qs \in$ *listset* [*A*] **by** *simp-all*
from *this*(*2*) **obtain** *b* **where** *qs*: $qs = [b]$ **by** (*rule listset-singletonE*)
with *a* **show** $qs = [a]$ **by** *simp*
qed *simp-all*
qed

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 −
from *assms*(*1*) **have** *1*: *inj-on f* {*0*..<*length xs*} **and** *2*: *f* ‘ {*0*..<*length xs*} = {*0*..<*length ys*}
by (*simp-all add*: *bij-betw-def lessThan-atLeast0*)
let *?f* = (!) *ys* ∘ *f*
have *xs* = *map ?f* [*0*..<*length xs*] **unfolding** *list-eq-iff-nth-eq*
proof (*intro conjI allI impI*)
fix *i*
assume $i <$ *length xs*

hence *xs* ! *i* = *ys* ! *f i* **by** (*rule assms*(*2*))
also from ‹*i* < *length xs*› **have** . . . = *map* ((!) *ys* ∘ *f*) [*0*..<*length xs*] ! *i* **by** *simp*
finally show *xs* ! *i* = *map* ((!) *ys* ∘ *f*) [*0*..<*length xs*] ! *i* **.**
qed *simp*
hence *mset xs* = *mset* (*map ?f* [*0*..<*length xs*]) **by** (*rule arg-cong*)
also have . . . = *image-mset* ((!) *ys*) (*image-mset f* (*mset-set* {*0*..<*length xs*}))
by (*simp flip*: *image-mset.comp*)
also from *1* **have** . . . = *image-mset* ((!) *ys*) (*mset-set* {*0*..<*length ys*})
by (*simp add*: *image-mset-mset-set 2*)
also have . . . = *mset* (*map* ((!) *ys*) [*0*..<*length ys*]) **by** *simp*
finally show *mset xs* = *mset ys* **by** (*simp only*: *map-nth*)
qed

lemma *direct-decomp-perm*:
assumes *direct-decomp A ss1* **and** *mset ss1* = *mset ss2*
shows *direct-decomp A ss2*
proof −
from *assms*(*2*) **have** *len-ss1*: *length ss1* = *length ss2*
using *mset-eq-length* **by** *blast*
from *assms*(*2*) **obtain** *f* **where** ‹*f permutes* {..<*length ss2*}›
‹*permute-list f ss2* = *ss1*›
by (*rule mset-eq-permutation*)
then have *f-bij*: *bij-betw f* {..<*length ss2*} {..<*length ss1*}
and *f*: $\bigwedge$*i*. *i* < *length ss2* ⟹ *ss1* ! *i* = *ss2* ! *f i*
by (*auto simp add*: *permutes-imp-bij permute-list-nth*)
define *g* **where** *g* = *inv-into* {..<*length ss2*} *f*
from *f-bij* **have** *g-bij*: *bij-betw g* {..<*length ss1*} {..<*length ss2*}
unfolding *g-def len-ss1* **by** (*rule bij-betw-inv-into*)
have *f-g*: *f* (*g i*) = *i* **if** *i* < *length ss1* **for** *i*
proof −
from *that f-bij* **have** *i* ∈ *f* ‘ {..<*length ss2*} **by** (*simp add*: *bij-betw-def len-ss1*)
thus *?thesis* **by** (*simp only*: *f-inv-into-f g-def*)
qed
have *g-f*: *g* (*f i*) = *i* **if** *i* < *length ss2* **for** *i*
proof −
from *f-bij* **have** *inj-on f* {..<*length ss2*} **by** (*simp only*: *bij-betw-def*)
moreover from *that* **have** *i* ∈ {..<*length ss2*} **by** *simp*
ultimately show *?thesis* **by** (*simp add*: *g-def*)
qed
have *g*: *ss2* ! *i* = *ss1* ! *g i* **if** *i* < *length ss1* **for** *i*
proof −
from *that* **have** *i* ∈ {..<*length ss2*} **by** (*simp add*: *len-ss1*)
hence *g i* ∈ *g* ‘ {..<*length ss2*} **by** (*rule imageI*)
also from *g-bij* **have** . . . = {..<*length ss2*} **by** (*simp only*: *len-ss1 bij-betw-def*)
finally have *g i* < *length ss2* **by** *simp*
hence *ss1* ! *g i* = *ss2* ! *f* (*g i*) **by** (*rule f*)
with *that* **show** *?thesis* **by** (*simp only*: *f-g*)
qed

show *?thesis*
proof (*rule direct-decompI-alt*)
fix *qs2*
assume $qs2 \in listset\ ss2$
then obtain *qs1* **where** *qs1-in*: $qs1 \in listset\ ss1$ **and** *len-qs1*: $length\ qs1 = length\ qs2$
and $*$: $\bigwedge i.\ i < length\ qs2 \Longrightarrow qs1\ !\ i = qs2\ !\ f\ i$ **using** *f-bij f* **by** (*rule listset-permE*) *blast+*
from ‹$qs2 \in listset\ ss2$› **have** $length\ qs2 = length\ ss2$ **by** (*rule listsetD*)
with *f-bij* **have** $bij\text{-}betw\ f\ \{..<length\ qs1\}\ \{..<length\ qs2\}$ **by** (*simp only*: *len-qs1 len-ss1*)
hence $mset\ qs1 = mset\ qs2$ **using** $*$ **by** (*rule mset-bij*) (*simp only*: *len-qs1*)
hence $sum\text{-}list\ qs2 = sum\text{-}list\ qs1$ **by** (*simp flip*: *sum-mset-sum-list*)
also from *assms*(*1*) *qs1-in* **have** $\ldots \in A$ **by** (*rule direct-decompD*)
finally show $sum\text{-}list\ qs2 \in A$ **.**
next
fix *a*
assume $a \in A$
with *assms*(*1*) **obtain** *qs* **where** *a*: $a = sum\text{-}list\ qs$ **and** *qs-in*: $qs \in listset\ ss1$
by (*rule direct-decompE*)
from *qs-in* **obtain** *qs2* **where** *qs2-in*: $qs2 \in listset\ ss2$ **and** *len-qs2*: $length\ qs2 = length\ qs$
and *1*: $\bigwedge i.\ i < length\ qs \Longrightarrow qs2\ !\ i = qs\ !\ g\ i$ **using** *g-bij g* **by** (*rule listset-permE*) *blast+*
show $\exists!qs\in listset\ ss2.\ a = sum\text{-}list\ qs$
proof (*intro ex1I conjI allI impI*)
from *qs-in* **have** *len-qs*: $length\ qs = length\ ss1$ **by** (*rule listsetD*)
with *g-bij* **have** *g-bij2*: $bij\text{-}betw\ g\ \{..<length\ qs2\}\ \{..<length\ qs\}$ **by** (*simp only*: *len-qs2 len-ss1*)
hence $mset\ qs2 = mset\ qs$ **using** *1* **by** (*rule mset-bij*) (*simp only*: *len-qs2*)
thus *a2*: $a = sum\text{-}list\ qs2$ **by** (*simp only*: *a flip*: *sum-mset-sum-list*)

fix qs'
assume $qs' \in listset\ ss2 \wedge a = sum\text{-}list\ qs'$
hence qs'*-in*: $qs' \in listset\ ss2$ **and** a': $a = sum\text{-}list\ qs'$ **by** *simp-all*
from *this*(*1*) **obtain** *qs1* **where** *qs1-in*: $qs1 \in listset\ ss1$ **and** *len-qs1*: $length\ qs1 = length\ qs'$
and *2*: $\bigwedge i.\ i < length\ qs' \Longrightarrow qs1\ !\ i = qs'\ !\ f\ i$ **using** *f-bij f* **by** (*rule listset-permE*) *blast+*
from ‹$qs' \in listset\ ss2$› **have** $length\ qs' = length\ ss2$ **by** (*rule listsetD*)
with *f-bij* **have** $bij\text{-}betw\ f\ \{..<length\ qs1\}\ \{..<length\ qs'\}$ **by** (*simp only*: *len-qs1 len-ss1*)
hence $mset\ qs1 = mset\ qs'$ **using** *2* **by** (*rule mset-bij*) (*simp only*: *len-qs1*)
hence $sum\text{-}list\ qs1 = sum\text{-}list\ qs'$ **by** (*simp flip*: *sum-mset-sum-list*)
hence $sum\text{-}list\ qs1 = sum\text{-}list\ qs$ **by** (*simp only*: *a flip*: a')
with *assms*(*1*) *qs1-in qs-in* **have** $qs1 = qs$ **by** (*rule direct-decomp-unique*)
show $qs' = qs2$ **unfolding** *list-eq-iff-nth-eq*
proof (*intro conjI allI impI*)
from qs'*-in* **have** $length\ qs' = length\ ss2$ **by** (*rule listsetD*)

```
      thus eq: length qs' = length qs2 by (simp only: len-qs2 len-qs len-ss1)

      fix i
      assume i < length qs'
      hence i < length qs2 by (simp only: eq)
      hence i ∈ {..<length qs2} and i < length qs and i < length ss1
        by (simp-all add: len-qs2 len-qs)
      from this(1) have g i ∈ g ` {..<length qs2} by (rule imageI)
      also from g-bij2 have ... = {..<length qs} by (simp only: bij-betw-def)
      finally have g i < length qs' by (simp add: eq len-qs2)
      from ‹i < length qs› have qs2 ! i = qs ! g i by (rule 1)
      also have ... = qs1 ! g i by (simp only: ‹qs1 = qs›)
      also from ‹g i < length qs'› have ... = qs' ! f (g i) by (rule 2)
      also from ‹i < length ss1› have ... = qs' ! i by (simp only: f-g)
      finally show qs' ! i = qs2 ! i by (rule sym)
    qed
  qed fact
 qed
qed

lemma direct-decomp-split-map:
  direct-decomp A (map f ss) ⟹ direct-decomp A (map f (filter P ss) @ map f
(filter (− P) ss))
proof (rule direct-decomp-perm)
  show mset (map f ss) = mset (map f (filter P ss) @ map f (filter (− P) ss))
    by simp (metis image-mset-union multiset-partition)
qed

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 (rule direct-decompI-alt)
  fix qs
  assume qs ∈ listset ?ss
  then obtain qs1 qs2 where qs1: qs1 ∈ listset ss and qs2: qs2 ∈ listset rs and
qs: qs = qs1 @ qs2
    by (rule listset-appendE)
  have sum-list qs = sum-list ((sum-list qs2) # qs1) by (simp add: qs add.commute)
  also from assms(1) have ... ∈ A
  proof (rule direct-decompD)
    from assms(2) qs2 have sum-list qs2 ∈ s by (rule direct-decompD)
    thus sum-list qs2 # qs1 ∈ listset (s # ss) using qs1 refl by (rule listset-ConsI)
  qed
  finally show sum-list qs ∈ A .
next
  fix a
  assume a ∈ A
```

with *assms*(*1*) **obtain** *qs1* **where** *qs1-in*: *qs1* ∈ *listset* (*s* # *ss*) **and** *a*: *a* = *sum-list qs1*
by (*rule direct-decompE*)
from *qs1-in* **obtain** *qs11 qs12* **where** *qs11* ∈ *s* **and** *qs12-in*: *qs12* ∈ *listset ss*
and *qs1*: *qs1* = *qs11* # *qs12* **by** (*rule listset-ConsE*)
from *assms*(*2*) *this*(*1*) **obtain** *qs2* **where** *qs2-in*: *qs2* ∈ *listset rs* **and** *qs11*: *qs11* = *sum-list qs2*
by (*rule direct-decompE*)
let *?qs* = *qs12* @ *qs2*
show ∃!*qs*∈*listset ?ss*. *a* = *sum-list qs*
proof (*intro ex1I conjI allI impI*)
from *qs12-in qs2-in refl* **show** *?qs* ∈ *listset ?ss* **by** (*rule listset-appendI*)

show *a* = *sum-list ?qs* **by** (*simp add*: *a qs1 qs11 add.commute*)

fix *qs0*
assume *qs0* ∈ *listset ?ss* ∧ *a* = *sum-list qs0*
hence *qs0-in*: *qs0* ∈ *listset ?ss* **and** *a2*: *a* = *sum-list qs0* **by** *simp-all*
from *this*(*1*) **obtain** *qs01 qs02* **where** *qs01-in*: *qs01* ∈ *listset ss* **and** *qs02-in*: *qs02* ∈ *listset rs*
and *qs0*: *qs0* = *qs01* @ *qs02* **by** (*rule listset-appendE*)
note *assms*(*1*)
moreover from - *qs01-in refl* **have** (*sum-list qs02*) # *qs01* ∈ *listset* (*s* # *ss*) (**is** *?qs'* ∈ -)
proof (*rule listset-ConsI*)
from *assms*(*2*) *qs02-in* **show** *sum-list qs02* ∈ *s* **by** (*rule direct-decompD*)
qed
moreover note *qs1-in*
moreover from *a2* **have** *sum-list ?qs'* = *sum-list qs1* **by** (*simp add*: *qs0 a add.commute*)
ultimately have *?qs'* = *qs11* # *qs12* **unfolding** *qs1* **by** (*rule direct-decomp-unique*)
hence *qs11* = *sum-list qs02* **and** *1*: *qs01* = *qs12* **by** *simp-all*
from *this*(*1*) **have** *sum-list qs02* = *sum-list qs2* **by** (*simp only*: *qs11*)
with *assms*(*2*) *qs02-in qs2-in* **have** *qs02* = *qs2* **by** (*rule direct-decomp-unique*)
thus *qs0* = *qs12* @ *qs2* **by** (*simp only*: *1 qs0*)
qed
qed

lemma *sum-list-map-times*: *sum-list* (*map* ((∗) *x*) *xs*) = (*x*::*'a*::*semiring-0*) ∗ *sum-list xs*
by (*induct xs*) (*simp-all add*: *algebra-simps*)

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 (*rule direct-decompI-alt*)
fix *qs*
assume *qs* ∈ *listset ?ss*

then obtain *qs0* where *qs0-in*: $qs0 \in listset\ ss$ and *qs*: $qs = map\ ((*)\ x)\ qs0$
  by (*rule listset-map-imageE*)
 have $sum\text{-}list\ qs = x * sum\text{-}list\ qs0$ by (*simp only*: *qs sum-list-map-times*)
 moreover from *assms*(*1*) *qs0-in* have $sum\text{-}list\ qs0 \in A$ by (*rule direct-decompD*)
 ultimately show $sum\text{-}list\ qs \in (*)\ x\ `\ A$ by (*rule image-eqI*)
next
 fix *a*
 assume $a \in ?A$
 then obtain a' where $a' \in A$ and *a*: $a = x * a'$ ..
 from *assms*(*1*) *this*(*1*) obtain qs' where qs'*-in*: $qs' \in listset\ ss$ and a': $a' = sum\text{-}list\ qs'$
  by (*rule direct-decompE*)
 define *qs* where $qs = map\ ((*)\ x)\ qs'$
 show $\exists !qs \in listset\ ?ss.\ a = sum\text{-}list\ qs$
 proof (*intro ex1I conjI allI impI*)
  from qs'*-in qs-def* show $qs \in listset\ ?ss$ by (*rule listset-map-imageI*)

  fix *qs0*
  assume $qs0 \in listset\ ?ss \wedge a = sum\text{-}list\ qs0$
  hence $qs0 \in listset\ ?ss$ and *a0*: $a = sum\text{-}list\ qs0$ by *simp-all*
  from *this*(*1*) obtain *qs1* where *qs1-in*: $qs1 \in listset\ ss$ and *qs0*: $qs0 = map\ ((*)\ x)\ qs1$
   by (*rule listset-map-imageE*)
  show $qs0 = qs$
  proof (*cases* $x = 0$)
   case *True*
   from *qs1-in* have $length\ qs1 = length\ ss$ by (*rule listsetD*)
   moreover from qs'*-in* have $length\ qs' = length\ ss$ by (*rule listsetD*)
   ultimately show *?thesis* by (*simp add*: *qs-def qs0 list-eq-iff-nth-eq True*)
  next
   case *False*
   have $x * sum\text{-}list\ qs1 = a$ by (*simp only*: *a0 qs0 sum-list-map-times*)
   also have $\ldots = x * sum\text{-}list\ qs'$ by (*simp only*: a' *a*)
   finally have $sum\text{-}list\ qs1 = sum\text{-}list\ qs'$ using *False* by (*rule assms*(*2*))
   with *assms*(*1*) *qs1-in* qs'*-in* have $qs1 = qs'$ by (*rule direct-decomp-unique*)
   thus *?thesis* by (*simp only*: *qs0 qs-def*)
  qed
 qed (*simp only*: *a* a' *qs-def sum-list-map-times*)
qed

lemma *direct-decomp-appendD*:
 assumes *direct-decomp A* (*ss1* @ *ss2*)
 shows $\{\} \notin set\ ss2 \Longrightarrow direct\text{-}decomp\ (sum\text{-}list\ `\ listset\ ss1)\ ss1$ (is - $\Longrightarrow$ *?thesis1*)
  and $\{\} \notin set\ ss1 \Longrightarrow direct\text{-}decomp\ (sum\text{-}list\ `\ listset\ ss2)\ ss2$ (is - $\Longrightarrow$ *?thesis2*)
  and *direct-decomp A* [$sum\text{-}list\ `\ listset\ ss1$, $sum\text{-}list\ `\ listset\ ss2$] (is *direct-decomp* - *?ss*)
proof −
 have *rl*: *direct-decomp* ($sum\text{-}list\ `\ listset\ ts1$) *ts1*

if *direct-decomp A* (*ts1* @ *ts2*) **and** {} ∉ *set ts2* **for** *ts1 ts2*
proof (*intro direct-decompI inj-onI refl*)
fix *qs1 qs2*
assume *qs1*: *qs1* ∈ *listset ts1* **and** *qs2*: *qs2* ∈ *listset ts1*
assume *eq*: *sum-list qs1* = *sum-list qs2*
from *that*(*2*) **have** *listset ts2* ≠ {} **by** (*simp add*: *listset-empty-iff*)
then obtain *qs3* **where** *qs3*: *qs3* ∈ *listset ts2* **by** *blast*
note *that*(*1*)
moreover from *qs1 qs3 refl* **have** *qs1* @ *qs3* ∈ *listset* (*ts1* @ *ts2*) **by** (*rule listset-appendI*)
moreover from *qs2 qs3 refl* **have** *qs2* @ *qs3* ∈ *listset* (*ts1* @ *ts2*) **by** (*rule listset-appendI*)
moreover have *sum-list* (*qs1* @ *qs3*) = *sum-list* (*qs2* @ *qs3*) **by** (*simp add*: *eq*)
ultimately have *qs1* @ *qs3* = *qs2* @ *qs3* **by** (*rule direct-decomp-unique*)
thus *qs1* = *qs2* **by** *simp*
qed

{
assume {} ∉ *set ss2*
with *assms* **show** *?thesis1* **by** (*rule rl*)
}

{
from *assms* **have** *direct-decomp A* (*ss2* @ *ss1*)
by (*rule direct-decomp-perm*) *simp*
moreover assume {} ∉ *set ss1*
ultimately show *?thesis2* **by** (*rule rl*)
}

show *direct-decomp A ?ss*
proof (*rule direct-decompI-alt*)
fix *qs*
assume *qs* ∈ *listset ?ss*
then obtain *q1 q2* **where** *q1*: *q1* ∈ *sum-list* ‘ *listset ss1* **and** *q2*: *q2* ∈ *sum-list* ‘ *listset ss2*
and *qs*: *qs* = [*q1*, *q2*] **by** (*rule listset-doubletonE*)
from *q1* **obtain** *qs1* **where** *qs1*: *qs1* ∈ *listset ss1* **and** *q1*: *q1* = *sum-list qs1* ..
from *q2* **obtain** *qs2* **where** *qs2*: *qs2* ∈ *listset ss2* **and** *q2*: *q2* = *sum-list qs2* ..
from *qs1 qs2 refl* **have** *qs1* @ *qs2* ∈ *listset* (*ss1* @ *ss2*) **by** (*rule listset-appendI*)
with *assms* **have** *sum-list* (*qs1* @ *qs2*) ∈ *A* **by** (*rule direct-decompD*)
thus *sum-list qs* ∈ *A* **by** (*simp add*: *qs q1 q2*)
next
fix *a*
assume *a* ∈ *A*
with *assms* **obtain** *qs0* **where** *qs0-in*: *qs0* ∈ *listset* (*ss1* @ *ss2*) **and** *a*: *a* = *sum-list qs0*

by (*rule direct-decompE*)
from *this*(*1*) **obtain** *qs1 qs2* **where** *qs1*: *qs1* ∈ *listset ss1* **and** *qs2*: *qs2* ∈ *listset ss2*
and *qs0*: *qs0* = *qs1* @ *qs2* **by** (*rule listset-appendE*)
from *qs1* **have** *len-qs1*: *length qs1* = *length ss1* **by** (*rule listsetD*)
define *qs* **where** *qs* = [*sum-list qs1*, *sum-list qs2*]
show ∃!*qs*∈*listset ?ss*. *a* = *sum-list qs*
proof (*intro ex1I conjI*)
from *qs1* **have** *sum-list qs1* ∈ *sum-list* ' *listset ss1* **by** (*rule imageI*)
moreover from *qs2* **have** *sum-list qs2* ∈ *sum-list* ' *listset ss2* **by** (*rule imageI*)
ultimately show *qs* ∈ *listset ?ss* **using** *qs-def* **by** (*rule listset-doubletonI*)

fix *qs'*
assume *qs'* ∈ *listset ?ss* ∧ *a* = *sum-list qs'*
hence *qs'* ∈ *listset ?ss* **and** *a'*: *a* = *sum-list qs'* **by** *simp-all*
from *this*(*1*) **obtain** *q1 q2* **where** *q1*: *q1* ∈ *sum-list* ' *listset ss1*
and *q2*: *q2* ∈ *sum-list* ' *listset ss2* **and** *qs'*: *qs'* = [*q1*, *q2*] **by** (*rule listset-doubletonE*)
from *q1* **obtain** *qs1'* **where** *qs1'*: *qs1'* ∈ *listset ss1* **and** *q1*: *q1* = *sum-list qs1'* **..**
from *q2* **obtain** *qs2'* **where** *qs2'*: *qs2'* ∈ *listset ss2* **and** *q2*: *q2* = *sum-list qs2'* **..**
from *qs1'* **have** *len-qs1'*: *length qs1'* = *length ss1* **by** (*rule listsetD*)
note *assms*
moreover from *qs1' qs2' refl* **have** *qs1'* @ *qs2'* ∈ *listset* (*ss1* @ *ss2*) **by** (*rule listset-appendI*)
moreover note *qs0-in*
moreover have *sum-list* (*qs1'* @ *qs2'*) = *sum-list qs0* **by** (*simp add*: *a' qs' flip*: *a q1 q2*)
ultimately have *qs1'* @ *qs2'* = *qs0* **by** (*rule direct-decomp-unique*)
also have ... = *qs1* @ *qs2* **by** *fact*
finally show *qs'* = *qs* **by** (*simp add*: *qs-def qs' q1 q2 len-qs1 len-qs1'*)
qed (*simp add*: *qs-def a qs0*)
qed
qed

lemma *direct-decomp-Cons-zeroI*:
assumes *direct-decomp A ss*
shows *direct-decomp A* ({*0*} # *ss*)
proof (*rule direct-decompI-alt*)
fix *qs*
assume *qs* ∈ *listset* ({*0*} # *ss*)
then obtain *q qs'* **where** *q* ∈ {*0*} **and** *qs'* ∈ *listset ss* **and** *qs* = *q* # *qs'*
by (*rule listset-ConsE*)
from *this*(*1*, *3*) **have** *sum-list qs* = *sum-list qs'* **by** *simp*
also from *assms* ‹*qs'* ∈ *listset ss*› **have** ... ∈ *A* **by** (*rule direct-decompD*)
finally show *sum-list qs* ∈ *A* **.**
next

fix *a*
assume $a \in A$
with *assms* **obtain** qs' **where** qs': $qs' \in listset\ ss$ **and** *a*: $a = sum\text{-}list\ qs'$
by (*rule direct-decompE*)
define *qs* **where** $qs = 0\ \#\ qs'$
show $\exists!qs.\ qs \in listset\ (\{0\}\ \#\ ss) \wedge a = sum\text{-}list\ qs$
proof (*intro ex1I conjI*)
from - qs' *qs-def* **show** $qs \in listset\ (\{0\}\ \#\ ss)$ **by** (*rule listset-ConsI*) *simp*
next
fix *qs0*
assume $qs0 \in listset\ (\{0\}\ \#\ ss) \wedge a = sum\text{-}list\ qs0$
hence $qs0 \in listset\ (\{0\}\ \#\ ss)$ **and** *a0*: $a = sum\text{-}list\ qs0$ **by** *simp-all*
from *this*(*1*) **obtain** $q0\ qs0'$ **where** $q0 \in \{0\}$ **and** $qs0'$: $qs0' \in listset\ ss$
and *qs0*: $qs0 = q0\ \#\ qs0'$ **by** (*rule listset-ConsE*)
from *this*(*1*, *3*) **have** $sum\text{-}list\ qs0' = sum\text{-}list\ qs'$ **by** (*simp add*: *a0 flip*: *a*)
with *assms* $qs0'$ qs' **have** $qs0' = qs'$ **by** (*rule direct-decomp-unique*)
with ‹$q0 \in \{0\}$› **show** $qs0 = qs$ **by** (*simp add*: *qs-def qs0*)
qed (*simp add*: *qs-def a*)
qed

lemma *direct-decomp-Cons-zeroD*:
assumes *direct-decomp* A ($\{0\}\ \#\ ss$)
shows *direct-decomp* A *ss*
proof −
have *direct-decomp* $\{0\}$ [] **by** (*simp add*: *direct-decomp-def bij-betw-def*)
with *assms* **have** *direct-decomp* A (*ss* @ []) **by** (*rule direct-decomp-direct-decomp*)
thus *?thesis* **by** *simp*
qed

lemma *direct-decomp-Cons-subsetI*:
assumes *direct-decomp* A ($s\ \#\ ss$) **and** $\bigwedge s0.\ s0 \in set\ ss \Longrightarrow 0 \in s0$
shows $s \subseteq A$
proof
fix *x*
assume $x \in s$
moreover from *assms*(*2*) **have** $map\ (\lambda\text{-}.\ 0)\ ss \in listset\ ss$
by (*induct ss*, *auto simp del*: *listset.simps*(*2*) *intro*: *listset-ConsI*)
ultimately have $x\ \#\ (map\ (\lambda\text{-}.\ 0)\ ss) \in listset\ (s\ \#\ ss)$ **using** *refl* **by** (*rule listset-ConsI*)
with *assms*(*1*) **have** $sum\text{-}list\ (x\ \#\ (map\ (\lambda\text{-}.\ 0)\ ss)) \in A$ **by** (*rule direct-decompD*)
thus $x \in A$ **by** *simp*
qed

lemma *direct-decomp-Int-zero*:
assumes *direct-decomp* A *ss* **and** $i < j$ **and** $j < length\ ss$ **and** $\bigwedge s.\ s \in set\ ss \Longrightarrow 0 \in s$
shows $ss\ !\ i \cap ss\ !\ j = \{0\}$
proof −
from *assms*(*2*, *3*) **have** $i < length\ ss$ **by** (*rule less-trans*)

hence *i-in*: *ss* ! *i* $\in$ *set ss* **by** *simp*
from *assms*(*3*) **have** *j-in*: *ss* ! *j* $\in$ *set ss* **by** *simp*
show *?thesis*
proof
show *ss* ! *i* $\cap$ *ss* ! *j* $\subseteq$ {*0*}
proof
fix *x*
assume *x* $\in$ *ss* ! *i* $\cap$ *ss* ! *j*
hence *x-i*: *x* $\in$ *ss* ! *i* **and** *x-j*: *x* $\in$ *ss* ! *j* **by** *simp-all*
have *1*: (*map* (λ-. *0*) *ss*)[*k* := *y*] $\in$ *listset ss* **if** *k* < *length ss* **and** *y* $\in$ *ss* ! *k* **for** *k y*
using *assms*(*4*) *that*
proof (*induct ss arbitrary*: *k*)
case *Nil*
from *Nil*(*2*) **show** *?case* **by** *simp*
next
case (*Cons s ss*)
have $*$: $\bigwedge s'.\ s' \in$ *set ss* $\Longrightarrow$ *0* $\in s'$ **by** (*rule Cons.prems*) *simp*
show *?case*
proof (*cases k*)
case *k*: *0*
with *Cons.prems*(*3*) **have** *y* $\in$ *s* **by** *simp*
moreover from $*$ **have** *map* (λ-. *0*) *ss* $\in$ *listset ss*
by (*induct ss*) (*auto simp del*: *listset.simps*(*2*) *intro*: *listset-ConsI*)
moreover have (*map* (λ-. *0*) (*s* # *ss*))[*k* := *y*] = *y* # *map* (λ-. *0*) *ss* **by** (*simp add*: *k*)
ultimately show *?thesis* **by** (*rule listset-ConsI*)
next
case *k*: (*Suc* k')
have *0* $\in$ *s* **by** (*rule Cons.prems*) *simp*
moreover from $*$ **have** (*map* (λ-. *0*) *ss*)[k' := *y*] $\in$ *listset ss*
proof (*rule Cons.hyps*)
from *Cons.prems*(*2*) **show** k' < *length ss* **by** (*simp add*: *k*)
next
from *Cons.prems*(*3*) **show** *y* $\in$ *ss* ! k' **by** (*simp add*: *k*)
qed
moreover have (*map* (λ-. *0*) (*s* # *ss*))[*k* := *y*] = *0* # (*map* (λ-. *0*) *ss*)[k' := *y*]
by (*simp add*: *k*)
ultimately show *?thesis* **by** (*rule listset-ConsI*)
qed
qed
have *2*: *sum-list* ((*map* (λ-. *0*) *ss*)[*k* := *y*]) = *y* **if** *k* < *length ss* **for** *k* **and** *y*::'*a*
using *that* **by** (*induct ss arbitrary*: *k*) (*auto simp*: *add-ac split*: *nat.split*)
define *qs1* **where** *qs1* = (*map* (λ-. *0*) *ss*)[*i* := *x*]
define *qs2* **where** *qs2* = (*map* (λ-. *0*) *ss*)[*j* := *x*]
note *assms*(*1*)
moreover from ‹*i* < *length ss*› *x-i* **have** *qs1* $\in$ *listset ss* **unfolding** *qs1-def*

by (*rule 1*)
moreover from *assms*(*3*) *x-j* **have** *qs2* ∈ *listset ss* **unfolding** *qs2-def* **by** (*rule 1*)
thm *sum-list-update*
moreover from ‹*i* < *length ss*› *assms*(*3*) **have** *sum-list qs1* = *sum-list qs2*
by (*simp add*: *qs1-def qs2-def 2*)
ultimately have *qs1* = *qs2* **by** (*rule direct-decomp-unique*)
hence *qs1* ! *i* = *qs2* ! *i* **by** *simp*
with ‹*i* < *length ss*› *assms*(*2*, *3*) **show** *x* ∈ {*0*} **by** (*simp add*: *qs1-def qs2-def*)
qed
next
from *i-in* **have** *0* ∈ *ss* ! *i* **by** (*rule assms*(*4*))
moreover from *j-in* **have** *0* ∈ *ss* ! *j* **by** (*rule assms*(*4*))
ultimately show {*0*} ⊆ *ss* ! *i* ∩ *ss* ! *j* **by** *simp*
qed
qed

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 (*rule pairwiseI*)
fix *s1 s2*
assume *s1* ∈ *set ss*
then obtain *i* **where** *i* < *length ss* **and** *s1*: *s1* = *ss* ! *i* **by** (*metis in-set-conv-nth*)
assume *s2* ∈ *set ss*
then obtain *j* **where** *j* < *length ss* **and** *s2*: *s2* = *ss* ! *j* **by** (*metis in-set-conv-nth*)
assume *s1* ≠ *s2*
hence *i* < *j* ∨ *j* < *i* **by** (*auto simp*: *s1 s2*)
thus *s1* ∩ *s2* = {*0*}
proof
assume *i* < *j*
with *assms*(*1*) **show** *?thesis* **unfolding** *s1 s2* **using** ‹*j* < *length ss*› *assms*(*2*)
by (*rule direct-decomp-Int-zero*)
next
assume *j* < *i*
with *assms*(*1*) **have** *s2* ∩ *s1* = {*0*} **unfolding** *s1 s2* **using** ‹*i* < *length ss*› *assms*(*2*)
by (*rule direct-decomp-Int-zero*)
thus *?thesis* **by** (*simp only*: *Int-commute*)
qed
qed

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 −
from *assms*(*2*) **obtain** *i j* **where** *i* < *j* **and** *j* < *length ss* **and** *1*: *ss* ! *i* = *X* **and** *2*: *ss* ! *j* = *X*

by (*rule count-list-gr-1-E*)
from *assms*(*1*) *this*(*1*, *2*) *assms*(*3*) **have** *ss* ! *i* ∩ *ss* ! *j* = {*0*} **by** (*rule direct-decomp-Int-zero*)
thus *?thesis* **by** (*simp add*: *1 2*)
qed

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 −
from *assms*(*2*) **obtain** *i* **where** *i* < *length ss* **and** *s1*: *s1* = *ss* ! *i* **by** (*metis in-set-conv-nth*)
from *this*(*1*) **have** *i*: *i* < *length* (*map f ss*) **by** *simp*
from *assms*(*3*) **obtain** *j* **where** *j* < *length ss* **and** *s2*: *s2* = *ss* ! *j* **by** (*metis in-set-conv-nth*)
from *this*(*1*) **have** *j*: *j* < *length* (*map f ss*) **by** *simp*
have ∗: *0* ∈ *s* **if** *s* ∈ *set* (*map f ss*) **for** *s*
proof −
from *that* **obtain** *s′* **where** *s′* ∈ *set ss* **and** *s*: *s* = *f s′* **unfolding** *set-map* **..**
from *this*(*1*) **show** *0* ∈ *s* **unfolding** *s* **by** (*rule assms*(*5*))
qed
show *?thesis*
proof (*rule linorder-cases*)
assume *i* < *j*
with *assms*(*1*) **have** (*map f ss*) ! *i* ∩ (*map f ss*) ! *j* = {*0*}
using *j* ∗ **by** (*rule direct-decomp-Int-zero*)
with *i j* **show** *?thesis* **by** (*simp add*: *s1 s2*)
next
assume *j* < *i*
with *assms*(*1*) **have** (*map f ss*) ! *j* ∩ (*map f ss*) ! *i* = {*0*}
using *i* ∗ **by** (*rule direct-decomp-Int-zero*)
with *i j* **show** *?thesis* **by** (*simp add*: *s1 s2 Int-commute*)
next
assume *i* = *j*
with *assms*(*4*) **show** *?thesis* **by** (*simp add*: *s1 s2*)
qed
qed

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*
  using *dim-span-eq-card-independent independent-empty* **by** *fastforce*

lemma *dim-zero* [*simp*]: *dim* {*0*} = *0*
  using *dim-span-eq-card-independent independent-empty* **by** *fastforce*

lemma *independent-UnI*:
  assumes *independent A* **and** *independent B* **and** *span A* ∩ *span B* = {*0*}
  shows *independent* (*A* ∪ *B*)
proof
  from *span-superset* **have** *A* ∩ *B* ⊆ *span A* ∩ *span B* **by** *blast*
  hence *A* ∩ *B* = {} **unfolding** *assms*(*3*) **using** *assms*(*1*, *2*) *dependent-zero* **by** *blast*
  assume *dependent* (*A* ∪ *B*)
  then obtain *T u v* **where** *finite T* **and** *T* ⊆ *A* ∪ *B* **and** *eq*: (∑*v*∈*T*. *u v* ∗*s v*) = *0*
    and *v* ∈ *T* **and** *u v* ≠ *0* **unfolding** *dependent-explicit* **by** *blast*
  define *TA* **where** *TA* = *T* ∩ *A*
  define *TB* **where** *TB* = *T* ∩ *B*
  from ‹*T* ⊆ *A* ∪ *B*› **have** *T*: *T* = *TA* ∪ *TB* **by** (*auto simp*: *TA-def TB-def*)
  from ‹*finite T*› **have** *finite TA* **and** *TA* ⊆ *A* **by** (*simp-all add*: *TA-def*)
  from ‹*finite T*› **have** *finite TB* **and** *TB* ⊆ *B* **by** (*simp-all add*: *TB-def*)
  from ‹*A* ∩ *B* = {}› ‹*TA* ⊆ *A*› *this*(*2*) **have** *TA* ∩ *TB* = {} **by** *blast*
  have *0* = (∑*v*∈*TA* ∪ *TB*. *u v* ∗*s v*) **by** (*simp only*: *eq flip*: *T*)
  also have . . . = (∑*v*∈*TA*. *u v* ∗*s v*) + (∑*v*∈*TB*. *u v* ∗*s v*) **by** (*rule sum.union-disjoint*) *fact*+
  finally have (∑*v*∈*TA*. *u v* ∗*s v*) = (∑*v*∈*TB*. (− *u*) *v* ∗*s v*) (**is** *?x* = *?y*)
    by (*simp add*: *sum-negf eq-neg-iff-add-eq-0*)
  from ‹*finite TB*› ‹*TB* ⊆ *B*› **have** *?y* ∈ *span B* **by** (*auto simp*: *span-explicit simp del*: *uminus-apply*)
  moreover from ‹*finite TA*› ‹*TA* ⊆ *A*› **have** *?x* ∈ *span A* **by** (*auto simp*: *span-explicit*)
  ultimately have *?y* ∈ *span A* ∩ *span B* **by** (*simp add*: ‹*?x* = *?y*›)
  hence *?x* = *0* **and** *?y* = *0* **by** (*simp-all add*: ‹*?x* = *?y*› *assms*(*3*))
  from ‹*v* ∈ *T*› **have** *v* ∈ *TA* ∪ *TB* **by** (*simp only*: *T*)
  hence *u v* = *0*
  proof
    assume *v* ∈ *TA*
      with *assms*(*1*) ‹*finite TA*› ‹*TA* ⊆ *A*› ‹*?x* = *0*› **show** *u v* = *0* **by** (*rule independentD*)
  next
    assume *v* ∈ *TB*
    with *assms*(*2*) ‹*finite TB*› ‹*TB* ⊆ *B*› ‹*?y* = *0*› **have** (− *u*) *v* = *0* **by** (*rule independentD*)

thus *u v* = *0* **by** *simp*
qed
with ‹*u v* ≠ *0*› **show** *False* **..**
qed

lemma *subspace-direct-decomp*:
assumes *direct-decomp A ss* **and** $\bigwedge$*s*. *s* ∈ *set ss* ⟹ *subspace s*
shows *subspace A*
proof (*rule subspaceI*)
let *?qs* = *map* (λ-. *0*) *ss*
from *assms*(*2*) **have** *?qs* ∈ *listset ss*
by (*induct ss*) (*auto simp del*: *listset.simps*(*2*) *dest*: *subspace-0 intro*: *listset-ConsI*)
with *assms*(*1*) **have** *sum-list ?qs* ∈ *A* **by** (*rule direct-decompD*)
thus *0* ∈ *A* **by** *simp*
next
fix *p q*
assume *p* ∈ *A*
with *assms*(*1*) **obtain** *ps* **where** *ps*: *ps* ∈ *listset ss* **and** *p*: *p* = *sum-list ps* **by** (*rule direct-decompE*)
assume *q* ∈ *A*
with *assms*(*1*) **obtain** *qs* **where** *qs*: *qs* ∈ *listset ss* **and** *q*: *q* = *sum-list qs* **by** (*rule direct-decompE*)
from *ps qs* **have** *l*: *length ps* = *length qs* **by** (*simp only*: *listsetD*)
from *ps qs* **have** *map2* (+) *ps qs* ∈ *listset ss* (**is** *?qs* ∈ -)
by (*rule listset-closed-map2*) (*auto dest*: *assms*(*2*) *subspace-add*)
with *assms*(*1*) **have** *sum-list ?qs* ∈ *A* **by** (*rule direct-decompD*)
thus *p* + *q* ∈ *A* **using** *l* **by** (*simp only*: *p q sum-list-map2-plus*)
next
fix *c p*
assume *p* ∈ *A*
with *assms*(*1*) **obtain** *ps* **where** *ps* ∈ *listset ss* **and** *p*: *p* = *sum-list ps* **by** (*rule direct-decompE*)
from *this*(*1*) **have** *map* ((∗*s*) *c*) *ps* ∈ *listset ss* (**is** *?qs* ∈ -)
by (*rule listset-closed-map*) (*auto dest*: *assms*(*2*) *subspace-scale*)
with *assms*(*1*) **have** *sum-list ?qs* ∈ *A* **by** (*rule direct-decompD*)
also have *sum-list ?qs* = *c* ∗*s sum-list ps* **by** (*induct ps*) (*simp-all add*: *scale-right-distrib*)
finally show *c* ∗*s p* ∈ *A* **by** (*simp only*: *p*)
qed

lemma *is-basis-alt*: *subspace V* ⟹ *is-basis V B* ⟷ (*independent B* ∧ *span B* = *V*)
by (*metis* (*full-types*) *is-basis-def dim-eq-card span-eq span-eq-iff*)

lemma *is-basis-finite*: *is-basis V A* ⟹ *is-basis V B* ⟹ *finite A* ⟷ *finite B*
unfolding *is-basis-def* **using** *independent-span-bound* **by** *auto*

lemma *some-basis-is-basis*: *is-basis V* (*some-basis V*)
proof −

obtain B **where** $B \subseteq V$ **and** *independent* B **and** $V \subseteq$ *span* B **and** *card* B = *dim* V
  by (*rule basis-exists*)
 hence *is-basis* V B **by** (*simp add*: *is-basis-def*)
 thus *?thesis* **unfolding** *some-basis-def* **by** (*rule someI*)
qed

corollary
 shows *some-basis-subset*: *some-basis* $V \subseteq V$
  and *independent-some-basis*: *independent* (*some-basis* V)
  and *span-some-basis-supset*: $V \subseteq$ *span* (*some-basis* V)
  and *card-some-basis*: *card* (*some-basis* V) = *dim* V
 using *some-basis-is-basis*[*of* V] **by** (*simp-all add*: *is-basis-def*)

lemma *some-basis-not-zero*: $0 \notin$ *some-basis* V
 using *independent-some-basis dependent-zero* **by** *blast*

lemma *span-some-basis*: *subspace* $V \Longrightarrow$ *span* (*some-basis* V) = V
 by (*simp add*: *span-subspace some-basis-subset span-some-basis-supset*)

lemma *direct-decomp-some-basis-pairwise-disjnt*:
 assumes *direct-decomp* A *ss* **and** $\bigwedge s.\ s \in$ *set ss* $\Longrightarrow$ *subspace* s
 shows *pairwise* ($\lambda s1\ s2.$ *disjnt* (*some-basis s1*) (*some-basis s2*)) (*set ss*)
proof (*rule pairwiseI*)
 fix *s1 s2*
 assume *s1* $\in$ *set ss* **and** *s2* $\in$ *set ss* **and** *s1* $\neq$ *s2*
 have *some-basis s1* $\cap$ *some-basis s2* $\subseteq$ *s1* $\cap$ *s2* **using** *some-basis-subset* **by** *blast*
 also from *direct-decomp-pairwise-zero* **have** ... = $\{0\}$
 proof (*rule pairwiseD*)
  fix s
  assume $s \in$ *set ss*
  hence *subspace* s **by** (*rule assms*(*2*))
  thus $0 \in s$ **by** (*rule subspace-0*)
 qed *fact*+
 finally have *some-basis s1* $\cap$ *some-basis s2* $\subseteq \{0\}$ **.**
 with *some-basis-not-zero* **show** *disjnt* (*some-basis s1*) (*some-basis s2*)
  unfolding *disjnt-def* **by** *blast*
qed

lemma *direct-decomp-span-some-basis*:
 assumes *direct-decomp* A *ss* **and** $\bigwedge s.\ s \in$ *set ss* $\Longrightarrow$ *subspace* s
 shows *span* ($\bigcup$(*some-basis* ‘ *set ss*)) = A
proof −
 from *assms*(*1*) **have** *eq0*[*symmetric*]: *sum-list* ‘ *listset ss* = A **by** (*rule direct-decompD*)
 show *?thesis* **unfolding** *eq0* **using** *assms*(*2*)
 proof (*induct ss*)
  case *Nil*
  show *?case* **by** *simp*

next
case (*Cons s ss*)
have *subspace s* **by** (*rule Cons.prems*) *simp*
hence *eq1*: *span* (*some-basis s*) = *s* **by** (*rule span-some-basis*)
have $\bigwedge s'$. $s' \in$ *set ss* $\Longrightarrow$ *subspace* s' **by** (*rule Cons.prems*) *simp*
hence *eq2*: *span* ($\bigcup$ (*some-basis* ‘ *set ss*)) = *sum-list* ‘ *listset ss* **by** (*rule Cons.hyps*)
have *span* ($\bigcup$ (*some-basis* ‘ *set* (*s* # *ss*))) = $\{x + y \mid x\ y.\ x \in s \wedge y \in$ *sum-list* ‘ *listset ss*$\}$
by (*simp add*: *span-Un eq1 eq2*)
also have ... = *sum-list* ‘ *listset* (*s* # *ss*) (**is** *?A* = *?B*)
proof
show *?A* $\subseteq$ *?B*
proof
fix *a*
assume *a* $\in$ *?A*
then obtain *x y* **where** *x* $\in$ *s* **and** *y* $\in$ *sum-list* ‘ *listset ss* **and** *a*: *a* = *x* + *y* **by** *blast*
from *this*(*2*) **obtain** *qs* **where** *qs* $\in$ *listset ss* **and** *y*: *y* = *sum-list qs* **..**
from ‹*x* $\in$ *s*› *this*(*1*) *refl* **have** *x* # *qs* $\in$ *listset* (*s* # *ss*) **by** (*rule listset-ConsI*)
hence *sum-list* (*x* # *qs*) $\in$ *?B* **by** (*rule imageI*)
also have *sum-list* (*x* # *qs*) = *a* **by** (*simp add*: *a y*)
finally show *a* $\in$ *?B* **.**
qed
next
show *?B* $\subseteq$ *?A*
proof
fix *a*
assume *a* $\in$ *?B*
then obtain qs' **where** $qs' \in$ *listset* (*s* # *ss*) **and** *a*: *a* = *sum-list* qs' **..**
from *this*(*1*) **obtain** *x qs* **where** *x* $\in$ *s* **and** *qs* $\in$ *listset ss* **and** qs': qs' = *x* # *qs*
by (*rule listset-ConsE*)
from *this*(*2*) **have** *sum-list qs* $\in$ *sum-list* ‘ *listset ss* **by** (*rule imageI*)
moreover have *a* = *x* + *sum-list qs* **by** (*simp add*: *a* qs')
ultimately show *a* $\in$ *?A* **using** ‹*x* $\in$ *s*› **by** *blast*
qed
qed
finally show *?case* **.**
qed
qed

lemma *direct-decomp-independent-some-basis*:
assumes *direct-decomp A ss* **and** $\bigwedge s$. *s* $\in$ *set ss* $\Longrightarrow$ *subspace s*
shows *independent* ($\bigcup$(*some-basis* ‘ *set ss*))
using *assms*
proof (*induct ss arbitrary*: *A*)
case *Nil*
from *independent-empty* **show** *?case* **by** *simp*

next
  case (*Cons s ss*)
  have *1*: $\bigwedge s'$. $s' \in$ *set ss* $\Longrightarrow$ *subspace* s' **by** (*rule Cons.prems*) *simp*
  have *subspace s* **by** (*rule Cons.prems*) *simp*
  hence $0 \in s$ **and** *eq1*: *span* (*some-basis s*) = *s* **by** (*rule subspace-0*, *rule span-some-basis*)
  from *Cons.prems*(*1*) **have** $*$: *direct-decomp A* ([*s*] @ *ss*) **by** *simp*
  moreover from ‹$0 \in s$› **have** {} $\notin$ *set* [*s*] **by** *auto*
  ultimately have *2*: *direct-decomp* (*sum-list* ‘ *listset ss*) *ss* **by** (*rule direct-decomp-appendD*)
  hence *eq2*: *span* ($\bigcup$ (*some-basis* ‘ *set ss*)) = *sum-list* ‘ *listset ss* **using** *1*
    by (*rule direct-decomp-span-some-basis*)

  note *independent-some-basis*[*of s*]
  moreover from *2 1* **have** *independent* ($\bigcup$ (*some-basis* ‘ *set ss*)) **by** (*rule Cons.hyps*)
  moreover have *span* (*some-basis s*) $\cap$ *span* ($\bigcup$ (*some-basis* ‘ *set ss*)) = {*0*}
  proof −
    from $*$ **have** *direct-decomp A* [*sum-list* ‘ *listset* [*s*], *sum-list* ‘ *listset ss*]
      by (*rule direct-decomp-appendD*)
    hence *direct-decomp A* [*s*, *sum-list* ‘ *listset ss*] **by** (*simp add*: *image-image*)
    moreover have $0 < (1{::}nat)$ **by** *simp*
    moreover have *1* < *length* [*s*, *sum-list* ‘ *listset ss*] **by** *simp*
    ultimately have [*s*, *sum-list* ‘ *listset ss*] ! *0* $\cap$ [*s*, *sum-list* ‘ *listset ss*] ! *1* = {*0*}
      by (*rule direct-decomp-Int-zero*) (*auto simp*: ‹$0 \in s$› *eq2*[*symmetric*] *span-zero*)
    thus *?thesis* **by** (*simp add*: *eq1 eq2*)
  qed
  ultimately have *independent* (*some-basis s* $\cup$ ($\bigcup$ (*some-basis* ‘ *set ss*)))
    by (*rule independent-UnI*)
  thus *?case* **by** *simp*
qed

corollary *direct-decomp-is-basis*:
  assumes *direct-decomp A ss* **and** $\bigwedge s$. $s \in$ *set ss* $\Longrightarrow$ *subspace s*
  shows *is-basis A* ($\bigcup$(*some-basis* ‘ *set ss*))
proof −
  from *assms* **have** *subspace A* **by** (*rule subspace-direct-decomp*)
  moreover from *assms* **have** *span* ($\bigcup$(*some-basis* ‘ *set ss*)) = *A*
    by (*rule direct-decomp-span-some-basis*)
  moreover from *assms* **have** *independent* ($\bigcup$(*some-basis* ‘ *set ss*))
    by (*rule direct-decomp-independent-some-basis*)
  ultimately show *?thesis* **by** (*simp add*: *is-basis-alt*)
qed

lemma *dim-direct-decomp*:
  assumes *direct-decomp A ss* **and** *finite B* **and** $A \subseteq$ *span B* **and** $\bigwedge s$. $s \in$ *set ss* $\Longrightarrow$ *subspace s*
  shows *dim A* = ($\sum$ *s*$\in$*set ss*. *dim s*)
proof −
  from *assms*(*1*, *4*) **have** *is-basis A* ($\bigcup$(*some-basis* ‘ *set ss*))

(**is** *is-basis A ?B*) **by** (*rule direct-decomp-is-basis*)
hence *dim A* = *card ?B* **and** *independent ?B* **and** *?B* ⊆ *A* **by** (*simp-all add*: *is-basis-def*)
from *this*(*3*) *assms*(*3*) **have** *?B* ⊆ *span B* **by** (*rule subset-trans*)
with *assms*(*2*) ‹*independent ?B*› **have** *finite ?B* **using** *independent-span-bound* **by** *blast*
note ‹*dim A* = *card ?B*›
also from *finite-set* **have** *card ?B* = (∑ *s*∈*set ss*. *card* (*some-basis s*))
proof (*intro card-UN-disjoint ballI impI*)
fix *s*
assume *s* ∈ *set ss*
with ‹*finite ?B*› **show** *finite* (*some-basis s*) **by** *auto*
next
fix *s1 s2*
have *pairwise* (λ*s t*. *disjnt* (*some-basis s*) (*some-basis t*)) (*set ss*)
using *assms*(*1*, *4*) **by** (*rule direct-decomp-some-basis-pairwise-disjnt*)
moreover assume *s1* ∈ *set ss* **and** *s2* ∈ *set ss* **and** *s1* ≠ *s2*
thm *pairwiseD*
ultimately have *disjnt* (*some-basis s1*) (*some-basis s2*) **by** (*rule pairwiseD*)
thus *some-basis s1* ∩ *some-basis s2* = {} **by** (*simp only*: *disjnt-def*)
qed
also from *refl card-some-basis* **have** ... = (∑ *s*∈*set ss*. *dim s*) **by** (*rule sum.cong*)
finally show *?thesis* **.**
qed

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 (*rule homogeneous-setI*)
fix *a n*
assume *a* ∈ *A*
with *assms*(*1*) **obtain** *qs* **where** *qs* ∈ *listset ss* **and** *a*: *a* = *sum-list qs* **by** (*rule direct-decompE*)
have *hom-component a n* = *hom-component* (*sum-list qs*) *n* **by** (*simp only*: *a*)
also have ... = *sum-list* (*map* (λ*q*. *hom-component q n*) *qs*)
by (*induct qs*) (*simp-all add*: *hom-component-plus*)
also from *assms*(*1*) **have** ... ∈ *A*
proof (*rule direct-decompD*)
show *map* (λ*q*. *hom-component q n*) *qs* ∈ *listset ss*
proof (*rule listset-closed-map*)
fix *s q*
assume *s* ∈ *set ss*
hence *homogeneous-set s* **by** (*rule assms*(*2*))
moreover assume *q* ∈ *s*
ultimately show *hom-component q n* ∈ *s* **by** (*rule homogeneous-setD*)

qed *fact*
qed
finally show *hom-component a n* ∈ *A* **.**
qed

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 −
from *assms* **obtain** *a* **where** *a* ∈ *A* **and** *p*: *p* = *hom-component a z* **unfolding** *hom-deg-set-def* **..**
show ∗: *homogeneous p* **by** (*simp only*: *p homogeneous-hom-component*)

assume *p* ≠ *0*
hence *keys p* ≠ {} **by** *simp*
then obtain *t* **where** *t* ∈ *keys p* **by** *blast*
with ∗ **have** *deg-pm t* = *poly-deg p* **by** (*rule homogeneousD-poly-deg*)
moreover from ‹*t* ∈ *keys p*› **have** *deg-pm t* = *z* **unfolding** *p* **by** (*rule keys-hom-componentD*)
ultimately show *poly-deg p* = *z* **by** *simp*
qed

lemma *zero-in-hom-deg-set*:
assumes *0* ∈ *A*
shows *0* ∈ *hom-deg-set z A*
proof −
have *0* = *hom-component 0 z* **by** *simp*
also from *assms* **have** ... ∈ *hom-deg-set z A* **unfolding** *hom-deg-set-def* **by** (*rule imageI*)
finally show *?thesis* **.**
qed

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 −
from *assms*(*2*) **obtain** *a* **where** *a* ∈ *A* **and** *p*: *p* = *hom-component a z* **unfolding** *hom-deg-set-def* **..**
from *this*(*1*) **have** − *a* ∈ *A* **by** (*rule assms*(*1*))
moreover have − *p* = *hom-component* (− *a*) *z* **by** (*simp add*: *p*)
ultimately show *?thesis* **unfolding** *hom-deg-set-def* **by** (*rule rev-image-eqI*)
qed

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 −
from *assms*(*2*) **obtain** *a1* **where** *a1* ∈ *A* **and** *p*: *p* = *hom-component a1 z* **unfolding** *hom-deg-set-def* **..**
from *assms*(*3*) **obtain** *a2* **where** *a2* ∈ *A* **and** *q*: *q* = *hom-component a2 z* **unfolding** *hom-deg-set-def* **..**
from ‹*a1* ∈ *A*› *this*(*1*) **have** *a1* + *a2* ∈ *A* **by** (*rule assms*(*1*))
moreover have *p* + *q* = *hom-component* (*a1* + *a2*) *z* **by** (*simp only*: *p q hom-component-plus*)
ultimately show *?thesis* **unfolding** *hom-deg-set-def* **by** (*rule rev-image-eqI*)
qed

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 −
from *assms*(*2*) **obtain** *a1* **where** *a1* ∈ *A* **and** *p*: *p* = *hom-component a1 z* **unfolding** *hom-deg-set-def* **..**
from *assms*(*3*) **obtain** *a2* **where** *a2* ∈ *A* **and** *q*: *q* = *hom-component a2 z* **unfolding** *hom-deg-set-def* **..**
from ‹*a1* ∈ *A*› *this*(*1*) **have** *a1* − *a2* ∈ *A* **by** (*rule assms*(*1*))
moreover have *p* − *q* = *hom-component* (*a1* − *a2*) *z* **by** (*simp only*: *p q hom-component-minus*)
ultimately show *?thesis* **unfolding** *hom-deg-set-def* **by** (*rule rev-image-eqI*)
qed

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 −
from *assms*(*2*) **obtain** *a* **where** *a* ∈ *A* **and** *p*: *p* = *hom-component a z* **unfolding** *hom-deg-set-def* **..**
from *this*(*1*) **have** *c* · *a* ∈ *A* **by** (*rule assms*(*1*))
moreover have *c* · *p* = *hom-component* (*c* · *a*) *z*
by (*simp add*: *p punit.map-scale-eq-monom-mult hom-component-monom-mult*)
ultimately show *?thesis* **unfolding** *hom-deg-set-def* **by** (*rule rev-image-eqI*)
qed

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*
using *assms*(*3*)
proof (*induct I rule*: *infinite-finite-induct*)
case (*infinite I*)
with *assms*(*1*) **show** *?case* **by** (*simp add*: *zero-in-hom-deg-set*)
next
case *empty*

with *assms*(*1*) **show** *?case* **by** (*simp add*: *zero-in-hom-deg-set*)
next
case (*insert j I*)
from *insert.hyps*(*1*, *2*) **have** *sum f* (*insert j I*) = *f j* + *sum f I* **by** *simp*
also from *assms*(*2*) **have** ... ∈ *hom-deg-set z A*
proof (*intro hom-deg-set-closed-plus insert.hyps*)
show *f j* ∈ *hom-deg-set z A* **by** (*rule insert.prems*) *simp*
next
fix *i*
assume *i* ∈ *I*
hence *i* ∈ *insert j I* **by** *simp*
thus *f i* ∈ *hom-deg-set z A* **by** (*rule insert.prems*)
qed
finally show *?case* **.**
qed

lemma *hom-deg-set-subset*: *homogeneous-set A* ⟹ *hom-deg-set z A* ⊆ *A*
by (*auto dest*: *homogeneous-setD simp*: *hom-deg-set-def*)

lemma *Polys-closed-hom-deg-set*:
assumes *A* ⊆ *P*[*X*]
shows *hom-deg-set z A* ⊆ *P*[*X*]
proof
fix *p*
assume *p* ∈ *hom-deg-set z A*
then obtain *p'* **where** *p'* ∈ *A* **and** *p*: *p* = *hom-component p' z* **unfolding** *hom-deg-set-def* **..**
from *this*(*1*) *assms* **have** *p'* ∈ *P*[*X*] **..**
have *keys p* ⊆ *keys p'* **by** (*simp add*: *p keys-hom-component*)
also from ‹*p'* ∈ *P*[*X*]› **have** ... ⊆ .[*X*] **by** (*rule PolysD*)
finally show *p* ∈ *P*[*X*] **by** (*rule PolysI*)
qed

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
show *?A* ⊆ *?B*
proof
fix *h*
assume *h* ∈ *?A*
also from *assms* **have** ... ⊆ *A* **by** (*rule hom-deg-set-subset*)
finally show *h* ∈ *?B* **using** ‹*h* ∈ *?A*› **by** (*auto dest*: *hom-deg-setD*)
qed
next
show *?B* ⊆ *?A*
proof
fix *h*

assume $h \in ?B$
hence $h \in A$ **and** *homogeneous* h **and** $h = 0 \vee$ *poly-deg* $h = z$ **by** *simp-all*
from *this*(*3*) **show** $h \in ?A$
proof
assume $h = 0$
with ‹$h \in A$› **have** $0 \in A$ **by** *simp*
thus *?thesis* **unfolding** ‹$h = 0$› **by** (*rule zero-in-hom-deg-set*)
next
assume *poly-deg* $h = z$
with ‹*homogeneous* h› **have** $h =$ *hom-component* h z **by** (*simp add*: *hom-component-of-homogeneous*)
with ‹$h \in A$› **show** *?thesis* **unfolding** *hom-deg-set-def* **by** (*rule rev-image-eqI*)
qed
qed
qed

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
show $?A \subseteq ?B$
proof
fix h
assume $h \in ?A$
then obtain a **where** $a \in A$ **and** h: $h =$ *hom-component a z* **unfolding** *hom-deg-set-def* **..**
from *this*(*1*) **obtain** *qs* **where** *qs* ∈ *listset ss* **and** a: $a =$ *sum-list qs* **unfolding** *assms* **..**
have $h =$ *hom-component* (*sum-list qs*) z **by** (*simp only*: *a h*)
also have ... = *sum-list* (*map* (λq. *hom-component q z*) *qs*)
by (*induct qs*) (*simp-all add*: *hom-component-plus*)
also have ... $\in ?B$
proof (*rule imageI*)
show *map* (λq. *hom-component q z*) *qs* ∈ *listset* (*map* (*hom-deg-set z*) *ss*)
unfolding *hom-deg-set-def* **using** ‹*qs* ∈ *listset ss*› *refl* **by** (*rule listset-map-imageI*)
qed
finally show $h \in ?B$ **.**
qed
next
show $?B \subseteq ?A$
proof
fix h
assume $h \in ?B$
then obtain *qs* **where** *qs* ∈ *listset* (*map* (*hom-deg-set z*) *ss*) **and** h: $h =$ *sum-list qs* **..**
from *this*(*1*) **obtain** *qs'* **where** *qs'* ∈ *listset ss* **and** *qs*: *qs* = *map* (λq. *hom-component q z*) *qs'*
unfolding *hom-deg-set-def* **by** (*rule listset-map-imageE*)

have *h* = *sum-list* (*map* (λ*q*. *hom-component q z*) *qs′*) **by** (*simp only*: *h qs*)
also have ... = *hom-component* (*sum-list qs′*) *z* **by** (*induct qs′*) (*simp-all add*: *hom-component-plus*)
finally have *h* = *hom-component* (*sum-list qs′*) *z* **.**
moreover have *sum-list qs′* ∈ *A* **unfolding** *assms* **using** ‹*qs′* ∈ *listset ss*› **by** (*rule imageI*)
ultimately show *h* ∈ *?A* **unfolding** *hom-deg-set-def* **by** (*rule image-eqI*)
qed
qed

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 (*rule direct-decompI*)
from *assms*(*1*) **have** *sum-list* ‘ *listset ss* = *A* **by** (*rule direct-decompD*)
from *this*[*symmetric*] **show** *sum-list* ‘ *listset* (*map* (*hom-deg-set z*) *ss*) = *hom-deg-set z A*
by (*simp only*: *hom-deg-set-sum-list-listset*)
next
from *assms*(*1*) **have** *inj-on sum-list* (*listset ss*) **by** (*rule direct-decompD*)
moreover have *listset* (*map* (*hom-deg-set z*) *ss*) ⊆ *listset ss*
proof (*rule listset-mono*)
fix *i*
assume *i* < *length ss*
hence *map* (*hom-deg-set z*) *ss* ! *i* = *hom-deg-set z* (*ss* ! *i*) **by** *simp*
also from ‹*i* < *length ss*› **have** ... ⊆ *ss* ! *i* **by** (*intro hom-deg-set-subset assms*(*2*) *nth-mem*)
finally show *map* (*hom-deg-set z*) *ss* ! *i* ⊆ *ss* ! *i* **.**
qed *simp*
ultimately show *inj-on sum-list* (*listset* (*map* (*hom-deg-set z*) *ss*)) **by** (*rule inj-on-subset*)
qed

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*))
using *ideal.span-zero ideal.span-add*
proof (*rule phull.subspaceI*)
fix *c p*
assume *p* ∈ *ideal F*
thus *c* · *p* ∈ *ideal F* **unfolding** *map-scale-eq-times* **by** (*rule ideal.span-scale*)
qed

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

using *zero-in-Polys Polys-closed-plus Polys-closed-map-scale* **by** (*rule phull.subspaceI*)

lemma *subspace-hom-deg-set*:
assumes *phull.subspace A*
shows *phull.subspace* (*hom-deg-set z A*) (**is** *phull.subspace ?A*)
proof (*rule phull.subspaceI*)
from *assms* **have** $0 \in A$ **by** (*rule phull.subspace-0*)
thus $0 \in ?A$ **by** (*rule zero-in-hom-deg-set*)
next
fix *p q*
assume $p \in ?A$ **and** $q \in ?A$
with *phull.subspace-add* **show** $p + q \in ?A$ **by** (*rule hom-deg-set-closed-plus*) (*rule assms*)
next
fix *c p*
assume $p \in ?A$
with *phull.subspace-scale* **show** $c \cdot p \in ?A$ **by** (*rule hom-deg-set-closed-scalar*) (*rule assms*)
qed

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
show $?A \subseteq ?B$
proof
fix *p*
assume $p \in ?A$
also from *this* **have** ... = $\{p \in P[X].$ *homogeneous p* $\wedge$ ($p = 0 \vee$ *poly-deg p* $= z)\}$
by (*simp only*: *hom-deg-set-alt-homogeneous-set*[*OF homogeneous-set-Polys*])
finally have $p \in P[X]$ **and** *homogeneous p* **and** $p \neq 0 \Longrightarrow$ *poly-deg p* $= z$ **by** *simp-all*
thus $p \in ?B$
proof (*induct p rule*: *poly-mapping-plus-induct*)
case *1*
from *phull.span-zero* **show** *?case* **.**
next
case (*2 p c t*)
let *?m* = *monomial c t*
from *2*(*1*) **have** $t \in$ *keys ?m* **by** *simp*
hence $t \in$ *keys* (*?m* + *p*) **using** *2*(*2*) **by** (*rule in-keys-plusI1*)
hence *?m* + *p* $\neq 0$ **by** *auto*
hence *poly-deg* (*monomial c t* + *p*) = *z* **by** (*rule 2*)
from *2*(*4*) **have** *keys* (*?m* + *p*) $\subseteq$.[*X*] **by** (*rule PolysD*)
with ‹$t \in$ *keys* (*?m* + *p*)› **have** $t \in$.[*X*] **..**
hence *?m* $\in$ *P*[*X*] **by** (*rule Polys-closed-monomial*)
have $t \in$ *deg-sect X z*
proof (*rule deg-sectI*)

```
        from 2(5) ‹t ∈ keys (?m + p)› have deg-pm t = poly-deg (?m + p)
          by (rule homogeneousD-poly-deg)
        also have ... = z by fact
        finally show deg-pm t = z .
      qed fact
      hence monomial 1 t ∈ monomial 1 ` deg-sect X z by (rule imageI)
      hence monomial 1 t ∈ ?B by (rule phull.span-base)
      hence c · monomial 1 t ∈ ?B by (rule phull.span-scale)
      hence ?m ∈ ?B by simp
      moreover have p ∈ ?B
      proof (rule 2)
      from 2(4) ‹?m ∈ P[X]› have (?m + p) − ?m ∈ P[X] by (rule Polys-closed-minus)
        thus p ∈ P[X] by simp
      next
        have 1: deg-pm s = z if s ∈ keys p for s
        proof −
          from that 2(2) have s ≠ t by blast
          hence s ∉ keys ?m by simp
          with that have s ∈ keys (?m + p) by (rule in-keys-plusI2)
            with 2(5) have deg-pm s = poly-deg (?m + p) by (rule homoge-
neousD-poly-deg)
          also have ... = z by fact
          finally show ?thesis .
        qed
        show homogeneous p by (rule homogeneousI) (simp add: 1)

        assume p ≠ 0
        show poly-deg p = z
        proof (rule antisym)
          show poly-deg p ≤ z by (rule poly-deg-leI) (simp add: 1)
        next
          from ‹p ≠ 0› have keys p ≠ {} by simp
          then obtain s where s ∈ keys p by blast
          hence z = deg-pm s by (simp only: 1)
          also from ‹s ∈ keys p› have ... ≤ poly-deg p by (rule poly-deg-max-keys)
          finally show z ≤ poly-deg p .
        qed
      qed
      ultimately show ?case by (rule phull.span-add)
    qed
  qed
next
  show ?B ⊆ ?A
  proof
    fix p
    assume p ∈ ?B
    then obtain M u where M ⊆ monomial 1 ` deg-sect X z and finite M and
p: p = (∑ m∈M. u m · m)
      by (auto simp: phull.span-explicit)
```

from *this*(*1*) **obtain** *T* **where** *T* ⊆ *deg-sect X z* **and** *M*: *M* = *monomial 1* ‘
T
and *inj*: *inj-on* (*monomial* (*1*::′*a*)) *T* **by** (*rule subset-imageE-inj*)
define *c* **where** *c* = (λ*t*. *u* (*monomial 1 t*))
from *inj* **have** *p* = (∑ *t*∈*T*. *monomial* (*c t*) *t*) **by** (*simp add*: *p M sum.reindex*
c-def)
also have . . . ∈ *?A*
proof (*intro hom-deg-set-closed-sum zero-in-Polys Polys-closed-plus*)
fix *t*
assume *t* ∈ *T*
hence *t* ∈ *deg-sect X z* **using** ‹*T* ⊆ *deg-sect X z*› **..**
hence *t* ∈ .[*X*] **and** *eq*: *deg-pm t* = *z* **by** (*rule deg-sectD*)+
from *this*(*1*) **have** *monomial* (*c t*) *t* ∈ *P*[*X*] (**is** *?m* ∈ -) **by** (*rule Polys-closed-monomial*)
thus *?m* ∈ *?A*
by (*simp add*: *hom-deg-set-alt-homogeneous-set*[*OF homogeneous-set-Polys*]
poly-deg-monomial
monomial-0-iff eq)
qed
finally show *p* ∈ *?A* **.**
qed
qed

9.5 (Projective) Hilbert Function

interpretation *phull*: *vector-space map-scale*
apply *standard*
subgoal by (*fact map-scale-distrib-left*)
subgoal by (*fact map-scale-distrib-right*)
subgoal by (*fact map-scale-assoc*)
subgoal by (*fact map-scale-one-left*)
done

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*
by (*rule ext*) (*simp add*: *Hilbert-fun-def hom-deg-set-def*)

lemma *Hilbert-fun-zero* [*simp*]: *Hilbert-fun* {*0*} = *0*
by (*rule ext*) (*simp add*: *Hilbert-fun-def hom-deg-set-def*)

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 −
from *assms*(*3*, *4*) **have** *dd*: *direct-decomp* (*hom-deg-set z A*) (*map* (*hom-deg-set*

z) ps)
  by (*rule direct-decomp-hom-deg-set*)
 have *Hilbert-fun* A z = *phull.dim* (*hom-deg-set* z A) **by** (*fact Hilbert-fun-def*)
 also from *dd* **have** ... = *sum phull.dim* (*set* (*map* (*hom-deg-set* z) *ps*))
 proof (*rule phull.dim-direct-decomp*)
  from *assms*(*1*) **have** *finite* (*deg-sect* X z) **by** (*rule finite-deg-sect*)
  thus *finite* (*monomial* (*1*::'*a*) ' *deg-sect* X z) **by** (*rule finite-imageI*)
 next
  from *assms*(*2*) **have** *hom-deg-set* z $A \subseteq$ *hom-deg-set* z $P[X]$
   unfolding *hom-deg-set-def* **by** (*rule image-mono*)
  thus *hom-deg-set* z $A \subseteq$ *phull.span* (*monomial 1* ' *deg-sect* X z)
   by (*simp only*: *hom-deg-set-Polys-eq-span*)
 next
  fix s
  assume $s \in$ *set* (*map* (*hom-deg-set* z) *ps*)
  then obtain s' **where** $s' \in$ *set ps* **and** s: s = *hom-deg-set* z s' **unfolding** *set-map* **..**
  from *this*(*1*) **have** *phull.subspace* s' **by** (*rule assms*(*5*))
  thus *phull.subspace* s **unfolding** s **by** (*rule subspace-hom-deg-set*)
 qed
 also have ... = *sum* (*phull.dim* $\circ$ *hom-deg-set* z) (*set ps*) **unfolding** *set-map* **using** *finite-set*
 proof (*rule sum.reindex-nontrivial*)
  fix *s1 s2*
  note *dd*
  moreover assume *s1* $\in$ *set ps* **and** *s2* $\in$ *set ps* **and** *s1* $\neq$ *s2*
  moreover have *0* $\in$ *hom-deg-set* z s **if** $s \in$ *set ps* **for** s
  proof (*rule zero-in-hom-deg-set*)
   from *that* **have** *phull.subspace* s **by** (*rule assms*(*5*))
   thus *0* $\in s$ **by** (*rule phull.subspace-0*)
  qed
  ultimately have *hom-deg-set* z *s1* $\cap$ *hom-deg-set* z *s2* = {*0*} **by** (*rule direct-decomp-map-Int-zero*)
  moreover assume *hom-deg-set* z *s1* = *hom-deg-set* z *s2*
  ultimately show *phull.dim* (*hom-deg-set* z *s1*) = *0* **by** *simp*
 qed
 also have ... = ($\sum$ *p*$\in$*set ps*. *Hilbert-fun* p z) **by** (*simp only*: *o-def Hilbert-fun-def*)
 finally show *?thesis* **.**
qed

context *pm-powerprod*
begin

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

assume $t \in ?A$
hence $t \in lpp$ ‘ $(A - \{0\})$ **and** *deg-t*[*symmetric*]: *deg-pm* $t = z$ **by** *simp-all*
from *this*(*1*) **obtain** p **where** $p \in A - \{0\}$ **and** t: $t = lpp\ p$ **..**
from *this*(*1*) **have** $p \in A$ **and** $p \neq 0$ **by** *simp-all*
from *this*(*1*) **have** *1*: *hom-component* $p\ z \in$ *hom-deg-set* $z\ A$ (**is** $?p \in$ -)
unfolding *hom-deg-set-def* **by** (*rule imageI*)
from ‹$p \neq 0$› **have** $?p \neq 0$ **and** *lpp* $?p = t$ **unfolding** *t deg-t* **by** (*rule hom-component-lpp*)+
note *this*(*2*)[*symmetric*]
moreover from *1* ‹$?p \neq 0$› **have** $?p \in$ *hom-deg-set* $z\ A - \{0\}$ **by** *simp*
ultimately show $t \in ?B$ **by** (*rule image-eqI*)
next
fix t
assume $t \in ?B$
then obtain p **where** $p \in$ *hom-deg-set* $z\ A - \{0\}$ **and** t: $t = lpp\ p$ **..**
from *this*(*1*) **have** $p \in$ *hom-deg-set* $z\ A$ **and** $p \neq 0$ **by** *simp-all*
with *assms* **have** $p \in A$ **and** *homogeneous* p **and** *poly-deg* $p = z$
by (*simp-all add*: *hom-deg-set-alt-homogeneous-set*)
from *this*(*1*) ‹$p \neq 0$› **have** $p \in A - \{0\}$ **by** *simp*
hence *1*: $t \in lpp$ ‘ $(A - \{0\})$ **using** t **by** (*rule rev-image-eqI*)
from ‹$p \neq 0$› **have** $t \in$ *keys* p **unfolding** t **by** (*rule punit.lt-in-keys*)
with ‹*homogeneous* p› **have** *deg-pm* t = *poly-deg* p **by** (*rule homogeneousD-poly-deg*)
with *1* **show** $t \in ?A$ **by** (*simp add*: ‹*poly-deg* $p = z$›)
qed

lemma *Hilbert-fun-alt*:
assumes *finite* X **and** $A \subseteq P[X]$ **and** *phull.subspace* A
shows *Hilbert-fun* $A\ z$ = *card* (*lpp* ‘ (*hom-deg-set* $z\ A - \{0\}$)) (**is** - = *card* $?A$)
proof −
have $?A \subseteq lpp$ ‘ (*hom-deg-set* $z\ A - \{0\}$) **by** *simp*
then obtain B **where** *sub*: $B \subseteq$ *hom-deg-set* $z\ A - \{0\}$ **and** *eq1*: $?A = lpp$ ‘ B
and *inj*: *inj-on lpp* B **by** (*rule subset-imageE-inj*)
have *Hilbert-fun* $A\ z$ = *phull.dim* (*hom-deg-set* $z\ A$) **by** (*fact Hilbert-fun-def*)
also have . . . = *card* B
proof (*rule phull.dim-eq-card*)
show *phull.span* B = *phull.span* (*hom-deg-set* $z\ A$)
proof
from *sub* **have** $B \subseteq$ *hom-deg-set* $z\ A$ **by** *blast*
thus *phull.span* $B \subseteq$ *phull.span* (*hom-deg-set* $z\ A$) **by** (*rule phull.span-mono*)
next
from *assms*(*3*) **have** *phull.subspace* (*hom-deg-set* $z\ A$) **by** (*rule subspace-hom-deg-set*)
hence *phull.span* (*hom-deg-set* $z\ A$) = *hom-deg-set* $z\ A$ **by** (*simp only*: *phull.span-eq-iff*)
also have . . . $\subseteq$ *phull.span* B
proof (*rule ccontr*)
assume ¬ *hom-deg-set* $z\ A \subseteq$ *phull.span* B
then obtain *p0* **where** *p0* $\in$ *hom-deg-set* $z\ A -$ *phull.span* B (**is** - $\in ?B$) **by** *blast*
note *assms*(*1*) *this*

moreover have *?B* $\subseteq$ *P*[*X*]
proof (*rule subset-trans*)
from *assms*(*2*) **show** *hom-deg-set z A* $\subseteq$ *P*[*X*] **by** (*rule Polys-closed-hom-deg-set*)
qed *blast*
ultimately obtain *p* **where** *p* $\in$ *?B* **and** *p-min*: $\bigwedge$*q*. *punit.ord-strict-p q p* $\Longrightarrow$ *q* $\notin$ *?B*
by (*rule punit.ord-p-minimum-dgrad-p-set*[*OF dickson-grading-varnum*, **where** *m=0*,
simplified dgrad-p-set-varnum]) *blast*
from *this*(*1*) **have** *p* $\in$ *hom-deg-set z A* **and** *p* $\notin$ *phull.span B* **by** *simp-all*
from *phull.span-zero this*(*2*) **have** *p* $\neq$ *0* **by** *blast*
with ‹*p* $\in$ *hom-deg-set z A*› **have** *p* $\in$ *hom-deg-set z A* $-$ {*0*} **by** *simp*
hence *lpp p* $\in$ *lpp* ‘ (*hom-deg-set z A* $-$ {*0*}) **by** (*rule imageI*)
also have ... = *lpp* ‘ *B* **by** (*simp only*: *eq1*)
finally obtain *b* **where** *b* $\in$ *B* **and** *eq2*: *lpp p* = *lpp b* **..**
from *this*(*1*) *sub* **have** *b* $\in$ *hom-deg-set z A* $-$ {*0*} **..**
hence *b* $\in$ *hom-deg-set z A* **and** *b* $\neq$ *0* **by** *simp-all*
from *this*(*2*) **have** *lcb*: *punit.lc b* $\neq$ *0* **by** (*rule punit.lc-not-0*)
from ‹*p* $\neq$ *0*› **have** *lcp*: *punit.lc p* $\neq$ *0* **by** (*rule punit.lc-not-0*)
from ‹*b* $\in$ *B*› **have** *b* $\in$ *phull.span B* **by** (*rule phull.span-base*)
hence (*punit.lc p* / *punit.lc b*) $\cdot$ *b* $\in$ *phull.span B* (**is** *?b* $\in$ -) **by** (*rule phull.span-scale*)
with ‹*p* $\notin$ *phull.span B*› **have** *p* $-$ *?b* $\neq$ *0* **by** *auto*
moreover from *lcb lcp* ‹*b* $\neq$ *0*› **have** *lpp ?b* = *lpp p*
by (*simp add*: *punit.map-scale-eq-monom-mult punit.lt-monom-mult eq2*)
moreover from *lcb* **have** *punit.lc ?b* = *punit.lc p* **by** (*simp add*: *punit.map-scale-eq-monom-mult*)
ultimately have *lpp* (*p* $-$ *?b*) $\prec$ *lpp p* **by** (*rule punit.lt-minus-lessI*)
hence *punit.ord-strict-p* (*p* $-$ *?b*) *p* **by** (*rule punit.lt-ord-p*)
hence *p* $-$ *?b* $\notin$ *?B* **by** (*rule p-min*)
hence *p* $-$ *?b* $\notin$ *hom-deg-set z A* $\vee$ *p* $-$ *?b* $\in$ *phull.span B* **by** *simp*
thus *False*
proof
assume $*$: *p* $-$ *?b* $\notin$ *hom-deg-set z A*
from *phull.subspace-scale* **have** *?b* $\in$ *hom-deg-set z A*
proof (*rule hom-deg-set-closed-scalar*)
show *phull.subspace A* **by** *fact*
next
show *b* $\in$ *hom-deg-set z A* **by** *fact*
qed
with *phull.subspace-diff* ‹*p* $\in$ *hom-deg-set z A*› **have** *p* $-$ *?b* $\in$ *hom-deg-set z A*
by (*rule hom-deg-set-closed-minus*) (*rule assms*(*3*))
with $*$ **show** *?thesis* **..**
next
assume *p* $-$ *?b* $\in$ *phull.span B*
hence *p* $-$ *?b* + *?b* $\in$ *phull.span B* **using** ‹*?b* $\in$ *phull.span B*› **by** (*rule phull.span-add*)
hence *p* $\in$ *phull.span B* **by** *simp*
with ‹*p* $\notin$ *phull.span B*› **show** *?thesis* **..**

qed
qed
finally show *phull.span* (*hom-deg-set z A*) $\subseteq$ *phull.span B* **.**
qed
next
show *phull.independent B*
proof
assume *phull.dependent B*
then obtain B' u b' **where** *finite* B' **and** $B' \subseteq B$ **and** $(\sum b{\in}B'.\ u\ b \cdot b) = 0$
and $b' \in B'$ **and** $u\ b' \neq 0$ **unfolding** *phull.dependent-explicit* **by** *blast*
define *B0* **where** $B0 = \{b \in B'.\ u\ b \neq 0\}$
have $B0 \subseteq B'$ **by** (*simp add*: *B0-def*)
with ‹*finite* B'› **have** $(\sum b{\in}B0.\ u\ b \cdot b) = (\sum b{\in}B'.\ u\ b \cdot b)$
by (*rule sum.mono-neutral-left*) (*simp add*: *B0-def*)
also have $\ldots = 0$ **by** *fact*
finally have *eq*: $(\sum b{\in}B0.\ u\ b \cdot b) = 0$ **.**
define *t* **where** $t =$ *ordered-powerprod-lin.Max* (*lpp* ‘ *B0*)
from ‹$b' \in B'$› ‹$u\ b' \neq 0$› **have** $b' \in B0$ **by** (*simp add*: *B0-def*)
hence *lpp* $b' \in$ *lpp* ‘ *B0* **by** (*rule imageI*)
hence *lpp* ‘ $B0 \neq \{\}$ **by** *blast*
from ‹$B0 \subseteq B'$› ‹*finite* B'› **have** *finite B0* **by** (*rule finite-subset*)
hence *finite* (*lpp* ‘ *B0*) **by** (*rule finite-imageI*)
hence $t \in$ *lpp* ‘ *B0* **unfolding** *t-def* **using** ‹*lpp* ‘ $B0 \neq \{\}$›
by (*rule ordered-powerprod-lin.Max-in*)
then obtain *b0* **where** $b0 \in B0$ **and** *t*: $t =$ *lpp b0* **..**
note *this*(*1*)
moreover from ‹$B0 \subseteq B'$› ‹$B' \subseteq B$› **have** $B0 \subseteq B$ **by** (*rule subset-trans*)
also have $\ldots \subseteq$ *hom-deg-set z A* $- \{0\}$ **by** *fact*
finally have $b0 \in$ *hom-deg-set z A* $- \{0\}$ **.**
hence $b0 \neq 0$ **by** *simp*
hence $t \in$ *keys b0* **unfolding** *t* **by** (*rule punit.lt-in-keys*)
have *lookup* $(\sum b{\in}B0.\ u\ b \cdot b)\ t = (\sum b{\in}B0.\ u\ b * lookup\ b\ t)$ **by** (*simp add*: *lookup-sum*)
also from ‹*finite B0*› **have** $\ldots = (\sum b{\in}\{b0\}.\ u\ b * lookup\ b\ t)$
proof (*rule sum.mono-neutral-right*)
from ‹$b0 \in B0$› **show** $\{b0\} \subseteq B0$ **by** *simp*
next
show $\forall\, b{\in}B0 - \{b0\}.\ u\ b * lookup\ b\ t = 0$
proof
fix *b*
assume $b \in B0 - \{b0\}$
hence $b \in B0$ **and** $b \neq b0$ **by** *simp-all*
from *this*(*1*) **have** *lpp* $b \in$ *lpp* ‘ *B0* **by** (*rule imageI*)
with ‹*finite* (*lpp* ‘ *B0*)› **have** *lpp* $b \preceq t$ **unfolding** *t-def*
by (*rule ordered-powerprod-lin.Max-ge*)
have $t \notin$ *keys b*
proof
assume $t \in$ *keys b*
hence $t \preceq$ *lpp b* **by** (*rule punit.lt-max-keys*)

with ‹*lpp b* $\preceq$ *t*› **have** *lpp b* = *lpp b0*
unfolding *t* **by** *simp*
from *inj* ‹*B0* $\subseteq$ *B*› **have** *inj-on lpp B0* **by** (*rule inj-on-subset*)
hence *b* = *b0* **using** ‹*lpp b* = *lpp b0*› ‹*b* $\in$ *B0*› ‹*b0* $\in$ *B0*› **by** (*rule inj-onD*)
with ‹*b* $\neq$ *b0*› **show** *False* **..**
qed
thus *u b* * *lookup b t* = *0* **by** (*simp add*: *in-keys-iff*)
qed
qed
also from ‹*t* $\in$ *keys b0*› ‹*b0* $\in$ *B0*› **have** ... $\neq$ *0* **by** (*simp add*: *B0-def in-keys-iff*)
finally show *False* **by** (*simp add*: *eq*)
qed
qed
also have ... = *card ?A* **unfolding** *eq1* **using** *inj* **by** (*rule card-image*[*symmetric*])
finally show *?thesis* **.**
qed

end

end

10 Cone Decompositions

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

10.1 More Properties of Reduced Gröbner Bases

context *pm-powerprod*
begin

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

end

10.2 Quotient Ideals

definition *quot-set* :: $'a$ *set* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$::*semigroup-mult set* (**infixl** ‹÷› *55*)
  where *quot-set* A x = $(*)$ x −‘ A

lemma *quot-set-iff*: $a \in A \div x \longleftrightarrow x * a \in A$
  by (*simp add*: *quot-set-def*)

lemma *quot-setI*: $x * a \in A \Longrightarrow a \in A \div x$
  by (*simp only*: *quot-set-iff*)

lemma *quot-setD*: $a \in A \div x \Longrightarrow x * a \in A$
  by (*simp only*: *quot-set-iff*)

lemma *quot-set-quot-set* [*simp*]: $A \div x \div y = A \div x * y$
  by (*rule set-eqI*) (*simp add*: *quot-set-iff mult.assoc*)

lemma *quot-set-one* [*simp*]: $A \div$ (*1*::-::*monoid-mult*) = A
  by (*rule set-eqI*) (*simp add*: *quot-set-iff*)

lemma *ideal-quot-set-ideal* [*simp*]: *ideal* (*ideal* $B \div x$) = (*ideal* B) $\div$ (x::-::*comm-ring*)
proof
  show *ideal* (*ideal* $B \div x$) $\subseteq$ *ideal* $B \div x$
  proof
    fix b
    assume $b \in$ *ideal* (*ideal* $B \div x$)
    thus $b \in$ *ideal* $B \div x$
    proof (*induct* b *rule*: *ideal.span-induct'*)
      case *base*
      show *?case* **by** (*simp add*: *quot-set-iff ideal.span-zero*)
    next
      case (*step* b q p)
      hence $x * b \in$ *ideal* B **and** $x * p \in$ *ideal* B **by** (*simp-all add*: *quot-set-iff*)
      hence $x * b + q * (x * p) \in$ *ideal* B
        by (*intro ideal.span-add ideal.span-scale*[**where** c=q])
      thus *?case* **by** (*simp only*: *quot-set-iff algebra-simps*)
    qed
  qed
qed (*fact ideal.span-superset*)

lemma *quot-set-image-times*: *inj* $((*)\ x) \Longrightarrow ((*)\ x$ ‘ $A) \div x = A$
  by (*simp add*: *quot-set-def inj-vimage-image-eq*)

10.3 Direct Decompositions of Polynomial Rings

context *pm-powerprod*
begin

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

where *normal-form F p* = (*SOME q*. (*punit.red* (*punit.reduced-GB F*))** *p q* ∧ ¬ *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* ⊆ *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 −
from *fin-X F-sub* **have** *finite* (*punit.reduced-GB F*) **by** (*rule finite-reduced-GB-Polys*)
hence *wfP* (*punit.red* (*punit.reduced-GB F*))$^{-1-1}$ **by** (*rule punit.red-wf-finite*)
then obtain *q* **where** (*punit.red* (*punit.reduced-GB F*))** *p q*
and ¬ *punit.is-red* (*punit.reduced-GB F*) *q* **unfolding** *punit.is-red-def not-not*
by (*rule relation.wf-imp-nf-ex*)
hence (*punit.red* (*punit.reduced-GB F*))** *p q* ∧ ¬ *punit.is-red* (*punit.reduced-GB F*) *q* **..**
hence *?thesis1* ∧ *?thesis2* **unfolding** *normal-form-def* **by** (*rule someI*)
thus *?thesis1* **and** *?thesis2* **by** *simp-all*
qed

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 (*rule relation.ChurchRosser-unique-final*)
from *fin-X F-sub* **have** *punit.is-Groebner-basis* (*punit.reduced-GB F*) **by** (*rule reduced-GB-is-GB-Polys*)
thus *relation.is-ChurchRosser* (*punit.red* (*punit.reduced-GB F*))
by (*simp only*: *punit.is-Groebner-basis-def*)
next
show (*punit.red* (*punit.reduced-GB F*))** *p* (*normal-form F p*) **by** (*rule normal-form*)
next
have ¬ *punit.is-red* (*punit.reduced-GB F*) (*normal-form F p*) **by** (*rule normal-form*)
thus *relation.is-final* (*punit.red* (*punit.reduced-GB F*)) (*normal-form F p*)
by (*simp add*: *punit.is-red-def*)
next
from *assms*(*2*) **show** *relation.is-final* (*punit.red* (*punit.reduced-GB F*)) *q*
by (*simp add*: *punit.is-red-def*)

qed *fact*

lemma *normal-form-id-iff*: *normal-form F p* = *p* ⟷ (¬ *punit.is-red* (*punit.reduced-GB F*) *p*)
proof
  assume *normal-form F p* = *p*
  with *normal-form(2)*[*of p*] **show** ¬ *punit.is-red* (*punit.reduced-GB F*) *p* **by** *simp*
next
  assume ¬ *punit.is-red* (*punit.reduced-GB F*) *p*
  with *rtranclp.rtrancl-refl* **show** *normal-form F p* = *p* **by** (*rule normal-form-unique*)
qed

lemma *normal-form-normal-form*: *normal-form F* (*normal-form F p*) = *normal-form F p*
  by (*simp add*: *normal-form-id-iff normal-form*)

lemma *normal-form-zero*: *normal-form F 0* = *0*
  by (*simp add*: *normal-form-id-iff punit.irred-0*)

lemma *normal-form-map-scale*: *normal-form F* (*c* · *p*) = *c* · (*normal-form F p*)
  by (*intro normal-form-unique punit.is-irred-map-scale normal-form*)
    (*simp add*: *punit.map-scale-eq-monom-mult punit.red-rtrancl-mult normal-form*)

lemma *normal-form-uminus*: *normal-form F* (− *p*) = − *normal-form F p*
  by (*intro normal-form-unique punit.red-rtrancl-uminus normal-form*)
    (*simp add*: *punit.is-red-uminus normal-form*)

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*
  by (*intro normal-form-unique rtranclp.rtrancl-refl punit.is-irred-plus normal-form*)

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*
  by (*intro normal-form-unique rtranclp.rtrancl-refl punit.is-irred-minus normal-form*)

lemma *normal-form-ideal-Polys*: *normal-form* (*ideal F* ∩ *P*[*X*]) = *normal-form F*
proof −
  let *?F* = *ideal F* ∩ *P*[*X*]
  from *fin-X* **have** *eq*: *punit.reduced-GB ?F* = *punit.reduced-GB F*
  proof (*rule reduced-GB-unique-Polys*)
    from *fin-X F-sub* **show** *punit.is-reduced-GB* (*punit.reduced-GB F*)
      by (*rule reduced-GB-is-reduced-GB-Polys*)
  next
    from *fin-X F-sub* **have** *ideal* (*punit.reduced-GB F*) = *ideal F* **by** (*rule reduced-GB-ideal-Polys*)
    also have . . . = *ideal* (*ideal F* ∩ *P*[*X*])
    proof (*intro subset-antisym ideal.span-subset-spanI*)

from *ideal.span-superset*[*of F*] *F-sub* **have** $F \subseteq$ *ideal* $F \cap P[X]$ **by** *simp*
thus $F \subseteq$ *ideal* (*ideal* $F \cap P[X]$) **using** *ideal.span-superset* **by** (*rule subset-trans*)
qed *blast*
finally show *ideal* (*punit.reduced-GB F*) = *ideal* (*ideal* $F \cap P[X]$) **.**
qed *blast*
show *?thesis* **by** (*rule ext*) (*simp only*: *normal-form-def eq*)
qed

lemma *normal-form-diff-in-ideal*: *p* − *normal-form F p* ∈ *ideal F*
proof −
from *normal-form*(*1*) **have** *p* − *normal-form F p* ∈ *ideal* (*punit.reduced-GB F*)
by (*rule punit.red-rtranclp-diff-in-pmdl*[*simplified*])
also from *fin-X F-sub* **have** ... = *ideal F* **by** (*rule reduced-GB-ideal-Polys*)
finally show *?thesis* **.**
qed

lemma *normal-form-zero-iff*: *normal-form F p* = *0* ⟷ *p* ∈ *ideal F*
proof
assume *normal-form F p* = *0*
with *normal-form-diff-in-ideal*[*of p*] **show** *p* ∈ *ideal F* **by** *simp*
next
assume *p* ∈ *ideal F*
hence *p* − (*p* − *normal-form F p*) ∈ *ideal F* **using** *normal-form-diff-in-ideal*
by (*rule ideal.span-diff*)
also from *fin-X F-sub* **have** ... = *ideal* (*punit.reduced-GB F*) **by** (*rule reduced-GB-ideal-Polys*[*symmetric*])
finally have ∗: *normal-form F p* ∈ *ideal* (*punit.reduced-GB F*) **by** *simp*
show *normal-form F p* = *0*
proof (*rule ccontr*)
from *fin-X F-sub* **have** *punit.is-Groebner-basis* (*punit.reduced-GB F*) **by** (*rule reduced-GB-is-GB-Polys*)
moreover note ∗
moreover assume *normal-form F p* ≠ *0*
ultimately obtain *g* **where** *g* ∈ *punit.reduced-GB F* **and** *g* ≠ *0*
and *a*: *lpp g adds lpp* (*normal-form F p*) **by** (*rule punit.GB-adds-lt*[*simplified*])
note *this*(*1*, *2*)
moreover from ‹*normal-form F p* ≠ *0*› **have** *lpp* (*normal-form F p*) ∈ *keys* (*normal-form F p*)
by (*rule punit.lt-in-keys*)
ultimately have *punit.is-red* (*punit.reduced-GB F*) (*normal-form F p*)
using *a* **by** (*rule punit.is-red-addsI*[*simplified*])
with *normal-form*(*2*) **show** *False* **..**
qed
qed

lemma *normal-form-eq-iff*: *normal-form F p* = *normal-form F q* ⟷ *p* − *q* ∈ *ideal F*
proof −

have $p - q - (\textit{normal-form } F\ p - \textit{normal-form } F\ q) = (p - \textit{normal-form } F\ p) - (q - \textit{normal-form } F\ q)$
by *simp*
also from *normal-form-diff-in-ideal normal-form-diff-in-ideal* **have** $\ldots \in \textit{ideal } F$
by (*rule ideal.span-diff*)
finally have $*$: $p - q - (\textit{normal-form } F\ p - \textit{normal-form } F\ q) \in \textit{ideal } F$ **.**
show *?thesis*
proof
assume $\textit{normal-form } F\ p = \textit{normal-form } F\ q$
with $*$ **show** $p - q \in \textit{ideal } F$ **by** *simp*
next
assume $p - q \in \textit{ideal } F$
hence $p - q - (p - q - (\textit{normal-form } F\ p - \textit{normal-form } F\ q)) \in \textit{ideal } F$ **using** $*$
by (*rule ideal.span-diff*)
hence $\textit{normal-form } F\ (\textit{normal-form } F\ p - \textit{normal-form } F\ q) = 0$ **by** (*simp add*: *normal-form-zero-iff*)
thus $\textit{normal-form } F\ p = \textit{normal-form } F\ q$ **by** (*simp add*: *normal-form-minus-normal-form*)
qed
qed

lemma *Polys-closed-normal-form*:
assumes $p \in P[X]$
shows $\textit{normal-form } F\ p \in P[X]$
proof $-$
from *fin-X F-sub* **have** $\textit{punit.reduced-GB } F \subseteq P[X]$ **by** (*rule reduced-GB-Polys*)
with *fin-X* **show** *?thesis* **using** *assms normal-form*(*1*)
by (*rule punit.dgrad-p-set-closed-red-rtrancl*[*OF dickson-grading-varnum*, **where** *m=0*, *simplified dgrad-p-set-varnum*])
qed

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
assume $p \in \textit{normal-form } F\ \text{‘}\ P[X]$
then obtain q **where** $q \in P[X]$ **and** p: $p = \textit{normal-form } F\ q$ **..**
from *this*(*1*) **show** $p \in P[X] \wedge \neg\ \textit{punit.is-red } (\textit{punit.reduced-GB } F)\ p$ **unfolding** p
by (*intro conjI Polys-closed-normal-form normal-form*)
next
assume $p \in P[X] \wedge \neg\ \textit{punit.is-red } (\textit{punit.reduced-GB } F)\ p$
hence $p \in P[X]$ **and** $\neg\ \textit{punit.is-red } (\textit{punit.reduced-GB } F)\ p$ **by** *simp-all*
from *this*(*2*) **have** $\textit{normal-form } F\ p = p$ **by** (*simp add*: *normal-form-id-iff*)
from *this*[*symmetric*] ‹$p \in P[X]$› **show** $p \in \textit{normal-form } F\ \text{‘}\ P[X]$ **by** (*rule image-eqI*)
qed

end

lemma *direct-decomp-ideal-insert*:
  fixes F **and** f
  defines $I \equiv ideal\ (insert\ f\ F)$
  defines $L \equiv (ideal\ F \div f) \cap P[X]$
  assumes $F \subseteq P[X]$ **and** $f \in P[X]$
  shows *direct-decomp* $(I \cap P[X])$ $[ideal\ F \cap P[X],\ (*)\ f\ \text{‘}\ normal\text{-}form\ L\ \text{‘}\ P[X]]$
    (**is** *direct-decomp* - *?ss*)
proof (*rule direct-decompI-alt*)
  fix *qs*
  assume $qs \in listset\ ?ss$
  then obtain $x\ y$ **where** x: $x \in ideal\ F \cap P[X]$ **and** y: $y \in (*)\ f\ \text{‘}\ normal\text{-}form$
$L\ \text{‘}\ P[X]$
    and *qs*: $qs = [x, y]$ **by** (*rule listset-doubletonE*)
  have $sum\text{-}list\ qs = x + y$ **by** (*simp add*: *qs*)
  also have $\ldots \in I \cap P[X]$ **unfolding** *I-def*
  proof (*intro IntI ideal.span-add Polys-closed-plus*)
    have $ideal\ F \subseteq ideal\ (insert\ f\ F)$ **by** (*rule ideal.span-mono*) *blast*
    with x **show** $x \in ideal\ (insert\ f\ F)$ **and** $x \in P[X]$ **by** *blast+*
  next
    from y **obtain** p **where** $p \in P[X]$ **and** y: $y = f * normal\text{-}form\ L\ p$ **by** *blast*
    have $f \in ideal\ (insert\ f\ F)$ **by** (*rule ideal.span-base*) *simp*
    hence $normal\text{-}form\ L\ p * f \in ideal\ (insert\ f\ F)$ **by** (*rule ideal.span-scale*)
    thus $y \in ideal\ (insert\ f\ F)$ **by** (*simp only*: *mult.commute* y)

    have $L \subseteq P[X]$ **by** (*simp add*: *L-def*)
    hence $normal\text{-}form\ L\ p \in P[X]$ **using** ‹$p \in P[X]$› **by** (*rule Polys-closed-normal-form*)
    with *assms*(*4*) **show** $y \in P[X]$ **unfolding** y **by** (*rule Polys-closed-times*)
  qed
  finally show $sum\text{-}list\ qs \in I \cap P[X]$ **.**
next
  fix a
  assume $a \in I \cap P[X]$
  hence $a \in I$ **and** $a \in P[X]$ **by** *simp-all*
  from *assms*(*3*, *4*) **have** $insert\ f\ F \subseteq P[X]$ **by** *simp*
  then obtain $F0\ q0$ **where** $F0 \subseteq insert\ f\ F$ **and** *finite F0* **and** *q0*: $\bigwedge f0.\ q0\ f0$
$\in P[X]$
    and a: $a = (\sum f0 \in F0.\ q0\ f0 * f0)$
    using ‹$a \in P[X]$› ‹$a \in I$› **unfolding** *I-def* **by** (*rule in-idealE-Polys*) *blast*
  obtain $q\ a'$ **where** a': $a' \in ideal\ F$ **and** $a' \in P[X]$ **and** $q \in P[X]$ **and** a: $a = q$
$* f + a'$
  proof (*cases* $f \in F0$)
    case *True*
    with ‹$F0 \subseteq insert\ f\ F$› **have** $F0 - \{f\} \subseteq F$ **by** *blast*
    show *?thesis*
    proof
    have $(\sum f0 \in F0 - \{f\}.\ q0\ f0 * f0) \in ideal\ (F0 - \{f\})$ **by** (*rule ideal.sum-in-spanI*)
      also from ‹$F0 - \{f\} \subseteq F$› **have** $\ldots \subseteq ideal\ F$ **by** (*rule ideal.span-mono*)
      finally show $(\sum f0 \in F0 - \{f\}.\ q0\ f0 * f0) \in ideal\ F$ **.**

next
show $(\sum f0 \in F0 - \{f\}.\ q0\ f0 * f0) \in P[X]$
proof (*intro Polys-closed-sum Polys-closed-times q0*)
fix *f0*
assume $f0 \in F0 - \{f\}$
also have $\ldots \subseteq F0$ **by** *blast*
also have $\ldots \subseteq$ *insert f F* **by** *fact*
also have $\ldots \subseteq P[X]$ **by** *fact*
finally show $f0 \in P[X]$ **.**
qed
next
from ‹*finite F0*› *True* **show** $a = q0\ f * f + (\sum f0 \in F0 - \{f\}.\ q0\ f0 * f0)$
by (*simp only*: *a sum.remove*)
qed *fact*
next
case *False*
with ‹$F0 \subseteq$ *insert f F*› **have** $F0 \subseteq F$ **by** *blast*
show *?thesis*
proof
have $a \in$ *ideal F0* **unfolding** *a* **by** (*rule ideal.sum-in-spanI*)
also from ‹$F0 \subseteq F$› **have** $\ldots \subseteq$ *ideal F* **by** (*rule ideal.span-mono*)
finally show $a \in$ *ideal F* **.**
next
show $a = 0 * f + a$ **by** *simp*
qed (*fact* ‹$a \in P[X]$›, *fact zero-in-Polys*)
qed
let $?a = f *$ (*normal-form L q*)
have $L \subseteq P[X]$ **by** (*simp add*: *L-def*)
hence *normal-form L q* $\in P[X]$ **using** ‹$q \in P[X]$› **by** (*rule Polys-closed-normal-form*)
with *assms*(*4*) **have** $?a \in P[X]$ **by** (*rule Polys-closed-times*)
from ‹$L \subseteq P[X]$› **have** $q -$ *normal-form L q* $\in$ *ideal L* **by** (*rule normal-form-diff-in-ideal*)
also have $\ldots \subseteq$ *ideal* (*ideal F* $\div$ *f*) **unfolding** *L-def* **by** (*rule ideal.span-mono*)
blast
finally have $f * (q -$ *normal-form L q*$) \in$ *ideal F* **by** (*simp add*: *quot-set-iff*)
with ‹$a' \in$ *ideal F*› **have** $a' + f * (q -$ *normal-form L q*$) \in$ *ideal F* **by** (*rule ideal.span-add*)
hence $a - ?a \in$ *ideal F* **by** (*simp add*: *a algebra-simps*)

define *qs* **where** $qs = [a - ?a,\ ?a]$
show $\exists! qs \in$ *listset ?ss*. $a =$ *sum-list qs*
proof (*intro ex1I conjI allI impI*)
have $a - ?a \in$ *ideal F* $\cap P[X]$
proof
from *assms*(*4*) ‹$a \in P[X]$› ‹*normal-form L q* $\in P[X]$› **show** $a - ?a \in P[X]$
by (*intro Polys-closed-minus Polys-closed-times*)
qed *fact*
moreover from ‹$q \in P[X]$› **have** $?a \in (*)\ f$ ‘ *normal-form L* ‘ $P[X]$ **by** (*intro imageI*)
ultimately show $qs \in$ *listset ?ss* **using** *qs-def* **by** (*rule listset-doubletonI*)

next
fix *qs0*
assume *qs0* $\in$ *listset ?ss* $\wedge$ *a* = *sum-list qs0*
hence *qs0* $\in$ *listset ?ss* **and** *a* = *sum-list qs0* **by** *simp-all*
from *this*(*1*) **obtain** *x y* **where** *x* $\in$ *ideal F* $\cap$ *P*[*X*] **and** *y* $\in$ ($*$) *f* ‘ *normal-form L* ‘ *P*[*X*]
and *qs0*: *qs0* = [*x*, *y*] **by** (*rule listset-doubletonE*)
from *this*(*2*) **obtain** *a0* **where** *a0* $\in$ *P*[*X*] **and** *y*: *y* = *f* $*$ *normal-form L a0* **by** *blast*
from ‹*x* $\in$ *ideal F* $\cap$ *P*[*X*]› **have** *x* $\in$ *ideal F* **by** *simp*
have *x*: *x* = *a* − *y* **by** (*simp add*: ‹*a* = *sum-list qs0*› *qs0*)
have *f* $*$ (*normal-form L q* − *normal-form L a0*) = *x* − (*a* − *?a*) **by** (*simp add*: *x y a algebra-simps*)
also from ‹*x* $\in$ *ideal F*› ‹*a* − *?a* $\in$ *ideal F*› **have** ... $\in$ *ideal F* **by** (*rule ideal.span-diff*)
finally have *normal-form L q* − *normal-form L a0* $\in$ *ideal F* $\div$ *f* **by** (*rule quot-setI*)
moreover from ‹*L* $\subseteq$ *P*[*X*]› ‹*q* $\in$ *P*[*X*]› ‹*a0* $\in$ *P*[*X*]› **have** *normal-form L q* − *normal-form L a0* $\in$ *P*[*X*]
by (*intro Polys-closed-minus Polys-closed-normal-form*)
ultimately have *normal-form L q* − *normal-form L a0* $\in$ *L* **by** (*simp add*: *L-def*)
also have ... $\subseteq$ *ideal L* **by** (*fact ideal.span-superset*)
finally have *normal-form L q* − *normal-form L a0* = *0* **using** ‹*L* $\subseteq$ *P*[*X*]›
by (*simp only*: *normal-form-minus-normal-form flip*: *normal-form-zero-iff*)
thus *qs0* = *qs* **by** (*simp add*: *qs0 qs-def x y*)
qed (*simp-all add*: *qs-def*)
qed

corollary *direct-decomp-ideal-normal-form*:
assumes *F* $\subseteq$ *P*[*X*]
shows *direct-decomp P*[*X*] [*ideal F* $\cap$ *P*[*X*], *normal-form F* ‘ *P*[*X*]]
proof −
from *assms one-in-Polys* **have** *direct-decomp* (*ideal* (*insert 1 F*) $\cap$ *P*[*X*]) [*ideal F* $\cap$ *P*[*X*],
($*$) *1* ‘ *normal-form* ((*ideal F* $\div$ *1*) $\cap$ *P*[*X*]) ‘ *P*[*X*]]
by (*rule direct-decomp-ideal-insert*)
moreover have *ideal* (*insert 1 F*) = *UNIV*
by (*simp add*: *ideal-eq-UNIV-iff-contains-one ideal.span-base*)
moreover from *refl* **have** (($*$) *1* $\circ$ *normal-form F*) ‘ *P*[*X*] = *normal-form F* ‘ *P*[*X*]
by (*rule image-cong*) *simp*
ultimately show *?thesis* **using** *assms* **by** (*simp add*: *image-comp normal-form-ideal-Polys*)
qed

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 $cone\ hU = (*)\ (fst\ hU)\ `\ P[snd\ hU]$

lemma *coneI*: $p = a * h \Longrightarrow a \in P[U] \Longrightarrow p \in cone\ (h,\ U)$
  by (*auto simp*: *cone-def mult.commute*[*of a*])

lemma *coneE*:
  assumes $p \in cone\ (h,\ U)$
  obtains *a* **where** $a \in P[U]$ **and** $p = a * h$
  using *assms* **by** (*auto simp*: *cone-def mult.commute*)

lemma *cone-empty*: $cone\ (h,\ \{\}) = range\ (\lambda c.\ c \cdot h)$
  by (*auto simp*: *Polys-empty map-scale-eq-times intro*: *coneI elim*!: *coneE*)

lemma *cone-zero* [*simp*]: $cone\ (0,\ U) = \{0\}$
  by (*auto simp*: *cone-def intro*: *zero-in-Polys*)

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

lemma *zero-in-cone*: $0 \in cone\ hU$
  by (*auto simp*: *cone-def intro*!: *image-eqI zero-in-Polys*)

corollary *empty-not-in-map-cone*: $\{\} \notin set\ (map\ cone\ ps)$
  using *zero-in-cone* **by** *fastforce*

lemma *tip-in-cone*: $h \in cone\ (h{::}\text{-} \Rightarrow_0 \text{-}{::}comm\text{-}semiring\text{-}1,\ U)$
  using - *one-in-Polys* **by** (*rule coneI*) *simp*

lemma *cone-closed-plus*:
  assumes $a \in cone\ hU$ **and** $b \in cone\ hU$
  shows $a + b \in cone\ hU$
proof −
  obtain *h U* **where** *hU*: $hU = (h,\ U)$ **using** *prod.exhaust* **by** *blast*
  with *assms* **have** $a \in cone\ (h,\ U)$ **and** $b \in cone\ (h,\ U)$ **by** *simp-all*
  from *this*(*1*) **obtain** a' **where** $a' \in P[U]$ **and** *a*: $a = a' * h$ **by** (*rule coneE*)
  from ‹$b \in cone\ (h,\ U)$› **obtain** b' **where** $b' \in P[U]$ **and** *b*: $b = b' * h$ **by** (*rule coneE*)
  have $a + b = (a' + b') * h$ **by** (*simp only*: *a b algebra-simps*)
  moreover from ‹$a' \in P[U]$› ‹$b' \in P[U]$› **have** $a' + b' \in P[U]$ **by** (*rule Polys-closed-plus*)
  ultimately show *?thesis* **unfolding** *hU* **by** (*rule coneI*)
qed

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

proof −
  obtain *h U* **where** *hU*: $hU = (h,\ U)$ **using** *prod.exhaust* **by** *blast*
  with *assms* **have** $a \in$ *cone* $(h,\ U)$ **by** *simp*
  from *this*(*1*) **obtain** a' **where** $a' \in P[U]$ **and** *a*: $a = a' * h$ **by** (*rule coneE*)
  have $-\ a = (-\ a') * h$ **by** (*simp add*: *a*)
  moreover from ‹$a' \in P[U]$› **have** $-\ a' \in P[U]$ **by** (*rule Polys-closed-uminus*)
  ultimately show *?thesis* **unfolding** *hU* **by** (*rule coneI*)
qed

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 −
  from *assms*(*2*) **have** $-\ b \in$ *cone hU* **by** (*rule cone-closed-uminus*)
  with *assms*(*1*) **have** $a + (-\ b) \in$ *cone hU* **by** (*rule cone-closed-plus*)
  thus *?thesis* **by** *simp*
qed

lemma *cone-closed-times*:
  assumes $a \in$ *cone* $(h,\ U)$ **and** $q \in P[U]$
  shows $q * a \in$ *cone* $(h,\ U)$
proof −
  from *assms*(*1*) **obtain** a' **where** $a' \in P[U]$ **and** *a*: $a = a' * h$ **by** (*rule coneE*)
  have $q * a = (q * a') * h$ **by** (*simp only*: *a ac-simps*)
  moreover from *assms*(*2*) ‹$a' \in P[U]$› **have** $q * a' \in P[U]$ **by** (*rule Polys-closed-times*)
  ultimately show *?thesis* **by** (*rule coneI*)
qed

corollary *cone-closed-monom-mult*:
  assumes $a \in$ *cone* $(h,\ U)$ **and** $t \in .[U]$
  shows *punit.monom-mult c t a* $\in$ *cone* $(h,\ U)$
proof −
  from *assms*(*2*) **have** *monomial c t* $\in P[U]$ **by** (*rule Polys-closed-monomial*)
  with *assms*(*1*) **have** *monomial c t* $* a \in$ *cone* $(h,\ U)$ **by** (*rule cone-closed-times*)
  thus *?thesis* **by** (*simp only*: *times-monomial-left*)
qed

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 −
  from *assms*(*1*) **obtain** *a* **where** *p*: $p = a * h$ **by** (*rule coneE*)
  with *assms*(*2*) **have** $a \neq 0$ **and** $h \neq 0$ **by** *auto*
  hence *lpp p* = *lpp a* + *lpp h* **unfolding** *p* **by** (*rule lp-times*)
  also have ... = *lpp h* + *lpp a* **by** (*rule add.commute*)
  finally show *?thesis* **by** (*rule addsI*)
qed

lemma *cone-mono-1*:

assumes $h' \in P[U]$
shows *cone* $(h' * h,\ U) \subseteq$ *cone* $(h,\ U)$
proof
fix p
assume $p \in$ *cone* $(h' * h,\ U)$
then obtain a' **where** $a' \in P[U]$ **and** $p = a' * (h' * h)$ **by** (*rule coneE*)
from *this*(*2*) **have** $p = a' * h' * h$ **by** (*simp only*: *mult.assoc*)
moreover from ‹$a' \in P[U]$› *assms* **have** $a' * h' \in P[U]$ **by** (*rule Polys-closed-times*)
ultimately show $p \in$ *cone* $(h,\ U)$ **by** (*rule coneI*)
qed

lemma *cone-mono-2*:
assumes *U1* $\subseteq$ *U2*
shows *cone* (*h*, *U1*) $\subseteq$ *cone* (*h*, *U2*)
proof
from *assms* **have** *P*[*U1*] $\subseteq$ *P*[*U2*] **by** (*rule Polys-mono*)
fix p
assume $p \in$ *cone* (*h*, *U1*)
then obtain a **where** $a \in$ *P*[*U1*] **and** $p = a * h$ **by** (*rule coneE*)
note *this*(*2*)
moreover from ‹$a \in$ *P*[*U1*]› ‹*P*[*U1*] $\subseteq$ *P*[*U2*]› **have** $a \in$ *P*[*U2*] **..**
ultimately show $p \in$ *cone* (*h*, *U2*) **by** (*rule coneI*)
qed

lemma *cone-subsetD*:
assumes *cone* (*h1*, *U1*) $\subseteq$ *cone* (*h2*::- $\Rightarrow_0$ *'a*::{*comm-ring-1*,*ring-no-zero-divisors*}, *U2*)
shows *h2 dvd h1* **and** *h1* $\neq$ *0* $\Longrightarrow$ *U1* $\subseteq$ *U2*
proof −
from *tip-in-cone assms* **have** *h1* $\in$ *cone* (*h2*, *U2*) **..**
then obtain *a1'* **where** *a1'* $\in$ *P*[*U2*] **and** *h1*: *h1* = *a1'* $*$ *h2* **by** (*rule coneE*)
from *this*(*2*) **have** *h1* = *h2* $*$ *a1'* **by** (*simp only*: *mult.commute*)
thus *h2 dvd h1* **..**

assume *h1* $\neq$ *0*
with *h1* **have** *a1'* $\neq$ *0* **and** *h2* $\neq$ *0* **by** *auto*
show *U1* $\subseteq$ *U2*
proof
fix x
assume $x \in$ *U1*
hence *monomial* (*1*::*'a*) (*Poly-Mapping.single x 1*) $\in$ *P*[*U1*] (**is** *?p* $\in$ -)
by (*intro Polys-closed-monomial PPs-closed-single*)
with *refl* **have** *?p* $*$ *h1* $\in$ *cone* (*h1*, *U1*) **by** (*rule coneI*)
hence *?p* $*$ *h1* $\in$ *cone* (*h2*, *U2*) **using** *assms* **..**
then obtain a **where** $a \in$ *P*[*U2*] **and** *?p* $*$ *h1* = a $*$ *h2* **by** (*rule coneE*)
from *this*(*2*) **have** (*?p* $*$ *a1'*) $*$ *h2* = a $*$ *h2* **by** (*simp only*: *h1 ac-simps*)
hence *?p* $*$ *a1'* = a **using** ‹*h2* $\neq$ *0*› **by** (*rule times-canc-right*)
with ‹$a \in$ *P*[*U2*]› **have** *a1'* $*$ *?p* $\in$ *P*[*U2*] **by** (*simp add*: *mult.commute*)
hence *?p* $\in$ *P*[*U2*] **using** ‹*a1'* $\in$ *P*[*U2*]› ‹*a1'* $\neq$ *0*› **by** (*rule times-in-PolysD*)

thus $x \in U2$ **by** (*simp add*: *Polys-def PPs-def*)
qed
qed

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

assume $h \neq 0$
show $U \subseteq X$
proof
fix x
assume $x \in U$
hence *monomial* (1::$'a$) (*Poly-Mapping.single* x 1) $\in P[U]$ (**is** $?p \in$ -)
by (*intro Polys-closed-monomial PPs-closed-single*)
with *refl* **have** $?p * h \in$ *cone* (h, U) **by** (*rule coneI*)
hence $?p * h \in P[X]$ **using** *assms* **..**
hence $h * ?p \in P[X]$ **by** (*simp only*: *mult.commute*)
hence $?p \in P[X]$ **using** ‹$h \in P[X]$› ‹$h \neq 0$› **by** (*rule times-in-PolysD*)
thus $x \in X$ **by** (*simp add*: *Polys-def PPs-def*)
qed
qed

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 (*cases* $h = 0$)
case *True*
thus *?thesis* **by** (*simp add*: *zero-in-Polys*)
next
case *False*
hence $U \subseteq X$ **by** (*rule assms*(*2*))
hence $P[U] \subseteq P[X]$ **by** (*rule Polys-mono*)
show *?thesis*
proof
fix a
assume $a \in$ *cone* (h, U)
then obtain q **where** $q \in P[U]$ **and** a: $a = q * h$ **by** (*rule coneE*)
from *this*(*1*) ‹$P[U] \subseteq P[X]$› **have** $q \in P[X]$ **..**
from *this assms*(*1*) **show** $a \in P[X]$ **unfolding** a **by** (*rule Polys-closed-times*)
qed
qed

lemma *cone-image-times*: ($*$) a ‘ *cone* (h, U) = *cone* ($a * h$, U)
by (*auto simp*: *ac-simps image-image intro*!: *image-eqI coneI elim*!: *coneE*)

lemma *cone-image-times'*: $(*)$ *a* ' *cone hU* = *cone* (*apfst* ((*) *a*) *hU*)
proof −
  obtain *h U* **where** *hU* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
  thus *?thesis* **by** (*simp add*: *cone-image-times*)
qed

lemma *homogeneous-set-coneI*:
  assumes *homogeneous h*
  shows *homogeneous-set* (*cone* (*h*, *U*))
proof (*rule homogeneous-setI*)
  fix *a n*
  assume *a* ∈ *cone* (*h*, *U*)
  then obtain *q* **where** *q* ∈ *P*[*U*] **and** *a*: *a* = *q* ∗ *h* **by** (*rule coneE*)
  from *this*(*1*) **show** *hom-component a n* ∈ *cone* (*h*, *U*) **unfolding** *a*
  proof (*induct q rule*: *poly-mapping-plus-induct*)
    case *1*
    show *?case* **by** (*simp add*: *zero-in-cone*)
  next
    case (*2 p c t*)
    have *p* ∈ *P*[*U*]
    proof (*intro PolysI subsetI*)
      fix *s*
      assume *s* ∈ *keys p*
      moreover from *2*(*2*) *this* **have** *s* ∉ *keys* (*monomial c t*) **by** *auto*
      ultimately have *s* ∈ *keys* (*monomial c t* + *p*) **by** (*rule in-keys-plusI2*)
      also from *2*(*4*) **have** ... ⊆ .[*U*] **by** (*rule PolysD*)
      finally show *s* ∈ .[*U*] **.**
    qed
    hence ∗: *hom-component* (*p* ∗ *h*) *n* ∈ *cone* (*h*, *U*) **by** (*rule 2*(*3*))
    from *2*(*1*) **have** *t* ∈ *keys* (*monomial c t*) **by** *simp*
    hence *t* ∈ *keys* (*monomial c t* + *p*) **using** *2*(*2*) **by** (*rule in-keys-plusI1*)
    also from *2*(*4*) **have** ... ⊆ .[*U*] **by** (*rule PolysD*)
    finally have *monomial c t* ∈ *P*[*U*] **by** (*rule Polys-closed-monomial*)
    with *refl* **have** *monomial c t* ∗ *h* ∈ *cone* (*h*, *U*) (**is** *?h* ∈ -) **by** (*rule coneI*)
    from *assms* **have** *homogeneous ?h* **by** (*simp add*: *homogeneous-times*)
    hence *hom-component ?h n* = (*?h when n* = *poly-deg ?h*) **by** (*rule hom-component-of-homogeneous*)
    with ‹*?h* ∈ *cone* (*h*, *U*)› **have** ∗∗: *hom-component ?h n* ∈ *cone* (*h*, *U*)
      by (*simp add*: *when-def zero-in-cone*)
    have *hom-component* ((*monomial c t* + *p*) ∗ *h*) *n* = *hom-component ?h n* + *hom-component* (*p* ∗ *h*) *n*
      by (*simp only*: *distrib-right hom-component-plus*)
    also from ∗∗ ∗ **have** ... ∈ *cone* (*h*, *U*) **by** (*rule cone-closed-plus*)
    finally show *?case* **.**
  qed
qed

lemma *subspace-cone*: *phull.subspace* (*cone hU*)
  using *zero-in-cone cone-closed-plus*
proof (*rule phull.subspaceI*)

fix *c a*
assume $a \in$ *cone hU*
moreover obtain *h U* **where** *hU*: $hU = (h,\ U)$ **using** *prod.exhaust* **by** *blast*
ultimately have $a \in$ *cone* $(h,\ U)$ **by** *simp*
thus $c \cdot a \in$ *cone hU* **unfolding** *hU punit.map-scale-eq-monom-mult* **using** *zero-in-PPs*
by (*rule cone-closed-monom-mult*)
qed

lemma *direct-decomp-cone-insert*:
fixes *h* :: - $\Rightarrow_0$ *'a*::{*comm-ring-1*,*ring-no-zero-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 −
let *?x* = *Poly-Mapping.single x* (*Suc 0*)
define *xx* **where** *xx* = *monomial* (*1*::*'a*) *?x*
show *direct-decomp* (*cone* (*h*, *insert x U*)) [*cone* (*h*, *U*), *cone* (*xx* $*$ *h*, *insert x U*)]
(**is** *direct-decomp* - *?ss*)
proof (*rule direct-decompI-alt*)
fix *qs*
assume $qs \in$ *listset ?ss*
then obtain *a b* **where** $a \in$ *cone* (*h*, *U*) **and** *b*: $b \in$ *cone* (*xx* $*$ *h*, *insert x U*)
and *qs*: *qs* = [*a*, *b*] **by** (*rule listset-doubletonE*)
note *this*(*1*)
also have *cone* (*h*, *U*) $\subseteq$ *cone* (*h*, *insert x U*) **by** (*rule cone-mono-2*) *blast*
finally have *a*: $a \in$ *cone* (*h*, *insert x U*) **.**
have *cone* (*xx* $*$ *h*, *insert x U*) $\subseteq$ *cone* (*h*, *insert x U*)
by (*rule cone-mono-1*) (*simp add*: *xx-def Polys-def PPs-closed-single*)
with *b* **have** $b \in$ *cone* (*h*, *insert x U*) **..**
with *a* **have** $a + b \in$ *cone* (*h*, *insert x U*) **by** (*rule cone-closed-plus*)
thus *sum-list qs* $\in$ *cone* (*h*, *insert x U*) **by** (*simp add*: *qs*)
next
fix *a*
assume $a \in$ *cone* (*h*, *insert x U*)
then obtain *q* **where** $q \in P$[*insert x U*] **and** *a*: $a = q * h$ **by** (*rule coneE*)
define *qU* **where** *qU* = *except q* (− .[*U*])
define *qx* **where** *qx* = *except q* .[*U*]
have *q*: $q = qU + qx$ **by** (*simp only*: *qU-def qx-def add.commute flip*: *except-decomp*)
have $qU \in P[U]$ **by** (*rule PolysI*) (*simp add*: *qU-def keys-except*)
have *x-adds*: *?x adds t* **if** $t \in$ *keys qx* **for** *t* **unfolding** *adds-poly-mapping le-fun-def*
proof
fix *y*
show *lookup ?x y* $\leq$ *lookup t y*
proof (*cases* $y = x$)

case *True*
from *that* **have** $t \in keys\ q$ **and** $t \notin .[U]$ **by** (*simp-all add*: *qx-def keys-except*)
from ‹$q \in P[insert\ x\ U]$› **have** $keys\ q \subseteq .[insert\ x\ U]$ **by** (*rule PolysD*)
with ‹$t \in keys\ q$› **have** $t \in .[insert\ x\ U]$ **..**
hence $keys\ t \subseteq insert\ x\ U$ **by** (*rule PPsD*)
moreover from ‹$t \notin .[U]$› **have** $\neg\ keys\ t \subseteq U$ **by** (*simp add*: *PPs-def*)
ultimately have $x \in keys\ t$ **by** *blast*
thus *?thesis* **by** (*simp add*: *lookup-single True in-keys-iff*)
next
case *False*
thus *?thesis* **by** (*simp add*: *lookup-single*)
qed
qed
define qx' **where** $qx' = Poly\text{-}Mapping.map\text{-}key\ ((+)\ ?x)\ qx$
have *lookup-qx'*: $lookup\ qx' = (\lambda t.\ lookup\ qx\ (?x + t))$
by (*rule ext*) (*simp add*: *qx'-def map-key.rep-eq*)
have $qx' * xx = punit.monom\text{-}mult\ 1\ ?x\ qx'$
by (*simp only*: *xx-def mult.commute flip*: *times-monomial-left*)
also have $\ldots = qx$
by (*auto simp*: *punit.lookup-monom-mult lookup-qx' add.commute*[*of ?x*] *adds-minus*
simp flip: *not-in-keys-iff-lookup-eq-zero dest*: *x-adds intro*!: *poly-mapping-eqI*)
finally have *qx*: $qx = qx' * xx$ **by** (*rule sym*)
have $qx' \in P[insert\ x\ U]$
proof (*intro PolysI subsetI*)
fix t
assume $t \in keys\ qx'$
hence $t + ?x \in keys\ qx$ **by** (*simp only*: *lookup-qx' in-keys-iff not-False-eq-True add.commute*)
also have $\ldots \subseteq keys\ q$ **by** (*auto simp*: *qx-def keys-except*)
also from ‹$q \in P[insert\ x\ U]$› **have** $\ldots \subseteq .[insert\ x\ U]$ **by** (*rule PolysD*)
finally have $(t + ?x) - ?x \in .[insert\ x\ U]$ **by** (*rule PPs-closed-minus*)
thus $t \in .[insert\ x\ U]$ **by** *simp*
qed
define *qs* **where** $qs = [qU * h,\ qx' * (xx * h)]$
show $\exists!qs{\in}listset\ ?ss.\ a = sum\text{-}list\ qs$
proof (*intro ex1I conjI allI impI*)
from *refl* ‹$qU \in P[U]$› **have** $qU * h \in cone\ (h,\ U)$ **by** (*rule coneI*)
moreover from *refl* ‹$qx' \in P[insert\ x\ U]$› **have** $qx' * (xx * h) \in cone\ (xx * h,\ insert\ x\ U)$
by (*rule coneI*)
ultimately show $qs \in listset\ ?ss$ **using** *qs-def* **by** (*rule listset-doubletonI*)
next
fix *qs0*
assume $qs0 \in listset\ ?ss \wedge a = sum\text{-}list\ qs0$
hence $qs0 \in listset\ ?ss$ **and** *a0*: $a = sum\text{-}list\ qs0$ **by** *simp-all*
from *this*(*1*) **obtain** *p1 p2* **where** $p1 \in cone\ (h,\ U)$ **and** *p2*: $p2 \in cone\ (xx * h,\ insert\ x\ U)$
and *qs0*: $qs0 = [p1,\ p2]$ **by** (*rule listset-doubletonE*)

from *this*(*1*) **obtain** *qU0* **where** *qU0* ∈ *P*[*U*] **and** *p1*: *p1* = *qU0* ∗ *h* **by** (*rule coneE*)
from *p2* **obtain** *qx0* **where** *p2*: *p2* = *qx0* ∗ (*xx* ∗ *h*) **by** (*rule coneE*)
show *qs0* = *qs*
proof (*cases h* = *0*)
case *True*
thus *?thesis* **by** (*simp add*: *qs-def qs0 p1 p2*)
next
case *False*
from *a0* **have** (*qU* − *qU0*) ∗ *h* = (*qx0* − *qx*′) ∗ *xx* ∗ *h*
by (*simp add*: *a qs0 p1 p2 q qx algebra-simps*)
hence *eq*: *qU* − *qU0* = (*qx0* − *qx*′) ∗ *xx* **using** *False* **by** (*rule times-canc-right*)
have *qx0* = *qx*′
proof (*rule ccontr*)
assume *qx0* ≠ *qx*′
hence *qx0* − *qx*′ ≠ *0* **by** *simp*
moreover have *xx* ≠ *0* **by** (*simp add*: *xx-def monomial-0-iff*)
ultimately have *lpp* ((*qx0* − *qx*′) ∗ *xx*) = *lpp* (*qx0* − *qx*′) + *lpp xx*
by (*rule lp-times*)
also have *lpp xx* = *?x* **by** (*simp add*: *xx-def punit.lt-monomial*)
finally have *?x adds lpp* (*qU* − *qU0*) **by** (*simp add*: *eq*)
hence *lookup ?x x* ≤ *lookup* (*lpp* (*qU* − *qU0*)) *x* **by** (*simp only*: *adds-poly-mapping le-fun-def*)
hence *x* ∈ *keys* (*lpp* (*qU* − *qU0*)) **by** (*simp add*: *in-keys-iff lookup-single*)
moreover have *lpp* (*qU* − *qU0*) ∈ *keys* (*qU* − *qU0*)
proof (*rule punit.lt-in-keys*)
from ‹*qx0* − *qx*′ ≠ *0*› ‹*xx* ≠ *0*› **show** *qU* − *qU0* ≠ *0* **unfolding** *eq* **by** (*rule times-not-zero*)
qed
ultimately have *x* ∈ *indets* (*qU* − *qU0*) **by** (*rule in-indetsI*)
from ‹*qU* ∈ *P*[*U*]› ‹*qU0* ∈ *P*[*U*]› **have** *qU* − *qU0* ∈ *P*[*U*] **by** (*rule Polys-closed-minus*)
hence *indets* (*qU* − *qU0*) ⊆ *U* **by** (*rule PolysD*)
with ‹*x* ∈ *indets* (*qU* − *qU0*)› **have** *x* ∈ *U* **..**
with *assms* **show** *False* **..**
qed
moreover from *this eq* **have** *qU0* = *qU* **by** *simp*
ultimately show *?thesis* **by** (*simp only*: *qs-def qs0 p1 p2*)
qed
qed (*simp-all add*: *qs-def a q qx*, *simp only*: *algebra-simps*)
qed
qed

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

definition *hom-decomp* :: ((('x $\Rightarrow_0$ nat) $\Rightarrow_0$ 'a::{one,zero}) $\times$ 'x set) list $\Rightarrow$ bool
where *hom-decomp ps* $\longleftrightarrow$ ($\forall$ *hU*$\in$*set ps. homogeneous* (*fst hU*))

definition *cone-decomp* :: (('x $\Rightarrow_0$ nat) $\Rightarrow_0$ 'a) set $\Rightarrow$
((('x $\Rightarrow_0$ nat) $\Rightarrow_0$ 'a::comm-semiring-0) $\times$ 'x set) list $\Rightarrow$ bool
where *cone-decomp T ps* $\longleftrightarrow$ *direct-decomp T* (*map cone ps*)

lemma *valid-decompI*:
($\bigwedge$*h U.* (*h, U*) $\in$ *set ps* $\Longrightarrow$ *h* $\in$ *P*[*X*]) $\Longrightarrow$ ($\bigwedge$*h U.* (*h, U*) $\in$ *set ps* $\Longrightarrow$ *h* $\neq$ *0*) $\Longrightarrow$
($\bigwedge$*h U.* (*h, U*) $\in$ *set ps* $\Longrightarrow$ *U* $\subseteq$ *X*) $\Longrightarrow$ *valid-decomp X ps*
unfolding *valid-decomp-def* **by** *blast*

lemma *valid-decompD*:
assumes *valid-decomp X ps* **and** (*h, U*) $\in$ *set ps*
shows *h* $\in$ *P*[*X*] **and** *h* $\neq$ *0* **and** *U* $\subseteq$ *X*
using *assms* **unfolding** *valid-decomp-def* **by** *blast+*

lemma *valid-decompD-finite*:
assumes *finite X* **and** *valid-decomp X ps* **and** (*h, U*) $\in$ *set ps*
shows *finite U*
proof −
from *assms*(*2, 3*) **have** *U* $\subseteq$ *X* **by** (*rule valid-decompD*)
thus *?thesis* **using** *assms*(*1*) **by** (*rule finite-subset*)
qed

lemma *valid-decomp-Nil*: *valid-decomp X* []
by (*simp add*: *valid-decomp-def*)

lemma *valid-decomp-concat*:
assumes $\bigwedge$*ps. ps* $\in$ *set pss* $\Longrightarrow$ *valid-decomp X ps*
shows *valid-decomp X* (*concat pss*)
proof (*rule valid-decompI*)
fix *h U*
assume (*h, U*) $\in$ *set* (*concat pss*)
then obtain *ps* **where** *ps* $\in$ *set pss* **and** (*h, U*) $\in$ *set ps* **unfolding** *set-concat* **..**
from *this*(*1*) **have** *valid-decomp X ps* **by** (*rule assms*)
thus *h* $\in$ *P*[*X*] **and** *h* $\neq$ *0* **and** *U* $\subseteq$ *X* **using** ‹(*h, U*) $\in$ *set ps*› **by** (*rule valid-decompD*)+
qed

corollary *valid-decomp-append*:
assumes *valid-decomp X ps* **and** *valid-decomp X qs*
shows *valid-decomp X* (*ps* @ *qs*)
proof −

have *valid-decomp X* (*concat* [*ps*, *qs*]) **by** (*rule valid-decomp-concat*) (*auto simp*: *assms*)
thus *?thesis* **by** *simp*
qed

lemma *valid-decomp-map-times*:
assumes *valid-decomp X ps* **and** $s \in P[X]$ **and** $s \neq (0::\text{-} \Rightarrow_0 \text{-}::semiring\text{-}no\text{-}zero\text{-}divisors)$
shows *valid-decomp X* (*map* (*apfst* ((∗) *s*)) *ps*)
proof (*rule valid-decompI*)
fix *h U*
assume (*h*, *U*) ∈ *set* (*map* (*apfst* ((∗) *s*)) *ps*)
then obtain *x* **where** *x* ∈ *set ps* **and** (*h*, *U*) = *apfst* ((∗) *s*) *x* **unfolding** *set-map* **..**
moreover obtain *a b* **where** *x* = (*a*, *b*) **using** *prod.exhaust* **by** *blast*
ultimately have *h*: *h* = *s* ∗ *a* **and** (*a*, *U*) ∈ *set ps* **by** *simp-all*
from *assms*(*1*) *this*(*2*) **have** $a \in P[X]$ **and** $a \neq 0$ **and** $U \subseteq X$ **by** (*rule valid-decompD*)+
from *assms*(*2*) *this*(*1*) **show** $h \in P[X]$ **unfolding** *h* **by** (*rule Polys-closed-times*)
from *assms*(*3*) ‹$a \neq 0$› **show** $h \neq 0$ **unfolding** *h* **by** (*rule times-not-zero*)
from ‹$U \subseteq X$› **show** $U \subseteq X$ **.**
qed

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
by (*auto simp*: *monomial-decomp-def*)

lemma *monomial-decompD*:
assumes *monomial-decomp ps* **and** (*h*, *U*) ∈ *set ps*
shows *is-monomial h* **and** *punit.lc h* = *1*
using *assms* **by** (*auto simp*: *monomial-decomp-def*)

lemma *monomial-decomp-append-iff*:
monomial-decomp (*ps* @ *qs*) ⟷ *monomial-decomp ps* ∧ *monomial-decomp qs*
by (*auto simp*: *monomial-decomp-def*)

lemma *monomial-decomp-concat*:
(⋀*ps*. *ps* ∈ *set pss* ⟹ *monomial-decomp ps*) ⟹ *monomial-decomp* (*concat pss*)
by (*induct pss*) (*auto simp*: *monomial-decomp-def*)

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 (*rule monomial-decompI*)
fix *h U*
assume (*h*, *U*) ∈ *set* (*map* (*apfst* ((∗) *f*)) *ps*)
then obtain *x* **where** *x* ∈ *set ps* **and** (*h*, *U*) = *apfst* ((∗) *f*) *x* **unfolding** *set-map* **..**

moreover obtain *a b* **where** *x* = (*a*, *b*) **using** *prod.exhaust* **by** *blast*
ultimately have *h*: *h* = *f* ∗ *a* **and** (*a*, *U*) ∈ *set ps* **by** *simp-all*
from *assms*(*1*) *this*(*2*) **have** *is-monomial a* **and** *punit.lc a* = *1* **by** (*rule monomial-decompD*)+
from *this*(*1*) **have** *monomial* (*punit.lc a*) (*lpp a*) = *a* **by** (*rule punit.monomial-eq-itself*)
moreover define *t* **where** *t* = *lpp a*
ultimately have *a*: *a* = *monomial 1 t* **by** (*simp only*: ‹*punit.lc a* = *1*›)
from *assms*(*2*) **have** *monomial* (*punit.lc f*) (*lpp f*) = *f* **by** (*rule punit.monomial-eq-itself*)
moreover define *s* **where** *s* = *lpp f*
ultimately have *f*: *f* = *monomial 1 s* **by** (*simp only*: *assms*(*3*))
show *is-monomial h* **by** (*simp add*: *h a f times-monomial-monomial monomial-is-monomial*)
show *punit.lc h* = *1* **by** (*simp add*: *h a f times-monomial-monomial*)
qed

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 (*cases t* ∈ *keys a*)
case *True*
obtain *h U* **where** *hU*: *hU* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
with *assms*(*2*) **have** (*h*, *U*) ∈ *set ps* **by** *simp*
with *assms*(*1*) **have** *is-monomial h* **by** (*rule monomial-decompD*)
then obtain *c s* **where** *h*: *h* = *monomial c s* **by** (*rule is-monomial-monomial*)
from *assms*(*3*) **obtain** *q* **where** *q* ∈ *P*[*U*] **and** *a* = *q* ∗ *h* **unfolding** *hU* **by** (*rule coneE*)
from *this*(*2*) **have** *a* = *h* ∗ *q* **by** (*simp only*: *mult.commute*)
also have ... = *punit.monom-mult c s q* **by** (*simp only*: *h times-monomial-left*)
finally have *a*: *a* = *punit.monom-mult c s q* **.**
with *True* **have** *t* ∈ *keys* (*punit.monom-mult c s q*) **by** *simp*
hence *t* ∈ (+) *s* ‘ *keys q* **using** *punit.keys-monom-mult-subset*[*simplified*] **..**
then obtain *u* **where** *u* ∈ *keys q* **and** *t*: *t* = *s* + *u* **..**
note *this*(*1*)
also from ‹*q* ∈ *P*[*U*]› **have** *keys q* ⊆ .[*U*] **by** (*rule PolysD*)
finally have *u* ∈ .[*U*] **.**
have *monomial* (*lookup a t*) *t* = *monomial* (*lookup q u*) *u* ∗ *h*
by (*simp add*: *a t punit.lookup-monom-mult h times-monomial-monomial mult.commute*)
moreover from ‹*u* ∈ .[*U*]› **have** *monomial* (*lookup q u*) *u* ∈ *P*[*U*] **by** (*rule Polys-closed-monomial*)
ultimately show *?thesis* **unfolding** *hU* **by** (*rule coneI*)
next
case *False*
thus *?thesis* **by** (*simp add*: *zero-in-cone in-keys-iff*)
qed

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 −
  from *assms*(*2*) **obtain** *qs* **where** *qs-in*: $qs \in$ *listset* (*map cone ps*) **and** *a*: $a =$ *sum-list qs* **..**
  from *assms*(*3*) *keys-sum-list-subset* **have** $t \in$ *Keys* (*set qs*) **unfolding** *a* **..**
  then obtain *q* **where** $q \in$ *set qs* **and** $t \in$ *keys q* **by** (*rule in-KeysE*)
  from *this*(*1*) **obtain** *i* **where** $i <$ *length qs* **and** *q*: $q = qs$! *i* **by** (*metis in-set-conv-nth*)
  moreover from *qs-in* **have** *length qs* = *length* (*map cone ps*) **by** (*rule listsetD*)
  ultimately have $i <$ *length* (*map cone ps*) **by** *simp*
  moreover from *qs-in this* **have** *qs* ! $i \in$ (*map cone ps*) ! *i* **by** (*rule listsetD*)
  ultimately have *ps* ! $i \in$ *set ps* **and** $q \in$ *cone* (*ps* ! *i*) **by** (*simp-all add*: *q*)
  with *assms*(*1*) **have** ∗: *monomial* (*lookup q t*) $t \in$ *cone* (*ps* ! *i*)
    by (*rule monomial-decomp-monomial-in-cone*)
  obtain *h U* **where** *psi*: *ps* ! $i = (h, U)$ **using** *prod.exhaust* **by** *blast*
  show *?thesis*
  proof
    from ‹*ps* ! $i \in$ *set ps*› **show** $(h, U) \in$ *set ps* **by** (*simp only*: *psi*)
  next
    from ‹$t \in$ *keys q*› **show** *lookup q t* $\neq$ *0* **by** (*simp add*: *in-keys-iff*)
  next
    from ∗ **show** *monomial* (*lookup q t*) $t \in$ *cone* (h, U) **by** (*simp only*: *psi*)
  qed
qed

lemma *hom-decompI*: ($\bigwedge h\ U.\ (h, U) \in$ *set ps* $\Longrightarrow$ *homogeneous h*) $\Longrightarrow$ *hom-decomp ps*
  by (*auto simp*: *hom-decomp-def*)

lemma *hom-decompD*: *hom-decomp ps* $\Longrightarrow$ $(h, U) \in$ *set ps* $\Longrightarrow$ *homogeneous h*
  by (*auto simp*: *hom-decomp-def*)

lemma *hom-decomp-append-iff*: *hom-decomp* (*ps* @ *qs*) $\longleftrightarrow$ *hom-decomp ps* $\wedge$ *hom-decomp qs*
  by (*auto simp*: *hom-decomp-def*)

lemma *hom-decomp-concat*: ($\bigwedge ps.\ ps \in$ *set pss* $\Longrightarrow$ *hom-decomp ps*) $\Longrightarrow$ *hom-decomp* (*concat pss*)
  by (*induct pss*) (*auto simp*: *hom-decomp-def*)

lemma *hom-decomp-map-times*:
  assumes *hom-decomp ps* **and** *homogeneous f*
  shows *hom-decomp* (*map* (*apfst* ((∗) *f*)) *ps*)
proof (*rule hom-decompI*)
  fix *h U*
  assume $(h, U) \in$ *set* (*map* (*apfst* ((∗) *f*)) *ps*)
  then obtain *x* **where** $x \in$ *set ps* **and** $(h, U) =$ *apfst* ((∗) *f*) *x* **unfolding** *set-map* **..**
  moreover obtain *a b* **where** $x = (a, b)$ **using** *prod.exhaust* **by** *blast*

ultimately have *h*: $h = f * a$ **and** $(a,\ U) \in$ *set ps* **by** *simp-all*
from *assms*(*1*) *this*(*2*) **have** *homogeneous a* **by** (*rule hom-decompD*)
with *assms*(*2*) **show** *homogeneous h* **unfolding** *h* **by** (*rule homogeneous-times*)
qed

lemma *monomial-decomp-imp-hom-decomp*:
assumes *monomial-decomp ps*
shows *hom-decomp ps*
proof (*rule hom-decompI*)
fix *h U*
assume $(h,\ U) \in$ *set ps*
with *assms* **have** *is-monomial h* **by** (*rule monomial-decompD*)
then obtain *c t* **where** *h*: $h =$ *monomial c t* **by** (*rule is-monomial-monomial*)
show *homogeneous h* **unfolding** *h* **by** (*fact homogeneous-monomial*)
qed

lemma *cone-decompI*: *direct-decomp T* (*map cone ps*) $\Longrightarrow$ *cone-decomp T ps*
unfolding *cone-decomp-def* **by** *blast*

lemma *cone-decompD*: *cone-decomp T ps* $\Longrightarrow$ *direct-decomp T* (*map cone ps*)
unfolding *cone-decomp-def* **by** *blast*

lemma *cone-decomp-cone-subset*:
assumes *cone-decomp T ps* **and** $hU \in$ *set ps*
shows *cone hU* $\subseteq T$
proof
fix *p*
assume $p \in$ *cone hU*
from *assms*(*2*) **obtain** *i* **where** $i <$ *length ps* **and** *hU*: $hU = ps\ !\ i$ **by** (*metis in-set-conv-nth*)
define *qs* **where** $qs = (map\ 0\ ps)[i := p]$
have *sum-list qs* $\in T$
proof (*intro direct-decompD listsetI*)
from *assms*(*1*) **show** *direct-decomp T* (*map cone ps*) **by** (*rule cone-decompD*)
next
fix *j*
assume $j <$ *length* (*map cone ps*)
with ‹$i <$ *length ps*› ‹$p \in$ *cone hU*› **show** $qs\ !\ j \in$ *map cone ps* $!\ j$
by (*auto simp*: *qs-def nth-list-update zero-in-cone hU*)
qed (*simp add*: *qs-def*)
also have *sum-list qs* $= qs\ !\ i$ **by** (*rule sum-list-eq-nthI*) (*simp-all add*: *qs-def* ‹$i < $ *length ps*›)
also from ‹$i <$ *length ps*› **have** $\ldots = p$ **by** (*simp add*: *qs-def*)
finally show $p \in T$ **.**
qed

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 −
  from *assms*(*1*, *3*) **have** *cone* (*h*, *U*) ⊆ *T* **by** (*rule cone-decomp-cone-subset*)
  hence *cone* (*h*, *U*) ⊆ *P*[*X*] **using** *assms*(*2*) **by** (*rule subset-trans*)
  thus *h* ∈ *P*[*X*] **and** *h* ≠ *0* ⟹ *U* ⊆ *X* **by** (*rule cone-subset-PolysD*)+
qed

lemma *cone-decomp-closed-plus*:
  assumes *cone-decomp T ps* **and** *a* ∈ *T* **and** *b* ∈ *T*
  shows *a* + *b* ∈ *T*
proof −
  from *assms*(*1*) **have** *dd*: *direct-decomp T* (*map cone ps*) **by** (*rule cone-decompD*)
  then obtain *qsa* **where** *qsa*: *qsa* ∈ *listset* (*map cone ps*) **and** *a*: *a* = *sum-list qsa* **using** *assms*(*2*)
    by (*rule direct-decompE*)
  from *dd assms*(*3*) **obtain** *qsb* **where** *qsb*: *qsb* ∈ *listset* (*map cone ps*) **and** *b*: *b* = *sum-list qsb*
    by (*rule direct-decompE*)
  from *qsa* **have** *length qsa* = *length* (*map cone ps*) **by** (*rule listsetD*)
  moreover from *qsb* **have** *length qsb* = *length* (*map cone ps*) **by** (*rule listsetD*)
  ultimately have *a* + *b* = *sum-list* (*map2* (+) *qsa qsb*) **by** (*simp only*: *sum-list-map2-plus a b*)
  also from *dd* **have** *sum-list* (*map2* (+) *qsa qsb*) ∈ *T*
  proof (*rule direct-decompD*)
    from *qsa qsb* **show** *map2* (+) *qsa qsb* ∈ *listset* (*map cone ps*)
    proof (*rule listset-closed-map2*)
      fix *c p1 p2*
      assume *c* ∈ *set* (*map cone ps*)
      then obtain *hU* **where** *c*: *c* = *cone hU* **by** *auto*
      assume *p1* ∈ *c* **and** *p2* ∈ *c*
      thus *p1* + *p2* ∈ *c* **unfolding** *c* **by** (*rule cone-closed-plus*)
    qed
  qed
  finally show *?thesis* **.**
qed

lemma *cone-decomp-closed-uminus*:
  assumes *cone-decomp T ps* **and** (*a*::- $\Rightarrow_0$ -::*comm-ring*) ∈ *T*
  shows − *a* ∈ *T*
proof −
  from *assms*(*1*) **have** *dd*: *direct-decomp T* (*map cone ps*) **by** (*rule cone-decompD*)
  then obtain *qsa* **where** *qsa*: *qsa* ∈ *listset* (*map cone ps*) **and** *a*: *a* = *sum-list qsa* **using** *assms*(*2*)
    by (*rule direct-decompE*)
  from *qsa* **have** *length qsa* = *length* (*map cone ps*) **by** (*rule listsetD*)
  have − *a* = *sum-list* (*map uminus qsa*) **unfolding** *a* **by** (*induct qsa*, *simp-all*)
  also from *dd* **have** . . . ∈ *T*
  proof (*rule direct-decompD*)
    from *qsa* **show** *map uminus qsa* ∈ *listset* (*map cone ps*)

proof (*rule listset-closed-map*)
fix *c p*
assume $c \in$ *set* (*map cone ps*)
then obtain *hU* **where** *c*: $c =$ *cone hU* **by** *auto*
assume $p \in c$
thus $- p \in c$ **unfolding** *c* **by** (*rule cone-closed-uminus*)
qed
qed
finally show *?thesis* **.**
qed

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 −
from *assms*(*1*, *3*) **have** $- b \in T$ **by** (*rule cone-decomp-closed-uminus*)
with *assms*(*1*, *2*) **have** $a + (- b) \in T$ **by** (*rule cone-decomp-closed-plus*)
thus *?thesis* **by** *simp*
qed

lemma *cone-decomp-Nil*: *cone-decomp* {*0*} []
by (*auto simp*: *cone-decomp-def intro*: *direct-decompI-alt*)

lemma *cone-decomp-singleton*: *cone-decomp* (*cone* (*t*, *U*)) [(*t*, *U*)]
by (*simp add*: *cone-decomp-def direct-decomp-singleton*)

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 (*rule cone-decompI*)
from *assms*(*2*) **have** *direct-decomp S1* (*map cone ps*) **by** (*rule cone-decompD*)
with *assms*(*1*) **have** *direct-decomp T* ([*S2*] @ *map cone ps*) **by** (*rule direct-decomp-direct-decomp*)
hence *direct-decomp T* (*S2* # *map cone ps*) **by** *simp*
moreover from *assms*(*3*) **have** *direct-decomp S2* (*map cone qs*) **by** (*rule cone-decompD*)
ultimately have *direct-decomp T* (*map cone ps* @ *map cone qs*) **by** (*intro direct-decomp-direct-decomp*)
thus *direct-decomp T* (*map cone* (*ps* @ *qs*)) **by** *simp*
qed

lemma *cone-decomp-concat*:
assumes *direct-decomp T ss* **and** *length pss* = *length ss*
and $\bigwedge i.\ i <$ *length ss* $\Longrightarrow$ *cone-decomp* (*ss* ! *i*) (*pss* ! *i*)
shows *cone-decomp T* (*concat pss*)
using *assms*(*2*, *1*, *3*)
proof (*induct pss ss arbitrary*: *T rule*: *list-induct2*)
case *Nil*
from *Nil*(*1*) **show** *?case* **by** (*simp add*: *cone-decomp-def*)
next

case (*Cons ps pss s ss*)
have *0* < *length* (*s* # *ss*) **by** *simp*
hence *cone-decomp* ((*s* # *ss*) ! *0*) ((*ps* # *pss*) ! *0*) **by** (*rule Cons.prems*)
hence *cone-decomp s ps* **by** *simp*
hence ∗: *direct-decomp s* (*map cone ps*) **by** (*rule cone-decompD*)
with *Cons.prems*(*1*) **have** *direct-decomp T* (*ss* @ *map cone ps*) **by** (*rule direct-decomp-direct-decomp*)
hence *1*: *direct-decomp T* [*sum-list* ‘ *listset ss*, *sum-list* ‘ *listset* (*map cone ps*)]
and *2*: *direct-decomp* (*sum-list* ‘ *listset ss*) *ss*
by (*auto dest*: *direct-decomp-appendD intro*!: *empty-not-in-map-cone*)
note *1*
moreover from *2* **have** *cone-decomp* (*sum-list* ‘ *listset ss*) (*concat pss*)
proof (*rule Cons.hyps*)
fix *i*
assume *i* < *length ss*
hence *Suc i* < *length* (*s* # *ss*) **by** *simp*
hence *cone-decomp* ((*s* # *ss*) ! *Suc i*) ((*ps* # *pss*) ! *Suc i*) **by** (*rule Cons.prems*)
thus *cone-decomp* (*ss* ! *i*) (*pss* ! *i*) **by** *simp*
qed
moreover have *cone-decomp* (*sum-list* ‘ *listset* (*map cone ps*)) *ps*
proof (*intro cone-decompI direct-decompI refl*)
from ∗ **show** *inj-on sum-list* (*listset* (*map cone ps*))
by (*simp only*: *direct-decomp-def bij-betw-def*)
qed
ultimately have *cone-decomp T* (*concat pss* @ *ps*) **by** (*rule cone-decomp-append*)
hence *direct-decomp T* (*map cone* (*concat pss*) @ *map cone ps*) **by** (*simp add*: *cone-decomp-def*)
hence *direct-decomp T* (*map cone ps* @ *map cone* (*concat pss*))
by (*auto intro*: *direct-decomp-perm*)
thus *?case* **by** (*simp add*: *cone-decomp-def*)
qed

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 (*rule cone-decompI*)
from *assms* **have** *direct-decomp T* (*map cone ps*) **by** (*rule cone-decompD*)
hence *direct-decomp* ((∗) *s* ‘ *T*) (*map* ((‘) ((∗) *s*)) (*map cone ps*))
by (*rule direct-decomp-image-times*) (*rule times-canc-left*)
also have *map* ((‘) ((∗) *s*)) (*map cone ps*) = *map cone* (*map* (*apfst* ((∗) *s*)) *ps*)
by (*simp add*: *cone-image-times′*)
finally show *direct-decomp* ((∗) *s* ‘ *T*) (*map cone* (*map* (*apfst* ((∗) *s*)) *ps*)) **.**
qed

lemma *cone-decomp-perm*:
assumes *cone-decomp T ps* **and** *mset ps* = *mset qs*
shows *cone-decomp T qs*
using *assms*(*1*) **unfolding** *cone-decomp-def*

proof (*rule direct-decomp-perm*)
  from ‹*mset ps* = *mset qs*› **show** ‹*mset* (*map cone ps*) = *mset* (*map cone qs*)›
    by *simp*
qed

lemma *valid-cone-decomp-subset-Polys*:
  assumes *valid-decomp X ps* **and** *cone-decomp T ps*
  shows $T \subseteq P[X]$
proof
  fix *p*
  assume $p \in T$
  from *assms*(*2*) **have** *direct-decomp T* (*map cone ps*) **by** (*rule cone-decompD*)
  then obtain *qs* **where** *qs* ∈ *listset* (*map cone ps*) **and** *p*: *p* = *sum-list qs* **using** ‹$p \in T$›
    by (*rule direct-decompE*)
  from *assms*(*1*) *this*(*1*) **show** $p \in P[X]$ **unfolding** *p*
  proof (*induct ps arbitrary*: *qs*)
    case *Nil*
    from *Nil*(*2*) **show** *?case* **by** (*simp add*: *zero-in-Polys*)
  next
    case (*Cons a ps*)
    obtain *h U* **where** *a*: *a* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
    hence (*h*, *U*) ∈ *set* (*a* # *ps*) **by** *simp*
    with *Cons.prems*(*1*) **have** $h \in P[X]$ **and** $U \subseteq X$ **by** (*rule valid-decompD*)+
    hence *cone a* ⊆ $P[X]$ **unfolding** *a* **by** (*rule cone-subset-PolysI*)
    from *Cons.prems*(*1*) **have** *valid-decomp X ps* **by** (*simp add*: *valid-decomp-def*)
    from *Cons.prems*(*2*) **have** *qs* ∈ *listset* (*cone a* # *map cone ps*) **by** *simp*
    then obtain *q qs′* **where** *q* ∈ *cone a* **and** *qs′*: *qs′* ∈ *listset* (*map cone ps*) **and** *qs*: *qs* = *q* # *qs′*
      by (*rule listset-ConsE*)
    from *this*(*1*) ‹*cone a* ⊆ $P[X]$› **have** $q \in P[X]$ **..**
    moreover from ‹*valid-decomp X ps*› *qs′* **have** *sum-list qs′* ∈ $P[X]$ **by** (*rule Cons.hyps*)
    ultimately have *q* + *sum-list qs′* ∈ $P[X]$ **by** (*rule Polys-closed-plus*)
    thus *?case* **by** (*simp add*: *qs*)
  qed
qed

lemma *homogeneous-set-cone-decomp*:
  assumes *cone-decomp T ps* **and** *hom-decomp ps*
  shows *homogeneous-set T*
proof (*rule homogeneous-set-direct-decomp*)
  from *assms*(*1*) **show** *direct-decomp T* (*map cone ps*) **by** (*rule cone-decompD*)
next
  fix *cn*
  assume *cn* ∈ *set* (*map cone ps*)
  then obtain *hU* **where** *hU* ∈ *set ps* **and** *cn*: *cn* = *cone hU* **unfolding** *set-map* **..**
  moreover obtain *h U* **where** *hU*: *hU* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*

ultimately have $(h,\ U) \in$ *set ps* **by** *simp*
with *assms*(*2*) **have** *homogeneous h* **by** (*rule hom-decompD*)
thus *homogeneous-set cn* **unfolding** *cn hU* **by** (*rule homogeneous-set-coneI*)
qed

lemma *subspace-cone-decomp*:
assumes *cone-decomp T ps*
shows *phull.subspace* (T::(- $\Rightarrow_0$ -::*field*) *set*)
proof (*rule phull.subspace-direct-decomp*)
from *assms* **show** *direct-decomp T* (*map cone ps*) **by** (*rule cone-decompD*)
next
fix *cn*
assume *cn* $\in$ *set* (*map cone ps*)
then obtain *hU* **where** *hU* $\in$ *set ps* **and** *cn*: *cn* = *cone hU* **unfolding** *set-map*
..
show *phull.subspace cn* **unfolding** *cn* **by** (*rule subspace-cone*)
qed

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

definition *standard-decomp* :: *nat* $\Rightarrow$ ((($'x \Rightarrow_0$ *nat*) $\Rightarrow_0$ $'a$::*zero*) $\times$ $'x$ *set*) *list* $\Rightarrow$ *bool*
where *standard-decomp k ps* $\longleftrightarrow$ ($\forall$ $(h,\ U)$$\in$*set* (*ps*$_+$). $k \le$ *poly-deg h* $\wedge$
($\forall$ d. $k \le d \longrightarrow d \le$ *poly-deg h* $\longrightarrow$
($\exists$ $(h',\ U')$$\in$*set ps*. *poly-deg h'* = d $\wedge$ *card U* $\le$ *card U'*)))

lemma *pos-decomp-Nil* [*simp*]: $[]_+ = []$
by (*simp add*: *pos-decomp-def*)

lemma *pos-decomp-subset*: *set* (*ps*$_+$) $\subseteq$ *set ps*
by (*simp add*: *pos-decomp-def*)

lemma *pos-decomp-append*: (*ps* @ *qs*)$_+$ = *ps*$_+$ @ *qs*$_+$
by (*simp add*: *pos-decomp-def*)

lemma *pos-decomp-concat*: (*concat pss*)$_+$ = *concat* (*map pos-decomp pss*)
by (*metis* (*mono-tags*, *lifting*) *filter-concat map-eq-conv pos-decomp-def*)

lemma *pos-decomp-map*: (*map* (*apfst f*) *ps*)$_+$ = *map* (*apfst f*) (*ps*$_+$)
by (*metis* (*mono-tags*, *lifting*) *pos-decomp-def filter-cong filter-map o-apply snd-apfst*)

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 $-$
have {h. $(h,\ \{\}) \in$ *set qs* $\wedge$ *P h*} = *fst* ‘ {$(h,\ U) \in$ *set qs* $-$ *set* (*qs*$_+$). *P h*}

by (*auto simp*: *pos-decomp-def image-Collect*)
also have *card* ... = *card* $\{(h, U) \in set\ qs - set\ (qs_+).\ P\ h\}$
by (*rule card-image*, *auto simp*: *pos-decomp-def intro*: *inj-onI*)
finally show *?thesis* **by** (*rule sym*)
qed

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*
unfolding *standard-decomp-def* **using** *assms* **by** *blast*

lemma *standard-decompD*: *standard-decomp k ps* $\Longrightarrow (h, U) \in set\ (ps_+) \Longrightarrow k \leq$ *poly-deg h*
unfolding *standard-decomp-def* **by** *blast*

lemma *standard-decompE*:
assumes *standard-decomp k ps* **and** $(h, U) \in set\ (ps_+)$ **and** $k \leq d$ **and** $d \leq$ *poly-deg h*
obtains $h'\ U'$ **where** $(h', U') \in set\ ps$ **and** *poly-deg* $h' = d$ **and** $card\ U \leq card\ U'$
using *assms* **unfolding** *standard-decomp-def* **by** *blast*

lemma *standard-decomp-Nil*: $ps_+ = [] \Longrightarrow$ *standard-decomp k ps*
by (*simp add*: *standard-decomp-def*)

lemma *standard-decomp-singleton*: *standard-decomp* (*poly-deg h*) $[(h, U)]$
by (*simp add*: *standard-decomp-def pos-decomp-def*)

lemma *standard-decomp-concat*:
assumes $\bigwedge ps.\ ps \in set\ pss \Longrightarrow$ *standard-decomp k ps*
shows *standard-decomp k* (*concat pss*)
proof (*rule standard-decompI*)
fix *h U*
assume $(h, U) \in set\ ((concat\ pss)_+)$
then obtain *ps* **where** $ps \in set\ pss$ **and** $*$: $(h, U) \in set\ (ps_+)$ **by** (*auto simp*: *pos-decomp-concat*)
from *this*(*1*) **have** *standard-decomp k ps* **by** (*rule assms*)
thus $k \leq$ *poly-deg h* **using** $*$ **by** (*rule standard-decompD*)

fix *d*
assume $k \leq d$ **and** $d \leq$ *poly-deg h*
with ‹*standard-decomp k ps*› $*$ **obtain** $h'\ U'$ **where** $(h', U') \in set\ ps$ **and** *poly-deg* $h' = d$
and $card\ U \leq card\ U'$ **by** (*rule standard-decompE*)
note *this*(*2*, *3*)
moreover from ‹$(h', U') \in set\ ps$› ‹$ps \in set\ pss$› **have** $(h', U') \in set\ (concat\ pss)$ **by** *auto*

ultimately show $\exists h'\ U'.\ (h',\ U') \in$ *set* (*concat pss*) $\wedge$ *poly-deg* $h' = d \wedge$ *card* $U \leq$ *card* U'
by *blast*
qed

corollary *standard-decomp-append*:
assumes *standard-decomp k ps* **and** *standard-decomp k qs*
shows *standard-decomp k* (*ps* @ *qs*)
proof −
have *standard-decomp k* (*concat* [*ps*, *qs*]) **by** (*rule standard-decomp-concat*) (*auto simp*: *assms*)
thus *?thesis* **by** *simp*
qed

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-decomp* (k + *poly-deg s*) (*map* (*apfst* ((∗) *s*)) *ps*)
proof (*rule standard-decompI*)
fix *h U*
assume $(h,\ U) \in$ *set* ((*map* (*apfst* ((∗) *s*)) *ps*)$_+$)
then obtain *h0* **where** *1*: $(h0,\ U) \in$ *set* (ps_+) **and** *h*: $h = s * h0$ **by** (*fastforce simp*: *pos-decomp-map*)
from *this*(*1*) *pos-decomp-subset* **have** $(h0,\ U) \in$ *set ps* **..**
with *assms*(*2*) **have** $h0 \neq 0$ **by** (*rule valid-decompD*)
with *assms*(*3*) **have** *deg-h*: *poly-deg h* = *poly-deg s* + *poly-deg h0* **unfolding** *h* **by** (*rule poly-deg-times*)
moreover from *assms*(*1*) *1* **have** $k \leq$ *poly-deg h0* **by** (*rule standard-decompD*)
ultimately show k + *poly-deg s* $\leq$ *poly-deg h* **by** *simp*

fix *d*
assume k + *poly-deg s* $\leq d$ **and** $d \leq$ *poly-deg h*
hence $k \leq d -$ *poly-deg s* **and** $d -$ *poly-deg s* $\leq$ *poly-deg h0* **by** (*simp-all add*: *deg-h*)
with *assms*(*1*) *1* **obtain** $h'\ U'$ **where** *2*: $(h',\ U') \in$ *set ps* **and** *poly-deg* $h' = d$ − *poly-deg s*
and *card* $U \leq$ *card* U' **by** (*rule standard-decompE*)
from *assms*(*2*) *this*(*1*) **have** $h' \neq 0$ **by** (*rule valid-decompD*)
with *assms*(*3*) **have** *deg-h'*: *poly-deg* $(s * h')$ = *poly-deg s* + *poly-deg* h' **by** (*rule poly-deg-times*)
from *2* **have** $(s * h',\ U') \in$ *set* (*map* (*apfst* ((∗) *s*)) *ps*) **by** *force*
moreover from ‹k + *poly-deg s* $\leq d$› ‹*poly-deg* $h' = d$ − *poly-deg s*› **have** *poly-deg* $(s * h') = d$
by (*simp add*: *deg-h'*)
ultimately show $\exists h'\ U'.\ (h',\ U') \in$ *set* (*map* (*apfst* ((∗) *s*)) *ps*) $\wedge$ *poly-deg* $h' = d \wedge$ *card* $U \leq$ *card* U'
using ‹*card* $U \leq$ *card* U'› **by** *fastforce*
qed

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-deg* ‘ *fst* ‘ *set* (ps_+))
proof −
let *?A* = *poly-deg* ‘ *fst* ‘ *set* (ps_+)
define *m* **where** *m* = *Min ?A*
have *finite ?A* **by** *simp*
moreover from *assms*(*4*) **have** *?A* ≠ {} **by** *simp*
ultimately have *m* ∈ *?A* **unfolding** *m-def* **by** (*rule Min-in*)
then obtain *h U* **where** (*h*, *U*) ∈ *set* (ps_+) **and** *m*: *m* = *poly-deg h* **by** *fastforce*
have *m-min*: *m* ≤ *poly-deg h′* **if** (*h′*, *U′*) ∈ *set* (ps_+) **for** *h′ U′*
proof −
from *that* **have** *poly-deg* (*fst* (*h′*, *U′*)) ∈ *?A* **by** (*intro imageI*)
with ‹*finite ?A*› **have** *m* ≤ *poly-deg* (*fst* (*h′*, *U′*)) **unfolding** *m-def* **by** (*rule Min-le*)
thus *?thesis* **by** *simp*
qed
show *?thesis*
proof (*rule linorder-cases*)
assume *k* < *m*
hence *k* ≤ *poly-deg h* **by** (*simp add*: *m*)
with *assms*(*3*) ‹(*h*, *U*) ∈ *set* (ps_+)› *le-refl* **obtain** *h′ U′*
where (*h′*, *U′*) ∈ *set ps* **and** *poly-deg h′* = *k* **and** *card U* ≤ *card U′* **by** (*rule standard-decompE*)
from *this*(*2*) ‹*k* < *m*› **have** ¬ *m* ≤ *poly-deg h′* **by** *simp*
with *m-min* **have** (*h′*, *U′*) ∉ *set* (ps_+) **by** *blast*
with ‹(*h′*, *U′*) ∈ *set ps*› **have** *U′* = {} **by** (*simp add*: *pos-decomp-def*)
with ‹*card U* ≤ *card U′*› **have** *U* = {} ∨ *infinite U* **by** (*simp add*: *card-eq-0-iff*)
thus *?thesis*
proof
assume *U* = {}
with ‹(*h*, *U*) ∈ *set* (ps_+)› **show** *?thesis* **by** (*simp add*: *pos-decomp-def*)
next
assume *infinite U*
moreover from *assms*(*1*, *2*) **have** *finite U*
proof (*rule valid-decompD-finite*)
from ‹(*h*, *U*) ∈ *set* (ps_+)› **show** (*h*, *U*) ∈ *set ps* **by** (*simp add*: *pos-decomp-def*)
qed
ultimately show *?thesis* **..**
qed
next
assume *m* < *k*
hence ¬ *k* ≤ *m* **by** *simp*
moreover from *assms*(*3*) ‹(*h*, *U*) ∈ *set* (ps_+)› **have** *k* ≤ *m* **unfolding** *m* **by** (*rule standard-decompD*)
ultimately show *?thesis* **..**
qed (*simp only*: *m-def*)
qed

lemma *standard-decomp-SucE*:
assumes *finite* X **and** $U \subseteq X$ **and** $h \in P[X]$ **and** $h \neq (0::\text{-} \Rightarrow_0 {}'a::\{comm\text{-}ring\text{-}1, ring\text{-}no\text{-}zero\text{-}divisors\})$
obtains *ps* **where** *valid-decomp* X *ps* **and** *cone-decomp* (*cone* (h, U)) *ps*
and *standard-decomp* (*Suc* (*poly-deg* h)) *ps*
and *is-monomial* $h \Longrightarrow$ *punit.lc* $h = 1 \Longrightarrow$ *monomial-decomp ps* **and** *homogeneous* $h \Longrightarrow$ *hom-decomp ps*
proof −
from *assms*(2, 1) **have** *finite* U **by** (*rule finite-subset*)
thus *?thesis* **using** *assms*(2) *that*
proof (*induct U arbitrary*: *thesis rule*: *finite-induct*)
case *empty*
from *assms*(3, 4) **have** *valid-decomp* X [(h, {})] **by** (*simp add*: *valid-decomp-def*)
moreover note *cone-decomp-singleton*
moreover have *standard-decomp* (*Suc* (*poly-deg* h)) [(h, {})]
by (*rule standard-decomp-Nil*) (*simp add*: *pos-decomp-def*)
ultimately show *?case* **by** (*rule empty*) (*simp-all add*: *monomial-decomp-def hom-decomp-def*)
next
case (*insert x U*)
from *insert.prems*(1) **have** $x \in X$ **and** $U \subseteq X$ **by** *simp-all*
from *this*(2) **obtain** *ps* **where** *0*: *valid-decomp* X *ps* **and** *1*: *cone-decomp* (*cone* (h, U)) *ps*
and *2*: *standard-decomp* (*Suc* (*poly-deg* h)) *ps*
and *3*: *is-monomial* $h \Longrightarrow$ *punit.lc* $h = 1 \Longrightarrow$ *monomial-decomp ps*
and *4*: *homogeneous* $h \Longrightarrow$ *hom-decomp ps* **by** (*rule insert.hyps*) *blast*
let $?x$ = *monomial* ($1::'a$) (*Poly-Mapping.single* x (*Suc* 0))
have $?x \neq 0$ **by** (*simp add*: *monomial-0-iff*)
with *assms*(4) **have** *deg*: *poly-deg* ($?x * h$) = *Suc* (*poly-deg* h)
by (*simp add*: *poly-deg-times poly-deg-monomial deg-pm-single*)
define *qs* **where** *qs* = [($?x * h$, *insert* x U)]
show *?case*
proof (*rule insert.prems*)
from ‹$x \in X$› **have** $?x \in P[X]$ **by** (*intro Polys-closed-monomial PPs-closed-single*)
hence $?x * h \in P[X]$ **using** *assms*(3) **by** (*rule Polys-closed-times*)
moreover from ‹$?x \neq 0$› *assms*(4) **have** $?x * h \neq 0$ **by** (*rule times-not-zero*)
ultimately have *valid-decomp* X *qs* **using** *insert.hyps*(1) ‹$x \in X$› ‹$U \subseteq X$›
by (*simp add*: *qs-def valid-decomp-def*)
with *0* **show** *valid-decomp* X (*ps* @ *qs*) **by** (*rule valid-decomp-append*)
next
show *cone-decomp* (*cone* (h, *insert* x U)) (*ps* @ *qs*)
proof (*rule cone-decomp-append*)
show *direct-decomp* (*cone* (h, *insert* x U)) [*cone* (h, U), *cone* ($?x * h$, *insert* x U)]
using *insert.hyps*(2) **by** (*rule direct-decomp-cone-insert*)
next
show *cone-decomp* (*cone* ($?x * h$, *insert* x U)) *qs*
by (*simp add*: *qs-def cone-decomp-singleton*)
qed (*fact 1*)
next

from *standard-decomp-singleton*[*of ?x* ∗ *h insert x U*]
have *standard-decomp* (*Suc* (*poly-deg h*)) *qs* **by** (*simp add*: *deg qs-def*)
with *2* **show** *standard-decomp* (*Suc* (*poly-deg h*)) (*ps* @ *qs*) **by** (*rule standard-decomp-append*)
next
assume *is-monomial h* **and** *punit.lc h* = *1*
hence *monomial-decomp ps* **by** (*rule 3*)
moreover have *monomial-decomp qs*
proof −
have *is-monomial* (*?x* ∗ *h*)
by (*metis* ‹*is-monomial h*› *is-monomial-monomial monomial-is-monomial mult.commute*
mult.right-neutral mult-single)
thus *?thesis* **by** (*simp add*: *monomial-decomp-def qs-def lc-times* ‹*punit.lc h* = *1*›)
qed
ultimately show *monomial-decomp* (*ps* @ *qs*) **by** (*simp only*: *monomial-decomp-append-iff*)
next
assume *homogeneous h*
hence *hom-decomp ps* **by** (*rule 4*)
moreover from ‹*homogeneous h*› **have** *hom-decomp qs*
by (*simp add*: *hom-decomp-def qs-def homogeneous-times*)
ultimately show *hom-decomp* (*ps* @ *qs*) **by** (*simp only*: *hom-decomp-append-iff*)
qed
qed
qed

lemma *standard-decomp-geE*:
assumes *finite X* **and** *valid-decomp X ps*
and *cone-decomp* (T::(('x $\Rightarrow_0$ *nat*) $\Rightarrow_0$ 'a::{*comm-ring-1*,*ring-no-zero-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-decomp ps* ⟹ *monomial-decomp qs* **and** *hom-decomp ps* ⟹ *hom-decomp qs*
proof −
have ∃ *qs*. *valid-decomp X qs* ∧ *cone-decomp T qs* ∧ *standard-decomp* ($k + i$) *qs* ∧
(*monomial-decomp ps* ⟶ *monomial-decomp qs*) ∧ (*hom-decomp ps* ⟶ *hom-decomp qs*) **for** *i*
proof (*induct i*)
case *0*
from *assms*(*2*, *3*, *4*) **show** *?case* **unfolding** *add-0-right* **by** *blast*
next
case (*Suc i*)
then obtain *qs* **where** *0*: *valid-decomp X qs* **and** *1*: *cone-decomp T qs*
and *2*: *standard-decomp* ($k + i$) *qs* **and** *3*: *monomial-decomp ps* ⟹ *monomial-decomp qs*

and *4*: *hom-decomp ps* $\Longrightarrow$ *hom-decomp qs* **by** *blast*
let *?P* = λhU. *poly-deg* (*fst hU*) $\neq$ *k* + *i*
define *rs* **where** *rs* = *filter* ($-$ *?P*) *qs*
define *ss* **where** *ss* = *filter* *?P* *qs*

have *set rs* $\subseteq$ *set qs* **by** (*auto simp*: *rs-def*)
have *set ss* $\subseteq$ *set qs* **by** (*auto simp*: *ss-def*)

define *f* **where** *f* = (λhU. *SOME ps'*. *valid-decomp X ps'* $\wedge$ *cone-decomp* (*cone hU*) *ps'* $\wedge$
standard-decomp (*Suc* (*poly-deg* ((*fst hU*)::('*x* $\Rightarrow_0$ -) $\Rightarrow_0$ '*a*))) *ps'* $\wedge$
(*monomial-decomp ps* $\longrightarrow$ *monomial-decomp ps'*) $\wedge$
(*hom-decomp ps* $\longrightarrow$ *hom-decomp ps'*))
have *valid-decomp X* (*f hU*) $\wedge$ *cone-decomp* (*cone hU*) (*f hU*) $\wedge$ *standard-decomp* (*Suc* (*k* + *i*)) (*f hU*) $\wedge$
(*monomial-decomp ps* $\longrightarrow$ *monomial-decomp* (*f hU*)) $\wedge$ (*hom-decomp ps* $\longrightarrow$ *hom-decomp* (*f hU*))
if *hU* $\in$ *set rs* **for** *hU*
proof $-$
obtain *h U* **where** *hU*: *hU* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
with *that* **have** *eq*: *poly-deg* (*fst hU*) = *k* + *i* **by** (*simp add*: *rs-def*)
from *that* ‹*set rs* $\subseteq$ *set qs*› **have** (*h*, *U*) $\in$ *set qs* **unfolding** *hU* **..**
with *0* **have** *U* $\subseteq$ *X* **and** *h* $\in$ *P*[*X*] **and** *h* $\neq$ *0* **by** (*rule valid-decompD*)+
with *assms*(*1*) **obtain** *ps'* **where** *valid-decomp X ps'* **and** *cone-decomp* (*cone* (*h*, *U*)) *ps'*
and *standard-decomp* (*Suc* (*poly-deg h*)) *ps'*
and *md*: *is-monomial h* $\Longrightarrow$ *punit.lc h* = *1* $\Longrightarrow$ *monomial-decomp ps'*
and *hd*: *homogeneous h* $\Longrightarrow$ *hom-decomp ps'* **by** (*rule standard-decomp-SucE*) *blast*
note *this*(*1*$-$*3*)
moreover have *monomial-decomp ps'* **if** *monomial-decomp ps*
proof $-$
from *that* **have** *monomial-decomp qs* **by** (*rule 3*)
hence *is-monomial h* **and** *punit.lc h* = *1* **using** ‹(*h*, *U*) $\in$ *set qs*› **by** (*rule monomial-decompD*)+
thus *?thesis* **by** (*rule md*)
qed
moreover have *hom-decomp ps'* **if** *hom-decomp ps*
proof $-$
from *that* **have** *hom-decomp qs* **by** (*rule 4*)
hence *homogeneous h* **using** ‹(*h*, *U*) $\in$ *set qs*› **by** (*rule hom-decompD*)
thus *?thesis* **by** (*rule hd*)
qed
ultimately have *valid-decomp X ps'* $\wedge$ *cone-decomp* (*cone hU*) *ps'* $\wedge$
standard-decomp (*Suc* (*poly-deg* (*fst hU*))) *ps'* $\wedge$ (*monomial-decomp ps* $\longrightarrow$ *monomial-decomp ps'*) $\wedge$
(*hom-decomp ps* $\longrightarrow$ *hom-decomp ps'*) **by** (*simp add*: *hU*)
thus *?thesis* **unfolding** *f-def eq* **by** (*rule someI*)

qed
hence *f1*: $\bigwedge ps.\ ps \in set\ (map\ f\ rs) \Longrightarrow valid\text{-}decomp\ X\ ps$
and *f2*: $\bigwedge hU.\ hU \in set\ rs \Longrightarrow cone\text{-}decomp\ (cone\ hU)\ (f\ hU)$
and *f3*: $\bigwedge ps.\ ps \in set\ (map\ f\ rs) \Longrightarrow standard\text{-}decomp\ (Suc\ (k + i))\ ps$
and *f4*: $\bigwedge ps'.\ monomial\text{-}decomp\ ps \Longrightarrow ps' \in set\ (map\ f\ rs) \Longrightarrow monomial\text{-}decomp\ ps'$
and *f5*: $\bigwedge ps'.\ hom\text{-}decomp\ ps \Longrightarrow ps' \in set\ (map\ f\ rs) \Longrightarrow hom\text{-}decomp\ ps'$
by *auto*

define *rs'* **where** $rs' = concat\ (map\ f\ rs)$
show *?case* **unfolding** *add-Suc-right*
proof (*intro exI conjI impI*)
have *valid-decomp X ss*
proof (*rule valid-decompI*)
fix *h U*
assume $(h,\ U) \in set\ ss$
hence $(h,\ U) \in set\ qs$ **using** ‹$set\ ss \subseteq set\ qs$› **..**
with *0* **show** $h \in P[X]$ **and** $h \neq 0$ **and** $U \subseteq X$ **by** (*rule valid-decompD*)+
qed
moreover have *valid-decomp X rs'*
unfolding *rs'-def* **using** *f1* **by** (*rule valid-decomp-concat*)
ultimately show *valid-decomp X (ss @ rs')* **by** (*rule valid-decomp-append*)
next
from *1* **have** *direct-decomp T (map cone qs)* **by** (*rule cone-decompD*)
hence *direct-decomp T ((map cone ss) @ (map cone rs))* **unfolding** *ss-def rs-def*
by (*rule direct-decomp-split-map*)
hence *ss*: *cone-decomp (sum-list ‘ listset (map cone ss)) ss*
and *cone-decomp (sum-list ‘ listset (map cone rs)) rs*
and *T*: *direct-decomp T [sum-list ‘ listset (map cone ss), sum-list ‘ listset (map cone rs)]*
by (*auto simp*: *cone-decomp-def dest*: *direct-decomp-appendD intro*!: *empty-not-in-map-cone*)
from *this(2)* **have** *direct-decomp (sum-list ‘ listset (map cone rs)) (map cone rs)*
by (*rule cone-decompD*)
hence *cone-decomp (sum-list ‘ listset (map cone rs)) rs'* **unfolding** *rs'-def*
proof (*rule cone-decomp-concat*)
fix *i*
assume $*$: $i < length\ (map\ cone\ rs)$
hence $rs\ !\ i \in set\ rs$ **by** *simp*
hence *cone-decomp (cone (rs ! i)) (f (rs ! i))* **by** (*rule f2*)
with $*$ **show** *cone-decomp (map cone rs ! i) (map f rs ! i)* **by** *simp*
qed *simp*
with *T ss* **show** *cone-decomp T (ss @ rs')* **by** (*rule cone-decomp-append*)
next
have *standard-decomp* $(Suc\ (k + i))$ *ss*
proof (*rule standard-decompI*)
fix *h U*
assume $(h,\ U) \in set\ (ss_+)$

hence $(h,\ U) \in$ *set* (qs_+) **and** *poly-deg* $h \neq k + i$ **by** (*simp-all add*: *pos-decomp-def ss-def*)
from *2 this*(*1*) **have** $k + i \leq$ *poly-deg* h **by** (*rule standard-decompD*)
with ‹*poly-deg* $h \neq k + i$› **show** *Suc* $(k + i) \leq$ *poly-deg* h **by** *simp*

fix d'
assume *Suc* $(k + i) \leq d'$ **and** $d' \leq$ *poly-deg* h
from *this*(*1*) **have** $k + i \leq d'$ **and** $d' \neq k + i$ **by** *simp-all*
from *2* ‹$(h,\ U) \in$ *set* (qs_+)› *this*(*1*) **obtain** h' U'
where $(h',\ U') \in$ *set qs* **and** *poly-deg* $h' = d'$ **and** *card* $U \leq$ *card* U'
using ‹$d' \leq$ *poly-deg* h› **by** (*rule standard-decompE*)
moreover from ‹$d' \neq k + i$› *this*(*1*, *2*) **have** $(h',\ U') \in$ *set ss* **by** (*simp add*: *ss-def*)
ultimately show $\exists\, h'\ U'.\ (h',\ U') \in$ *set ss* $\wedge$ *poly-deg* $h' = d' \wedge$ *card* $U \leq$ *card* U' **by** *blast*
qed
moreover have *standard-decomp* (*Suc* $(k + i)$) rs'
unfolding rs'*-def* **using** *f3* **by** (*rule standard-decomp-concat*)
ultimately show *standard-decomp* (*Suc* $(k + i)$) (*ss* @ rs') **by** (*rule standard-decomp-append*)
next
assume $*$: *monomial-decomp ps*
hence *monomial-decomp qs* **by** (*rule 3*)
hence *monomial-decomp ss* **by** (*simp add*: *monomial-decomp-def ss-def*)
moreover have *monomial-decomp* rs'
unfolding rs'*-def* **using** *f4*[*OF* $*$] **by** (*rule monomial-decomp-concat*)
ultimately show *monomial-decomp* (*ss* @ rs') **by** (*simp only*: *monomial-decomp-append-iff*)
next
assume $*$: *hom-decomp ps*
hence *hom-decomp qs* **by** (*rule 4*)
hence *hom-decomp ss* **by** (*simp add*: *hom-decomp-def ss-def*)
moreover have *hom-decomp* rs' **unfolding** rs'*-def* **using** *f5*[*OF* $*$] **by** (*rule hom-decomp-concat*)
ultimately show *hom-decomp* (*ss* @ rs') **by** (*simp only*: *hom-decomp-append-iff*)
qed
qed
then obtain *qs* **where** *1*: *valid-decomp X qs* **and** *2*: *cone-decomp T qs*
and *standard-decomp* $(k + (d - k))$ *qs* **and** *4*: *monomial-decomp ps* $\Longrightarrow$ *monomial-decomp qs*
and *5*: *hom-decomp ps* $\Longrightarrow$ *hom-decomp qs* **by** *blast*
from *this*(*3*) *assms*(*5*) **have** *standard-decomp d qs* **by** *simp*
with *1 2* **show** *?thesis* **using** *4 5* **..**
qed

10.5 Splitting w.r.t. Ideals

context
fixes X :: *'x set*
begin

definition *splits-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*
  unfolding *splits-wrt-def fst-conv snd-conv*
proof (*intro conjI ballI*)
  fix *hU*
  assume $hU \in set\ ps$
  moreover obtain *h U* **where** *hU*: $hU = (h, U)$ **using** *prod.exhaust* **by** *blast*
  ultimately have $(h, U) \in set\ ps$ **by** *simp*
  hence $cone\ (h, U) \subseteq P[X]$ **and** $h \in ideal\ F$ **by** (*rule assms*)+
  from - *this*(*1*) **show** $cone\ hU \subseteq ideal\ F \cap P[X]$ **unfolding** *hU*
  proof (*rule Int-greatest*)
    show $cone\ (h, U) \subseteq ideal\ F$
    proof
      fix *a*
      assume $a \in cone\ (h, U)$
      then obtain a' **where** $a' \in P[U]$ **and** *a*: $a = a' * h$ **by** (*rule coneE*)
      from ‹$h \in ideal\ F$› **show** $a \in ideal\ F$ **unfolding** *a* **by** (*rule ideal.span-scale*)
    qed
  qed
next
  fix *hU*
  assume $hU \in set\ qs$
  moreover obtain *h U* **where** *hU*: $hU = (h, U)$ **using** *prod.exhaust* **by** *blast*
  ultimately have $(h, U) \in set\ qs$ **by** *simp*
  hence $cone\ (h, U) \subseteq P[X]$ **and** $\bigwedge a.\ a \in cone\ (h, U) \Longrightarrow a \in ideal\ F \Longrightarrow a = 0$ **by** (*rule assms*)+
  moreover have $0 \in cone\ (h, U) \cap ideal\ F$
    by (*simp add*: *zero-in-cone ideal.span-zero*)
  ultimately show *case hU of* $(h, U) \Rightarrow cone\ (h, U) \subseteq P[X] \wedge cone\ (h, U) \cap ideal\ F = \{0\}$
    by (*fastforce simp*: *hU*)
qed (*fact assms*)+

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*
 using *assms*(*3*) - - - *assms*(*5*)
proof (*rule splits-wrtI*)
 fix *h U*
 assume $(h,\ U) \in$ *set ps*
 thus $h \in$ *ideal F* **by** (*rule assms*(*4*))
 from *assms*(*1*) ‹$(h,\ U) \in$ *set ps*› **have** $h \in P[X]$ **and** $U \subseteq X$ **by** (*rule valid-decompD*)+
 thus *cone* $(h,\ U) \subseteq P[X]$ **by** (*rule cone-subset-PolysI*)
next
 fix *h U*
 assume $(h,\ U) \in$ *set qs*
 with *assms*(*2*) **have** $h \in P[X]$ **by** (*rule valid-decompD*)
 moreover from *assms*(*2*) ‹$(h,\ U) \in$ *set qs*› **have** $U \subseteq X$ **by** (*rule valid-decompD*)
 ultimately show *cone* $(h,\ U) \subseteq P[X]$ **by** (*rule cone-subset-PolysI*)
qed

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* $F = \{0\}$
 using *assms* **by** (*fastforce simp*: *splits-wrt-def*)+

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
 fix *x*
 assume *x-in*: $x \in$ *sum-list* ‘ *listset* (*map cone ps*)
 have *listset* (*map cone ps*) $\subseteq$ *listset* (*map* (λ-. *ideal F* $\cap$ $P[X]$) *ps*)
 proof (*rule listset-mono*)
  fix *i*
  assume *i*: $i <$ *length* (*map* (λ-. *ideal F* $\cap$ $P[X]$) *ps*)
  hence *ps* ! $i \in$ *set ps* **by** *simp*
  with *assms*(*1*) **have** *cone* (*ps* ! *i*) $\subseteq$ *ideal F* $\cap$ $P[X]$ **by** (*rule splits-wrtD*)
  with *i* **show** *map cone ps* ! $i \subseteq$ *map* (λ-. *ideal F* $\cap$ $P[X]$) *ps* ! *i* **by** *simp*
 qed *simp*
 hence *sum-list* ‘ *listset* (*map cone ps*) $\subseteq$ *sum-list* ‘ *listset* (*map* (λ-. *ideal F* $\cap$ $P[X]$) *ps*)
  by (*rule image-mono*)
 with *x-in* **have** $x \in$ *sum-list* ‘ *listset* (*map* (λ-. *ideal F* $\cap$ $P[X]$) *ps*) **..**
 then obtain *xs* **where** *xs*: $xs \in$ *listset* (*map* (λ-. *ideal F* $\cap$ $P[X]$) *ps*) **and** *x*: x = *sum-list xs* **..**
 have *1*: $y \in$ *ideal F* $\cap$ $P[X]$ **if** $y \in$ *set xs* **for** *y*

proof −
from *that* **obtain** i **where** $i <$ *length xs* **and** y: $y = xs \mathbin{!} i$ **by** (*metis in-set-conv-nth*)
moreover from *xs* **have** *length xs* = *length* (*map* (λ-. *ideal* $F \cap P[X]$) *ps*)
by (*rule listsetD*)
ultimately have $i <$ *length* (*map* (λ-. *ideal* $F \cap P[X]$) *ps*) **by** *simp*
moreover from *xs this* **have** $xs \mathbin{!} i \in$ (*map* (λ-. *ideal* $F \cap P[X]$) *ps*) $\mathbin{!} i$ **by** (*rule listsetD*)
ultimately show $y \in$ *ideal* $F \cap P[X]$ **by** (*simp add*: y)
qed
show $x \in$ *ideal* $F \cap P[X]$ **unfolding** x
proof
show *sum-list xs* $\in$ *ideal F*
proof (*rule ideal.span-closed-sum-list*[*simplified*])
fix y
assume $y \in$ *set xs*
hence $y \in$ *ideal* $F \cap P[X]$ **by** (*rule 1*)
thus $y \in$ *ideal F* **by** *simp*
qed
next
show *sum-list xs* $\in P[X]$
proof (*rule Polys-closed-sum-list*)
fix y
assume $y \in$ *set xs*
hence $y \in$ *ideal* $F \cap P[X]$ **by** (*rule 1*)
thus $y \in P[X]$ **by** *simp*
qed
qed
qed

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
fix x
assume *x-in*: $x \in$ *sum-list* ‘ *listset* (*map cone qs*)
have *listset* (*map cone qs*) $\subseteq$ *listset* (*map* (λ-. $P[X]$) *qs*)
proof (*rule listset-mono*)
fix i
assume i: $i <$ *length* (*map* (λ-. $P[X]$) *qs*)
hence $qs \mathbin{!} i \in$ *set qs* **by** *simp*
with *assms*(*1*) **have** *cone* ($qs \mathbin{!} i$) $\subseteq P[X]$ **by** (*rule splits-wrtD*)
with i **show** *map cone qs* $\mathbin{!} i \subseteq$ *map* (λ-. $P[X]$) *qs* $\mathbin{!} i$ **by** *simp*
qed *simp*
hence *sum-list* ‘ *listset* (*map cone qs*) $\subseteq$ *sum-list* ‘ *listset* (*map* (λ-. $P[X]$) *qs*)
by (*rule image-mono*)
with *x-in* **have** $x \in$ *sum-list* ‘ *listset* (*map* (λ-. $P[X]$) *qs*) **..**
then obtain *xs* **where** *xs*: *xs* $\in$ *listset* (*map* (λ-. $P[X]$) *qs*) **and** x: x = *sum-list xs* **..**

show $x \in P[X]$ **unfolding** x
proof (*rule Polys-closed-sum-list*)
fix y
assume $y \in$ *set xs*
then obtain i **where** $i <$ *length xs* **and** y: $y = xs \mathbin{!} i$ **by** (*metis in-set-conv-nth*)
moreover from *xs* **have** *length xs* = *length* (*map* (λ-. $P[X]$::(- $\Rightarrow_0$ $'a$) *set*) *qs*)
by (*rule listsetD*)
ultimately have $i <$ *length* (*map* (λ-. $P[X]$) *qs*) **by** *simp*
moreover from *xs this* **have** *xs* ! *i* ∈ (*map* (λ-. $P[X]$) *qs*) ! *i* **by** (*rule listsetD*)
ultimately show $y \in P[X]$ **by** (*simp add*: *y*)
qed
qed

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 ∩ *ideal F*) *ps*
proof −
from *assms*(*1*) **have** ∗: *cone-decomp T* (*ps* @ *qs*) **by** (*rule splits-wrtD*)
hence *direct-decomp T* (*map cone ps* @ *map cone qs*) **by** (*simp add*: *cone-decomp-def*)
hence *1*: *direct-decomp* (*sum-list* ‘ *listset* (*map cone ps*)) (*map cone ps*)
and *2*: *direct-decomp T* [*sum-list* ‘ *listset* (*map cone ps*), *sum-list* ‘ *listset* (*map cone qs*)]
by (*auto dest*: *direct-decomp-appendD intro*!: *empty-not-in-map-cone*)
let *?ss* = [*sum-list* ‘ *listset* (*map cone ps*), *sum-list* ‘ *listset* (*map cone qs*)]
show *?thesis*
proof (*intro cone-decompI direct-decompI*)
from *1* **show** *inj-on sum-list* (*listset* (*map cone ps*)) **by** (*simp only*: *direct-decomp-def bij-betw-def*)
next
from *assms*(*1*) **have** *sum-list* ‘ *listset* (*map cone ps*) ⊆ *ideal F* ∩ $P[X]$
by (*rule splits-wrt-image-sum-list-fst-subset*)
hence *sub*: *sum-list* ‘ *listset* (*map cone ps*) ⊆ *ideal F* **by** *simp*
show *sum-list* ‘ *listset* (*map cone ps*) = T ∩ *ideal F*
proof (*rule set-eqI*)
fix x
show $x \in$ *sum-list* ‘ *listset* (*map cone ps*) ⟷ $x \in T$ ∩ *ideal F*
proof
assume *x-in*: $x \in$ *sum-list* ‘ *listset* (*map cone ps*)
show $x \in T$ ∩ *ideal F*
proof (*intro IntI*)
have *map* (λ-. *0*) *qs* ∈ *listset* (*map cone qs*) (**is** *?ys* ∈ -)
by (*induct qs*) (*auto intro*: *listset-ConsI zero-in-cone simp del*: *listset.simps*(*2*))
hence *sum-list ?ys* ∈ *sum-list* ‘ *listset* (*map cone qs*) **by** (*rule imageI*)
hence *0* ∈ *sum-list* ‘ *listset* (*map cone qs*) **by** *simp*
with *x-in* **have** [*x*, *0*] ∈ *listset ?ss* **using** *refl* **by** (*rule listset-doubletonI*)
with *2* **have** *sum-list* [*x*, *0*] ∈ *T* **by** (*rule direct-decompD*)

thus $x \in T$ **by** *simp*
next
from *x-in sub* **show** $x \in$ *ideal* F **..**
qed
next
assume $x \in T \cap$ *ideal* F
hence $x \in T$ **and** $x \in$ *ideal* F **by** *simp-all*
from *2 this*(*1*) **obtain** *xs* **where** *xs* $\in$ *listset ?ss* **and** x: $x =$ *sum-list xs*
by (*rule direct-decompE*)
from *this*(*1*) **obtain** p q **where** p: $p \in$ *sum-list* ‘ *listset* (*map cone ps*)
and q: $q \in$ *sum-list* ‘ *listset* (*map cone qs*) **and** *xs*: *xs* $= [p, q]$
by (*rule listset-doubletonE*)
from ‹$x \in$ *ideal* F› **have** $p + q \in$ *ideal* F **by** (*simp add*: *x xs*)
moreover from *p sub* **have** $p \in$ *ideal* F **..**
ultimately have $p + q - p \in$ *ideal* F **by** (*rule ideal.span-diff*)
hence $q \in$ *ideal* F **by** *simp*
have $q = 0$
proof (*rule ccontr*)
assume $q \neq 0$
hence *keys* $q \neq \{\}$ **by** *simp*
then obtain t **where** $t \in$ *keys* q **by** *blast*
with *assms*(*2*) q **obtain** c h U **where** $(h, U) \in$ *set qs* **and** $c \neq 0$
and *monomial c t* $\in$ *cone* (h, U) **by** (*rule monomial-decomp-sum-list-monomial-in-cone*)
moreover from *assms*(*3*) ‹$q \in$ *ideal* F› ‹$t \in$ *keys* q› **have** *monomial c t*
$\in$ *ideal* F
by (*rule punit.monomial-pmdl-field*[*simplified*])
ultimately have *monomial c t* $\in$ *cone* $(h, U) \cap$ *ideal* F **by** *simp*
also from *assms*(*1*) ‹$(h, U) \in$ *set qs*› **have** $\ldots = \{0\}$ **by** (*rule splits-wrtD*)
finally have $c = 0$ **by** (*simp add*: *monomial-0-iff*)
with ‹$c \neq 0$› **show** *False* **..**
qed
with p **show** $x \in$ *sum-list* ‘ *listset* (*map cone ps*) **by** (*simp add*: *x xs*)
qed
qed
qed
qed

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* ($T \cap$ *normal-form* F ‘ $P[X]$) *qs*
proof $-$
from *assms*(*2*) **have** $*$: *cone-decomp* T (*ps* @ *qs*) **by** (*rule splits-wrtD*)
hence *direct-decomp* T (*map cone ps* @ *map cone qs*) **by** (*simp add*: *cone-decomp-def*)
hence *1*: *direct-decomp* (*sum-list* ‘ *listset* (*map cone qs*)) (*map cone qs*)
and *2*: *direct-decomp* T [*sum-list* ‘ *listset* (*map cone ps*), *sum-list* ‘ *listset* (*map*

cone qs)]
  by (*auto dest*: *direct-decomp-appendD intro*!: *empty-not-in-map-cone*)
 let *?ss* = [*sum-list* ‘ *listset* (*map cone ps*), *sum-list* ‘ *listset* (*map cone qs*)]
 let *?G* = *punit.reduced-GB F*
 from *assms*(*1*, *5*) **have** *?G* ⊆ *P*[*X*] **and** *G-is-GB*: *punit.is-Groebner-basis ?G*
  and *ideal-G*: *ideal ?G* = *ideal F*
  by (*rule reduced-GB-Polys*, *rule reduced-GB-is-GB-Polys*, *rule reduced-GB-ideal-Polys*)
 show *?thesis*
 proof (*intro cone-decompI direct-decompI*)
   from *1* **show** *inj-on sum-list* (*listset* (*map cone qs*)) **by** (*simp only*: *direct-decomp-def bij-betw-def*)
 next
  from *assms*(*2*) **have** *sum-list* ‘ *listset* (*map cone ps*) ⊆ *ideal F* ∩ *P*[*X*]
   by (*rule splits-wrt-image-sum-list-fst-subset*)
  hence *sub*: *sum-list* ‘ *listset* (*map cone ps*) ⊆ *ideal F* **by** *simp*
  show *sum-list* ‘ *listset* (*map cone qs*) = *T* ∩ *normal-form F* ‘ *P*[*X*]
  proof (*rule set-eqI*)
   fix *x*
   show *x* ∈ *sum-list* ‘ *listset* (*map cone qs*) ⟷ *x* ∈ *T* ∩ *normal-form F* ‘ *P*[*X*]
   proof
    assume *x-in*: *x* ∈ *sum-list* ‘ *listset* (*map cone qs*)
    show *x* ∈ *T* ∩ *normal-form F* ‘ *P*[*X*]
    proof (*intro IntI*)
     have *map* (λ-. *0*) *ps* ∈ *listset* (*map cone ps*) (**is** *?ys* ∈ -)
       by (*induct ps*) (*auto intro*: *listset-ConsI zero-in-cone simp del*: *listset.simps*(*2*))
     hence *sum-list ?ys* ∈ *sum-list* ‘ *listset* (*map cone ps*) **by** (*rule imageI*)
     hence *0* ∈ *sum-list* ‘ *listset* (*map cone ps*) **by** *simp*
    from *this x-in* **have** [*0*, *x*] ∈ *listset ?ss* **using** *refl* **by** (*rule listset-doubletonI*)
     with *2* **have** *sum-list* [*0*, *x*] ∈ *T* **by** (*rule direct-decompD*)
     thus *x* ∈ *T* **by** *simp*
    next
     from *assms*(*2*) **have** *sum-list* ‘ *listset* (*map cone qs*) ⊆ *P*[*X*]
      by (*rule splits-wrt-image-sum-list-snd-subset*)
     with *x-in* **have** *x* ∈ *P*[*X*] **..**
     moreover have ¬ *punit.is-red ?G x*
     proof
      assume *punit.is-red ?G x*
      then obtain *g t* **where** *g* ∈ *?G* **and** *t* ∈ *keys x* **and** *g* ≠ *0* **and** *adds*: *lpp g adds t*
       by (*rule punit.is-red-addsE*[*simplified*])
      from *assms*(*3*) *x-in this*(*2*) **obtain** *c h U* **where** (*h*, *U*) ∈ *set qs* **and** *c* ≠ *0*
     and *monomial c t* ∈ *cone* (*h*, *U*) **by** (*rule monomial-decomp-sum-list-monomial-in-cone*)
      note *this*(*3*)
      moreover have *monomial c t* ∈ *ideal ?G*
      proof (*rule punit.is-red-monomial-monomial-set-in-pmdl*[*simplified*])
        from ‹*c* ≠ *0*› **show** *is-monomial* (*monomial c t*) **by** (*rule monomial-is-monomial*)

next
from *assms*(*1*, *5*, *4*) **show** *is-monomial-set ?G* **by** (*rule reduced-GB-is-monomial-set-Polys*)
next
from ‹*c* ≠ *0*› **have** *t* ∈ *keys* (*monomial c t*) **by** *simp*
with ‹*g* ∈ *?G*› ‹*g* ≠ *0*› **show** *punit.is-red ?G* (*monomial c t*) **using** *adds*
by (*rule punit.is-red-addsI*[*simplified*])
qed
ultimately have *monomial c t* ∈ *cone* (*h*, *U*) ∩ *ideal F* **by** (*simp add*: *ideal-G*)
also from *assms*(*2*) ‹(*h*, *U*) ∈ *set qs*› **have** . . . = {*0*} **by** (*rule splits-wrtD*)
finally have *c* = *0* **by** (*simp add*: *monomial-0-iff*)
with ‹*c* ≠ *0*› **show** *False* **..**
qed
ultimately show *x* ∈ *normal-form F* ‘ *P*[*X*]
using *assms*(*1*, *5*) **by** (*simp add*: *image-normal-form-iff*)
qed
next
assume *x* ∈ *T* ∩ *normal-form F* ‘ *P*[*X*]
hence *x* ∈ *T* **and** *x* ∈ *normal-form F* ‘ *P*[*X*] **by** *simp-all*
from *this*(*2*) *assms*(*1*, *5*) **have** *x* ∈ *P*[*X*] **and** *irred*: ¬ *punit.is-red ?G x*
by (*simp-all add*: *image-normal-form-iff*)
from *2* ‹*x* ∈ *T*› **obtain** *xs* **where** *xs* ∈ *listset ?ss* **and** *x*: *x* = *sum-list xs*
by (*rule direct-decompE*)
from *this*(*1*) **obtain** *p q* **where** *p*: *p* ∈ *sum-list* ‘ *listset* (*map cone ps*)
and *q*: *q* ∈ *sum-list* ‘ *listset* (*map cone qs*) **and** *xs*: *xs* = [*p*, *q*]
by (*rule listset-doubletonE*)
have *x* = *p* + *q* **by** (*simp add*: *x xs*)
from *p sub* **have** *p* ∈ *ideal F* **..**
have *p* = *0*
proof (*rule ccontr*)
assume *p* ≠ *0*
hence *keys p* ≠ {} **by** *simp*
then obtain *t* **where** *t* ∈ *keys p* **by** *blast*
from *assms*(*4*) ‹*p* ∈ *ideal F*› ‹*t* ∈ *keys p*› **have** *3*: *monomial c t* ∈ *ideal F* **for** *c*
by (*rule punit.monomial-pmdl-field*[*simplified*])
have *t* ∉ *keys q*
proof
assume *t* ∈ *keys q*
with *assms*(*3*) *q* **obtain** *c h U* **where** (*h*, *U*) ∈ *set qs* **and** *c* ≠ *0*
and *monomial c t* ∈ *cone* (*h*, *U*) **by** (*rule monomial-decomp-sum-list-monomial-in-cone*)
from *this*(*3*) *3* **have** *monomial c t* ∈ *cone* (*h*, *U*) ∩ *ideal F* **by** *simp*
also from *assms*(*2*) ‹(*h*, *U*) ∈ *set qs*› **have** . . . = {*0*} **by** (*rule splits-wrtD*)
finally have *c* = *0* **by** (*simp add*: *monomial-0-iff*)
with ‹*c* ≠ *0*› **show** *False* **..**
qed
with ‹*t* ∈ *keys p*› **have** *t* ∈ *keys x* **unfolding** ‹*x* = *p* + *q*› **by** (*rule*

```
in-keys-plusI1)
          have punit.is-red ?G x
          proof -
            note G-is-GB
          moreover from 3 have monomial 1 t ∈ ideal ?G by (simp only: ideal-G)
            moreover have monomial (1::'a) t ≠ 0 by (simp add: monomial-0-iff)
            ultimately obtain g where g ∈ ?G and g ≠ 0
          and lpp g adds lpp (monomial (1::'a) t) by (rule punit.GB-adds-lt[simplified])
            from this(3) have lpp g adds t by (simp add: punit.lt-monomial)
         with ‹g ∈ ?G› ‹g ≠ 0› ‹t ∈ keys x› show ?thesis by (rule punit.is-red-addsI[simplified])
          qed
          with irred show False ..
        qed
        with q show x ∈ sum-list ‘ listset (map cone qs) by (simp add: x xs)
      qed
    qed
  qed
qed

lemma quot-monomial-ideal-monomial:
  ideal (monomial 1 ‘ S) ÷ monomial 1 (Poly-Mapping.single (x::'x) (1::nat)) =
    ideal (monomial (1::'a::comm-ring-1) ‘ (λs. s − Poly-Mapping.single x 1) ‘ S)
proof (rule set-eqI)
  let ?x = Poly-Mapping.single x (1::nat)
  fix a
  have a ∈ ideal (monomial 1 ‘ S) ÷ monomial 1 ?x ⟷ punit.monom-mult 1 ?x
a ∈ ideal (monomial (1::'a) ‘ S)
    by (simp only: quot-set-iff times-monomial-left)
  also have ... ⟷ a ∈ ideal (monomial 1 ‘ (λs. s − ?x) ‘ S)
  proof (induct a rule: poly-mapping-plus-induct)
    case 1
    show ?case by (simp add: ideal.span-zero)
  next
    case (2 a c t)
    let ?S = monomial (1::'a) ‘ (λs. s − ?x) ‘ S
    show ?case
    proof
      assume 0: punit.monom-mult 1 ?x (monomial c t + a) ∈ ideal (monomial 1
‘ S)
      have is-monomial-set (monomial (1::'a) ‘ S)
        by (auto intro!: is-monomial-setI monomial-is-monomial)
      moreover from 0 have 1: monomial c (?x + t) + punit.monom-mult 1 ?x
a ∈ ideal (monomial 1 ‘ S)
        by (simp add: punit.monom-mult-monomial punit.monom-mult-dist-right)
      moreover have ?x + t ∈ keys (monomial c (?x + t) + punit.monom-mult
1 ?x a)
      proof (intro in-keys-plusI1 notI)
        from 2(1) show ?x + t ∈ keys (monomial c (?x + t)) by simp
      next
```

assume *?x* + *t* ∈ *keys* (*punit.monom-mult 1 ?x a*)
also have . . . ⊆ (+) *?x* ' *keys a* **by** (*rule punit.keys-monom-mult-subset*[*simplified*])
finally obtain *s* **where** *s* ∈ *keys a* **and** *?x* + *t* = *?x* + *s* **..**
from *this*(*2*) **have** *t* = *s* **by** *simp*
with ‹*s* ∈ *keys a*› *2*(*2*) **show** *False* **by** *simp*
qed
ultimately obtain *f* **where** *f* ∈ *monomial* (*1*::'*a*) ' *S* **and** *adds*: *lpp f adds ?x* + *t*
by (*rule punit.keys-monomial-pmdl*[*simplified*])
from *this*(*1*) **obtain** *s* **where** *s* ∈ *S* **and** *f*: *f* = *monomial 1 s* **..**
from *adds* **have** *s adds ?x* + *t* **by** (*simp add*: *f punit.lt-monomial*)
hence *s* − *?x adds t*
by (*metis* (*no-types*, *lifting*) *add-minus-2 adds-minus adds-triv-right plus-minus-assoc-pm-nat-1*)
then obtain *s'* **where** *t*: *t* = (*s* − *?x*) + *s'* **by** (*rule addsE*)
from ‹*s* ∈ *S*› **have** *monomial 1* (*s* − *?x*) ∈ *?S* **by** (*intro imageI*)
also have . . . ⊆ *ideal ?S* **by** (*rule ideal.span-superset*)
finally have *monomial c s'* ∗ *monomial 1* (*s* − *?x*) ∈ *ideal ?S*
by (*rule ideal.span-scale*)
hence *monomial c t* ∈ *ideal ?S* **by** (*simp add*: *times-monomial-monomial t add.commute*)
moreover have *a* ∈ *ideal ?S*
proof −
from ‹*f* ∈ *monomial 1* ' *S*› **have** *f* ∈ *ideal* (*monomial 1* ' *S*) **by** (*rule ideal.span-base*)
hence *punit.monom-mult c* (*?x* + *t* − *s*) *f* ∈ *ideal* (*monomial 1* ' *S*)
by (*rule punit.pmdl-closed-monom-mult*[*simplified*])
with ‹*s adds ?x* + *t*› **have** *monomial c* (*?x* + *t*) ∈ *ideal* (*monomial 1* ' *S*)
by (*simp add*: *f punit.monom-mult-monomial adds-minus*)
with *1* **have** *monomial c* (*?x* + *t*) + *punit.monom-mult 1 ?x a* − *monomial c* (*?x* + *t*) ∈ *ideal* (*monomial 1* ' *S*)
by (*rule ideal.span-diff*)
thus *?thesis* **by** (*simp add*: *2*(*3*) *del*: *One-nat-def*)
qed
ultimately show *monomial c t* + *a* ∈ *ideal ?S*
by (*rule ideal.span-add*)
next
have *is-monomial-set ?S* **by** (*auto intro*!: *is-monomial-setI monomial-is-monomial*)
moreover assume *1*: *monomial c t* + *a* ∈ *ideal ?S*
moreover from - *2*(*2*) **have** *t* ∈ *keys* (*monomial c t* + *a*)
proof (*rule in-keys-plusI1*)
from *2*(*1*) **show** *t* ∈ *keys* (*monomial c t*) **by** *simp*
qed
ultimately obtain *f* **where** *f* ∈ *?S* **and** *adds*: *lpp f adds t*
by (*rule punit.keys-monomial-pmdl*[*simplified*])
from *this*(*1*) **obtain** *s* **where** *s* ∈ *S* **and** *f*: *f* = *monomial 1* (*s* − *?x*) **by** *blast*
from *adds* **have** *s* − *?x adds t* **by** (*simp add*: *f punit.lt-monomial*)
hence *s adds ?x* + *t*
by (*auto simp*: *adds-poly-mapping le-fun-def lookup-add lookup-minus*

lookup-single when-def
split: *if-splits*)
then obtain s' **where** t: *?x* + t = s + s' **by** (*rule addsE*)
from ‹$s \in S$› **have** *monomial 1 s* ∈ *monomial 1 ‘ S* **by** (*rule imageI*)
also have ... ⊆ *ideal* (*monomial 1 ‘ S*) **by** (*rule ideal.span-superset*)
finally have *monomial c s′* ∗ *monomial 1 s* ∈ *ideal* (*monomial 1 ‘ S*)
by (*rule ideal.span-scale*)
hence *monomial c* (*?x* + t) ∈ *ideal* (*monomial 1 ‘ S*)
by (*simp only*: t) (*simp add*: *times-monomial-monomial add.commute*)
moreover have *punit.monom-mult 1 ?x a* ∈ *ideal* (*monomial 1 ‘ S*)
proof −
from ‹$f \in$ *?S*› **have** f ∈ *ideal ?S* **by** (*rule ideal.span-base*)
hence *punit.monom-mult c* (t − (s − *?x*)) f ∈ *ideal ?S*
by (*rule punit.pmdl-closed-monom-mult*[*simplified*])
with ‹s − *?x adds t*› **have** *monomial c t* ∈ *ideal ?S*
by (*simp add*: *f punit.monom-mult-monomial adds-minus*)
with *1* **have** *monomial c t* + a − *monomial c t* ∈ *ideal ?S*
by (*rule ideal.span-diff*)
thus *?thesis* **by** (*simp add*: *2*(*3*) *del*: *One-nat-def*)
qed
ultimately have *monomial c* (*?x* + t) + *punit.monom-mult 1 ?x a* ∈ *ideal* (*monomial 1 ‘ S*)
by (*rule ideal.span-add*)
thus *punit.monom-mult 1 ?x* (*monomial c t* + a) ∈ *ideal* (*monomial 1 ‘ S*)
by (*simp add*: *punit.monom-mult-monomial punit.monom-mult-dist-right*)
qed
qed
finally show a ∈ *ideal* (*monomial 1 ‘ S*) ÷ *monomial 1 ?x* ⟷ a ∈ *ideal* (*monomial 1 ‘* (λ*s*. s − *?x*) *‘ S*) **.**
qed

lemma *lem-4-2-1*:
assumes *ideal F* ÷ *monomial 1 t* = *ideal* (*monomial* (*1*::′*a*::*comm-ring-1*) *‘ S*)
shows *cone* (*monomial 1 t*, *U*) ⊆ *ideal F* ⟷ *0* ∈ *S*
proof
have *monomial 1 t* ∈ *cone* (*monomial* (*1*::′*a*) *t*, *U*) **by** (*rule tip-in-cone*)
also assume *cone* (*monomial 1 t*, *U*) ⊆ *ideal F*
finally have ∗: *monomial 1 t* ∗ *1* ∈ *ideal F* **by** *simp*
have *is-monomial-set* (*monomial* (*1*::′*a*) *‘ S*)
by (*auto intro*!: *is-monomial-setI monomial-is-monomial*)
moreover from ∗ **have** *1* ∈ *ideal* (*monomial* (*1*::′*a*) *‘ S*) **by** (*simp only*: *quot-set-iff flip*: *assms*)
moreover have *0* ∈ *keys* (*1*::- $\Rightarrow_0$ ′*a*) **by** *simp*
ultimately obtain g **where** g ∈ *monomial* (*1*::′*a*) *‘ S* **and** *adds*: *lpp g adds 0*
by (*rule punit.keys-monomial-pmdl*[*simplified*])
from *this*(*1*) **obtain** s **where** $s \in S$ **and** g: g = *monomial 1 s* **..**
from *adds* **have** *s adds 0* **by** (*simp add*: *g punit.lt-monomial flip*: *single-one*)
with ‹$s \in S$› **show** *0* ∈ *S* **by** (*simp only*: *adds-zero*)
next

  assume *0* ∈ *S*
  hence *monomial 1 0* ∈ *monomial* (*1*::′*a*) ‘ *S* **by** (*rule imageI*)
  hence *1* ∈ *ideal* (*monomial* (*1*::′*a*) ‘ *S*) **unfolding** *single-one* **by** (*rule ideal.span-base*)
  hence *eq*: *ideal F* ÷ *monomial 1 t* = *UNIV* (**is** - ÷ *?t* = -)
    by (*simp only*: *assms ideal-eq-UNIV-iff-contains-one*)
  show *cone* (*monomial 1 t*, *U*) ⊆ *ideal F*
  proof
    fix *a*
    assume *a* ∈ *cone* (*?t*, *U*)
    then obtain *q* **where** *a*: *a* = *q* ∗ *?t* **by** (*rule coneE*)
    have *q* ∈ *ideal F* ÷ *?t* **by** (*simp add*: *eq*)
    thus *a* ∈ *ideal F* **by** (*simp only*: *a quot-set-iff mult.commute*)
  qed
qed

lemma *lem-4-2-2*:
  assumes *ideal F* ÷ *monomial 1 t* = *ideal* (*monomial* (*1*::′*a*::*comm-ring-1*) ‘ *S*)
  shows *cone* (*monomial 1 t*, *U*) ∩ *ideal F* = {*0*} ⟷ *S* ∩ .[*U*] = {}
proof
  let *?t* = *monomial* (*1*::′*a*) *t*
  assume *eq*: *cone* (*?t*, *U*) ∩ *ideal F* = {*0*}
  {
    fix *s*
    assume *s* ∈ *S*
    hence *monomial 1 s* ∈ *monomial* (*1*::′*a*) ‘ *S* (**is** *?s* ∈ -) **by** (*rule imageI*)
    hence *?s* ∈ *ideal* (*monomial 1* ‘ *S*) **by** (*rule ideal.span-base*)
    also have … = *ideal F* ÷ *?t* **by** (*simp only*: *assms*)
    finally have ∗: *?s* ∗ *?t* ∈ *ideal F* **by** (*simp only*: *quot-set-iff mult.commute*)
    assume *s* ∈ .[*U*]
    hence *?s* ∈ *P*[*U*] **by** (*rule Polys-closed-monomial*)
    with *refl* **have** *?s* ∗ *?t* ∈ *cone* (*?t*, *U*) **by** (*rule coneI*)
    with ∗ **have** *?s* ∗ *?t* ∈ *cone* (*?t*, *U*) ∩ *ideal F* **by** *simp*
    hence *False* **by** (*simp add*: *eq times-monomial-monomial monomial-0-iff*)
  }
  thus *S* ∩ .[*U*] = {} **by** *blast*
next
  let *?t* = *monomial* (*1*::′*a*) *t*
  assume *eq*: *S* ∩ .[*U*] = {}
  {
    fix *a*
    assume *a* ∈ *cone* (*?t*, *U*)
    then obtain *q* **where** *q* ∈ *P*[*U*] **and** *a*: *a* = *q* ∗ *?t* **by** (*rule coneE*)
    assume *a* ∈ *ideal F*
    have *a* = *0*
    proof (*rule ccontr*)
      assume *a* ≠ *0*
      hence *q* ≠ *0* **by** (*auto simp*: *a*)
      from ‹*a* ∈ *ideal F*› **have** ∗: *q* ∈ *ideal F* ÷ *?t* **by** (*simp only*: *quot-set-iff a mult.commute*)

have *is-monomial-set* (*monomial* (*1*::′*a*) ‘ *S*)
by (*auto intro*!: *is-monomial-setI monomial-is-monomial*)
moreover from ∗ **have** *q-in*: $q \in$ *ideal* (*monomial 1* ‘ *S*) **by** (*simp only*: *assms*)
moreover from ‹$q \neq 0$› **have** *lpp q* ∈ *keys q* **by** (*rule punit.lt-in-keys*)
ultimately obtain *g* **where** *g* ∈ *monomial* (*1*::′*a*) ‘ *S* **and** *adds*: *lpp g adds lpp q*
by (*rule punit.keys-monomial-pmdl*[*simplified*])
from *this*(*1*) **obtain** *s* **where** *s* ∈ *S* **and** *g*: *g* = *monomial 1 s* **..**
from ‹$q \neq 0$› **have** *lpp q* ∈ *keys q* **by** (*rule punit.lt-in-keys*)
also from ‹$q \in P[U]$› **have** . . . ⊆ .[*U*] **by** (*rule PolysD*)
finally have *lpp q* ∈ .[*U*] **.**
moreover from *adds* **have** *s adds lpp q* **by** (*simp add*: *g punit.lt-monomial*)
ultimately have *s* ∈ .[*U*] **by** (*rule PPs-closed-adds*)
with *eq* ‹*s* ∈ *S*› **show** *False* **by** *blast*
qed
}
thus *cone* (*?t*, *U*) ∩ *ideal F* = {*0*} **using** *zero-in-cone ideal.span-zero* **by** *blast*
qed

10.6 Function *split*

definition *max-subset* :: ′*a set* ⇒ (′*a set* ⇒ *bool*) ⇒ ′*a set*
where *max-subset A P* = (*ARG-MAX card B*. *B* ⊆ *A* ∧ *P B*)

lemma *max-subset*:
assumes *finite A* **and** *B* ⊆ *A* **and** *P B*
shows *max-subset A P* ⊆ *A* (**is** *?thesis1*)
and *P* (*max-subset A P*) (**is** *?thesis2*)
and *card B* ≤ *card* (*max-subset A P*) (**is** *?thesis3*)
proof −
from *assms*(*2*, *3*) **have** *B* ⊆ *A* ∧ *P B* **by** *simp*
moreover have ∀ *C*. *C* ⊆ *A* ∧ *P C* ⟶ *card C* < *Suc* (*card A*)
proof (*intro allI impI*, *elim conjE*)
fix *C*
assume *C* ⊆ *A*
with *assms*(*1*) **have** *card C* ≤ *card A* **by** (*rule card-mono*)
thus *card C* < *Suc* (*card A*) **by** *simp*
qed
ultimately have *?thesis1* ∧ *?thesis2* **and** *?thesis3* **unfolding** *max-subset-def*
by (*rule arg-max-natI*, *rule arg-max-nat-le*)
thus *?thesis1* **and** *?thesis2* **and** *?thesis3* **by** *simp-all*
qed

function (*domintros*) *split* :: (′*x* $\Rightarrow_0$ *nat*) ⇒ ′*x set* ⇒ (′*x* $\Rightarrow_0$ *nat*) *set* ⇒
(((((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*) × (′*x set*)) *list*) ×
((((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::{*zero*,*one*}) × (′*x set*)) *list*))
where
split t U S =

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

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** $fst\ (snd\ args) \subseteq X$ **and** *finite* ($snd\ (snd\ args)$)
shows *split-dom* $TYPE('a::\{zero,one\})$ *args*
proof –
let $?m = \lambda args'.\ card\ (fst\ (snd\ args')) + sum\ deg\text{-}pm\ (snd\ (snd\ args'))$
from *wf-measure*[*of* ?m] *assms*(2, 3) **show** *?thesis*
proof (*induct args*)
case (*less args*)
obtain $t\ U\ F$ **where** *args*: $args = (t, U, F)$ **using** *prod.exhaust* **by** *metis*
from *less.prems* **have** $U \subseteq X$ **and** *finite* F **by** (*simp-all only*: *args fst-conv snd-conv*)
from *this*(1) *assms*(1) **have** *finite* U **by** (*rule finite-subset*)
have *IH*: *split-dom* $TYPE('a)$ (t', U', F')
if $U' \subseteq X$ **and** *finite* F' **and** $card\ U' + sum\ deg\text{-}pm\ F' < card\ U + sum\ deg\text{-}pm\ F$
for $t'\ U'\ F'$
using *less.hyps that* **by** (*simp add*: *args*)

define S **where** $S = max\text{-}subset\ U\ (\lambda V.\ F \cap .[V] = \{\})$
define x **where** $x = (SOME\ x'.\ x' \in U \land x' \notin S)$
show *?case* **unfolding** *args*
proof (*rule split.domintros*, *simp-all only*: *x-def*[*symmetric*] *S-def*[*symmetric*])
fix f
assume $0 \notin F$ **and** $f \in F$ **and** $f \in .[U]$
from *this*(1) **have** $F \cap .[\{\}] = \{\}$ **by** *simp*
with ‹*finite* U› *empty-subsetI* **have** $S \subseteq U$ **and** $F \cap .[S] = \{\}$
unfolding *S-def* **by** (*rule max-subset*)+
have $x \in U \land x \notin S$ **unfolding** *x-def*
proof (*rule someI-ex*)
from ‹$f \in F$› ‹$f \in .[U]$› ‹$F \cap .[S] = \{\}$› **have** $S \neq U$ **by** *blast*
with ‹$S \subseteq U$› **show** $\exists y.\ y \in U \land y \notin S$ **by** *blast*
qed
hence $x \in U$ **and** $x \notin S$ **by** *simp-all*
{
assume $\neg$ *split-dom* $TYPE('a)$ $(t, U - \{x\}, F)$

moreover from - ‹*finite F*› **have** *split-dom TYPE('a)* (*t*, *U* − {*x*}, *F*)
proof (*rule IH*)
from ‹*U* ⊆ *X*› **show** *U* − {*x*} ⊆ *X* **by** *blast*
next
from ‹*finite U*› ‹*x* ∈ *U*› **have** *card* (*U* − {*x*}) < *card U* **by** (*rule card-Diff1-less*)
thus *card* (*U* − {*x*}) + *sum deg-pm F* < *card U* + *sum deg-pm F* **by** *simp*
qed
ultimately show *False* **..**
}
{
let *?args* = (*Poly-Mapping.single x* (*Suc 0*) + *t*, *U*, (*λf*. *f* − *Poly-Mapping.single x* (*Suc 0*)) ‘ *F*)
assume ¬ *split-dom TYPE('a) ?args*
moreover from ‹*U* ⊆ *X*› **have** *split-dom TYPE('a) ?args*
proof (*rule IH*)
from ‹*finite F*› **show** *finite* ((*λf*. *f* − *Poly-Mapping.single x* (*Suc 0*)) ‘ *F*)
by (*rule finite-imageI*)
next
have *sum deg-pm* ((*λf*. *f* − *Poly-Mapping.single x* (*Suc 0*)) ‘ *F*) ≤
sum (*deg-pm* ∘ (*λf*. *f* − *Poly-Mapping.single x* (*Suc 0*))) *F*
using ‹*finite F*› **by** (*rule sum-image-le*) *simp*
also from ‹*finite F*› **have** ... < *sum deg-pm F*
proof (*rule sum-strict-mono-ex1*)
show ∀*f*∈*F*. (*deg-pm* ∘ (*λf*. *f* − *Poly-Mapping.single x* (*Suc 0*))) *f* ≤ *deg-pm f*
by (*simp add*: *deg-pm-minus-le*)
next
show ∃*f*∈*F*. (*deg-pm* ∘ (*λf*. *f* − *Poly-Mapping.single x* (*Suc 0*))) *f* < *deg-pm f*
proof (*rule ccontr*)
assume ∗: ¬ (∃*f*∈*F*. (*deg-pm* ∘ (*λf*. *f* − *Poly-Mapping.single x* (*Suc 0*))) *f* < *deg-pm f*)
note ‹*finite U*›
moreover from ‹*x* ∈ *U*› ‹*S* ⊆ *U*› **have** *insert x S* ⊆ *U* **by** (*rule insert-subsetI*)
moreover have *F* ∩ .[*insert x S*] = {}
proof −
{
fix *s*
assume *s* ∈ *F*
with ∗ **have** ¬ *deg-pm* (*s* − *Poly-Mapping.single x* (*Suc 0*)) < *deg-pm s* **by** *simp*
with *deg-pm-minus-le*[*of s Poly-Mapping.single x* (*Suc 0*)]
have *deg-pm* (*s* − *Poly-Mapping.single x* (*Suc 0*)) = *deg-pm s* **by** *simp*
hence *keys s* ∩ *keys* (*Poly-Mapping.single x* (*Suc 0*)) = {}
by (*simp only*: *deg-pm-minus-id-iff*)

hence $x \notin$ *keys s* **by** *simp*
moreover assume $s \in .[\textit{insert } x\ S]$
ultimately have $s \in .[S]$ **by** (*fastforce simp*: *PPs-def*)
with ‹$s \in F$› ‹$F \cap .[S] = \{\}$› **have** *False* **by** *blast*
}
thus *?thesis* **by** *blast*
qed
ultimately have *card* (*insert x S*) $\leq$ *card S* **unfolding** *S-def* **by** (*rule max-subset*)
moreover from ‹$S \subseteq U$› ‹*finite U*› **have** *finite S* **by** (*rule finite-subset*)
ultimately show *False* **using** ‹$x \notin S$› **by** *simp*
qed
qed
finally show *card U* + *sum deg-pm* (($\lambda f.\ f$ − *monomial* (*Suc 0*) *x*) ‘ *F*) < *card U* + *sum deg-pm F*
by *simp*
qed
ultimately show *False* **..**
}
qed
qed
qed

corollary *split-domI*: *finite X* $\Longrightarrow U \subseteq X \Longrightarrow$ *finite S* $\Longrightarrow$ *split-dom TYPE*($'a$::{*zero*,*one*}) (*t*, *U*, *S*)
using *split-domI'*[*of* (*t*, *U*, *S*)] **by** *simp*

lemma *split-empty*:
assumes *finite X* **and** $U \subseteq X$
shows *split t U* {} = ([], [(*monomial* (*1*::$'a$::{*zero*,*one*}) *t*, *U*)])
proof −
have *finite* {} **..**
with *assms* **have** *split-dom TYPE*($'a$) (*t*, *U*, {}) **by** (*rule split-domI*)
thus *?thesis* **by** (*simp add*: *split.psimps*)
qed

lemma *split-induct* [*consumes 3*, *case-names base1 base2 step*]:
fixes *P* :: ($'x \Rightarrow_0$ *nat*) $\Rightarrow$ -
assumes *finite X* **and** $U \subseteq X$ **and** *finite S*
assumes $\bigwedge t\ U\ S.\ U \subseteq X \Longrightarrow$ *finite S* $\Longrightarrow 0 \in S \Longrightarrow P\ t\ U\ S$ ([(*monomial* (*1*::$'a$::{*zero*,*one*}) *t*, *U*)], [])
assumes $\bigwedge t\ U\ S.\ U \subseteq X \Longrightarrow$ *finite S* $\Longrightarrow 0 \notin S \Longrightarrow S \cap .[U] = \{\} \Longrightarrow P\ t\ U\ S$ ([], [(*monomial 1 t*, *U*)])
assumes $\bigwedge t\ U\ S\ V\ x$ *ps0 ps1 qs0 qs1*. $U \subseteq X \Longrightarrow$ *finite S* $\Longrightarrow 0 \notin S \Longrightarrow S \cap .[U] \neq \{\} \Longrightarrow V \subseteq U \Longrightarrow$
$S \cap .[V] = \{\} \Longrightarrow (\bigwedge V'.\ V' \subseteq U \Longrightarrow S \cap .[V'] = \{\} \Longrightarrow$ *card* $V' \leq$ *card* $V) \Longrightarrow$
$x \in U \Longrightarrow x \notin V \Longrightarrow V =$ *max-subset* U ($\lambda V'.\ S \cap .[V'] = \{\}$) $\Longrightarrow x$ = (*SOME* $x'.\ x' \in U - V$) $\Longrightarrow$

$(ps0, qs0)$ = *split* t $(U - \{x\})$ $S \Longrightarrow$
$(ps1, qs1)$ = *split* (*Poly-Mapping.single* x *1* + t) U (($\lambda f.$ f − *Poly-Mapping.single* x *1*) ‘ S) $\Longrightarrow$
split t U S = (*ps0* @ *ps1*, *qs0* @ *qs1*) $\Longrightarrow$
P t $(U - \{x\})$ S (*ps0*, *qs0*) $\Longrightarrow$
P (*Poly-Mapping.single* x *1* + t) U (($\lambda f.$ f − *Poly-Mapping.single* x *1*) ‘ S) (*ps1*, *qs1*) $\Longrightarrow$
P t U S (*ps0* @ *ps1*, *qs0* @ *qs1*)
shows P t U S (*split* t U S)
proof −
from *assms*(*1*−*3*) **have** *split-dom* *TYPE*(′*a*) (t, U, S) **by** (*rule split-domI*)
thus *?thesis* **using** *assms*(*2*,*3*)
proof (*induct* t U S *rule*: *split.pinduct*)
case *step*: (*1* t U F)
from *step*(*4*) *assms*(*1*) **have** *finite* U **by** (*rule finite-subset*)
define S **where** S = *max-subset* U ($\lambda V.$ $F \cap .[V] = \{\}$)
define x **where** x = (*SOME* x'. $x' \in U \wedge x' \notin S$)
show *?case*
proof (*simp add*: *split.psimps*[*OF step*(*1*)] *S-def*[*symmetric*] *x-def*[*symmetric*] *split*: *prod.split*, *intro allI conjI impI*)
assume *0* $\in F$
with *step*(*4*, *5*) **show** P t U F ([(*monomial 1 t*, U)], []) **by** (*rule assms*(*4*))
thus P t U F ([(*monomial 1 t*, U)], []) **.**
next
assume *0* $\notin F$ **and** $F \cap .[U] = \{\}$
with *step*(*4*, *5*) **show** P t U F ([], [(*monomial 1 t*, U)]) **by** (*rule assms*(*5*))
next
fix *ps0 qs0 ps1 qs1* :: ((- $\Rightarrow_0$ ′*a*) × -) *list*
assume *split* (*Poly-Mapping.single* x (*Suc 0*) + t) U (($\lambda f.$ f − *Poly-Mapping.single* x (*Suc 0*)) ‘ F) = (*ps1*, *qs1*)
hence *PQ1*[*symmetric*]: *split* (*Poly-Mapping.single* x *1* + t) U (($\lambda f.$ f − *Poly-Mapping.single* x *1*) ‘ F) = (*ps1*, *qs1*)
by *simp*
assume *PQ0*[*symmetric*]: *split* t $(U - \{x\})$ F = (*ps0*, *qs0*)
assume $F \cap .[U] \neq \{\}$ **and** *0* $\notin F$
from *this*(*2*) **have** $F \cap .[\{\}] = \{\}$ **by** *simp*
with ‹*finite* U› *empty-subsetI* **have** $S \subseteq U$ **and** $F \cap .[S] = \{\}$
unfolding *S-def* **by** (*rule max-subset*)+
have *S-max*: *card* S' ≤ *card* S **if** $S' \subseteq U$ **and** $F \cap .[S'] = \{\}$ **for** S'
using ‹*finite* U› *that* **unfolding** *S-def* **by** (*rule max-subset*)
have $x \in U \wedge x \notin S$ **unfolding** *x-def*
proof (*rule someI-ex*)
from ‹$F \cap .[U] \neq \{\}$› ‹$F \cap .[S] = \{\}$› **have** $S \neq U$ **by** *blast*
with ‹$S \subseteq U$› **show** $\exists y.$ $y \in U \wedge y \notin S$ **by** *blast*
qed
hence $x \in U$ **and** $x \notin S$ **by** *simp-all*
from *step*(*4*, *5*) ‹*0* $\notin F$› ‹$F \cap .[U] \neq \{\}$› ‹$S \subseteq U$› ‹$F \cap .[S] = \{\}$› *S-max* ‹$x \in U$› ‹$x \notin S$› *S-def* - *PQ0 PQ1*
show P t U F (*ps0* @ *ps1*, *qs0* @ *qs1*)

proof (*rule assms*(*6*))
show *P t* (*U* − {*x*}) *F* (*ps0*, *qs0*)
unfolding *PQ0* **using** ‹*0* ∉ *F*› ‹*F* ∩ .[*U*] ≠ {}› - - *step*(*5*)
proof (*rule step*(*2*))
from ‹*U* ⊆ *X*› **show** *U* − {*x*} ⊆ *X* **by** *fastforce*
qed (*simp add*: *x-def S-def*)
next
show *P* (*Poly-Mapping.single x 1* + *t*) *U* ((λ*f*. *f* − *Poly-Mapping.single x 1*) ‘ *F*) (*ps1*, *qs1*)
unfolding *PQ1* **using** ‹*0* ∉ *F*› ‹*F* ∩ .[*U*] ≠ {}› - *refl PQ0* ‹*U* ⊆ *X*›
proof (*rule step*(*3*))
from ‹*finite F*› **show** *finite* ((λ*f*. *f* − *Poly-Mapping.single x 1*) ‘ *F*) **by** (*rule finite-imageI*)
qed (*simp add*: *x-def S-def*)
next
show *split t U F* = (*ps0* @ *ps1*, *qs0* @ *qs1*) **using** ‹*0* ∉ *F*› ‹*F* ∩ .[*U*] ≠ {}›
by (*simp add*: *split.psimps*[*OF step*(*1*)] *Let-def flip*: *S-def x-def PQ0 PQ1 del*: *One-nat-def*)
qed (*assumption*+, *simp add*: *x-def S-def*)
qed
qed
qed

lemma *valid-decomp-split*:
assumes *finite X* **and** *U* ⊆ *X* **and** *finite S* **and** *t* ∈ .[*X*]
shows *valid-decomp X* (*fst* ((*split t U S*)::(- × (((- ⇒$_0$ ′*a*::*zero-neq-one*) × -) *list*))))
and *valid-decomp X* (*snd* ((*split t U S*)::(- × (((- ⇒$_0$ ′*a*::*zero-neq-one*) × -) *list*))))
(**is** *valid-decomp* - (*snd ?s*))
proof −
from *assms* **have** *valid-decomp X* (*fst ?s*) ∧ *valid-decomp X* (*snd ?s*)
proof (*induct t U S rule*: *split-induct*)
case (*base1 t U S*)
from *base1*(*1*, *4*) **show** *?case* **by** (*simp add*: *valid-decomp-def monomial-0-iff Polys-closed-monomial*)
next
case (*base2 t U S*)
from *base2*(*1*, *5*) **show** *?case* **by** (*simp add*: *valid-decomp-def monomial-0-iff Polys-closed-monomial*)
next
case (*step t U S V x ps0 ps1 qs0 qs1*)
from *step.hyps*(*8*, *1*) **have** *x* ∈ *X* **..**
hence *Poly-Mapping.single x 1* ∈ .[*X*] **by** (*rule PPs-closed-single*)
hence *Poly-Mapping.single x 1* + *t* ∈ .[*X*] **using** *step.prems* **by** (*rule PPs-closed-plus*)
with *step.hyps*(*15*, *16*) *step.prems* **show** *?case* **by** (*simp add*: *valid-decomp-append*)
qed
thus *valid-decomp X* (*fst ?s*) **and** *valid-decomp X* (*snd ?s*) **by** *simp-all*
qed

lemma *monomial-decomp-split*:
  assumes *finite X* **and** $U \subseteq X$ **and** *finite S*
  shows *monomial-decomp* (*fst* ((*split t U S*)::(- × (((- $\Rightarrow_0$ 'a::*zero-neq-one*) × -) *list*))))
    and *monomial-decomp* (*snd* ((*split t U S*)::(- × (((- $\Rightarrow_0$ 'a::*zero-neq-one*) × -) *list*))))
        (**is** *monomial-decomp* (*snd ?s*))
proof −
  from *assms* **have** *monomial-decomp* (*fst ?s*) ∧ *monomial-decomp* (*snd ?s*)
  proof (*induct t U S rule*: *split-induct*)
    case (*base1 t U S*)
   from *base1*(*1*) **show** *?case* **by** (*simp add*: *monomial-decomp-def monomial-is-monomial*)
  next
    case (*base2 t U S*)
   from *base2*(*1*) **show** *?case* **by** (*simp add*: *monomial-decomp-def monomial-is-monomial*)
  next
    case (*step t U S V x ps0 ps1 qs0 qs1*)
    from *step.hyps*(*15*, *16*) **show** *?case* **by** (*auto simp*: *monomial-decomp-def*)
  qed
  thus *monomial-decomp* (*fst ?s*) **and** *monomial-decomp* (*snd ?s*) **by** *simp-all*
qed

lemma *split-splits-wrt*:
  assumes *finite X* **and** $U \subseteq X$ **and** *finite S* **and** $t \in .[X]$
    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*
  using *assms*
proof (*induct t U S rule*: *split-induct*)
  case (*base1 t U S*)
  from *base1*(*3*) **have** *cone* (*monomial 1 t*, *U*) ⊆ *ideal F* **by** (*simp only*: *lem-4-2-1* *base1*(*5*))
  show *?case*
  proof (*rule splits-wrtI*)
    fix *h0 U0*
    assume (*h0*, *U0*) ∈ *set* [(*monomial* (*1*::'a) *t*, *U*)]
    hence *h0*: *h0* = *monomial 1 t* **and** *U0* = *U* **by** *simp-all*
    note *this*(*1*)
    also have *monomial 1 t* ∈ *cone* (*monomial* (*1*::'a) *t*, *U*) **by** (*fact tip-in-cone*)
    also have ... ⊆ *ideal F* **by** *fact*
    finally show *h0* ∈ *ideal F* **.**

   from *base1*(*4*) **have** *h0* ∈ *P*[*X*] **unfolding** *h0* **by** (*rule Polys-closed-monomial*)
    moreover from *base1*(*1*) **have** *U0* ⊆ *X* **by** (*simp only*: ‹*U0* = *U*›)
    ultimately show *cone* (*h0*, *U0*) ⊆ *P*[*X*] **by** (*rule cone-subset-PolysI*)
  qed (*simp-all add*: *cone-decomp-singleton* ‹*U* ⊆ *X*›)
next
  case (*base2 t U S*)

from *base2(4)* **have** *cone (monomial 1 t, U) ∩ ideal F = {0}* **by** *(simp only: lem-4-2-2 base2(6))*
show *?case*
proof *(rule splits-wrtI)*
fix *h0 U0*
assume *(h0, U0) ∈ set [(monomial (1::'a) t, U)]*
hence *h0: h0 = monomial 1 t* **and** *U0 = U* **by** *simp-all*
note *this(1)*
also from *base2(5)* **have** *monomial 1 t ∈ P[X]* **by** *(rule Polys-closed-monomial)*
finally have *h0 ∈ P[X]* **.**
moreover from *base2(1)* **have** *U0 ⊆ X* **by** *(simp only: ‹U0 = U›)*
ultimately show *cone (h0, U0) ⊆ P[X]* **by** *(rule cone-subset-PolysI)*
next
fix *h0 U0 a*
assume *(h0, U0) ∈ set [(monomial (1::'a) t, U)]* **and** *a ∈ cone (h0, U0)*
hence *a ∈ cone (monomial 1 t, U)* **by** *simp*
moreover assume *a ∈ ideal F*
ultimately have *a ∈ cone (monomial 1 t, U) ∩ ideal F* **by** *(rule IntI)*
also have *. . . = {0}* **by** *fact*
finally show *a = 0* **by** *simp*
qed *(simp-all add: cone-decomp-singleton ‹U ⊆ X›)*
next
case *(step t U S V x ps0 ps1 qs0 qs1)*
let *?x = Poly-Mapping.single x 1*
from *step.prems* **have** *0: splits-wrt (ps0, qs0) (cone (monomial 1 t, U − {x})) F* **by** *(rule step.hyps)*
have *1: splits-wrt (ps1, qs1) (cone (monomial 1 (?x + t), U)) F*
proof *(rule step.hyps)*
from *step.hyps(8, 1)* **have** *x ∈ X* **..**
hence *?x ∈ .[X]* **by** *(rule PPs-closed-single)*
thus *?x + t ∈ .[X]* **using** *step.prems(1)* **by** *(rule PPs-closed-plus)*
next
have *ideal F ÷ monomial 1 (?x + t) = ideal F ÷ monomial 1 t ÷ monomial 1 ?x*
by *(simp add: times-monomial-monomial add.commute)*
also have *. . . = ideal (monomial 1 ‘ S) ÷ monomial 1 ?x* **by** *(simp only: step.prems)*
finally show *ideal F ÷ monomial 1 (?x + t) = ideal (monomial 1 ‘ (λs. s − ?x) ‘ S)*
by *(simp only: quot-monomial-ideal-monomial)*
qed

show *?case*
proof *(rule splits-wrtI)*
from *step.hyps(8)* **have** *U: insert x U = U* **by** *blast*
have *direct-decomp (cone (monomial (1::'a) t, insert x (U − {x})))*
[cone (monomial 1 t, U − {x}),
cone (monomial 1 (monomial (Suc 0) x) ∗ monomial 1 t, insert x (U − {x}))]

by (*rule direct-decomp-cone-insert*) *simp*
hence *direct-decomp* (*cone* (*monomial* (*1*::′*a*) *t*, *U*))
[*cone* (*monomial 1 t*, $U - \{x\}$), *cone* (*monomial 1* (*?x* + *t*), *U*)]
by (*simp add*: *U times-monomial-monomial*)
moreover from *0* **have** *cone-decomp* (*cone* (*monomial 1 t*, $U - \{x\}$)) (*ps0* @ *qs0*)
by (*rule splits-wrtD*)
moreover from *1* **have** *cone-decomp* (*cone* (*monomial 1* (*?x* + *t*), *U*)) (*ps1* @ *qs1*)
by (*rule splits-wrtD*)
ultimately have *cone-decomp* (*cone* (*monomial 1 t*, *U*)) ((*ps0* @ *qs0*) @ (*ps1* @ *qs1*))
by (*rule cone-decomp-append*)
thus *cone-decomp* (*cone* (*monomial 1 t*, *U*)) ((*ps0* @ *ps1*) @ *qs0* @ *qs1*)
by (*rule cone-decomp-perm*) *simp*
next
fix *h0 U0*
assume (*h0*, *U0*) ∈ *set* (*ps0* @ *ps1*)
hence (*h0*, *U0*) ∈ *set ps0* ∪ *set ps1* **by** *simp*
hence *cone* (*h0*, *U0*) ⊆ *ideal F* ∩ *P*[*X*]
proof
assume (*h0*, *U0*) ∈ *set ps0*
with *0* **show** *?thesis* **by** (*rule splits-wrtD*)
next
assume (*h0*, *U0*) ∈ *set ps1*
with *1* **show** *?thesis* **by** (*rule splits-wrtD*)
qed
hence ∗: *cone* (*h0*, *U0*) ⊆ *ideal F* **and** *cone* (*h0*, *U0*) ⊆ *P*[*X*] **by** *simp-all*
from *this*(*2*) **show** *cone* (*h0*, *U0*) ⊆ *P*[*X*] **.**

from *tip-in-cone* ∗ **show** *h0* ∈ *ideal F* **..**
next
fix *h0 U0*
assume (*h0*, *U0*) ∈ *set* (*qs0* @ *qs1*)
hence (*h0*, *U0*) ∈ *set qs0* ∪ *set qs1* **by** *simp*
thus *cone* (*h0*, *U0*) ⊆ *P*[*X*]
proof
assume (*h0*, *U0*) ∈ *set qs0*
with *0* **show** *?thesis* **by** (*rule splits-wrtD*)
next
assume (*h0*, *U0*) ∈ *set qs1*
with *1* **show** *?thesis* **by** (*rule splits-wrtD*)
qed

from ‹(*h0*, *U0*) ∈ *set qs0* ∪ *set qs1*› **have** *cone* (*h0*, *U0*) ∩ *ideal F* = {*0*}
proof
assume (*h0*, *U0*) ∈ *set qs0*
with *0* **show** *?thesis* **by** (*rule splits-wrtD*)
next

```
      assume (h0, U0) ∈ set qs1
      with 1 show ?thesis by (rule splits-wrtD)
    qed
    thus ⋀a. a ∈ cone (h0, U0) ⟹ a ∈ ideal F ⟹ a = 0 by blast
  qed
qed

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 –
  let ?t = monomial (1::'a) t
  let ?t' = monomial (1::'a) t'
  from assms(6) have 1: cone (?t', V) ⊆ cone (?t, U) and 2: cone (?t', V) ⊆ normal-form F ‘ P[X]
    by blast+
  from this(1) show V ⊆ U by (rule cone-subsetD) (simp add: monomial-0-iff)

  show S ∩ .[V] = {}
  proof
    let ?t = monomial (1::'a) t
    let ?t' = monomial (1::'a) t'
    show S ∩ .[V] ⊆ {}
    proof
      fix s
      assume s ∈ S ∩ .[V]
      hence s ∈ S and s ∈ .[V] by simp-all
     from this(2) have monomial (1::'a) s ∈ P[V] (is ?s ∈ -) by (rule Polys-closed-monomial)
      with refl have ?s ∗ ?t ∈ cone (?t, V) by (rule coneI)
      from tip-in-cone 1 have ?t' ∈ cone (?t, U) ..
      then obtain s' where s' ∈ P[U] and t': ?t' = s' ∗ ?t by (rule coneE)
      note this(1)
      also from assms(2) have P[U] ⊆ P[X] by (rule Polys-mono)
      finally have s' ∈ P[X] .
      have s' ∗ ?s ∗ ?t = ?s ∗ ?t' by (simp add: t')
      also from refl ‹?s ∈ P[V]› have ... ∈ cone (?t', V) by (rule coneI)
      finally have s' ∗ ?s ∗ ?t ∈ cone (?t', V) .
      hence 1: s' ∗ ?s ∗ ?t ∈ normal-form F ‘ P[X] using 2 ..
      from ‹s ∈ S› have ?s ∈ monomial 1 ‘ S by (rule imageI)
      hence ?s ∈ ideal (monomial 1 ‘ S) by (rule ideal.span-base)
      hence s' ∗ ?s ∈ ideal (monomial 1 ‘ S) by (rule ideal.span-scale)
      hence s' ∗ ?s ∈ ideal F ÷ ?t by (simp only: assms(5))
      hence s' ∗ ?s ∗ ?t ∈ ideal F by (simp only: quot-set-iff mult.commute)
     hence s' ∗ ?s ∗ ?t ∈ ideal F ∩ normal-form F ‘ P[X] using 1 by (rule IntI)
      also from assms(1, 4) have ... ⊆ {0}
        by (auto simp: normal-form-normal-form simp flip: normal-form-zero-iff)
```

```
      finally have ?s * ?t' = 0 by (simp add: t' ac-simps)
      thus s ∈ {} by (simp add: times-monomial-monomial monomial-0-iff)
    qed
  qed (fact empty-subsetI)
qed

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 −
  let ?t = monomial (1::'a) t
  let ?t' = monomial (1::'a) t'
  from assms(7) have cone (?t', V) ⊆ cone (?t, U) and cone (?t', V) ⊆ nor-
mal-form F ‘ P[X]
    by blast+
  from assms(1, 2, 4, 5, 6, 7) have V ⊆ U and S ∩ .[V] = {} by (rule lem-4-5)+
  with assms(1, 2, 3) show ?thesis using that
  proof (induct t U S arbitrary: V thesis rule: split-induct)
    case (base1 t U S)
    from base1.hyps(3) have 0 ∈ S ∩ .[V] using zero-in-PPs by (rule IntI)
    thus ?case by (simp add: base1.prems(2))
  next
    case (base2 t U S)
    show ?case
    proof (rule base2.prems)
      from base2.hyps(1) assms(1) have finite U by (rule finite-subset)
      thus card V ≤ card U using base2.prems(1) by (rule card-mono)
    qed simp
  next
    case (step t U S V0 x ps0 ps1 qs0 qs1)
    from step.prems(1, 2) have 0: card V ≤ card V0 by (rule step.hyps)
    from step.hyps(5, 9) have V0 ⊆ U − {x} by blast
    then obtain V' where 1: (monomial 1 t, V') ∈ set (snd (ps0, qs0)) and 2:
card V0 ≤ card V'
      using step.hyps(6) by (rule step.hyps)
    show ?case
    proof (rule step.prems)
      from 1 show (monomial 1 t, V') ∈ set (snd (ps0 @ ps1, qs0 @ qs1)) by
simp
    next
      from 0 2 show card V ≤ card V' by (rule le-trans)
    qed
  qed
qed
```

lemma *lem-4-7*:
assumes *finite* X **and** $S \subseteq .[X]$ **and** $g \in$ *punit.reduced-GB* (*monomial* ($1::'a$) ‘ S)
and *cone-decomp* ($P[X] \cap$ *ideal* (*monomial* ($1::'a::field$) ‘ S)) *ps*
and *monomial-decomp ps*
obtains U **where** $(g, U) \in$ *set ps*
proof −
let $?S$ = *monomial* ($1::'a$) ‘ S
let $?G$ = *punit.reduced-GB* $?S$
note *assms*(*1*)
moreover from *assms*(*2*) **have** $?S \subseteq P[X]$ **by** (*auto intro*: *Polys-closed-monomial*)
moreover have *is-monomial-set* $?S$
by (*auto intro*!: *is-monomial-setI monomial-is-monomial*)
ultimately have *is-monomial-set* $?G$ **by** (*rule reduced-GB-is-monomial-set-Polys*)
hence *is-monomial* g **using** *assms*(*3*) **by** (*rule is-monomial-setD*)
hence $g \neq 0$ **by** (*rule monomial-not-0*)
moreover from *assms*(*1*) ‹$?S \subseteq P[X]$› **have** *punit.is-monic-set* $?G$
by (*rule reduced-GB-is-monic-set-Polys*)
ultimately have *punit.lc* $g = 1$ **using** *assms*(*3*) **by** (*simp add*: *punit.is-monic-set-def*)
moreover define t **where** t = *lpp* g
moreover from ‹*is-monomial* g› **have** *monomial* (*punit.lc* g) (*lpp* g) = g
by (*rule punit.monomial-eq-itself*)
ultimately have g: g = *monomial* 1 t **by** *simp*
hence $t \in$ *keys* g **by** *simp*
from *assms*(*3*) **have** $g \in$ *ideal* $?G$ **by** (*rule ideal.span-base*)
also from *assms*(*1*) ‹$?S \subseteq P[X]$› **have** *ideal-G*: ... = *ideal* $?S$ **by** (*rule reduced-GB-ideal-Polys*)
finally have $g \in$ *ideal* $?S$ **.**
moreover from *assms*(*3*) **have** $g \in P[X]$ **by** *rule* (*intro reduced-GB-Polys assms*(*1*) ‹$?S \subseteq P[X]$›)
ultimately have $g \in P[X] \cap$ *ideal* $?S$ **by** *simp*
with *assms*(*4*) **have** $g \in$ *sum-list* ‘ *listset* (*map cone ps*)
by (*simp only*: *cone-decomp-def direct-decompD*)
with *assms*(*5*) **obtain** d h U **where** $*$: $(h, U) \in$ *set ps* **and** $d \neq 0$ **and** *monomial* d $t \in$ *cone* (h, U)
using ‹$t \in$ *keys* g› **by** (*rule monomial-decomp-sum-list-monomial-in-cone*)
from *this*(*3*) *zero-in-PPs* **have** *punit.monom-mult* (1 / d) 0 (*monomial* d t) $\in$ *cone* (h, U)
by (*rule cone-closed-monom-mult*)
with ‹$d \neq 0$› **have** $g \in$ *cone* (h, U) **by** (*simp add*: g *punit.monom-mult-monomial*)
then obtain q **where** $q \in P[U]$ **and** g': $g = q * h$ **by** (*rule coneE*)
from ‹$g \neq 0$› **have** $q \neq 0$ **and** $h \neq 0$ **by** (*auto simp*: g')
hence *lt-g'*: *lpp* g = *lpp* q + *lpp* h **unfolding** g' **by** (*rule lp-times*)
hence *adds1*: *lpp* h *adds* t **by** (*simp add*: *t-def*)
from *assms*(*5*) $*$ **have** *is-monomial* h **and** *punit.lc* $h = 1$ **by** (*rule monomial-decompD*)+
moreover from *this*(*1*) **have** *monomial* (*punit.lc* h) (*lpp* h) = h
by (*rule punit.monomial-eq-itself*)
moreover define s **where** s = *lpp* h

ultimately have *h*: *h* = *monomial 1 s* **by** *simp*
have *punit.lc q* = *punit.lc g* **by** (*simp add*: *g′ lc-times* ‹*punit.lc h* = *1*›)
hence *punit.lc q* = *1* **by** (*simp only*: ‹*punit.lc g* = *1*›)
note *tip-in-cone*
also from *assms*(*4*) ∗ **have** *cone* (*h*, *U*) ⊆ *P*[*X*] ∩ *ideal ?S* **by** (*rule cone-decomp-cone-subset*)
also have . . . ⊆ *ideal ?G* **by** (*simp add*: *ideal-G*)
finally have *h* ∈ *ideal ?G* **.**
from *assms*(*1*) ‹*?S* ⊆ *P*[*X*]› **have** *punit.is-Groebner-basis ?G* **by** (*rule reduced-GB-is-GB-Polys*)
then obtain *g′* **where** *g′* ∈ *?G* **and** *g′* ≠ *0* **and** *adds2*: *lpp g′ adds lpp h*
using ‹*h* ∈ *ideal ?G*› ‹*h* ≠ *0*› **by** (*rule punit.GB-adds-lt*[*simplified*])
from *this*(*3*) *adds1* **have** *lpp g′ adds t* **by** (*rule adds-trans*)
with - ‹*g′* ≠ *0*› ‹*t* ∈ *keys g*› **have** *punit.is-red* {*g′*} *g*
by (*rule punit.is-red-addsI*[*simplified*]) *simp*
have *g′* = *g*
proof (*rule ccontr*)
assume *g′* ≠ *g*
with ‹*g′* ∈ *?G*› **have** {*g′*} ⊆ *?G* − {*g*} **by** *simp*
with ‹*punit.is-red* {*g′*} *g*› **have** *red*: *punit.is-red* (*?G* − {*g*}) *g* **by** (*rule punit.is-red-subset*)
from *assms*(*1*) ‹*?S* ⊆ *P*[*X*]› **have** *punit.is-auto-reduced ?G* **by** (*rule reduced-GB-is-auto-reduced-Polys*)
hence ¬ *punit.is-red* (*?G* − {*g*}) *g* **using** *assms*(*3*) **by** (*rule punit.is-auto-reducedD*)
thus *False* **using** *red* **..**
qed
with *adds2* **have** *t adds lpp h* **by** (*simp only*: *t-def*)
with *adds1* **have** *lpp h* = *t* **by** (*rule adds-antisym*)
hence *lpp q* = *0* **using** *lt-g′* **by** (*simp add*: *t-def*)
hence *monomial* (*punit.lc q*) *0* = *q* **by** (*rule punit.lt-eq-min-term-monomial*[*simplified*])
hence *g* = *h* **by** (*simp add*: ‹*punit.lc q* = *1*› *g′*)
with ∗ **have** (*g*, *U*) ∈ *set ps* **by** *simp*
thus *?thesis* **..**
qed

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*))
using *assms*
proof (*induct t U S arbitrary*: *thesis rule*: *split-induct*)
case (*base1 t U S*)
from *base1.prems*(*2*) *base1.hyps*(*3*) **show** *?case* **..**
next
case (*base2 t U S*)
from *subset-refl* **show** *?case* **by** (*rule base2.prems*) *simp*
next
case (*step t U S V0 x ps0 ps1 qs0 qs1*)
from *step.hyps*(*3*) **obtain** *V* **where** *1*: *V* ⊆ *U* − {*x*} **and** *2*: (*monomial 1 t*, *V*) ∈ *set* (*snd* (*ps0*, *qs0*))
using *step.hyps*(*15*) **by** *blast*

show *?case*
proof (*rule step.prems*)
from *1* **show** $V \subseteq U$ **by** *blast*
next
from *2* **show** (*monomial 1 t*, V) $\in$ *set* (*snd* (*ps0* @ *ps1*, *qs0* @ *qs1*)) **by** *fastforce*
qed
qed

lemma *fst-splitE*:
assumes *finite X* **and** $U \subseteq X$ **and** *finite S* **and** $0 \notin S$
and (*monomial* (*1*::'*a*) *s*, V) $\in$ *set* (*fst* (*split t U S*))
obtains t' x **where** $t' \in .[X]$ **and** $x \in X$ **and** $V \subseteq U$ **and** $0 \notin (\lambda s.\ s - t')$ ‘ S
and $s = t' + t +$ *Poly-Mapping.single x 1*
and (*monomial* (*1*::'*a*::*zero-neq-one*) *s*, V) $\in$ *set* (*fst* (*split* ($t' + t$) V (($\lambda s.\ s - t'$) ‘ S)))
and *set* (*snd* (*split* ($t' + t$) V (($\lambda s.\ s - t'$) ‘ S))) $\subseteq$ (*set* (*snd* (*split t U S*)) :: ((- $\Rightarrow_0$ '*a*) $\times$ -) *set*)
using *assms*
proof (*induct t U S arbitrary*: *thesis rule*: *split-induct*)
case (*base1 t U S*)
from *base1.prems*(*2*) *base1.hyps*(*3*) **show** *?case* **..**
next
case (*base2 t U S*)
from *base2.prems*(*3*) **show** *?case* **by** *simp*
next
case (*step t U S V0 x ps0 ps1 qs0 qs1*)
from *step.prems*(*3*) **have** (*monomial 1 s*, V) $\in$ *set ps0* $\cup$ *set ps1* **by** *simp*
thus *?case*
proof
assume (*monomial 1 s*, V) $\in$ *set ps0*
hence (*monomial* (*1*::'*a*) *s*, V) $\in$ *set* (*fst* (*ps0*, *qs0*)) **by** (*simp only*: *fst-conv*)
with *step.hyps*(*3*) **obtain** t' x' **where** $t' \in .[X]$ **and** $x' \in X$ **and** $V \subseteq U - \{x\}$
and $0 \notin (\lambda s.\ s - t')$ ‘ S **and** $s = t' + t +$ *Poly-Mapping.single* x' *1*
and (*monomial* (*1*::'*a*) *s*, V) $\in$ *set* (*fst* (*split* ($t' + t$) V (($\lambda s.\ s - t'$) ‘ S)))
and *set* (*snd* (*split* ($t' + t$) V (($\lambda s.\ s - t'$) ‘ S))) $\subseteq$ *set* (*snd* (*ps0*, *qs0*))
using *step.hyps*(*15*) **by** *blast*
note *this*(*7*)
also have *set* (*snd* (*ps0*, *qs0*)) $\subseteq$ *set* (*snd* (*ps0* @ *ps1*, *qs0* @ *qs1*)) **by** *simp*
finally have *set* (*snd* (*split* ($t' + t$) V (($\lambda s.\ s - t'$) ‘ S))) $\subseteq$ *set* (*snd* (*ps0* @ *ps1*, *qs0* @ *qs1*)) **.**
from ‹$V \subseteq U - \{x\}$› **have** $V \subseteq U$ **by** *blast*
show *?thesis* **by** (*rule step.prems*) *fact+*
next
assume (*monomial 1 s*, V) $\in$ *set ps1*
show *?thesis*
proof (*cases* $0 \in (\lambda f.\ f -$ *Poly-Mapping.single x 1*) ‘ S)
case *True*

from *step.hyps*(*2*) **have** *fin*: *finite* ((*λf. f* − *Poly-Mapping.single x 1*) ‘ *S*)
  by (*rule finite-imageI*)
have *split* (*Poly-Mapping.single x 1* + *t*) *U* ((*λf. f* − *Poly-Mapping.single x 1*) ‘ *S*) =
    ([(*monomial* (*1*::′*a*) (*Poly-Mapping.single x 1* + *t*), *U*)], [])
  by (*simp only*: *split.psimps*[*OF split-domI*, *OF assms*(*1*) *step.hyps*(*1*) *fin*] *True if-True*)
hence *ps1* = [(*monomial 1* (*Poly-Mapping.single x 1* + *t*), *U*)]
  by (*simp only*: *step.hyps*(*13*)[*symmetric*] *prod.inject*)
with ‹(*monomial 1 s*, *V*) ∈ *set ps1*› **have** *s*: *s* = *Poly-Mapping.single x 1* + *t* **and** *V* = *U*
  by (*auto dest*!: *monomial-inj*)
show *?thesis*
proof (*rule step.prems*)
  show *0* ∈ .[*X*] **by** (*fact zero-in-PPs*)
next
  from *step.hyps*(*8*, *1*) **show** *x* ∈ *X* **..**
next
  show *V* ⊆ *U* **by** (*simp add*: ‹*V* = *U*›)
next
  from *step.hyps*(*3*) **show** *0* ∉ (*λs. s* − *0*) ‘ *S* **by** *simp*
next
  show *s* = *0* + *t* + *Poly-Mapping.single x 1* **by** (*simp add*: *s add.commute*)
next
  show (*monomial* (*1*::′*a*) *s*, *V*) ∈ *set* (*fst* (*split* (*0* + *t*) *V* ((*λs. s* − *0*) ‘ *S*)))
    using ‹(*monomial 1 s*, *V*) ∈ *set ps1*› **by** (*simp add*: *step.hyps*(*14*) ‹*V* = *U*›)
next
  show *set* (*snd* (*split* (*0* + *t*) *V* ((*λs. s* − *0*) ‘ *S*))) ⊆ *set* (*snd* (*ps0* @ *ps1*, *qs0* @ *qs1*))
    by (*simp add*: *step.hyps*(*14*) ‹*V* = *U*›)
qed
next
case *False*
moreover from ‹(*monomial 1 s*, *V*) ∈ *set ps1*› **have** (*monomial 1 s*, *V*) ∈ *set* (*fst* (*ps1*, *qs1*))
  by (*simp only*: *fst-conv*)
ultimately obtain *t′ x′* **where** *t′* ∈ .[*X*] **and** *x′* ∈ *X* **and** *V* ⊆ *U*
  and *1*: *0* ∉ (*λs. s* − *t′*) ‘ (*λf. f* − *Poly-Mapping.single x 1*) ‘ *S*
  and *s*: *s* = *t′* + (*Poly-Mapping.single x 1* + *t*) + *Poly-Mapping.single x′ 1*
  and *2*: (*monomial* (*1*::′*a*) *s*, *V*) ∈ *set* (*fst* (*split* (*t′* + (*Poly-Mapping.single x 1* + *t*)) *V*
    ((*λs. s* − *t′*) ‘ (*λf. f* − *Poly-Mapping.single x 1*) ‘ *S*)))
  and *3*: *set* (*snd* (*split* (*t′* + (*Poly-Mapping.single x 1* + *t*)) *V* ((*λs. s* − *t′*) ‘ (*λf. f* − *monomial 1 x*) ‘ *S*))) ⊆
    set (*snd* (*ps1*, *qs1*))
  using *step.hyps*(*16*) **by** *blast*
have *eq*: (*λs. s* − *t′*) ‘ (*λf. f* − *Poly-Mapping.single x 1*) ‘ *S* =

$(\lambda s.\ s - (t' + \textit{Poly-Mapping.single}\ x\ 1))$ ‘ S
by (*simp add*: *image-image add.commute diff-diff-eq*)
show *?thesis*
proof (*rule step.prems*)
from *step.hyps*(*8*, *1*) **have** $x \in X$ **..**
hence *Poly-Mapping.single* $x\ 1 \in .[X]$ **by** (*rule PPs-closed-single*)
with ‹$t' \in .[X]$› **show** $t' +$ *Poly-Mapping.single* $x\ 1 \in .[X]$ **by** (*rule PPs-closed-plus*)
next
from *1* **show** $0 \notin (\lambda s.\ s - (t' + \textit{Poly-Mapping.single}\ x\ 1))$ ‘ S
by (*simp only*: *eq not-False-eq-True*)
next
show $s = t' + \textit{Poly-Mapping.single}\ x\ 1 + t + \textit{Poly-Mapping.single}\ x'\ 1$ **by** (*simp only*: *s ac-simps*)
next
show (*monomial* (*1*::′*a*) *s*, *V*) ∈ *set* (*fst* (*split* (t' + *Poly-Mapping.single x 1* + *t*) *V*
(($\lambda s.\ s - (t' + \textit{Poly-Mapping.single}\ x\ 1))$ ‘ S)))
using *2* **by** (*simp only*: *eq add.assoc*)
next
have *set* (*snd* (*split* (t' + *Poly-Mapping.single x 1* + *t*) *V* (($\lambda s.\ s - (t' +$ *Poly-Mapping.single x 1*)) ‘ *S*))) ⊆
set (*snd* (*ps1*, *qs1*)) (**is** *?x* ⊆ -) **using** *3* **by** (*simp only*: *eq add.assoc*)
also have ... ⊆ *set* (*snd* (*ps0* @ *ps1*, *qs0* @ *qs1*)) **by** *simp*
finally show *?x* ⊆ *set* (*snd* (*ps0* @ *ps1*, *qs0* @ *qs1*)) **.**
qed *fact+*
qed
qed
qed

lemma *lem-4-8*:
assumes *finite X* **and** *finite S* **and** $S \subseteq .[X]$ **and** $0 \notin S$
and $g \in$ *punit.reduced-GB* (*monomial* (*1*::′*a*) ‘ *S*)
obtains *t U* **where** $U \subseteq X$ **and** (*monomial* (*1*::′*a*::*field*) *t*, *U*) ∈ *set* (*snd* (*split 0 X S*))
and *poly-deg g* = *Suc* (*deg-pm t*)
proof −
let *?S* = *monomial* (*1*::′*a*) ‘ *S*
let *?G* = *punit.reduced-GB ?S*
have *md1*: *monomial-decomp* (*fst* ((*split 0 X S*)::(- × (((- $\Rightarrow_0$ ′*a*) × -) *list*))))
and *md2*: *monomial-decomp* (*snd* ((*split 0 X S*)::(- × (((- $\Rightarrow_0$ ′*a*) × -) *list*))))
using *assms*(*1*) *subset-refl assms*(*2*) **by** (*rule monomial-decomp-split*)+
from *assms*(*3*) **have** *0*: *?S* ⊆ *P*[*X*] **by** (*auto intro*: *Polys-closed-monomial*)
with *assms*(*1*) **have** *punit.is-auto-reduced ?G* **and** *punit.is-monic-set ?G*
and *ideal-G*: *ideal ?G* = *ideal ?S* **and** $0 \notin$ *?G*
by (*rule reduced-GB-is-auto-reduced-Polys*, *rule reduced-GB-is-monic-set-Polys*,
rule reduced-GB-ideal-Polys, *rule reduced-GB-nonzero-Polys*)
from *this*(*2*, *4*) *assms*(*5*) **have** *punit.lc g* = *1* **by** (*auto simp*: *punit.is-monic-set-def*)

have *is-monomial-set ?S* **by** (*auto intro*!: *is-monomial-setI monomial-is-monomial*)
with *assms*(*1*) *0* **have** *is-monomial-set ?G* **by** (*rule reduced-GB-is-monomial-set-Polys*)
hence *is-monomial g* **using** *assms*(*5*) **by** (*rule is-monomial-setD*)
moreover define *s* **where** $s = lpp\ g$
ultimately have *g*: $g = monomial\ 1\ s$ **using** ‹*punit.lc g = 1*› **by** (*metis punit.monomial-eq-itself*)
note *assms*(*1*) *subset-refl assms*(*2*) *zero-in-PPs*
moreover have *ideal ?G* ÷ *monomial 1 0* = *ideal ?S* **by** (*simp add*: *ideal-G*)
ultimately have *splits-wrt* (*split 0 X S*) (*cone* (*monomial* (*1*::′*a*) *0*, *X*)) *?G* **by** (*rule split-splits-wrt*)
hence *splits-wrt* (*fst* (*split 0 X S*), *snd* (*split 0 X S*)) *P*[*X*] *?G* **by** *simp*
hence *cone-decomp* (*P*[*X*] ∩ *ideal ?G*) (*fst* (*split 0 X S*))
using *md2* ‹*is-monomial-set ?G*› **by** (*rule splits-wrt-cone-decomp-1*)
hence *cone-decomp* (*P*[*X*] ∩ *ideal ?S*) (*fst* (*split 0 X S*)) **by** (*simp only*: *ideal-G*)
with *assms*(*1*, *3*, *5*) **obtain** *U* **where** (*g*, *U*) ∈ *set* (*fst* (*split 0 X S*)) **using** *md1* **by** (*rule lem-4-7*)
with *assms*(*1*) *subset-refl assms*(*2*, *4*) **obtain** *t*′ *x* **where** *t*′ ∈ .[*X*] **and** *x* ∈ *X* **and** *U* ⊆ *X*
and *0* ∉ (λ*s*. *s* − *t*′) ‘ *S* **and** *s*: *s* = *t*′ + *0* + *Poly-Mapping.single x 1*
and (*g*, *U*) ∈ *set* (*fst* (*split* (*t*′ + *0*) *U* ((λ*s*. *s* − *t*′) ‘ *S*)))
and *set* (*snd* (*split* (*t*′ + *0*) *U* ((λ*s*. *s* − *t*′) ‘ *S*))) ⊆ (*set* (*snd* (*split 0 X S*)) :: ((- $\Rightarrow_0$ ′*a*) × -) *set*)
unfolding *g* **by** (*rule fst-splitE*)
let *?S* = (λ*s*. *s* − *t*′) ‘ *S*
from *assms*(*2*) **have** *finite ?S* **by** (*rule finite-imageI*)
with *assms*(*1*) ‹*U* ⊆ *X*› **obtain** *V* **where** *V* ⊆ *U*
and (*monomial* (*1*::′*a*) (*t*′ + *0*), *V*) ∈ *set* (*snd* (*split* (*t*′ + *0*) *U ?S*))
using ‹*0* ∉ *?S*› **by** (*rule snd-splitI*)
note *this*(*2*)
also have ... ⊆ *set* (*snd* (*split 0 X S*)) **by** *fact*
finally have (*monomial* (*1*::′*a*) *t*′, *V*) ∈ *set* (*snd* (*split 0 X S*)) **by** *simp*
have *poly-deg g* = *Suc* (*deg-pm t*′) **by** (*simp add*: *g s deg-pm-plus deg-pm-single poly-deg-monomial*)
from ‹*V* ⊆ *U*› ‹*U* ⊆ *X*› **have** *V* ⊆ *X* **by** (*rule subset-trans*)
show *?thesis* **by** *rule fact*+
qed

corollary *cor-4-9*:
assumes *finite X* **and** *finite S* **and** *S* ⊆ .[*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 (*cases 0* ∈ *S*)
case *True*
hence *1* ∈ *monomial* (*1*::′*a*) ‘ *S* **by** (*rule rev-image-eqI*) (*simp only*: *single-one*)
hence *1* ∈ *ideal* (*monomial* (*1*::′*a*) ‘ *S*) **by** (*rule ideal.span-base*)
hence *ideal* (*monomial* (*1*::′*a*) ‘ *S*) = *UNIV* **by** (*simp only*: *ideal-eq-UNIV-iff-contains-one*)
moreover from *assms*(*3*) **have** *monomial* (*1*::′*a*) ‘ *S* ⊆ *P*[*X*] **by** (*auto intro*:

Polys-closed-monomial)
ultimately have *punit.reduced-GB* (*monomial* ($1::'a$) ‘ S) = {*1*}
using *assms*(*1*) **by** (*simp only*: *ideal-eq-UNIV-iff-reduced-GB-eq-one-Polys*)
with *assms*(*4*) **show** *?thesis* **by** *simp*
next
case *False*
from *finite-set* **have** *fin*: *finite* (*poly-deg* ‘ *fst* ‘ *?S*) **by** (*intro finite-imageI*)
obtain *t U* **where** (*monomial 1 t*, *U*) ∈ *?S* **and** *g*: *poly-deg g* = *Suc* (*deg-pm t*)
using *assms*(*1*−*3*) *False assms*(*4*) **by** (*rule lem-4-8*)
from *this*(*1*) **have** *poly-deg* (*fst* (*monomial* ($1::'a$) *t*, *U*)) ∈ *poly-deg* ‘ *fst* ‘ *?S*
by (*intro imageI*)
hence *deg-pm t* ∈ *poly-deg* ‘ *fst* ‘ *?S* **by** (*simp add*: *poly-deg-monomial*)
with *fin* **have** *deg-pm t* ≤ *Max* (*poly-deg* ‘ *fst* ‘ *?S*) **by** (*rule Max-ge*)
thus *poly-deg g* ≤ *Suc* (*Max* (*poly-deg* ‘ *fst* ‘ *?S*)) **by** (*simp add*: *g*)
qed

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*)
using *assms*
proof (*induct t U S rule*: *split-induct*)
case (*base1 t U S*)
show *?case* **by** (*simp add*: *standard-decomp-Nil*)
next
case (*base2 t U S*)
have *deg-pm t* = *poly-deg* (*monomial* ($1::'a$) *t*) **by** (*simp add*: *poly-deg-monomial*)
thus *?case* **by** (*simp add*: *standard-decomp-singleton*)
next
case (*step t U S V x ps0 ps1 qs0 qs1*)
from *step.hyps*(*15*) *step.prems* **have** *qs0*: *standard-decomp* (*deg-pm t*) *qs0* **by** (*simp only*: *snd-conv*)
have ($\lambda s.\ s -$ *Poly-Mapping.single x 1*) ‘ $S \subseteq .[X]$
proof
fix *u*
assume $u \in$ ($\lambda s.\ s -$ *Poly-Mapping.single x 1*) ‘ *S*
then obtain *s* **where** $s \in S$ **and** *u*: $u = s -$ *Poly-Mapping.single x 1* **..**
from *this*(*1*) *step.prems*(*1*) **have** $s \in .[X]$ **..**
thus $u \in .[X]$ **unfolding** *u* **by** (*rule PPs-closed-minus*)
qed
moreover from - *step.prems*(*2*) **have** *Poly-Mapping.single x 1* + $t \in .[X]$
proof (*rule PPs-closed-plus*)
from *step.hyps*(*8*, *1*) **have** $x \in X$ **..**
thus *Poly-Mapping.single x 1* $\in .[X]$ **by** (*rule PPs-closed-single*)
qed
ultimately have *qs1*: *standard-decomp* (*Suc* (*deg-pm t*)) *qs1* **using** *step.hyps*(*16*)
by (*simp add*: *deg-pm-plus deg-pm-single*)
show *?case* **unfolding** *snd-conv*
proof (*rule standard-decompI*)
fix *h U0*

assume $(h,\ U0) \in$ *set* $((qs0\ @\ qs1)_+)$
hence $*$: $(h,\ U0) \in$ *set* $(qs0_+) \cup$ *set* $(qs1_+)$ **by** (*simp add*: *pos-decomp-append*)
thus *deg-pm t* $\leq$ *poly-deg h*
proof
assume $(h,\ U0) \in$ *set* $(qs0_+)$
with *qs0* **show** *?thesis* **by** (*rule standard-decompD*)
next
assume $(h,\ U0) \in$ *set* $(qs1_+)$
with *qs1* **have** *Suc* (*deg-pm t*) $\leq$ *poly-deg h* **by** (*rule standard-decompD*)
thus *?thesis* **by** *simp*
qed

fix *d*
assume *d1*: *deg-pm t* $\leq$ *d* **and** *d2*: *d* $\leq$ *poly-deg h*
from $*$ **show** $\exists\, t'\ U'.\ (t',\ U') \in$ *set* (*qs0 @ qs1*) $\wedge$ *poly-deg* $t' = d \wedge$ *card U0* $\leq$ *card* U'
proof
assume $(h,\ U0) \in$ *set* $(qs0_+)$
with *qs0* **obtain** $h'\ U'$ **where** $(h',\ U') \in$ *set qs0* **and** *poly-deg* $h' = d$ **and** *card U0* $\leq$ *card* U'
using *d1 d2* **by** (*rule standard-decompE*)
moreover from *this*(*1*) **have** $(h',\ U') \in$ *set* (*qs0 @ qs1*) **by** *simp*
ultimately show *?thesis* **by** *blast*
next
assume $(h,\ U0) \in$ *set* $(qs1_+)$
hence $(h,\ U0) \in$ *set qs1* **by** (*simp add*: *pos-decomp-def*)
from *assms*(*1*) *step.hyps*(*1*, *2*) **have** *monomial-decomp* (*snd* (*split t U S*) :: $((\text{-} \Rightarrow_0 {}'a) \times \text{-})$ *list*)
by (*rule monomial-decomp-split*)
hence *md*: *monomial-decomp* (*qs0 @ qs1*) **by** (*simp add*: *step.hyps*(*14*))
moreover from ‹$(h,\ U0) \in$ *set qs1*› **have** $(h,\ U0) \in$ *set* (*qs0 @ qs1*) **by** *simp*
ultimately have *is-monomial h* **and** *punit.lc h* = *1* **by** (*rule monomial-decompD*)+
moreover from *this*(*1*) **have** *monomial* (*punit.lc h*) (*lpp h*) = *h* **by** (*rule punit.monomial-eq-itself*)
moreover define *s* **where** *s* = *lpp h*
ultimately have *h*: *h* = *monomial 1 s* **by** *simp*
from *d1* **have** *deg-pm t* = *d* $\vee$ *Suc* (*deg-pm t*) $\leq$ *d* **by** *auto*
thus *?thesis*
proof
assume *deg-pm t* = *d*
define *F* **where** *F* = $(*)$ (*monomial 1 t*) ‘ *monomial* (*1*::$'a$) ‘ *S*
have $F \subseteq P[X]$
proof
fix *f*
assume $f \in F$
then obtain *u* **where** $u \in S$ **and** *f*: *f* = *monomial 1* (*t* + *u*)
by (*auto simp*: *F-def times-monomial-monomial*)
from *this*(*1*) *step.prems*(*1*) **have** $u \in .[X]$ **..**

with *step.prems*(*2*) **have** $t + u \in .[X]$ **by** (*rule PPs-closed-plus*)
thus $f \in P[X]$ **unfolding** *f* **by** (*rule Polys-closed-monomial*)
qed
have *ideal F* = (∗) (*monomial 1 t*) ‘ *ideal* (*monomial 1* ‘ *S*)
by (*simp only*: *ideal.span-image-scale-eq-image-scale F-def*)
moreover have *inj* ((∗) (*monomial* (*1*::′*a*) *t*))
by (*auto intro*!: *injI simp*: *times-monomial-left monomial-0-iff dest*!: *punit.monom-mult-inj-3*)
ultimately have *eq*: *ideal F* ÷ *monomial 1 t* = *ideal* (*monomial 1* ‘ *S*)
by (*simp only*: *quot-set-image-times*)
with *assms*(*1*) *step.hyps*(*1*, *2*) *step.prems*(*2*)
have *splits-wrt* (*split t U S*) (*cone* (*monomial* (*1*::′*a*) *t*, *U*)) *F* **by** (*rule split-splits-wrt*)
hence *splits-wrt* (*ps0* @ *ps1*, *qs0* @ *qs1*) (*cone* (*monomial 1 t*, *U*)) *F* **by** (*simp only*: *step.hyps*(*14*))
with *assms*(*1*) **have** *cone-decomp* (*cone* (*monomial* (*1*::′*a*) *t*, *U*) ∩ *normal-form F* ‘ *P*[*X*]) (*qs0* @ *qs1*)
using *md* - ‹$F \subseteq P[X]$›
by (*rule splits-wrt-cone-decomp-2*)
(*auto intro*!: *is-monomial-setI monomial-is-monomial simp*: *F-def times-monomial-monomial*)
hence *cone* (*monomial 1 s*, *U0*) ⊆ *cone* (*monomial* (*1*::′*a*) *t*, *U*) ∩ *normal-form F* ‘ *P*[*X*]
using ‹(*h*, *U0*) ∈ *set* (*qs0* @ *qs1*)› **unfolding** *h* **by** (*rule cone-decomp-cone-subset*)
with *assms*(*1*) *step.hyps*(*1*, *2*) *step.prems*(*2*) ‹$F \subseteq P[X]$› *eq*
obtain *U*′ **where** (*monomial* (*1*::′*a*) *t*, *U*′) ∈ *set* (*snd* (*split t U S*)) **and** *card U0* ≤ *card U*′
by (*rule lem-4-6*)
from *this*(*1*) **have** (*monomial 1 t*, *U*′) ∈ *set* (*qs0* @ *qs1*) **by** (*simp add*: *step.hyps*(*14*))
show *?thesis*
proof (*intro exI conjI*)
show *poly-deg* (*monomial* (*1*::′*a*) *t*) = *d* **by** (*simp add*: *poly-deg-monomial* ‹*deg-pm t* = *d*›)
qed *fact*+
next
assume *Suc* (*deg-pm t*) ≤ *d*
with *qs1* ‹(*h*, *U0*) ∈ *set* ($qs1_+$)› **obtain** *h*′ *U*′ **where** (*h*′, *U*′) ∈ *set qs1* **and** *poly-deg h*′ = *d*
and *card U0* ≤ *card U*′ **using** *d2* **by** (*rule standard-decompE*)
moreover from *this*(*1*) **have** (*h*′, *U*′) ∈ *set* (*qs0* @ *qs1*) **by** *simp*
ultimately show *?thesis* **by** *blast*
qed
qed
qed
qed

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

defines $G \equiv$ *punit.reduced-GB F*
defines *ss* $\equiv$ (*split 0 X* (*lpp* ‘ *G*)) :: ((- $\Rightarrow_0$ *'a::field*) $\times$ -) *list* $\times$ -
defines $d \equiv$ *Suc* (*Max* (*poly-deg* ‘ *fst* ‘ *set* (*snd ss*)))
assumes *finite X* **and** $F \subseteq P[X]$
shows *standard-decomp 0* (*snd ss*) (**is** *?thesis1*)
and *cone-decomp* (*normal-form F* ‘ *P*[*X*]) (*snd ss*) (**is** *?thesis2*)
and ($\bigwedge f.\ f \in F \Longrightarrow$ *homogeneous f*) $\Longrightarrow g \in G \Longrightarrow$ *poly-deg g* $\leq d$
proof −
have *ideal G* = *ideal F* **and** *punit.is-Groebner-basis G* **and** *finite G* **and** $0 \notin G$
and $G \subseteq P[X]$ **and** *punit.is-reduced-GB G* **using** *assms*(*4*, *5*) **unfolding** *G-def*
by (*rule reduced-GB-ideal-Polys*, *rule reduced-GB-is-GB-Polys*, *rule finite-reduced-GB-Polys*,
rule reduced-GB-nonzero-Polys, *rule reduced-GB-Polys*, *rule reduced-GB-is-reduced-GB-Polys*)
define *S* **where** *S* = *lpp* ‘ *G*
note *assms*(*4*) *subset-refl*
moreover from ‹*finite G*› **have** *finite S* **unfolding** *S-def* **by** (*rule finite-imageI*)
moreover from ‹$G \subseteq P[X]$› **have** $S \subseteq .[X]$ **unfolding** *S-def* **by** (*rule PPs-closed-image-lpp*)
ultimately have *standard-decomp* (*deg-pm* (*0::'x* $\Rightarrow_0$ *nat*)) (*snd ss*)
using *zero-in-PPs* **unfolding** *ss-def S-def* **by** (*rule standard-decomp-snd-split*)
thus *?thesis1* **by** *simp*

let *?S* = *monomial* (*1::'a*) ‘ *S*
from ‹$S \subseteq .[X]$› **have** *?S* $\subseteq P[X]$ **by** (*auto intro*: *Polys-closed-monomial*)
have *splits-wrt ss* (*cone* (*monomial 1 0*, *X*)) *?S*
using *assms*(*4*) *subset-refl* ‹*finite S*› *zero-in-PPs* **unfolding** *ss-def S-def*
by (*rule split-splits-wrt*) *simp*
hence *splits-wrt* (*fst ss*, *snd ss*) *P*[*X*] *?S* **by** *simp*
with *assms*(*4*) **have** *cone-decomp* ($P[X] \cap$ *normal-form ?S* ‘ *P*[*X*]) (*snd ss*)
using - - ‹*?S* $\subseteq P[X]$›
proof (*rule splits-wrt-cone-decomp-2*)
from *assms*(*4*) *subset-refl* ‹*finite S*› **show** *monomial-decomp* (*snd ss*)
unfolding *ss-def S-def* **by** (*rule monomial-decomp-split*)
qed (*auto intro*!: *is-monomial-setI monomial-is-monomial*)
moreover have *normal-form ?S* ‘ *P*[*X*] = *normal-form F* ‘ *P*[*X*]
by (*rule set-eqI*)
(*simp add*: *image-normal-form-iff*[*OF assms*(*4*)] *assms*(*5*) ‹*?S* $\subseteq P[X]$›,
simp add: *S-def is-red-reduced-GB-monomial-lt-GB-Polys*[*OF assms*(*4*)] ‹*G*
$\subseteq P[X]$› ‹$0 \notin G$› *flip*: *G-def*)
moreover from *assms*(*4*, *5*) **have** *normal-form F* ‘ *P*[*X*] $\subseteq P[X]$
by (*auto intro*: *Polys-closed-normal-form*)
ultimately show *?thesis2* **by** (*simp only*: *Int-absorb1*)

assume $\bigwedge f.\ f \in F \Longrightarrow$ *homogeneous f*
moreover note ‹*punit.is-reduced-GB G*› ‹*ideal G* = *ideal F*›
moreover assume $g \in G$
ultimately have *homogeneous g* **by** (*rule is-reduced-GB-homogeneous*)
moreover have *lpp g* $\in$ *keys g*
proof (*rule punit.lt-in-keys*)
from ‹$g \in G$› ‹$0 \notin G$› **show** $g \neq 0$ **by** *blast*
qed

ultimately have *deg-lt*: *deg-pm* (*lpp g*) = *poly-deg g* **by** (*rule homogeneousD-poly-deg*)
from ‹$g \in G$› **have** *monomial 1* (*lpp g*) $\in$ *?S* **unfolding** *S-def* **by** (*intro imageI*)
also have ... = *punit.reduced-GB ?S* **unfolding** *S-def G-def* **using** *assms*(*4*, *5*)
by (*rule reduced-GB-monomial-lt-reduced-GB-Polys*[*symmetric*])
finally have *monomial 1* (*lpp g*) $\in$ *punit.reduced-GB ?S* **.**
with *assms*(*4*) ‹*finite S*› ‹$S \subseteq .[X]$› **have** *poly-deg* (*monomial* (*1*::'*a*) (*lpp g*)) $\leq$ *d*
unfolding *d-def ss-def S-def*[*symmetric*] **by** (*rule cor-4-9*)
thus *poly-deg g* $\leq$ *d* **by** (*simp add*: *poly-deg-monomial deg-lt*)
qed

10.7 Splitting Ideals

qualified definition *ideal-decomp-aux* :: (('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*) *set* $\Rightarrow$ (('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*) $\Rightarrow$
((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*::*field*) *set* $\times$ ((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*) $\times$ '*x set*) *list*)
where *ideal-decomp-aux F f* =
(*let J* = *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 –
define *J* **where** *J* = *ideal F*
define *L* **where** *L* = ($J \div f$) $\cap$ *P*[*X*]
define *S* **where** *S* = ($*$) *f* ‘ *normal-form L* ‘ *P*[*X*]
define *L'* **where** *L'* = *lpp* ‘ *punit.reduced-GB L*

have *eq*: *ideal-decomp-aux F f* = (*S*, *map* (*apfst* (($*$) *f*)) (*snd* (*split 0 X L'*)))
by (*simp add*: *J-def ideal-decomp-aux-def Let-def L-def L'-def S-def*)

have *L-sub*: $L \subseteq P[X]$ **by** (*simp add*: *L-def*)

show *?thesis1* **unfolding** *eq fst-conv*
proof
fix *s*
assume $s \in S$
then obtain *q* **where** $s = \textit{normal-form } L\ q * f$ **unfolding** *S-def* **by** (*elim imageE*) *auto*
also have $\ldots \in \textit{ideal } \{f\}$ **by** (*intro ideal.span-scale ideal.span-base singletonI*)
finally show $s \in \textit{ideal } \{f\}$ **.**
qed

show *?thesis2*
proof (*rule set-eqI*)
fix *h*
show $h \in \textit{ideal } F \cap \textit{fst } (\textit{ideal-decomp-aux } F\ f) \longleftrightarrow h \in \{0\}$
proof
assume $h \in \textit{ideal } F \cap \textit{fst } (\textit{ideal-decomp-aux } F\ f)$
hence $h \in J$ **and** $h \in S$ **by** (*simp-all add*: *J-def S-def eq*)
from *this*(*2*) **obtain** *q* **where** $q \in P[X]$ **and** *h*: $h = f * \textit{normal-form } L\ q$ **by** (*auto simp*: *S-def*)
from *fin-X L-sub this*(*1*) **have** $\textit{normal-form } L\ q \in P[X]$ **by** (*rule Polys-closed-normal-form*)
moreover from ‹$h \in J$› **have** $f * \textit{normal-form } L\ q \in J$ **by** (*simp add*: *h*)
ultimately have $\textit{normal-form } L\ q \in L$ **by** (*simp add*: *L-def quot-set-iff*)
hence $\textit{normal-form } L\ q \in \textit{ideal } L$ **by** (*rule ideal.span-base*)
with *normal-form-diff-in-ideal*[*OF fin-X L-sub*] **have** $(q - \textit{normal-form } L\ q) + \textit{normal-form } L\ q \in \textit{ideal } L$
by (*rule ideal.span-add*)
hence $\textit{normal-form } L\ q = 0$ **using** *fin-X L-sub* **by** (*simp add*: *normal-form-zero-iff*)
thus $h \in \{0\}$ **by** (*simp add*: *h*)
next
assume $h \in \{0\}$
moreover have $0 \in (*)\ f\ `\ \textit{normal-form } L\ `\ P[X]$
proof (*intro image-eqI*)
from *fin-X L-sub* **show** $0 = \textit{normal-form } L\ 0$ **by** (*simp only*: *normal-form-zero*)
qed (*simp-all add*: *zero-in-Polys*)
ultimately show $h \in \textit{ideal } F \cap \textit{fst } (\textit{ideal-decomp-aux } F\ f)$ **by** (*simp add*: *ideal.span-zero eq S-def*)
qed
qed

have $\textit{direct-decomp } (\textit{ideal } (\textit{insert } f\ F) \cap P[X])\ [\textit{ideal } F \cap P[X], \textit{fst } (\textit{ideal-decomp-aux } F\ f)]$
unfolding *eq fst-conv S-def L-def J-def* **using** *fin-X assms*(*2*, *3*) **by** (*rule direct-decomp-ideal-insert*)
thus *?thesis3* **by** (*rule direct-decomp-perm*) *simp*

have *std*: $\textit{standard-decomp } 0\ (\textit{snd } (\textit{split } 0\ X\ L') :: ((\text{-} \Rightarrow_0 {}'a) \times \text{-})\ \textit{list})$
and $\textit{cone-decomp } (\textit{normal-form } L\ `\ P[X])\ (\textit{snd } (\textit{split } 0\ X\ L'))$

unfolding L'-*def* **using** *fin-X* ‹$L \subseteq P[X]$› **by** (*rule standard-cone-decomp-snd-split*)+
from *this*(*2*) **show** *?thesis4* **unfolding** *eq fst-conv snd-conv S-def* **by** (*rule cone-decomp-map-times*)

from *fin-X* ‹$L \subseteq P[X]$› **have** *finite* (*punit.reduced-GB L*) **by** (*rule finite-reduced-GB-Polys*)
hence *finite* L' **unfolding** L'-*def* **by** (*rule finite-imageI*)
{
have *monomial-decomp* (*snd* (*split 0 X* L') :: ((- $\Rightarrow_0$ $'a$) $\times$ -) *list*)
using *fin-X subset-refl* ‹*finite* L'› **by** (*rule monomial-decomp-split*)
hence *hom-decomp* (*snd* (*split 0 X* L') :: ((- $\Rightarrow_0$ $'a$) $\times$ -) *list*)
by (*rule monomial-decomp-imp-hom-decomp*)
moreover assume *homogeneous f*
ultimately show *?thesis7* **unfolding** *eq snd-conv* **by** (*rule hom-decomp-map-times*)
}

have *vd*: *valid-decomp X* (*snd* (*split 0 X* L') :: ((- $\Rightarrow_0$ $'a$) $\times$ -) *list*)
using *fin-X subset-refl* ‹*finite* L'› *zero-in-PPs* **by** (*rule valid-decomp-split*)
moreover note *assms*(*3*)
moreover assume $f \neq 0$
ultimately show *?thesis5* **unfolding** *eq snd-conv* **by** (*rule valid-decomp-map-times*)

from *std vd* ‹$f \neq 0$› **have** *standard-decomp* (*0* + *poly-deg f*) (*map* (*apfst* ((*) *f*)) (*snd* (*split 0 X* L')))
by (*rule standard-decomp-map-times*)
thus *?thesis6* **by** (*simp add*: *eq*)
qed

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*]
using *assms*(*1*, *2*, *4*)
proof (*induct F arbitrary*: *thesis*)
case *empty*
show *?case*
proof (*rule empty.prems*)
show *valid-decomp X* [] **by** (*rule valid-decompI*) *simp-all*
next
show *standard-decomp* (*poly-deg f0*) [] **by** (*rule standard-decompI*) *simp-all*
next
show *cone-decomp* {*0*} [] **by** (*rule cone-decompI*) (*simp add*: *direct-decomp-def bij-betw-def*)
next
have *direct-decomp* (*ideal* {*f0*} $\cap$ $P[X]$) [*ideal* {*f0*} $\cap$ $P[X]$]
by (*fact direct-decomp-singleton*)

hence *direct-decomp* (*ideal* {*f0*} ∩ *P*[*X*]) [{*0*}, *ideal* {*f0*} ∩ *P*[*X*]] **by** (*rule direct-decomp-Cons-zeroI*)
thus *direct-decomp* (*ideal* {*f0*} ∩ *P*[*X*]) [*ideal* {*f0*} ∩ *P*[*X*], {*0*}]
by (*rule direct-decomp-perm*) *simp*
qed (*simp add*: *hom-decomp-def*)
next
case (*insert f F*)
from *insert.prems*(*2*) **have** *F* ⊆ *P*[*X*] **by** *simp*
moreover have *poly-deg f′* ≤ *poly-deg f0* **if** *f′* ∈ *F* **for** *f′*
proof −
from *that* **have** *f′* ∈ *insert f F* **by** *simp*
thus *?thesis* **by** (*rule insert.prems*)
qed
ultimately obtain *T ps* **where** *valid-ps*: *valid-decomp X ps* **and** *std-ps*: *standard-decomp* (*poly-deg f0*) *ps*
and *cn-ps*: *cone-decomp T ps* **and** *dd*: *direct-decomp* (*ideal* (*insert f0 F*) ∩ *P*[*X*]) [*ideal* {*f0*} ∩ *P*[*X*], *T*]
and *hom-ps*: (⋀*f*. *f* ∈ *F* ⟹ *homogeneous f*) ⟹ *hom-decomp ps*
using *insert.hyps*(*3*) **by** *metis*
show *?case*
proof (*cases f* = *0*)
case *True*
show *?thesis*
proof (*rule insert.prems*)
from *dd* **show** *direct-decomp* (*ideal* (*insert f0* (*insert f F*)) ∩ *P*[*X*]) [*ideal* {*f0*} ∩ *P*[*X*], *T*]
by (*simp only*: *insert-commute*[*of f0*] *True ideal.span-insert-zero*)
next
assume ⋀*f′*. *f′* ∈ *insert f F* ⟹ *homogeneous f′*
hence ⋀*f*. *f* ∈ *F* ⟹ *homogeneous f* **by** *blast*
thus *hom-decomp ps* **by** (*rule hom-ps*)
qed *fact+*
next
case *False*
let *?D* = *ideal-decomp-aux* (*insert f0 F*) *f*
from *insert.hyps*(*1*) **have** *f0F-fin*: *finite* (*insert f0 F*) **by** *simp*
moreover from ‹*F* ⊆ *P*[*X*]› *assms*(*3*) **have** *f0F-sub*: *insert f0 F* ⊆ *P*[*X*] **by** *simp*
moreover from *insert.prems*(*2*) **have** *f* ∈ *P*[*X*] **by** *simp*
ultimately have *eq*: *ideal* (*insert f0 F*) ∩ *fst ?D* = {*0*} **and** *valid-decomp X* (*snd ?D*)
and *cn-D*: *cone-decomp* (*fst ?D*) (*snd ?D*)
and *standard-decomp* (*poly-deg f*) (*snd ?D*)
and *dd′*: *direct-decomp* (*ideal* (*insert f* (*insert f0 F*)) ∩ *P*[*X*])
[*fst ?D*, *ideal* (*insert f0 F*) ∩ *P*[*X*]]
and *hom-D*: *homogeneous f* ⟹ *hom-decomp* (*snd ?D*)
by (*rule ideal-decomp-aux*, *auto intro*: *ideal-decomp-aux simp*: *False*)
note *fin-X this*(*2*−*4*)
moreover have *poly-deg f* ≤ *poly-deg f0* **by** (*rule insert.prems*) *simp*

ultimately obtain *qs* **where** *valid-qs*: *valid-decomp X qs* **and** *cn-qs*: *cone-decomp* (*fst ?D*) *qs*
and *std-qs*: *standard-decomp* (*poly-deg f0*) *qs*
and *hom-qs*: *hom-decomp* (*snd ?D*) $\Longrightarrow$ *hom-decomp qs* **by** (*rule standard-decomp-geE*) *blast*
let *?T* = *sum-list* ' *listset* [*T*, *fst ?D*]
let *?ps* = *ps* @ *qs*
show *?thesis*
proof (*rule insert.prems*)
from *valid-ps valid-qs* **show** *valid-decomp X ?ps* **by** (*rule valid-decomp-append*)
next
from *std-ps std-qs* **show** *standard-decomp* (*poly-deg f0*) *?ps* **by** (*rule standard-decomp-append*)
next
from *dd* **have** *direct-decomp* (*ideal* (*insert f0 F*) $\cap$ *P*[*X*]) [*T*, *ideal* {*f0*} $\cap$ *P*[*X*]]
by (*rule direct-decomp-perm*) *simp*
hence *T* $\subseteq$ *ideal* (*insert f0 F*) $\cap$ *P*[*X*]
by (*rule direct-decomp-Cons-subsetI*) (*simp add*: *ideal.span-zero zero-in-Polys*)
hence *T* $\cap$ *fst ?D* $\subseteq$ *ideal* (*insert f0 F*) $\cap$ *fst ?D* **by** *blast*
hence *T* $\cap$ *fst ?D* $\subseteq$ {*0*} **by** (*simp only*: *eq*)
from *refl* **have** *direct-decomp ?T* [*T*, *fst ?D*]
proof (*intro direct-decompI inj-onI*)
fix *xs ys*
assume *xs* $\in$ *listset* [*T*, *fst ?D*]
then obtain *x1 x2* **where** *x1* $\in$ *T* **and** *x2* $\in$ *fst ?D* **and** *xs*: *xs* = [*x1*, *x2*]
by (*rule listset-doubletonE*)
assume *ys* $\in$ *listset* [*T*, *fst ?D*]
then obtain *y1 y2* **where** *y1* $\in$ *T* **and** *y2* $\in$ *fst ?D* **and** *ys*: *ys* = [*y1*, *y2*]
by (*rule listset-doubletonE*)
assume *sum-list xs* = *sum-list ys*
hence *x1* − *y1* = *y2* − *x2* **by** (*simp add*: *xs ys*) (*metis add-diff-cancel-left add-diff-cancel-right*)
moreover from *cn-ps* ‹*x1* $\in$ *T*› ‹*y1* $\in$ *T*› **have** *x1* − *y1* $\in$ *T* **by** (*rule cone-decomp-closed-minus*)
moreover from *cn-D* ‹*y2* $\in$ *fst ?D*› ‹*x2* $\in$ *fst ?D*› **have** *y2* − *x2* $\in$ *fst ?D*
by (*rule cone-decomp-closed-minus*)
ultimately have *y2* − *x2* $\in$ *T* $\cap$ *fst ?D* **by** *simp*
also have ... $\subseteq$ {*0*} **by** *fact*
finally have *x2* = *y2* **by** *simp*
with ‹*x1* − *y1* = *y2* − *x2*› **show** *xs* = *ys* **by** (*simp add*: *xs ys*)
qed
thus *cone-decomp ?T ?ps* **using** *cn-ps cn-qs* **by** (*rule cone-decomp-append*)
next
assume $\bigwedge f'$. $f' \in$ *insert f F* $\Longrightarrow$ *homogeneous* f'
hence *homogeneous f* **and** $\bigwedge f'$. $f' \in F \Longrightarrow$ *homogeneous* f' **by** *blast+*
from *this*(*2*) **have** *hom-decomp ps* **by** (*rule hom-ps*)
moreover from ‹*homogeneous f*› **have** *hom-decomp qs* **by** (*intro hom-qs hom-D*)

ultimately show *hom-decomp* (*ps* @ *qs*) **by** (*simp only*: *hom-decomp-append-iff*)
next
from *dd'* **have** *direct-decomp* (*ideal* (*insert f0* (*insert f F*)) $\cap$ $P[X]$)
[*ideal* (*insert f0 F*) $\cap$ $P[X]$, *fst ?D*]
by (*simp add*: *insert-commute direct-decomp-perm*)
hence *direct-decomp* (*ideal* (*insert f0* (*insert f F*)) $\cap$ $P[X]$)
([*fst ?D*] @ [*ideal* {*f0*} $\cap$ $P[X]$, *T*]) **using** *dd* **by** (*rule direct-decomp-direct-decomp*)
hence *direct-decomp* (*ideal* (*insert f0* (*insert f F*)) $\cap$ $P[X]$) ([*ideal* {*f0*} $\cap$ $P[X]$] @ [*T*, *fst ?D*])
by (*rule direct-decomp-perm*) *auto*
hence *direct-decomp* (*ideal* (*insert f0* (*insert f F*)) $\cap$ $P[X]$) [*sum-list* ' *listset* [*ideal* {*f0*} $\cap$ $P[X]$], *?T*]
by (*rule direct-decomp-appendD*)
thus *direct-decomp* (*ideal* (*insert f0* (*insert f F*)) $\cap$ $P[X]$) [*ideal* {*f0*} $\cap$ $P[X]$, *?T*]
by (*simp add*: *image-image*)
qed
qed
qed

10.8 Exact Cone Decompositions

definition *exact-decomp* :: $nat \Rightarrow ((('x \Rightarrow_0 nat) \Rightarrow_0 'a::zero) \times 'x\ set)\ list \Rightarrow bool$
where *exact-decomp m ps* $\longleftrightarrow$ $(\forall (h, U) \in set\ ps.\ h \in P[X] \land U \subseteq X) \land$
$(\forall (h, U) \in set\ ps.\ \forall (h', U') \in set\ ps.\ poly\text{-}deg\ h = poly\text{-}deg\ h' \longrightarrow$
$m < card\ U \longrightarrow m < card\ U' \longrightarrow (h, U) = (h', U'))$

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

lemma *exact-decompD*:
assumes *exact-decomp m ps* **and** $(h, U) \in set\ ps$
shows $h \in P[X]$ **and** $U \subseteq X$
and $(h', U') \in set\ ps \Longrightarrow poly\text{-}deg\ h = poly\text{-}deg\ h' \Longrightarrow m < card\ U \Longrightarrow m < card\ U' \Longrightarrow$
$(h, U) = (h', U')$
using *assms* **unfolding** *exact-decomp-def* **by** *fastforce+*

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 $\bigwedge h\ h'\ U\ U'.\ (h,\ U) \in set\ (ps_+) \Longrightarrow (h',\ U') \in set\ (ps_+) \Longrightarrow poly\text{-}deg\ h = poly\text{-}deg\ h' \Longrightarrow$
$(h,\ U) = (h',\ U')$
shows *exact-decomp 0 ps*
using *assms*(*1*, *2*)
proof (*rule exact-decompI*)
fix $h\ h'$ **and** $U\ U' :: 'x\ set$
assume $0 < card\ U$
hence $U \neq \{\}$ **by** *auto*
moreover assume $(h,\ U) \in set\ ps$
ultimately have $(h,\ U) \in set\ (ps_+)$ **by** (*simp add*: *pos-decomp-def*)
assume $0 < card\ U'$
hence $U' \neq \{\}$ **by** *auto*
moreover assume $(h',\ U') \in set\ ps$
ultimately have $(h',\ U') \in set\ (ps_+)$ **by** (*simp add*: *pos-decomp-def*)
assume $poly\text{-}deg\ h = poly\text{-}deg\ h'$
with ‹$(h,\ U) \in set\ (ps_+)$› ‹$(h',\ U') \in set\ (ps_+)$› **show** $(h,\ U) = (h',\ U')$ **by** (*rule assms*(*3*))
qed

lemma *exact-decompD-zero*:
assumes *exact-decomp 0 ps* **and** $(h,\ U) \in set\ (ps_+)$ **and** $(h',\ U') \in set\ (ps_+)$
and $poly\text{-}deg\ h = poly\text{-}deg\ h'$
shows $(h,\ U) = (h',\ U')$
proof −
from *assms*(*2*) **have** $(h,\ U) \in set\ ps$ **and** $U \neq \{\}$ **by** (*simp-all add*: *pos-decomp-def*)
from *assms*(*1*) *this*(*1*) **have** $U \subseteq X$ **by** (*rule exact-decompD*)
hence *finite U* **using** *fin-X* **by** (*rule finite-subset*)
with ‹$U \neq \{\}$› **have** $0 < card\ U$ **by** (*simp add*: *card-gt-0-iff*)
from *assms*(*3*) **have** $(h',\ U') \in set\ ps$ **and** $U' \neq \{\}$ **by** (*simp-all add*: *pos-decomp-def*)
from *assms*(*1*) *this*(*1*) **have** $U' \subseteq X$ **by** (*rule exact-decompD*)
hence *finite* U' **using** *fin-X* **by** (*rule finite-subset*)
with ‹$U' \neq \{\}$› **have** $0 < card\ U'$ **by** (*simp add*: *card-gt-0-iff*)
show *?thesis* **by** (*rule exact-decompD*) *fact+*
qed

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 (*rule valid-decompI*)
fix *h U*
assume $*$: $(h,\ U) \in set\ ps$
with *assms*(*1*) **show** $h \in P[X]$ **and** $U \subseteq X$ **by** (*rule exact-decompD*)+
from $*$ **show** $h \neq 0$ **by** (*rule assms*(*2*))
qed

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

shows *exact-decomp m ps*
proof (*rule exact-decompI*)
fix *h U*
assume $(h,\ U) \in set\ ps$
with *assms*(*1*) **show** $h \in P[X]$ **and** $U \subseteq X$ **by** (*rule valid-decompD*)+
next
fix *h1 h2 U1 U2*
assume $(h1,\ U1) \in set\ ps$
with *assms*(*1*) **have** $U1 \subseteq X$ **by** (*rule valid-decompD*)
with *fin-X* **have** $card\ U1 \leq card\ X$ **by** (*rule card-mono*)
also have $\ldots \leq m$ **by** (*fact assms*(*2*))
also assume $m < card\ U1$
finally show $(h1,\ U1) = (h2,\ U2)$ **by** *simp*
qed

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

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

lemma a: $standard\text{-}decomp\ k\ ps \Longrightarrow standard\text{-}decomp$ (a *ps*) *ps*
unfolding a-*def* **by** (*rule LeastI*)

lemma a-*Nil*:
assumes $ps_+ = []$
shows a $ps = 0$
proof −
from *assms* **have** *standard-decomp 0 ps* **by** (*rule standard-decomp-Nil*)
thus *?thesis* **unfolding** a-*def* **by** (*rule Least-eq-0*)
qed

lemma a-*nonempty*:
assumes *valid-decomp X ps* **and** *standard-decomp k ps* **and** $ps_+ \neq []$
shows a $ps = Min\ (poly\text{-}deg\ `\ fst\ `\ set\ (ps_+))$
using *fin-X assms*(*1*) - *assms*(*3*)
proof (*rule standard-decomp-nonempty-unique*)
from *assms*(*2*) **show** *standard-decomp* (a *ps*) *ps* **by** (*rule* a)
qed

lemma a-*nonempty-unique*:
assumes *valid-decomp X ps* **and** *standard-decomp k ps* **and** $ps_+ \neq []$
shows a $ps = k$
proof −
from *assms* **have** a $ps = Min\ (poly\text{-}deg\ `\ fst\ `\ set\ (ps_+))$ **by** (*rule* a-*nonempty*)
moreover from *fin-X assms* **have** $k = Min\ (poly\text{-}deg\ `\ fst\ `\ set\ (ps_+))$
by (*rule standard-decomp-nonempty-unique*)
ultimately show *?thesis* **by** *simp*

qed

lemma b:
  shows a *ps* ≤ b *ps* *i* **and** (*h*, *U*) ∈ *set ps* ⟹ *i* ≤ *card U* ⟹ *poly-deg h* < b *ps*
i
proof −
  let *?A* = *poly-deg* ' *fst* ' *set ps*
  define *A* **where** *A* = *insert* (a *ps*) *?A*
  define *m* **where** *m* = *Suc* (*Max A*)
  from *finite-set* **have** *finite ?A* **by** (*intro finite-imageI*)
  hence *finite A* **by** (*simp add*: *A-def*)
  have a *ps* ≤ b *ps* *i* ∧ (∀ (*h'*, *U'*)∈*set ps*. *i* ≤ *card U'* ⟶ *poly-deg h'* < b *ps* *i*)
unfolding b-*def*
  proof (*rule LeastI*)
    have a *ps* ∈ *A* **by** (*simp add*: *A-def*)
    with ‹*finite A*› **have** a *ps* ≤ *Max A* **by** (*rule Max-ge*)
    hence a *ps* ≤ *m* **by** (*simp add*: *m-def*)
    moreover {
      fix *h U*
      assume (*h*, *U*) ∈ *set ps*
      hence *poly-deg* (*fst* (*h*, *U*)) ∈ *?A* **by** (*intro imageI*)
      hence *poly-deg h* ∈ *A* **by** (*simp add*: *A-def*)
      with ‹*finite A*› **have** *poly-deg h* ≤ *Max A* **by** (*rule Max-ge*)
      hence *poly-deg h* < *m* **by** (*simp add*: *m-def*)
    }
    ultimately show a *ps* ≤ *m* ∧ (∀ (*h*, *U*)∈*set ps*. *i* ≤ *card U* ⟶ *poly-deg h* <
m) **by** *blast*
  qed
  thus a *ps* ≤ b *ps* *i* **and** (*h*, *U*) ∈ *set ps* ⟹ *i* ≤ *card U* ⟹ *poly-deg h* < b *ps* *i*
by *blast+*
qed

lemma b-*le*:
  a *ps* ≤ *d* ⟹ (⋀*h' U'*. (*h'*, *U'*) ∈ *set ps* ⟹ *i* ≤ *card U'* ⟹ *poly-deg h'* < *d*)
⟹ b *ps* *i* ≤ *d*
  unfolding b-*def* **by** (*intro Least-le*) *blast*

lemma b-*decreasing*:
  assumes *i* ≤ *j*
  shows b *ps* *j* ≤ b *ps* *i*
proof (*rule* b-*le*)
  fix *h U*
  assume (*h*, *U*) ∈ *set ps*
  assume *j* ≤ *card U*
  with *assms*(*1*) **have** *i* ≤ *card U* **by** (*rule le-trans*)
  with ‹(*h*, *U*) ∈ *set ps*› **show** *poly-deg h* < b *ps* *i* **by** (*rule* b)
qed (*fact* b)

lemma b-*Nil*:

assumes $ps_+ = []$ **and** *Suc 0* $\leq$ *i*
shows b *ps i* = *0*
unfolding b-*def*
proof (*rule Least-eq-0*)
from *assms*(*1*) **have** a *ps* = *0* **by** (*rule* a-*Nil*)
moreover **{**
fix *h* **and** *U*::′*x set*
note *assms*(*2*)
also assume *i* $\leq$ *card U*
finally have *U* $\neq$ {} **by** *auto*
moreover assume (*h*, *U*) $\in$ *set ps*
ultimately have (*h*, *U*) $\in$ *set* (ps_+) **by** (*simp add*: *pos-decomp-def*)
hence *False* **by** (*simp add*: *assms*)
}
ultimately show a *ps* $\leq$ *0* $\wedge$ ($\forall$(*h*, *U*)$\in$*set ps*. *i* $\leq$ *card U* $\longrightarrow$ *poly-deg h* < *0*)
by *blast*
qed

lemma b-*zero*:
assumes *ps* $\neq$ []
shows *Suc* (*Max* (*poly-deg* ‘ *fst* ‘ *set ps*)) $\leq$ b *ps 0*
proof −
from *finite-set* **have** *finite* (*poly-deg* ‘ *fst* ‘ *set ps*) **by** (*intro finite-imageI*)
moreover from *assms* **have** *poly-deg* ‘ *fst* ‘ *set ps* $\neq$ {} **by** *simp*
moreover have $\forall$ *a*$\in$*poly-deg* ‘ *fst* ‘ *set ps*. *a* < b *ps 0*
proof
fix *d*
assume *d* $\in$ *poly-deg* ‘ *fst* ‘ *set ps*
then obtain *p* **where** *p* $\in$ *set ps* **and** *d* = *poly-deg* (*fst p*) **by** *blast*
moreover obtain *h U* **where** *p* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
ultimately have (*h*, *U*) $\in$ *set ps* **and** *d*: *d* = *poly-deg h* **by** *simp-all*
from *this*(*1*) *le0* **show** *d* < b *ps 0* **unfolding** *d* **by** (*rule* b)
qed
ultimately have *Max* (*poly-deg* ‘ *fst* ‘ *set ps*) < b *ps 0* **by** *simp*
thus *?thesis* **by** *simp*
qed

corollary b-*zero-gr*:
assumes (*h*, *U*) $\in$ *set ps*
shows *poly-deg h* < b *ps 0*
proof −
have *poly-deg h* $\leq$ *Max* (*poly-deg* ‘ *fst* ‘ *set ps*)
proof (*rule Max-ge*)
from *finite-set* **show** *finite* (*poly-deg* ‘ *fst* ‘ *set ps*) **by** (*intro finite-imageI*)
next
from *assms* **have** *poly-deg* (*fst* (*h*, *U*)) $\in$ *poly-deg* ‘ *fst* ‘ *set ps* **by** (*intro imageI*)
thus *poly-deg h* $\in$ *poly-deg* ‘ *fst* ‘ *set ps* **by** *simp*
qed
also have ... < *Suc* ... **by** *simp*

also have ... $\leq$ b *ps 0*
proof (*rule* b-*zero*)
from *assms* **show** $ps \neq []$ **by** *auto*
qed
finally show *?thesis* **.**
qed

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-deg* ' *fst* ' *set* (ps_+))))
proof (*cases* $ps_+ = []$)
case *True*
hence b *ps* (*Suc 0*) = *0* **using** *le-refl* **by** (*rule* b-*Nil*)
with *True* **show** *?thesis* **by** *simp*
next
case *False*
with *assms* **have** *aP*: a *ps* = *Min* (*poly-deg* ' *fst* ' *set* (ps_+)) (**is** - = *Min ?A*) **by** (*rule* a-*nonempty*)
from *pos-decomp-subset finite-set* **have** *finite* (*set* (ps_+)) **by** (*rule finite-subset*)
hence *finite ?A* **by** (*intro finite-imageI*)
from *False* **have** $?A \neq \{\}$ **by** *simp*
have b *ps* (*Suc 0*) = *Suc* (*Max ?A*) **unfolding** b-*def*
proof (*rule Least-equality*)
from ‹*finite ?A*› ‹$?A \neq \{\}$› **have** a $ps \in ?A$ **unfolding** *aP* **by** (*rule Min-in*)
with ‹*finite ?A*› **have** a $ps \leq Max\ ?A$ **by** (*rule Max-ge*)
hence a $ps \leq Suc\ (Max\ ?A)$ **by** *simp*
moreover {
fix *h U*
assume $(h,\ U) \in set\ ps$
with *fin-X assms*(*1*) **have** *finite U* **by** (*rule valid-decompD-finite*)
moreover assume $Suc\ 0 \leq card\ U$
ultimately have $U \neq \{\}$ **by** *auto*
with ‹$(h,\ U) \in set\ ps$› **have** $(h,\ U) \in set\ (ps_+)$ **by** (*simp add*: *pos-decomp-def*)
hence *poly-deg* (*fst* (*h*, *U*)) $\in ?A$ **by** (*intro imageI*)
hence *poly-deg* $h \in ?A$ **by** (*simp only*: *fst-conv*)
with ‹*finite ?A*› **have** *poly-deg* $h \leq Max\ ?A$ **by** (*rule Max-ge*)
hence *poly-deg* $h < Suc\ (Max\ ?A)$ **by** *simp*
}
ultimately show a $ps \leq Suc\ (Max\ ?A) \wedge (\forall (h,\ U) \in set\ ps.\ Suc\ 0 \leq card\ U \longrightarrow poly\text{-}deg\ h < Suc\ (Max\ ?A))$
by *blast*
next
fix *d*
assume a $ps \leq d \wedge (\forall (h,\ U) \in set\ ps.\ Suc\ 0 \leq card\ U \longrightarrow poly\text{-}deg\ h < d)$
hence *rl*: *poly-deg* $h < d$ **if** $(h,\ U) \in set\ ps$ **and** $0 < card\ U$ **for** *h U* **using** *that* **by** *auto*
have $Max\ ?A < d$ **unfolding** *Max-less-iff*[*OF* ‹*finite ?A*› ‹$?A \neq \{\}$›]
proof

```
      fix d0
      assume d0 ∈ poly-deg ‘ fst ‘ set (ps₊)
       then obtain h U where (h, U) ∈ set (ps₊) and d0: d0 = poly-deg h by
auto
     from this(1) have (h, U) ∈ set ps and U ≠ {} by (simp-all add: pos-decomp-def)
      from fin-X assms(1) this(1) have finite U by (rule valid-decompD-finite)
      with ‹U ≠ {}› have 0 < card U by (simp add: card-gt-0-iff)
      with ‹(h, U) ∈ set ps› show d0 < d unfolding d0 by (rule rl)
    qed
    thus Suc (Max ?A) ≤ d by simp
  qed
  with False show ?thesis by simp
qed

corollary b-one-gr:
  assumes valid-decomp X ps and standard-decomp k ps and (h, U) ∈ set (ps₊)
  shows poly-deg h < b ps (Suc 0)
proof −
  from assms(3) have ps₊ ≠ [] by auto
  with assms(1, 2) have eq: b ps (Suc 0) = Suc (Max (poly-deg ‘ fst ‘ set (ps₊)))
    by (simp add: b-one)
  have poly-deg h ≤ Max (poly-deg ‘ fst ‘ set (ps₊))
  proof (rule Max-ge)
    from finite-set show finite (poly-deg ‘ fst ‘ set (ps₊)) by (intro finite-imageI)
  next
    from assms(3) have poly-deg (fst (h, U)) ∈ poly-deg ‘ fst ‘ set (ps₊) by (intro
imageI)
    thus poly-deg h ∈ poly-deg ‘ fst ‘ set (ps₊) by simp
  qed
  also have ... < b ps (Suc 0) by (simp add: eq)
  finally show ?thesis .
qed

lemma b-card-X:
  assumes exact-decomp m ps and Suc (card X) ≤ i
  shows b ps i = a ps
  unfolding b-def
proof (rule Least-equality)
  {
    fix h U
    assume (h, U) ∈ set ps
    with assms(1) have U ⊆ X by (rule exact-decompD)
    note assms(2)
    also assume i ≤ card U
    finally have card X < card U by simp
    with fin-X have ¬ U ⊆ X by (auto dest: card-mono leD)
    hence False using ‹U ⊆ X› ..
  }
  thus a ps ≤ a ps ∧ (∀ (h, U)∈set ps. i ≤ card U ⟶ poly-deg h < a ps) by blast
```

```
qed simp

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 −
  have ps₊ ≠ []
  proof
    assume ps₊ = []
    hence b ps i = 0 using assms(3) by (rule b-Nil)
    with assms(6) show False by simp
  qed
  have eq1: b ps (Suc (card X)) = a ps using assms(2) le-refl by (rule b-card-X)
  from assms(1) have std: standard-decomp (b ps (Suc (card X))) ps unfolding
eq1 by (rule a)
  from assms(4) have Suc i ≤ Suc (card X) ..
  hence b ps (Suc (card X)) ≤ b ps (Suc i) by (rule b-decreasing)
  hence a ps ≤ b ps (Suc i) by (simp only: eq1)
  have ∃h U. (h, U) ∈ set ps ∧ i ≤ card U ∧ b ps i ≤ Suc (poly-deg h)
  proof (rule ccontr)
    assume *: ∄h U. (h, U) ∈ set ps ∧ i ≤ card U ∧ b ps i ≤ Suc (poly-deg h)
    note ‹a ps ≤ b ps (Suc i)›
    also from assms(5, 6) have b ps (Suc i) < b ps i by (rule le-less-trans)
    finally have a ps < b ps i .
    hence a ps ≤ b ps i − 1 by simp
    hence b ps i ≤ b ps i − 1
    proof (rule b-le)
      fix h U
      assume (h, U) ∈ set ps and i ≤ card U
      show poly-deg h < b ps i − 1
      proof (rule ccontr)
        assume ¬ poly-deg h < b ps i − 1
        hence b ps i ≤ Suc (poly-deg h) by simp
        with * ‹(h, U) ∈ set ps› ‹i ≤ card U› show False by auto
      qed
    qed
    thus False using ‹a ps < b ps i› by linarith
  qed
  then obtain h U where (h, U) ∈ set ps and i ≤ card U and b ps i ≤ Suc
(poly-deg h) by blast
  from assms(3) this(2) have U ≠ {} by auto
  with ‹(h, U) ∈ set ps› have (h, U) ∈ set (ps₊) by (simp add: pos-decomp-def)
  note std this
  moreover have b ps (Suc (card X)) ≤ d unfolding eq1 using ‹a ps ≤ b ps
(Suc i)› assms(5)
    by (rule le-trans)
  moreover have d ≤ poly-deg h
  proof −
```

from *assms*(*6*) ‹b *ps* $i \le$ *Suc* (*poly-deg* *h*)› **have** $d <$ *Suc* (*poly-deg* *h*) **by** (*rule* *less-le-trans*)
thus *?thesis* **by** *simp*
qed
ultimately obtain h' U' **where** $(h', U') \in$ *set ps* **and** *d*: *poly-deg* $h' = d$ **and** *card* $U \le$ *card* U'
by (*rule standard-decompE*)
from ‹$i \le$ *card* *U*› *this*(*3*) **have** $i \le$ *card* U' **by** (*rule le-trans*)
with *assms*(*3*) **have** $U' \neq \{\}$ **by** *auto*
with ‹$(h', U') \in$ *set ps*› **have** $(h', U') \in$ *set* (ps_+) **by** (*simp add*: *pos-decomp-def*)
moreover note ‹*poly-deg* $h' = d$›
moreover have *card* $U' = i$
proof (*rule ccontr*)
assume *card* $U' \neq i$
with ‹$i \le$ *card* U'› **have** *Suc* $i \le$ *card* U' **by** *simp*
with ‹$(h', U') \in$ *set ps*› **have** *poly-deg* $h' <$ b *ps* (*Suc* *i*) **by** (*rule* b)
with *assms*(*5*) **show** *False* **by** (*simp add*: *d*)
qed
ultimately show *?thesis* ..
qed

corollary *lem-6-1-2*:
assumes *standard-decomp k ps* **and** *exact-decomp 0 ps* **and** *Suc 0* $\le i$
and $i \le$ *card X* **and** b *ps* (*Suc* *i*) $\le d$ **and** $d <$ b *ps i*
obtains *h U* **where** $\{(h', U') \in$ *set* (ps_+). *poly-deg* $h' = d\} = \{(h, U)\}$ **and** *card* $U = i$
proof −
from *assms* **obtain** *h U* **where** $(h, U) \in$ *set* (ps_+) **and** *poly-deg* $h = d$ **and** *card* $U = i$
by (*rule lem-6-1-1*)
hence $\{(h, U)\} \subseteq \{(h', U') \in$ *set* (ps_+). *poly-deg* $h' = d\}$ (**is** - $\subseteq$ *?A*) **by** *simp*
moreover have *?A* $\subseteq \{(h, U)\}$
proof
fix *x*
assume $x \in$ *?A*
then obtain h' U' **where** $(h', U') \in$ *set* (ps_+) **and** *poly-deg* $h' = d$ **and** *x*: $x = (h', U')$
by *blast*
note *assms*(*2*) ‹$(h, U) \in$ *set* (ps_+)› *this*(*1*)
moreover have *poly-deg* h = *poly-deg* h' **by** (*simp only*: ‹*poly-deg* $h = d$› ‹*poly-deg* $h' = d$›)
ultimately have $(h, U) = (h', U')$ **by** (*rule exact-decompD-zero*)
thus $x \in \{(h, U)\}$ **by** (*simp add*: *x*)
qed
ultimately have $\{(h, U)\}$ = *?A* ..
hence *?A* $= \{(h, U)\}$ **by** (*rule sym*)
thus *?thesis* **using** ‹*card* $U = i$› ..
qed

corollary *lem-6-1-2'*:
assumes *standard-decomp k ps* **and** *exact-decomp 0 ps* **and** $Suc\ 0 \le i$
and $i \le card\ X$ **and** b *ps* $(Suc\ i) \le d$ **and** $d <$ b *ps i*
shows $card\ \{(h', U') \in set\ (ps_+).\ poly\text{-}deg\ h' = d\} = 1$ (**is** $card\ ?A = -$)
and $\{(h', U') \in set\ (ps_+).\ poly\text{-}deg\ h' = d \wedge card\ U' = i\} = \{(h', U') \in set\ (ps_+).\ poly\text{-}deg\ h' = d\}$
(**is** $?B = -$)
and $card\ \{(h', U') \in set\ (ps_+).\ poly\text{-}deg\ h' = d \wedge card\ U' = i\} = 1$
proof −
from *assms* **obtain** *h U* **where** $?A = \{(h, U)\}$ **and** $card\ U = i$ **by** (*rule lem-6-1-2*)
from *this*(*1*) **show** $card\ ?A = 1$ **by** *simp*
moreover show $?B = ?A$
proof
have $(h, U) \in ?A$ **by** (*simp add*: ‹$?A = \{(h, U)\}$›)
have $?A = \{(h, U)\}$ **by** *fact*
also from ‹$(h, U) \in ?A$› ‹$card\ U = i$› **have** $\ldots \subseteq ?B$ **by** *simp*
finally show $?A \subseteq ?B$ **.**
qed *blast*
ultimately show $card\ ?B = 1$ **by** *simp*
qed

corollary *lem-6-1-3*:
assumes *standard-decomp k ps* **and** *exact-decomp 0 ps* **and** $Suc\ 0 \le i$
and $i \le card\ X$ **and** $(h, U) \in set\ (ps_+)$ **and** $card\ U = i$
shows b *ps* $(Suc\ i) \le poly\text{-}deg\ h$
proof (*rule ccontr*)
define *j* **where** $j = (LEAST\ j'.$ b $ps\ j' \le poly\text{-}deg\ h)$
assume $\neg$ b *ps* $(Suc\ i) \le poly\text{-}deg\ h$
hence $poly\text{-}deg\ h <$ b *ps* $(Suc\ i)$ **by** *simp*
from *assms*(*2*) *le-refl* **have** b *ps* $(Suc\ (card\ X)) =$ a *ps* **by** (*rule* b-*card-X*)
also from - *assms*(*5*) **have** $\ldots \le poly\text{-}deg\ h$
proof (*rule standard-decompD*)
from *assms*(*1*) **show** *standard-decomp* (a *ps*) *ps* **by** (*rule* a)
qed
finally have b *ps* $(Suc\ (card\ X)) \le poly\text{-}deg\ h$ **.**
hence *1*: b $ps\ j \le poly\text{-}deg\ h$ **unfolding** *j-def* **by** (*rule LeastI*)
have $Suc\ i < j$
proof (*rule ccontr*)
assume $\neg\ Suc\ i < j$
hence $j \le Suc\ i$ **by** *simp*
hence b *ps* $(Suc\ i) \le$ b *ps j* **by** (*rule* b-*decreasing*)
also have $\ldots \le poly\text{-}deg\ h$ **by** *fact*
finally show *False* **using** ‹$poly\text{-}deg\ h <$ b *ps* $(Suc\ i)$› **by** *simp*
qed
hence *eq*: $Suc\ (j - 1) = j$ **by** *simp*
note *assms*(*1*, *2*)
moreover from *assms*(*3*) **have** $Suc\ 0 \le j - 1$
proof (*rule le-trans*)

from ‹*Suc i* < *j*› **show** $i \le j - 1$ **by** *simp*
qed
moreover have $j - 1 \le$ *card X*
proof −
have $j \le$ *Suc* (*card X*) **unfolding** *j-def* **by** (*rule Least-le*) *fact*
thus *?thesis* **by** *simp*
qed
moreover from *1* **have** b *ps* (*Suc* ($j - 1$)) ≤ *poly-deg h* **by** (*simp only*: *eq*)
moreover have *poly-deg h* < b *ps* ($j - 1$)
proof (*rule ccontr*)
assume ¬ *poly-deg h* < b *ps* ($j - 1$)
hence b *ps* ($j - 1$) ≤ *poly-deg h* **by** *simp*
hence $j \le j - 1$ **unfolding** *j-def* **by** (*rule Least-le*)
also have . . . < *Suc* ($j - 1$) **by** *simp*
finally show *False* **by** (*simp only*: *eq*)
qed
ultimately obtain *h0 U0*
where *eq1*: {(*h'*, *U'*). (*h'*, *U'*) ∈ *set* (ps_+) ∧ *poly-deg h'* = *poly-deg h*} = {(*h0*, *U0*)}
and *card U0* = $j - 1$ **by** (*rule lem-6-1-2*)
from *assms*(*5*) **have** (*h*, *U*) ∈ {(*h'*, *U'*). (*h'*, *U'*) ∈ *set* (ps_+) ∧ *poly-deg h'* = *poly-deg h*} **by** *simp*
hence (*h*, *U*) ∈ {(*h0*, *U0*)} **by** (*simp only*: *eq1*)
hence *U* = *U0* **by** *simp*
hence *card U* = $j - 1$ **by** (*simp only*: ‹*card U0* = $j - 1$›)
hence $i = j - 1$ **by** (*simp only*: *assms*(*6*))
hence *Suc i* = *j* **by** (*simp only*: *eq*)
with ‹*Suc i* < *j*› **show** *False* **by** *simp*
qed

qualified 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 −
let *?x* = *Poly-Mapping.single x* (*1*::*nat*)
obtain *h U* **where** *hU*: *hU* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
with *assms*(*2*) **have** (*h*, *U*) ∈ *set ps* **by** *simp*
with *assms*(*1*) **have** *1*: *is-monomial h* **and** *2*: *lcf h* = *1* **by** (*rule monomial-decompD*)+
from *this*(*1*) **have** *monomial* (*lcf h*) (*lpp h*) = *h* **by** (*rule punit.monomial-eq-itself*)

moreover define *t* **where** $t = lpp\ h$
ultimately have $h = monomial\ 1\ t$ **by** (*simp only*: *2*)
hence *is-monomial* (*punit.monom-mult 1 ?x h*) **and** *lcf* (*punit.monom-mult 1 ?x h*) = *1*
by (*simp-all add*: *punit.monom-mult-monomial monomial-is-monomial*)
with *assms*(*1*) *1 2* **show** *?thesis* **by** (*simp add*: *shift-list.simps monomial-decomp-def hU*)
qed

lemma *hom-decomp-shift-list*:
assumes *hom-decomp ps* **and** $hU \in set\ ps$
shows *hom-decomp* (*shift-list hU x ps*)
proof −
let *?x* = *Poly-Mapping.single x* (*1*::*nat*)
obtain *h U* **where** *hU*: *hU* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
with *assms*(*2*) **have** $(h, U) \in set\ ps$ **by** *simp*
with *assms*(*1*) **have** *1*: *homogeneous h* **by** (*rule hom-decompD*)
hence *homogeneous* (*punit.monom-mult 1 ?x h*) **by** (*simp only*: *homogeneous-monom-mult*)
with *assms*(*1*) *1* **show** *?thesis* **by** (*simp add*: *shift-list.simps hom-decomp-def hU*)
qed

lemma *valid-decomp-shift-list*:
assumes *valid-decomp X ps* **and** $(h, U) \in set\ ps$ **and** $x \in U$
shows *valid-decomp X* (*shift-list* (*h*, *U*) *x ps*)
proof −
let *?x* = *Poly-Mapping.single x* (*1*::*nat*)
from *assms*(*1*, *2*) **have** $h \in P[X]$ **and** $h \neq 0$ **and** $U \subseteq X$ **by** (*rule valid-decompD*)+
moreover from *this*(*1*) **have** *punit.monom-mult 1 ?x h* $\in P[X]$
proof (*intro Polys-closed-monom-mult PPs-closed-single*)
from ‹$x \in U$› ‹$U \subseteq X$› **show** $x \in X$ **..**
qed
moreover from ‹$U \subseteq X$› **have** $U - \{x\} \subseteq X$ **by** *blast*
ultimately show *?thesis*
using *assms*(*1*) ‹$h \neq 0$› **by** (*simp add*: *valid-decomp-def punit.monom-mult-eq-zero-iff shift-list.simps*)
qed

lemma *standard-decomp-shift-list*:
assumes *standard-decomp k ps* **and** $(h1, U1) \in set\ ps$ **and** $(h2, U2) \in set\ ps$
and *poly-deg h1* = *poly-deg h2* **and** $card\ U2 \leq card\ U1$ **and** $(h1, U1) \neq (h2, U2)$ **and** $x \in U2$
shows *standard-decomp k* (*shift-list* (*h2*, *U2*) *x ps*)
proof (*rule standard-decompI*)
let *?p1* = (*punit.monom-mult 1* (*Poly-Mapping.single x 1*) *h2*, *U2*)
let *?p2* = $(h2, U2 - \{x\})$
let *?qs* = *removeAll* (*h2*, *U2*) *ps*
fix *h U*
assume $(h, U) \in set\ ((shift\text{-}list\ (h2, U2)\ x\ ps)_+)$

hence *disj*: $(h,\ U) = ?p1 \vee ((h,\ U) = ?p2 \wedge U2 - \{x\} \neq \{\}) \vee (h,\ U) \in set$ (ps_+)
by (*auto simp*: *pos-decomp-def shift-list.simps split*: *if-split-asm*)
from *assms*(*7*) **have** $U2 \neq \{\}$ **by** *blast*
with *assms*(*3*) **have** $(h2,\ U2) \in set\ (ps_+)$ **by** (*simp add*: *pos-decomp-def*)
with *assms*(*1*) **have** *k-le*: $k \leq$ *poly-deg h2* **by** (*rule standard-decompD*)

let *?x* = *Poly-Mapping.single x 1*
from *disj* **show** $k \leq$ *poly-deg h*
proof (*elim disjE*)
assume $(h,\ U) = ?p1$
hence *h*: *h* = *punit.monom-mult* (*1*::*'a*) *?x h2* **by** *simp*
note *k-le*
also have *poly-deg h2* $\leq$ *poly-deg h* **by** (*cases h2* = *0*) (*simp-all add*: *h poly-deg-monom-mult*)
finally show *?thesis* **.**
next
assume $(h,\ U) = ?p2 \wedge U2 - \{x\} \neq \{\}$
with *k-le* **show** *?thesis* **by** *simp*
next
assume $(h,\ U) \in set\ (ps_+)$
with *assms*(*1*) **show** *?thesis* **by** (*rule standard-decompD*)
qed

fix *d*
assume $k \leq d$ **and** $d \leq$ *poly-deg h*
from *disj* **obtain** *h' U'* **where** *1*: $(h',\ U') \in set\ (?p1\ \#\ ps)$ **and** *poly-deg h'* = *d*
and *card U* $\leq$ *card U'*
proof (*elim disjE*)
assume $(h,\ U) = ?p1$
hence *h*: *h* = *punit.monom-mult 1 ?x h2* **and** *U* = *U2* **by** *simp-all*
from ‹$d \leq$ *poly-deg h*› **have** $d \leq$ *poly-deg h2* $\vee$ *poly-deg h* = *d*
by (*cases h2* = *0*) (*auto simp*: *h poly-deg-monom-mult deg-pm-single*)
thus *?thesis*
proof
assume $d \leq$ *poly-deg h2*
with *assms*(*1*) ‹$(h2,\ U2) \in set\ (ps_+)$› ‹$k \leq d$› **obtain** *h' U'*
where $(h',\ U') \in set\ ps$ **and** *poly-deg h'* = *d* **and** *card U2* $\leq$ *card U'*
by (*rule standard-decompE*)
from *this*(*1*) **have** $(h',\ U') \in set\ (?p1\ \#\ ps)$ **by** *simp*
moreover note ‹*poly-deg h'* = *d*›
moreover from ‹*card U2* $\leq$ *card U'*› **have** *card U* $\leq$ *card U'* **by** (*simp only*: ‹*U* = *U2*›)
ultimately show *?thesis* **..**
next
have $(h,\ U) \in set\ (?p1\ \#\ ps)$ **by** (*simp add*: ‹$(h,\ U) = ?p1$›)
moreover assume *poly-deg h* = *d*
ultimately show *?thesis* **using** *le-refl* **..**

qed
next
assume $(h, U) = ?p2 \wedge U2 - \{x\} \neq \{\}$
hence $h = h2$ **and** U: $U = U2 - \{x\}$ **by** *simp-all*
from ‹$d \leq$ *poly-deg* h› *this*(*1*) **have** $d \leq$ *poly-deg* $h2$ **by** *simp*
with *assms*(*1*) ‹$(h2, U2) \in$ *set* (ps_+)› ‹$k \leq d$› **obtain** h' U'
where $(h', U') \in$ *set ps* **and** *poly-deg* $h' = d$ **and** *card* $U2 \leq$ *card* U'
by (*rule standard-decompE*)
from *this*(*1*) **have** $(h', U') \in$ *set* (*?p1* # *ps*) **by** *simp*
moreover note ‹*poly-deg* $h' = d$›
moreover from - ‹*card* $U2 \leq$ *card* U'› **have** *card* $U \leq$ *card* U' **unfolding** U
by (*rule le-trans*) (*metis Diff-empty card-Diff1-le card.infinite finite-Diff-insert order-refl*)
ultimately show *?thesis* **..**
next
assume $(h, U) \in$ *set* (ps_+)
from *assms*(*1*) *this* ‹$k \leq d$› ‹$d \leq$ *poly-deg* h› **obtain** h' U'
where $(h', U') \in$ *set ps* **and** *poly-deg* $h' = d$ **and** *card* $U \leq$ *card* U'
by (*rule standard-decompE*)
from *this*(*1*) **have** $(h', U') \in$ *set* (*?p1* # *ps*) **by** *simp*
thus *?thesis* **using** ‹*poly-deg* $h' = d$› ‹*card* $U \leq$ *card* U'› **..**
qed
show $\exists h'\ U'.\ (h', U') \in$ *set* (*shift-list* $(h2, U2)$ *x ps*) $\wedge$ *poly-deg* $h' = d \wedge$ *card* $U \leq$ *card* U'
proof (*cases* $(h', U') = (h2, U2)$)
case *True*
hence $h' = h2$ **and** $U' = U2$ **by** *simp-all*
from *assms*(*2*, *6*) **have** $(h1, U1) \in$ *set* (*shift-list* $(h2, U2)$ *x ps*) **by** (*simp add*: *shift-list.simps*)
moreover from ‹*poly-deg* $h' = d$› **have** *poly-deg* $h1 = d$ **by** (*simp only*: ‹$h' = h2$› *assms*(*4*))
moreover from ‹*card* $U \leq$ *card* U'› *assms*(*5*) **have** *card* $U \leq$ *card* $U1$ **by** (*simp add*: ‹$U' = U2$›)
ultimately show *?thesis* **by** *blast*
next
case *False*
with *1* **have** $(h', U') \in$ *set* (*shift-list* $(h2, U2)$ *x ps*) **by** (*auto simp*: *shift-list.simps*)
thus *?thesis* **using** ‹*poly-deg* $h' = d$› ‹*card* $U \leq$ *card* U'› **by** *blast*
qed
qed

lemma *cone-decomp-shift-list*:
assumes *valid-decomp X ps* **and** *cone-decomp T ps* **and** $(h, U) \in$ *set ps* **and** $x \in U$
shows *cone-decomp T* (*shift-list* (h, U) *x ps*)
proof –
let *?p1* = (*punit.monom-mult 1* (*Poly-Mapping.single x 1*) *h*, *U*)
let *?p2* = $(h, U - \{x\})$
let *?qs* = *removeAll* (h, U) *ps*

from *assms*(*3*) **obtain** *ps1 ps2* **where** *ps*: $ps = ps1$ @ (h, U) # $ps2$ **and** $*$: $(h, U) \notin set\ ps1$
by (*meson split-list-first*)
have $count\text{-}list\ ps2\ (h, U) = 0$
proof (*rule ccontr*)
from *assms*(*1*, *3*) **have** $h \neq 0$ **by** (*rule valid-decompD*)
assume $count\text{-}list\ ps2\ (h, U) \neq 0$
hence $1 < count\text{-}list\ ps\ (h, U)$ **by** (*simp add*: *ps*)
also have $\ldots \leq count\text{-}list\ (map\ cone\ ps)\ (cone\ (h, U))$ **by** (*fact count-list-map-ge*)
finally have $1 < count\text{-}list\ (map\ cone\ ps)\ (cone\ (h, U))$ **.**
with *cone-decompD* **have** $cone\ (h, U) = \{0\}$
proof (*rule direct-decomp-repeated-eq-zero*)
fix *s*
assume $s \in set\ (map\ cone\ ps)$
thus $0 \in s$ **by** (*auto intro*: *zero-in-cone*)
qed (*fact assms*(*2*))
with *tip-in-cone*[*of h U*] **have** $h = 0$ **by** *simp*
with ‹$h \neq 0$› **show** *False* **..**
qed
hence $**$: $(h, U) \notin set\ ps2$ **by** (*simp add*: *count-list-0-iff*)
have *mset ps* = *mset* ((*h*, *U*) # *ps1* @ *ps2*) (**is** *mset* - = *mset ?ps*)
by (*simp add*: *ps*)
with *assms*(*2*) **have** *cone-decomp T ?ps* **by** (*rule cone-decomp-perm*)
hence *direct-decomp T* (*map cone ?ps*) **by** (*rule cone-decompD*)
hence *direct-decomp T* (*cone* (*h*, *U*) # *map cone* (*ps1* @ *ps2*)) **by** *simp*
hence *direct-decomp T* ((*map cone* (*ps1* @ *ps2*)) @ [*cone ?p1*, *cone ?p2*])
proof (*rule direct-decomp-direct-decomp*)
let *?x* = *Poly-Mapping.single x* (*Suc 0*)
have *direct-decomp* (*cone* (*h*, *insert x* ($U - \{x\}$)))
[*cone* (*h*, $U - \{x\}$), *cone* (*monomial* (*1*::'*a*) *?x* $*$ *h*, *insert x* ($U - \{x\}$))]
by (*rule direct-decomp-cone-insert*) *simp*
with *assms*(*4*) **show** *direct-decomp* (*cone* (*h*, *U*)) [*cone ?p1*, *cone ?p2*]
by (*simp add*: *insert-absorb times-monomial-left direct-decomp-perm*)
qed
hence *direct-decomp T* (*map cone* (*ps1* @ *ps2* @ [*?p1*, *?p2*])) **by** *simp*
hence *cone-decomp T* (*ps1* @ *ps2* @ [*?p1*, *?p2*]) **by** (*rule cone-decompI*)
moreover have *mset* (*ps1* @ *ps2* @ [*?p1*, *?p2*]) = *mset* (*?p1* # *?p2* # (*ps1* @ *ps2*))
by *simp*
ultimately have *cone-decomp T* (*?p1* # *?p2* # (*ps1* @ *ps2*)) **by** (*rule cone-decomp-perm*)
also from $*$ $**$ **have** *ps1* @ *ps2* = *removeAll* (*h*, *U*) *ps* **by** (*simp add*: *remove1-append ps*)
finally show *?thesis* **by** (*simp only*: *shift-list.simps*)
qed

10.9 Functions *shift* and *exact*

context

fixes $k\ m :: nat$
begin

context
fixes $d :: nat$
begin

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

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

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

lemma $shift2\text{-}invD$:
assumes $shift2\text{-}inv\ qs$
shows $valid\text{-}decomp\ X\ qs$ **and** $standard\text{-}decomp\ k\ qs$ **and** $exact\text{-}decomp\ (Suc\ m)\ qs$
and $d0 < d \Longrightarrow card\ \{q \in set\ qs.\ poly\text{-}deg\ (fst\ q) = d0 \wedge m < card\ (snd\ q)\} \leq 1$
using *assms* **by** (*simp-all add*: *shift2-inv-def*)

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

lemma $shift1\text{-}invD$:
assumes $shift1\text{-}inv\ (qs,\ B)$
shows $B = \{q \in set\ qs.\ poly\text{-}deg\ (fst\ q) = d \wedge m < card\ (snd\ q)\}$ **and** $shift2\text{-}inv\ qs$
using *assms* **by** *simp-all*

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

lemma $shift1\text{-}inv\text{-}finite\text{-}snd$:
assumes $shift1\text{-}inv\ (qs,\ B)$
shows $finite\ B$

proof (*rule finite-subset*)
  from *assms* **have** $B = \{q \in set\ qs.\ poly\text{-}deg\ (fst\ q) = d \wedge m < card\ (snd\ q)\}$ **by** (*rule shift1-invD*)
  also have $\ldots \subseteq set\ qs$ **by** *blast*
  finally show $B \subseteq set\ qs$ **.**
qed (*fact finite-set*)

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** $poly\text{-}deg\ h = d$ **and** $card\ U = Suc\ m$
proof −
  define A **where** $A = \{q \in B.\ card\ (snd\ q) = Suc\ m\}$
  define Y **where** $Y = \{q \in set\ qs.\ poly\text{-}deg\ (fst\ q) = d \wedge Suc\ m < card\ (snd\ q)\}$
  from *assms*(*1*) **have** *B*: $B = \{q \in set\ qs.\ poly\text{-}deg\ (fst\ q) = d \wedge m < card\ (snd\ q)\}$
    and *inv2*: *shift2-inv qs* **by** (*rule shift1-invD*)+
  have *B′*: $B = A \cup Y$ **by** (*auto simp*: *B A-def Y-def*)
  have *finite A*
  proof (*rule finite-subset*)
    show $A \subseteq B$ **unfolding** *A-def* **by** *blast*
  next
    from *assms*(*1*) **show** *finite B* **by** (*rule shift1-inv-finite-snd*)
  qed
  moreover have *finite Y*
  proof (*rule finite-subset*)
    show $Y \subseteq set\ qs$ **unfolding** *Y-def* **by** *blast*
  qed (*fact finite-set*)
  moreover have $A \cap Y = \{\}$ **by** (*auto simp*: *A-def Y-def*)
  ultimately have $card\ (A \cup Y) = card\ A + card\ Y$ **by** (*rule card-Un-disjoint*)
  with *assms*(*2*) **have** $1 < card\ A + card\ Y$ **by** (*simp only*: *B′*)
  thm *card-le-Suc0-iff-eq*[*OF* ‹*finite Y*›]
  moreover have $card\ Y \leq 1$ **unfolding** *One-nat-def card-le-Suc0-iff-eq*[*OF* ‹*finite Y*›]
  proof (*intro ballI*)
    fix $q1\ q2 :: (('x \Rightarrow_0 nat) \Rightarrow_0 'a) \times 'x\ set$
    obtain *h1 U1* **where** *q1*: $q1 = (h1, U1)$ **using** *prod.exhaust* **by** *blast*
    obtain *h2 U2* **where** *q2*: $q2 = (h2, U2)$ **using** *prod.exhaust* **by** *blast*
    assume $q1 \in Y$
     hence $(h1, U1) \in set\ qs$ **and** $poly\text{-}deg\ h1 = d$ **and** $Suc\ m < card\ U1$ **by** (*simp-all add*: *q1 Y-def*)
    assume $q2 \in Y$
     hence $(h2, U2) \in set\ qs$ **and** $poly\text{-}deg\ h2 = d$ **and** $Suc\ m < card\ U2$ **by** (*simp-all add*: *q2 Y-def*)
     from *this*(*2*) **have** $poly\text{-}deg\ h1 = poly\text{-}deg\ h2$ **by** (*simp only*: ‹$poly\text{-}deg\ h1 = d$›)
    from *inv2* **have** *exact-decomp* (*Suc m*) *qs* **by** (*rule shift2-invD*)
    thus $q1 = q2$ **unfolding** *q1 q2* **by** (*rule exact-decompD*) *fact*+

qed
ultimately have $0 < card\ A$ **by** *simp*
hence $A \neq \{\}$ **by** *auto*
then obtain a **where** $a \in A$ **by** *blast*
have $(h,\ U) \in B \wedge card\ (snd\ (h,\ U)) = Suc\ m$ **unfolding** *assms*(*3*)
proof (*rule someI*)
from ‹$a \in A$› **show** $a \in B \wedge card\ (snd\ a) = Suc\ m$ **by** (*simp add*: *A-def*)
qed
thus $(h,\ U) \in B$ **and** $card\ U = Suc\ m$ **by** *simp-all*
from *this*(*1*) **show** $(h,\ U) \in set\ qs$ **and** $poly\text{-}deg\ h = d$ **by** (*simp-all add*: *B*)
qed

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 −
let $?p1 = (punit.monom\text{-}mult\ 1\ (Poly\text{-}Mapping.single\ x\ 1)\ h,\ U)$
let $?p2 = (h,\ U - \{x\})$
let $?qs = removeAll\ (h,\ U)\ qs$
let $?B = B - \{(h,\ U)\}$
from *assms*(*1*, *2*, *3*) **have** $(h,\ U) \in B$ **and** $(h,\ U) \in set\ qs$ **and** *deg-h*: $poly\text{-}deg\ h = d$
and *card-U*: $card\ U = Suc\ m$ **by** (*rule shift1-inv-some-snd*)+
from *card-U* **have** $U \neq \{\}$ **by** *auto*
then obtain y **where** $y \in U$ **by** *blast*
hence $x \in U$ **unfolding** *assms*(*4*) **by** (*rule someI*)
with *card-U* **have** *card-Ux*: $card\ (U - \{x\}) = m$
by (*metis card-Diff-singleton card.infinite diff-Suc-1 nat.simps*(*3*))
from *assms*(*1*) **have** *B*: $B = \{q \in set\ qs.\ poly\text{-}deg\ (fst\ q) = d \wedge m < card\ (snd\ q)\}$
and *inv2*: *shift2-inv qs* **by** (*rule shift1-invD*)+
from *inv2* **have** *valid-qs*: *valid-decomp X qs* **by** (*rule shift2-invD*)
hence $h \neq 0$ **using** ‹$(h,\ U) \in set\ qs$› **by** (*rule valid-decompD*)
show *?thesis*
proof (*intro shift1-invI shift2-invI*)
show $?B = \{q \in set\ (shift\text{-}list\ (h,\ U)\ x\ qs).\ poly\text{-}deg\ (fst\ q) = d \wedge m < card\ (snd\ q)\}$ (**is** - = *?C*)
proof (*rule Set.set-eqI*)
fix b
show $b \in ?B \longleftrightarrow b \in ?C$
proof
assume $b \in ?C$
hence $b \in insert\ ?p1\ (insert\ ?p2\ (set\ ?qs))$ **and** *b1*: $poly\text{-}deg\ (fst\ b) = d$
and *b2*: $m < card\ (snd\ b)$ **by** (*simp-all add*: *shift-list.simps*)
from *this*(*1*) **show** $b \in ?B$
proof (*elim insertE*)
assume $b = ?p1$

```
          with ‹h ≠ 0› have poly-deg (fst b) = Suc d
            by (simp add: poly-deg-monom-mult deg-pm-single deg-h)
          thus ?thesis by (simp add: b1)
        next
          assume b = ?p2
          hence card (snd b) = m by (simp add: card-Ux)
          with b2 show ?thesis by simp
        next
          assume b ∈ set ?qs
          with b1 b2 show ?thesis by (auto simp: B)
        qed
      qed (auto simp: B shift-list.simps)
    qed
  next
    from valid-qs ‹(h, U) ∈ set qs› ‹x ∈ U› show valid-decomp X (shift-list (h, U) x qs)
      by (rule valid-decomp-shift-list)
  next
    from inv2 have std: standard-decomp k qs by (rule shift2-invD)
    have ?B ≠ {}
    proof
      assume ?B = {}
      hence B ⊆ {(h, U)} by simp
      with - have card B ≤ card {(h, U)} by (rule card-mono) simp
      with assms(2) show False by simp
    qed
    then obtain h' U' where (h', U') ∈ B and (h', U') ≠ (h, U) by auto
    from this(1) have (h', U') ∈ set qs and poly-deg h' = d and Suc m ≤ card U'
      by (simp-all add: B)
    note std this(1) ‹(h, U) ∈ set qs›
    moreover from ‹poly-deg h' = d› have poly-deg h' = poly-deg h by (simp only: deg-h)
    moreover from ‹Suc m ≤ card U'› have card U ≤ card U' by (simp only: card-U)
    ultimately show standard-decomp k (shift-list (h, U) x qs)
      by (rule standard-decomp-shift-list) fact+
  next
    from inv2 have exct: exact-decomp (Suc m) qs by (rule shift2-invD)
    show exact-decomp (Suc m) (shift-list (h, U) x qs)
    proof (rule exact-decompI)
      fix h' U'
      assume (h', U') ∈ set (shift-list (h, U) x qs)
      hence *: (h', U') ∈ insert ?p1 (insert ?p2 (set ?qs)) by (simp add: shift-list.simps)
      thus h' ∈ P[X]
      proof (elim insertE)
        assume (h', U') = ?p1
        hence h': h' = punit.monom-mult 1 (Poly-Mapping.single x 1) h by simp
        from exct ‹(h, U) ∈ set qs› have U ⊆ X by (rule exact-decompD)
```

with ‹$x \in U$› **have** $x \in X$ **..**
hence *Poly-Mapping.single* x *1* $\in .[X]$ **by** (*rule PPs-closed-single*)
moreover from *exct* ‹$(h, U) \in$ *set qs*› **have** $h \in P[X]$ **by** (*rule exact-decompD*)
ultimately show *?thesis* **unfolding** h' **by** (*rule Polys-closed-monom-mult*)
next
assume $(h', U') =$ *?p2*
hence $h' = h$ **by** *simp*
also from *exct* ‹$(h, U) \in$ *set qs*› **have** $\ldots \in P[X]$ **by** (*rule exact-decompD*)
finally show *?thesis* **.**
next
assume $(h', U') \in$ *set ?qs*
hence $(h', U') \in$ *set qs* **by** *simp*
with *exct* **show** *?thesis* **by** (*rule exact-decompD*)
qed

from $*$ **show** $U' \subseteq X$
proof (*elim insertE*)
assume $(h', U') =$ *?p1*
hence $U' = U$ **by** *simp*
also from *exct* ‹$(h, U) \in$ *set qs*› **have** $\ldots \subseteq X$ **by** (*rule exact-decompD*)
finally show *?thesis* **.**
next
assume $(h', U') =$ *?p2*
hence $U' = U - \{x\}$ **by** *simp*
also have $\ldots \subseteq U$ **by** *blast*
also from *exct* ‹$(h, U) \in$ *set qs*› **have** $\ldots \subseteq X$ **by** (*rule exact-decompD*)
finally show *?thesis* **.**
next
assume $(h', U') \in$ *set ?qs*
hence $(h', U') \in$ *set qs* **by** *simp*
with *exct* **show** *?thesis* **by** (*rule exact-decompD*)
qed
next
fix *h1 h2 U1 U2*
assume $(h1, U1) \in$ *set* (*shift-list* (h, U) *x qs*) **and** *Suc m* $<$ *card U1*
hence $(h1, U1) \in$ *set qs* **using** *card-U card-Ux* **by** (*auto simp*: *shift-list.simps*)
assume $(h2, U2) \in$ *set* (*shift-list* (h, U) *x qs*) **and** *Suc m* $<$ *card U2*
hence $(h2, U2) \in$ *set qs* **using** *card-U card-Ux* **by** (*auto simp*: *shift-list.simps*)
assume *poly-deg h1* $=$ *poly-deg h2*
from *exct* **show** $(h1, U1) = (h2, U2)$ **by** (*rule exact-decompD*) *fact+*
qed
next
fix *d0*
assume *d0* $<$ *d*
have *finite* $\{q \in$ *set qs*. *poly-deg* (*fst q*) $=$ *d0* $\wedge$ *m* $<$ *card* (*snd q*)$\}$ (**is** *finite ?A*)
by *auto*
moreover have $\{q \in$ *set* (*shift-list* (h, U) *x qs*). *poly-deg* (*fst q*) $=$ *d0* $\wedge$ *m* $<$

card (*snd* *q*)} ⊆ *?A*
      (**is** *?C* ⊆ -)
    proof
      fix *q*
      assume *q* ∈ *?C*
      hence *q* = *?p1* ∨ *q* = *?p2* ∨ *q* ∈ *set* *?qs* **and** *1*: *poly-deg* (*fst* *q*) = *d0* **and** *2*: *m* < *card* (*snd* *q*)
        by (*simp-all add*: *shift-list.simps*)
      from *this*(*1*) **show** *q* ∈ *?A*
      proof (*elim disjE*)
        assume *q* = *?p1*
        with ‹*h* ≠ *0*› **have** *d* ≤ *poly-deg* (*fst* *q*) **by** (*simp add*: *poly-deg-monom-mult deg-h*)
        with ‹*d0* < *d*› **show** *?thesis* **by** (*simp only*: *1*)
      next
        assume *q* = *?p2*
        hence *d* ≤ *poly-deg* (*fst* *q*) **by** (*simp add*: *deg-h*)
        with ‹*d0* < *d*› **show** *?thesis* **by** (*simp only*: *1*)
      next
        assume *q* ∈ *set* *?qs*
        with *1 2* **show** *?thesis* **by** *simp*
      qed
    qed
    ultimately have *card* *?C* ≤ *card* *?A* **by** (*rule card-mono*)
    also from *inv2* ‹*d0* < *d*› **have** ... ≤ *1* **by** (*rule shift2-invD*)
    finally show *card* *?C* ≤ *1* **.**
  qed
qed

function (*domintros*) *shift1* :: (((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*) × '*x* *set*) *list* × ((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*) × '*x* *set*) *set*) ⇒
                    (((('*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*) *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*))
  by *auto*

lemma *shift1-domI*:
  assumes *shift1-inv args*
  shows *shift1-dom args*
proof −
  from *wf-measure*[*of card* ∘ *snd*] **show** *?thesis* **using** *assms*
  proof (*induct*)

case (*less args*)
obtain *qs B* **where** *args*: *args* = (*qs*, *B*) **using** *prod.exhaust* **by** *blast*
have *IH*: *shift1-dom* (*qs0*, *B0*) **if** *card B0* < *card B* **and** *shift1-inv* (*qs0*, *B0*)
for *qs0* **and** *B0*::((- $\Rightarrow_0$ '*a*) × -) *set*
using - *that*(*2*)
proof (*rule less*)
from *that*(*1*) **show** ((*qs0*, *B0*), *args*) ∈ *measure* (*card* ∘ *snd*) **by** (*simp add*: *args*)
qed
from *less*(*2*) **have** *inv*: *shift1-inv* (*qs*, *B*) **by** (*simp only*: *args*)
show *?case* **unfolding** *args*
proof (*rule shift1.domintros*)
fix *h U*
assume *hU*: (*h*, *U*) = (*SOME b*. *b* ∈ *B* ∧ *card* (*snd b*) = *Suc m*)
define *x* **where** *x* = (*SOME y*. *y* ∈ *U*)
assume *Suc 0* < *card B*
hence *1* < *card B* **by** *simp*
have *shift1-dom* (*shift-list* (*h*, *U*) *x qs*, *B* − {(*h*, *U*)})
proof (*rule IH*)
from *inv* **have** *finite B* **by** (*rule shift1-inv-finite-snd*)
moreover from *inv* ‹*1* < *card B*› *hU* **have** (*h*, *U*) ∈ *B* **by** (*rule shift1-inv-some-snd*)
ultimately show *card* (*B* − {(*h*, *U*)}) < *card B* **by** (*rule card-Diff1-less*)
next
from *inv* ‹*1* < *card B*› *hU x-def* **show** *shift1-inv* (*shift-list* (*h*, *U*) *x qs*, (*B* − {(*h*, *U*)}))
by (*rule shift1-inv-preserved*)
qed
thus *shift1-dom* (*shift-list* (*SOME b*. *b* ∈ *B* ∧ *card* (*snd b*) = *Suc m*) (*SOME y*. *y* ∈ *U*) *qs*,
B − {*SOME b*. *b* ∈ *B* ∧ *card* (*snd b*) = *Suc m*}) **by** (*simp add*: *hU x-def*)
qed
qed
qed

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 −
from *assms*(*1*) **have** *shift1-dom args* **by** (*rule shift1-domI*)

```
  thus ?thesis using assms(1)
  proof (induct args rule: shift1.pinduct)
    case step: (1 qs B)
    obtain h U where hU: (h, U) = (SOME b. b ∈ B ∧ card (snd b) = Suc m)
by (smt prod.exhaust)
    define x where x = (SOME y. y ∈ U)
    show ?case
   proof (simp add: shift1.psimps[OF step.hyps(1)] flip: hU x-def del: One-nat-def,
          intro conjI impI)
      let ?args = (shift-list (h, U) x qs, B − {(h, U)})
      assume 1 < card B
    with step.prems have card-U: card U = Suc m using hU by (rule shift1-inv-some-snd)
      from card-U have finite U using card.infinite by fastforce
      from card-U have U ≠ {} by auto
      then obtain y where y ∈ U by blast
      hence x ∈ U unfolding x-def by (rule someI)
       with step.prems ‹1 < card B› hU x-def ‹finite U› show P (qs, B) (shift1
?args)
      proof (rule assms(3))
       from ‹finite U› ‹x ∈ U› show card (U − {x}) = m by (simp add: card-U)
      next
        from ‹1 < card B› refl hU x-def show P ?args (shift1 ?args)
        proof (rule step.hyps)
           from step.prems ‹1 < card B› hU x-def show shift1-inv ?args by (rule
shift1-inv-preserved)
        qed
      qed
    next
      assume ¬ 1 < card B
      hence card B ≤ 1 by simp
      with step.prems show P (qs, B) (qs, B) by (rule assms(2))
    qed
  qed
qed

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
  using assms(1)
proof (induct args rule: shift1-induct)
  case (base qs B)
  from assms(2) have d0 < d ∨ d0 = d by auto
  thus ?case
  proof
    from base(1) have shift2-inv qs by (rule shift1-invD)
    moreover assume d0 < d
    ultimately show ?thesis unfolding fst-conv by (rule shift2-invD)
  next
```

assume *d0* = *d*
from *base*(*1*) **have** *B* = {*q* ∈ *set* (*fst* (*qs*, *B*)). *poly-deg* (*fst q*) = *d0* ∧ *m* < *card* (*snd q*)}
unfolding *fst-conv* ‹*d0* = *d*› **by** (*rule shift1-invD*)
with *base*(*2*) **show** *?thesis* **by** *simp*
qed
qed

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 (*induct args rule*: *shift1-induct*)
case (*base qs B*)
show *?case* **..**
next
case (*step qs B h U x*)
let *?x* = *Poly-Mapping.single x* (*1*::*nat*)
let *?p1* = (*punit.monom-mult 1 ?x h*, *U*)
let *?A* = {*q* ∈ *set qs*. *m* < *card* (*snd q*)}
from *step*(*1*−*3*) **have** *card-U*: *card U* = *Suc m* **and** (*h*, *U*) ∈ *set qs* **by** (*rule shift1-inv-some-snd*)+
from *step*(*1*) **have** *shift2-inv qs* **by** (*rule shift1-invD*)
hence *valid-decomp X qs* **by** (*rule shift2-invD*)
hence *h* ≠ *0* **using** ‹(*h*, *U*) ∈ *set qs*› **by** (*rule valid-decompD*)
have *fin1*: *finite ?A* **by** *auto*
hence *fin2*: *finite* (*insert ?p1 ?A*) **by** *simp*
from ‹(*h*, *U*) ∈ *set qs*› **have** *hU-in*: (*h*, *U*) ∈ *insert ?p1 ?A* **by** (*simp add*: *card-U*)
have *?p1* ≠ (*h*, *U*)
proof
assume *?p1* = (*h*, *U*)
hence *lpp* (*punit.monom-mult 1 ?x h*) = *lpp h* **by** *simp*
with ‹*h* ≠ *0*› **show** *False* **by** (*simp add*: *punit.lt-monom-mult monomial-0-iff*)
qed
let *?qs* = *shift-list* (*h*, *U*) *x qs*
have {*q* ∈ *set* (*fst* (*?qs*, *B* − {(*h*, *U*)})). *m* < *card* (*snd q*)} = (*insert ?p1 ?A*) − {(*h*, *U*)}
using *step*(*7*) *card-U* ‹*?p1* ≠ (*h*, *U*)› **by** (*fastforce simp*: *shift-list.simps*)
also from *fin2 hU-in* **have** *card* . . . = *card* (*insert ?p1 ?A*) − *1* **by** (*simp add*: *card-Diff-singleton-if*)
also from *fin1* **have** . . . ≤ *Suc* (*card ?A*) − *1* **by** (*simp add*: *card-insert-if*)
also have . . . = *card* {*q* ∈ *set* (*fst* (*qs*, *B*)). *m* < *card* (*snd q*)} **by** *simp*
finally have *card* {*q* ∈ *set* (*fst* (*?qs*, *B* − {(*h*, *U*)})). *m* < *card* (*snd q*)} ≤
card {*q* ∈ *set* (*fst* (*qs*, *B*)). *m* < *card* (*snd q*)} **.**
with *step*(*8*) **show** *?case* **by** (*rule le-trans*)
qed

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

(*fst* (*shift1 args*))
proof (*induct args rule*: *shift1-induct*)
  case (*base qs B*)
  from *base*(*3*) **show** *?case* **.**
next
  case (*step qs B h U x*)
  from *step.hyps*(*1*) **have** *shift2-inv qs* **by** (*rule shift1-invD*)
  hence *valid-decomp X qs* **by** (*rule shift2-invD*)
  moreover from *step.prems* **have** *cone-decomp T qs* **by** (*simp only*: *fst-conv*)
  moreover from *step.hyps*(*1*−*3*) **have** $(h, U) \in$ *set qs* **by** (*rule shift1-inv-some-snd*)
  ultimately have *cone-decomp T* (*fst* (*shift-list* (h, U) *x qs*, $B - \{(h, U)\}$))
    unfolding *fst-conv* **using** *step.hyps*(*6*) **by** (*rule cone-decomp-shift-list*)
  thus *?case* **by** (*rule step.hyps*(*8*))
qed

lemma *shift1-4*:
  shift1-inv args $\Longrightarrow$
    Max (*poly-deg* ' *fst* ' *set* (*fst args*)) $\leq$ *Max* (*poly-deg* ' *fst* ' *set* (*fst* (*shift1 args*)))
proof (*induct args rule*: *shift1-induct*)
  case (*base qs B*)
  show *?case* **..**
next
  case (*step qs B h U x*)
  let *?x* = *Poly-Mapping.single x 1*
  let *?p1* = (*punit.monom-mult 1 ?x h*, *U*)
  let *?qs* = *shift-list* (h, U) *x qs*
  from *step*(*1*) **have** $B = \{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\}$
    and *inv2*: *shift2-inv qs* **by** (*rule shift1-invD*)+
  from *this*(*1*) **have** $B \subseteq$ *set qs* **by** *auto*
  with *step*(*2*) **have** *set qs* $\neq \{\}$ **by** *auto*
  from *finite-set* **have** *fin*: *finite* (*poly-deg* ' *fst* ' *set ?qs*) **by** (*intro finite-imageI*)
  have *Max* (*poly-deg* ' *fst* ' *set* (*fst* (*qs*, *B*))) $\leq$ *Max* (*poly-deg* ' *fst* ' *set* (*fst* (*?qs*,
$B - \{(h, U)\}$)))
    unfolding *fst-conv*
  proof (*rule Max.boundedI*)
    from *finite-set* **show** *finite* (*poly-deg* ' *fst* ' *set qs*) **by** (*intro finite-imageI*)
  next
    from ‹*set qs* $\neq \{\}$› **show** *poly-deg* ' *fst* ' *set qs* $\neq \{\}$ **by** *simp*
  next
    fix *a*
    assume $a \in$ *poly-deg* ' *fst* ' *set qs*
    then obtain *q* **where** $q \in$ *set qs* **and** *a*: $a =$ *poly-deg* (*fst q*) **by** *blast*
    show $a \leq$ *Max* (*poly-deg* ' *fst* ' *set ?qs*)
    proof (*cases* $q = (h, U)$)
      case *True*
     hence $a \leq$ *poly-deg* (*fst ?p1*) **by** (*cases* $h = 0$) (*simp-all add*: *a poly-deg-monom-mult*)
      also from *fin* **have** $\ldots \leq$ *Max* (*poly-deg* ' *fst* ' *set ?qs*)
      proof (*rule Max-ge*)
        have *?p1* $\in$ *set ?qs* **by** (*simp add*: *shift-list.simps*)

thus *poly-deg* (*fst ?p1*) ∈ *poly-deg* ‘ *fst* ‘ *set ?qs* **by** (*intro imageI*)
qed
finally show *?thesis* **.**
next
case *False*
with ‹*q* ∈ *set qs*› **have** *q* ∈ *set ?qs* **by** (*simp add*: *shift-list.simps*)
hence *a* ∈ *poly-deg* ‘ *fst* ‘ *set ?qs* **unfolding** *a* **by** (*intro imageI*)
with *fin* **show** *?thesis* **by** (*rule Max-ge*)
qed
qed
thus *?case* **using** *step*(*8*) **by** (*rule le-trans*)
qed

lemma *shift1-5*: *shift1-inv args* ⟹ *fst* (*shift1 args*) = [] ⟷ *fst args* = []
proof (*induct args rule*: *shift1-induct*)
case (*base qs B*)
show *?case* **..**
next
case (*step qs B h U x*)
let *?p1* = (*punit.monom-mult 1* (*Poly-Mapping.single x 1*) *h*, *U*)
let *?qs* = *shift-list* (*h*, *U*) *x qs*
from *step*(*1*) **have** *B* = {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d* ∧ *m* < *card* (*snd q*)}
and *inv2*: *shift2-inv qs* **by** (*rule shift1-invD*)+
from *this*(*1*) **have** *B* ⊆ *set qs* **by** *auto*
with *step*(*2*) **have** *qs* ≠ [] **by** *auto*
moreover have *fst* (*shift1* (*?qs*, *B* − {(*h*, *U*)})) ≠ []
by (*simp add*: *step.hyps*(*8*) *del*: *One-nat-def*) (*simp add*: *shift-list.simps*)
ultimately show *?case* **by** *simp*
qed

lemma *shift1-6*: *shift1-inv args* ⟹ *monomial-decomp* (*fst args*) ⟹ *monomial-decomp* (*fst* (*shift1 args*))
proof (*induct args rule*: *shift1-induct*)
case (*base qs B*)
from *base*(*3*) **show** *?case* **.**
next
case (*step qs B h U x*)
from *step*(*1−3*) **have** (*h*, *U*) ∈ *set qs* **by** (*rule shift1-inv-some-snd*)
with *step.prems* **have** *monomial-decomp* (*fst* (*shift-list* (*h*, *U*) *x qs*, *B* − {(*h*, *U*)}))
unfolding *fst-conv* **by** (*rule monomial-decomp-shift-list*)
thus *?case* **by** (*rule step.hyps*)
qed

lemma *shift1-7*: *shift1-inv args* ⟹ *hom-decomp* (*fst args*) ⟹ *hom-decomp* (*fst* (*shift1 args*))
proof (*induct args rule*: *shift1-induct*)
case (*base qs B*)
from *base*(*3*) **show** *?case* **.**

next
  case (*step qs B h U x*)
  from *step*(*1*−*3*) **have** $(h,\ U) \in$ *set qs* **by** (*rule shift1-inv-some-snd*)
  with *step.prems* **have** *hom-decomp* (*fst* (*shift-list* $(h,\ U)$ *x qs*, $B - \{(h,\ U)\}$))
    unfolding *fst-conv* **by** (*rule hom-decomp-shift-list*)
  thus *?case* **by** (*rule step.hyps*)
qed

end

lemma *shift2-inv-preserved*:
  assumes *shift2-inv d qs*
  shows *shift2-inv* (*Suc d*) (*fst* (*shift1* (*qs*, $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m$ $<$ *card* (*snd q*)$\}$)))
proof −
  define *args* **where** *args* = (*qs*, $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\}$)
  from *refl assms* **have** *inv1*: *shift1-inv d args* **unfolding** *args-def*
    by (*rule shift1-invI*)
  hence *shift1-inv d* (*shift1 args*) **by** (*induct args rule*: *shift1-induct*)
  hence *shift1-inv d* (*fst* (*shift1 args*), *snd* (*shift1 args*)) **by** *simp*
  hence *shift2-inv d* (*fst* (*shift1 args*)) **by** (*rule shift1-invD*)
  hence *valid-decomp X* (*fst* (*shift1 args*)) **and** *standard-decomp k* (*fst* (*shift1 args*))
    and *exact-decomp* (*Suc m*) (*fst* (*shift1 args*)) **by** (*rule shift2-invD*)+
  thus *shift2-inv* (*Suc d*) (*fst* (*shift1 args*))
  proof (*rule shift2-invI*)
    fix *d0*
    assume $d0 <$ *Suc d*
    hence $d0 \le d$ **by** *simp*
    with *inv1* **show** *card* $\{q \in$ *set* (*fst* (*shift1 args*)). *poly-deg* (*fst q*) $= d0 \wedge m <$ *card* (*snd q*)$\} \le 1$
      by (*rule shift1-1*)
  qed
qed

function *shift2* :: $nat \Rightarrow nat \Rightarrow ((('x \Rightarrow_0 nat) \Rightarrow_0 'a) \times 'x\ set)\ list \Rightarrow$
    $((('x \Rightarrow_0 nat) \Rightarrow_0 'a{::}\{comm\text{-}ring\text{-}1, ring\text{-}no\text{-}zero\text{-}divisors\}) \times 'x\ set)\ list$ **where**
  shift2 c d qs =
    (*if* $c \le d$ *then qs*
    else shift2 c (*Suc d*) (*fst* (*shift1* (*qs*, $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\}$))))
  by *auto*
termination proof
  show *wf* (*measure* ($\lambda(c,\ d,\ \text{-}).\ c - d$)) **by** (*fact wf-measure*)
qed *simp*

lemma *shift2-1*: *shift2-inv d qs* $\Longrightarrow$ *shift2-inv c* (*shift2 c d qs*)
proof (*induct c d qs rule*: *shift2.induct*)

case *IH*: (*1 c d qs*)
show *?case*
proof (*subst shift2.simps*, *simp del*: *shift2.simps*, *intro conjI impI*)
assume $c \le d$
show *shift2-inv c qs*
proof (*rule shift2-invI*)
from *IH*(*2*) **show** *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp* (*Suc m*) *qs*
by (*rule shift2-invD*)+
next
fix *d0*
assume $d0 < c$
hence $d0 < d$ **using** ‹$c \le d$› **by** (*rule less-le-trans*)
with *IH*(*2*) **show** *card* $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d0 \wedge m <$ *card* (*snd q*)$\} \le 1$
by (*rule shift2-invD*)
qed
next
assume $\neg\ c \le d$
thus *shift2-inv c* (*shift2 c* (*Suc d*) (*fst* (*shift1* (*qs*, $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\}$))))
proof (*rule IH*)
from *IH*(*2*) **show** *shift2-inv* (*Suc d*) (*fst* (*shift1* (*qs*, $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\}$)))
by (*rule shift2-inv-preserved*)
qed
qed
qed

lemma *shift2-2*:
shift2-inv d qs $\Longrightarrow$
card $\{q \in$ *set* (*shift2 c d qs*). $m <$ *card* (*snd q*)$\} \le$ *card* $\{q \in$ *set qs*. $m <$ *card* (*snd q*)$\}$
proof (*induct c d qs rule*: *shift2.induct*)
case *IH*: (*1 c d qs*)
let *?A* $= \{q \in$ *set* (*shift2 c* (*Suc d*) (*fst* (*shift1* (*qs*, $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\}$)))). $m <$ *card* (*snd q*)$\}$
show *?case*
proof (*subst shift2.simps*, *simp del*: *shift2.simps*, *intro impI*)
assume $\neg\ c \le d$
hence *card ?A* $\le$ *card* $\{q \in$ *set* (*fst* (*shift1* (*qs*, $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\}$))). $m <$ *card* (*snd q*)$\}$
proof (*rule IH*)
show *shift2-inv* (*Suc d*) (*fst* (*shift1* (*qs*, $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\}$)))
using *IH*(*2*) **by** (*rule shift2-inv-preserved*)
qed
also have $\ldots \le$ *card* $\{q \in$ *set* (*fst* (*qs*, $\{q \in$ *set qs*. *poly-deg* (*fst q*) $= d \wedge m <$ *card* (*snd q*)$\}$)). $m <$ *card* (*snd q*)$\}$

using *refl IH*(*2*) **by** (*intro shift1-2 shift1-invI*)
finally show *card ?A* ≤ *card* {*q* ∈ *set qs*. *m* < *card* (*snd q*)} **by** (*simp only*: *fst-conv*)
qed
qed

lemma *shift2-3*: *shift2-inv d qs* ⟹ *cone-decomp T qs* ⟹ *cone-decomp T* (*shift2 c d qs*)
proof (*induct c d qs rule*: *shift2.induct*)
case *IH*: (*1 c d qs*)
have *inv2*: *shift2-inv* (*Suc d*) (*fst* (*shift1* (*qs*, {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d* ∧ *m* < *card* (*snd q*)})))
using *IH*(*2*) **by** (*rule shift2-inv-preserved*)
show *?case*
proof (*subst shift2.simps*, *simp add*: *IH.prems del*: *shift2.simps*, *intro impI*)
assume ¬ *c* ≤ *d*
moreover note *inv2*
moreover have *cone-decomp T* (*fst* (*shift1* (*qs*, {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d* ∧ *m* < *card* (*snd q*)})))
proof (*rule shift1-3*)
from *refl IH*(*2*) **show** *shift1-inv d* (*qs*, {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d* ∧ *m* < *card* (*snd q*)})
by (*rule shift1-invI*)
qed (*simp add*: *IH.prems*)
ultimately show *cone-decomp T* (*shift2 c* (*Suc d*) (*fst* (*shift1* (*qs*, {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d* ∧ *m* < *card* (*snd q*)}))))
by (*rule IH*)
qed
qed

lemma *shift2-4*:
shift2-inv d qs ⟹ *Max* (*poly-deg* ‘ *fst* ‘ *set qs*) ≤ *Max* (*poly-deg* ‘ *fst* ‘ *set* (*shift2 c d qs*))
proof (*induct c d qs rule*: *shift2.induct*)
case *IH*: (*1 c d qs*)
let *?args* = (*qs*, {*q* ∈ *set qs*. *poly-deg* (*fst q*) = *d* ∧ *m* < *card* (*snd q*)})
show *?case*
proof (*subst shift2.simps*, *simp del*: *shift2.simps*, *intro impI*)
assume ¬ *c* ≤ *d*
have *Max* (*poly-deg* ‘ *fst* ‘ *set* (*fst ?args*)) ≤ *Max* (*poly-deg* ‘ *fst* ‘ *set* (*fst* (*shift1 ?args*)))
using *refl IH*(*2*) **by** (*intro shift1-4 shift1-invI*)
also from ‹¬ *c* ≤ *d*› **have** ... ≤ *Max* (*poly-deg* ‘ *fst* ‘ *set* (*shift2 c* (*Suc d*) (*fst* (*shift1 ?args*))))
proof (*rule IH*)
from *IH*(*2*) **show** *shift2-inv* (*Suc d*) (*fst* (*shift1 ?args*))
by (*rule shift2-inv-preserved*)
qed
finally show *Max* (*poly-deg* ‘ *fst* ‘ *set qs*) ≤ *Max* (*poly-deg* ‘ *fst* ‘ *set* (*shift2 c*

$(Suc\ d)$ $(fst\ (shift1\ ?args))))$
    by $(simp\ only$: $fst\text{-}conv)$
  qed
qed

lemma $shift2\text{-}5$:
  $shift2\text{-}inv\ d\ qs \Longrightarrow shift2\ c\ d\ qs = [] \longleftrightarrow qs = []$
proof $(induct\ c\ d\ qs\ rule$: $shift2.induct)$
  case IH: $(1\ c\ d\ qs)$
  let $?args = (qs, \{q \in set\ qs.\ poly\text{-}deg\ (fst\ q) = d \wedge m < card\ (snd\ q)\})$
  show $?case$
  proof $(subst\ shift2.simps$, $simp\ del$: $shift2.simps$, $intro\ impI)$
    assume $\neg\ c \leq d$
    hence $shift2\ c\ (Suc\ d)\ (fst\ (shift1\ ?args)) = [] \longleftrightarrow fst\ (shift1\ ?args) = []$
    proof $(rule\ IH)$
      from $IH(2)$ **show** $shift2\text{-}inv\ (Suc\ d)\ (fst\ (shift1\ ?args))$
        by $(rule\ shift2\text{-}inv\text{-}preserved)$
    qed
    also from $refl\ IH(2)$ **have** $\ldots \longleftrightarrow fst\ ?args = []$ **by** $(intro\ shift1\text{-}5\ shift1\text{-}invI)$
    finally show $shift2\ c\ (Suc\ d)\ (fst\ (shift1\ ?args)) = [] \longleftrightarrow qs = []$ **by** $(simp$
$only$: $fst\text{-}conv)$
  qed
qed

lemma $shift2\text{-}6$:
  $shift2\text{-}inv\ d\ qs \Longrightarrow monomial\text{-}decomp\ qs \Longrightarrow monomial\text{-}decomp\ (shift2\ c\ d\ qs)$
proof $(induct\ c\ d\ qs\ rule$: $shift2.induct)$
  case IH: $(1\ c\ d\ qs)$
  let $?args = (qs, \{q \in set\ qs.\ poly\text{-}deg\ (fst\ q) = d \wedge m < card\ (snd\ q)\})$
  show $?case$
  proof $(subst\ shift2.simps$, $simp\ del$: $shift2.simps$, $intro\ conjI\ impI\ IH)$
   from $IH(2)$ **show** $shift2\text{-}inv\ (Suc\ d)\ (fst\ (shift1\ ?args))$ **by** $(rule\ shift2\text{-}inv\text{-}preserved)$
  next
    from $refl\ IH(2)$ **have** $shift1\text{-}inv\ d\ ?args$ **by** $(rule\ shift1\text{-}invI)$
    moreover from $IH(3)$ **have** $monomial\text{-}decomp\ (fst\ ?args)$ **by** $simp$
    ultimately show $monomial\text{-}decomp\ (fst\ (shift1\ ?args))$ **by** $(rule\ shift1\text{-}6)$
  qed
qed

lemma $shift2\text{-}7$:
  $shift2\text{-}inv\ d\ qs \Longrightarrow hom\text{-}decomp\ qs \Longrightarrow hom\text{-}decomp\ (shift2\ c\ d\ qs)$
proof $(induct\ c\ d\ qs\ rule$: $shift2.induct)$
  case IH: $(1\ c\ d\ qs)$
  let $?args = (qs, \{q \in set\ qs.\ poly\text{-}deg\ (fst\ q) = d \wedge m < card\ (snd\ q)\})$
  show $?case$
  proof $(subst\ shift2.simps$, $simp\ del$: $shift2.simps$, $intro\ conjI\ impI\ IH)$
   from $IH(2)$ **show** $shift2\text{-}inv\ (Suc\ d)\ (fst\ (shift1\ ?args))$ **by** $(rule\ shift2\text{-}inv\text{-}preserved)$
  next
    from $refl\ IH(2)$ **have** $shift1\text{-}inv\ d\ ?args$ **by** $(rule\ shift1\text{-}invI)$

moreover from *IH*(*3*) **have** *hom-decomp* (*fst ?args*) **by** *simp*
ultimately show *hom-decomp* (*fst* (*shift1 ?args*)) **by** (*rule shift1-7*)
qed
qed

definition *shift* :: ((($'x \Rightarrow_0$ *nat*) $\Rightarrow_0$ $'a$) $\times$ $'x$ *set*) *list* $\Rightarrow$
((($'x \Rightarrow_0$ *nat*) $\Rightarrow_0$ $'a$::{*comm-ring-1*,*ring-no-zero-divisors*}) $\times$
$'x$ *set*) *list*
where *shift qs* = *shift2* (*k* + *card* {*q* $\in$ *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*
using *assms*
proof (*rule shift2-invI*)
fix *d0*
assume *d0* < *k*
have {*q* $\in$ *set qs*. *poly-deg* (*fst q*) = *d0* $\wedge$ *m* < *card* (*snd q*)} = {}
proof –
{
fix *q*
assume *q* $\in$ *set qs*
obtain *h U* **where** *q*: *q* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
assume *poly-deg* (*fst q*) = *d0* **and** *m* < *card* (*snd q*)
hence *poly-deg h* < *k* **and** *m* < *card U* **using** ‹*d0* < *k*› **by** (*simp-all add*: *q*)
from *this*(*2*) **have** *U* $\neq$ {} **by** *auto*
with ‹*q* $\in$ *set qs*› **have** (*h*, *U*) $\in$ *set* (qs_+) **by** (*simp add*: *q pos-decomp-def*)
with *assms*(*2*) **have** *k* $\leq$ *poly-deg h* **by** (*rule standard-decompD*)
with ‹*poly-deg h* < *k*› **have** *False* **by** *simp*
}
thus *?thesis* **by** *blast*
qed
thus *card* {*q* $\in$ *set qs*. *poly-deg* (*fst q*) = *d0* $\wedge$ *m* < *card* (*snd q*)} $\leq$ *1* **by** (*simp only*: *card.empty*)
qed

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 –
define *c* **where** *c* = *card* {*q* $\in$ *set qs*. *m* < *card* (*snd q*)}
define *A* **where** *A* = {*q* $\in$ *set* (*shift qs*). *m* < *card* (*snd q*)}
from *assms* **have** *shift2-inv k qs* **by** (*rule shift2-inv-init*)
hence *inv2*: *shift2-inv* (*k* + *c*) (*shift qs*) **and** *card A* $\leq$ *c*
unfolding *shift-def c-def A-def* **by** (*rule shift2-1*, *rule shift2-2*)
from *inv2* **have** *fin*: *valid-decomp X* (*shift qs*) **and** *std*: *standard-decomp k* (*shift*

qs)
  and *exct*: *exact-decomp* (*Suc m*) (*shift qs*)
  by (*rule shift2-invD*)+
 show *valid-decomp X* (*shift qs*) **and** *standard-decomp k* (*shift qs*) **by** *fact*+
 have *finite A* **by** (*auto simp*: *A-def*)

 show *exact-decomp m* (*shift qs*)
 proof (*rule exact-decompI*)
  fix *h U*
  assume (*h*, *U*) ∈ *set* (*shift qs*)
  with *exct* **show** *h* ∈ *P*[*X*] **and** *U* ⊆ *X* **by** (*rule exact-decompD*)+
 next
  fix *h1 h2 U1 U2*
  assume *1*: (*h1*, *U1*) ∈ *set* (*shift qs*) **and** *2*: (*h2*, *U2*) ∈ *set* (*shift qs*)
  assume *3*: *poly-deg h1* = *poly-deg h2* **and** *4*: *m* < *card U1* **and** *5*: *m* < *card U2*
  from *5* **have** *U2* ≠ {} **by** *auto*
  with *2* **have** (*h2*, *U2*) ∈ *set* ((*shift qs*)$_+$) **by** (*simp add*: *pos-decomp-def*)
  let *?C* = {*q* ∈ *set* (*shift qs*). *poly-deg* (*fst q*) = *poly-deg h2* ∧ *m* < *card* (*snd q*)}
  define *B* **where** *B* = {*q* ∈ *A*. *k* ≤ *poly-deg* (*fst q*) ∧ *poly-deg* (*fst q*) ≤ *poly-deg h2*}
  have *Suc* (*poly-deg h2*) − *k* ≤ *card B*
  proof −
   have *B* = (⋃ *d0*∈{*k*..*poly-deg h2*}. {*q* ∈ *A*. *poly-deg* (*fst q*) = *d0*}) **by** (*auto simp*: *B-def*)
   also have *card* ... = (∑ *d0*=*k*..*poly-deg h2*. *card* {*q* ∈ *A*. *poly-deg* (*fst q*) = *d0*})
   proof (*intro card-UN-disjoint ballI impI*)
    fix *d0*
     from - ‹*finite A*› **show** *finite* {*q* ∈ *A*. *poly-deg* (*fst q*) = *d0*} **by** (*rule finite-subset*) *blast*
   next
    fix *d0 d1* :: *nat*
    assume *d0* ≠ *d1*
    thus {*q* ∈ *A*. *poly-deg* (*fst q*) = *d0*} ∩ {*q* ∈ *A*. *poly-deg* (*fst q*) = *d1*} = {} **by** *blast*
   qed (*fact finite-atLeastAtMost*)
   also have ... ≥ (∑ *d0*=*k*..*poly-deg h2*. *1*)
   proof (*rule sum-mono*)
    fix *d0*
    assume *d0* ∈ {*k*..*poly-deg h2*}
    hence *k* ≤ *d0* **and** *d0* ≤ *poly-deg h2* **by** *simp-all*
    with *std* ‹(*h2*, *U2*) ∈ *set* ((*shift qs*)$_+$)› **obtain** *h′ U′* **where** (*h′*, *U′*) ∈ *set* (*shift qs*)
     and *poly-deg h′* = *d0* **and** *card U2* ≤ *card U′* **by** (*rule standard-decompE*)
    from *5 this*(*3*) **have** *m* < *card U′* **by** (*rule less-le-trans*)
    with ‹(*h′*, *U′*) ∈ *set* (*shift qs*)› **have** (*h′*, *U′*) ∈ {*q* ∈ *A*. *poly-deg* (*fst q*) = *d0*}

by (*simp add*: *A-def* ‹*poly-deg h′* = *d0*›)
hence $\{q \in A.\ poly\text{-}deg\ (fst\ q) = d0\} \neq \{\}$ **by** *blast*
moreover from - ‹*finite A*› **have** *finite* $\{q \in A.\ poly\text{-}deg\ (fst\ q) = d0\}$
by (*rule finite-subset*) *blast*
ultimately show $1 \leq card\ \{q \in A.\ poly\text{-}deg\ (fst\ q) = d0\}$
by (*simp add*: *card-gt-0-iff Suc-le-eq*)
qed
also have ($\sum d0{=}k..poly\text{-}deg\ h2.\ 1$) = *Suc* (*poly-deg h2*) − *k* **by** *auto*
finally show *?thesis* **.**
qed
also from ‹*finite A*› - **have** ... ≤ *card A* **by** (*rule card-mono*) (*auto simp*: *B-def*)
also have ... ≤ *c* **by** *fact*
finally have *poly-deg h2* < *k* + *c* **by** *simp*
with *inv2* **have** *card ?C* ≤ *1* **by** (*rule shift2-invD*)
have *finite ?C* **by** *auto*
moreover note ‹*card ?C* ≤ *1*›
moreover from *1 3 4* **have** (*h1*, *U1*) ∈ *?C* **by** *simp*
moreover from *2 5* **have** (*h2*, *U2*) ∈ *?C* **by** *simp*
ultimately show (*h1*, *U1*) = (*h2*, *U2*) **by** (*auto simp*: *card-le-Suc0-iff-eq*)
qed
qed

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 −
from *assms*(*1*, *2*, *3*) **have** *shift2-inv k qs* **by** (*rule shift2-inv-init*)
thus *?thesis* **unfolding** *shift-def* **using** *assms*(*4*) **by** (*rule shift2-6*)
qed

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 −
from *assms*(*1*, *2*, *3*) **have** *shift2-inv k qs* **by** (*rule shift2-inv-init*)
thus *?thesis* **unfolding** *shift-def* **using** *assms*(*4*) **by** (*rule shift2-7*)
qed

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 −

from *assms*(*1*, *2*, *3*) **have** *shift2-inv k qs* **by** (*rule shift2-inv-init*)
thus *?thesis* **unfolding** *shift-def* **using** *assms*(*4*) **by** (*rule shift2-3*)
qed

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*) $\leq$ *Max* (*poly-deg* ‘ *fst* ‘ *set* (*shift qs*))
proof −
from *assms*(*1*−*3*) **have** *shift2-inv k qs* **by** (*rule shift2-inv-init*)
thus *?thesis* **unfolding** *shift-def* **by** (*rule shift2-4*)
qed

lemma *shift-Nil-iff*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *exact-decomp* (*Suc m*) *qs*
shows *shift qs* = [] $\longleftrightarrow$ *qs* = []
proof −
from *assms*(*1*−*3*) **have** *shift2-inv k qs* **by** (*rule shift2-inv-init*)
thus *?thesis* **unfolding** *shift-def* **by** (*rule shift2-5*)
qed

end

primrec *exact-aux* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ (((*'x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'a*) $\times$ *'x set*) *list* $\Rightarrow$
(((*'x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'a*::{*comm-ring-1*,*ring-no-zero-divisors*}) $\times$ *'x set*) *list* **where**
exact-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 −
from *assms* **have** *?thesis1* $\wedge$ *?thesis2* $\wedge$ *?thesis3*
proof (*induct m arbitrary*: *qs*)
case *0*
thus *?case* **by** *simp*
next
case (*Suc m*)
let *?qs* = *shift k m qs*
have *valid-decomp X* (*exact-aux k m ?qs*) $\wedge$ *standard-decomp k* (*exact-aux k m ?qs*) $\wedge$
exact-decomp 0 (*exact-aux k m ?qs*)
proof (*rule Suc*)
from *Suc.prems* **show** *valid-decomp X ?qs* **and** *standard-decomp k ?qs* **and** *exact-decomp m ?qs*

```
      by (rule shift)+
    qed
    thus ?case by simp
  qed
  thus ?thesis1 and ?thesis2 and ?thesis3 by simp-all
qed

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)
  using assms
proof (induct m arbitrary: qs)
  case 0
  thus ?case by simp
next
  case (Suc m)
  let ?qs = shift k m qs
  have monomial-decomp (exact-aux k m ?qs)
  proof (rule Suc)
    show valid-decomp X ?qs and standard-decomp k ?qs and exact-decomp m ?qs
      using Suc.prems(1, 2, 3) by (rule shift)+
  next
    from Suc.prems show monomial-decomp ?qs by (rule monomial-decomp-shift)
  qed
  thus ?case by simp
qed

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)
  using assms
proof (induct m arbitrary: qs)
  case 0
  thus ?case by simp
next
  case (Suc m)
  let ?qs = shift k m qs
  have hom-decomp (exact-aux k m ?qs)
  proof (rule Suc)
    show valid-decomp X ?qs and standard-decomp k ?qs and exact-decomp m ?qs
      using Suc.prems(1, 2, 3) by (rule shift)+
  next
    from Suc.prems show hom-decomp ?qs by (rule hom-decomp-shift)
  qed
  thus ?case by simp
qed
```

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*)
using *assms*
proof (*induct m arbitrary*: *qs*)
case *0*
thus *?case* **by** *simp*
next
case (*Suc m*)
let *?qs* = *shift k m qs*
have *cone-decomp T* (*exact-aux k m ?qs*)
proof (*rule Suc*)
show *valid-decomp X ?qs* **and** *standard-decomp k ?qs* **and** *exact-decomp m ?qs*
using *Suc.prems*(*1*, *2*, *3*) **by** (*rule shift*)+
next
from *Suc.prems* **show** *cone-decomp T ?qs* **by** (*rule cone-decomp-shift*)
qed
thus *?case* **by** *simp*
qed

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*))
using *assms*
proof (*induct m arbitrary*: *qs*)
case *0*
thus *?case* **by** *simp*
next
case (*Suc m*)
let *?qs* = *shift k m qs*
from *Suc.prems* **have** *Max* (*poly-deg* ‘ *fst* ‘ *set qs*) ≤ *Max* (*poly-deg* ‘ *fst* ‘ *set ?qs*)
by (*rule Max-shift-ge*)
also have ... ≤ *Max* (*poly-deg* ‘ *fst* ‘ *set* (*exact-aux k m ?qs*))
proof (*rule Suc*)
from *Suc.prems* **show** *valid-decomp X ?qs* **and** *standard-decomp k ?qs* **and** *exact-decomp m ?qs*
by (*rule shift*)+
qed
finally show *?case* **by** *simp*
qed

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* = []
using *assms*
proof (*induct m arbitrary*: *qs*)

case *0*
thus *?case* **by** *simp*
next
case (*Suc m*)
let *?qs* = *shift k m qs*
have *exact-aux k m ?qs* = [] ⟷ *?qs* = []
proof (*rule Suc*)
from *Suc.prems* **show** *valid-decomp X ?qs* **and** *standard-decomp k ?qs* **and** *exact-decomp m ?qs*
by (*rule shift*)+
qed
also from *Suc.prems* **have** ... ⟷ *qs* = [] **by** (*rule shift-Nil-iff*)
finally show *?case* **by** *simp*
qed

definition *exact* :: *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 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 −
from *assms*(*1*) *le-refl* **have** *exact-decomp* (*card X*) *qs* **by** (*rule exact-decomp-card-X*)
with *assms* **show** *?thesis1* **and** *?thesis2* **and** *?thesis3* **unfolding** *exact-def* **by** (*rule exact-aux*)+
qed

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 −
from *assms*(*1*) *le-refl* **have** *exact-decomp* (*card X*) *qs* **by** (*rule exact-decomp-card-X*)
with *assms*(*1*, *2*) **show** *?thesis* **unfolding** *exact-def* **using** *assms*(*3*) **by** (*rule monomial-decomp-exact-aux*)
qed

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 −
from *assms*(*1*) *le-refl* **have** *exact-decomp* (*card X*) *qs* **by** (*rule exact-decomp-card-X*)
with *assms*(*1*, *2*) **show** *?thesis* **unfolding** *exact-def* **using** *assms*(*3*) **by** (*rule hom-decomp-exact-aux*)
qed

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 −
from *assms*(*1*) *le-refl* **have** *exact-decomp* (*card X*) *qs* **by** (*rule exact-decomp-card-X*)
with *assms*(*1*, *2*) **show** *?thesis* **unfolding** *exact-def* **using** *assms*(*3*) **by** (*rule cone-decomp-exact-aux*)
qed

lemma *Max-exact-ge*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs*
shows *Max* (*poly-deg* ' *fst* ' *set qs*) $\leq$ *Max* (*poly-deg* ' *fst* ' *set* (*exact k qs*))
proof −
from *assms*(*1*) *le-refl* **have** *exact-decomp* (*card X*) *qs* **by** (*rule exact-decomp-card-X*)
with *assms*(*1*, *2*) **show** *?thesis* **unfolding** *exact-def* **by** (*rule Max-exact-aux-ge*)
qed

lemma *exact-Nil-iff*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs*
shows *exact k qs* = [] $\longleftrightarrow$ *qs* = []
proof −
from *assms*(*1*) *le-refl* **have** *exact-decomp* (*card X*) *qs* **by** (*rule exact-decomp-card-X*)
with *assms*(*1*, *2*) **show** *?thesis* **unfolding** *exact-def* **by** (*rule exact-aux-Nil-iff*)
qed

corollary b-*zero-exact*:
assumes *valid-decomp X qs* **and** *standard-decomp k qs* **and** *qs* $\neq$ []
shows *Suc* (*Max* (*poly-deg* ' *fst* ' *set qs*)) $\leq$ b (*exact k qs*) *0*
proof −
from *assms*(*1*, *2*) **have** *Max* (*poly-deg* ' *fst* ' *set qs*) $\leq$ *Max* (*poly-deg* ' *fst* ' *set* (*exact k qs*))
by (*rule Max-exact-ge*)
also have *Suc* ... $\leq$ b (*exact k qs*) *0*
proof (*rule* b-*zero*)
from *assms* **show** *exact k qs* $\neq$ [] **by** (*simp add*: *exact-Nil-iff*)
qed
finally show *?thesis* **by** *simp*
qed

lemma *normal-form-exact-decompE*:
assumes *F* $\subseteq$ *P*[*X*]
obtains *qs* **where** *valid-decomp X qs* **and** *standard-decomp 0 qs* **and** *monomial-decomp qs*
and *cone-decomp* (*normal-form F* ' *P*[*X*]) *qs* **and** *exact-decomp 0 qs*
and $\bigwedge$*g*. ($\bigwedge$*f*. *f* $\in$ *F* $\Longrightarrow$ *homogeneous f*) $\Longrightarrow$ *g* $\in$ *punit.reduced-GB F* $\Longrightarrow$ *poly-deg g* $\leq$ b *qs 0*
proof −
let *?G* = *punit.reduced-GB F*
let *?S* = *lpp* ' *?G*

let *?N* = *normal-form F* ‘ *P*[*X*]
define *qs*::((- $\Rightarrow_0$ *'a*) × -) *list* **where** *qs* = *snd* (*split 0 X ?S*)
from *fin-X assms* **have** *std*: *standard-decomp 0 qs* **and** *cn*: *cone-decomp ?N qs*
unfolding *qs-def* **by** (*rule standard-cone-decomp-snd-split*)+
from *fin-X assms* **have** *finite ?G* **by** (*rule finite-reduced-GB-Polys*)
hence *finite ?S* **by** (*rule finite-imageI*)
with *fin-X subset-refl* **have** *valid*: *valid-decomp X qs* **unfolding** *qs-def* **using** *zero-in-PPs*
by (*rule valid-decomp-split*)
from *fin-X subset-refl* ‹*finite ?S*› **have** *md*: *monomial-decomp qs*
unfolding *qs-def* **by** (*rule monomial-decomp-split*)
let *?qs* = *exact 0 qs*
from *valid std* **have** *valid-decomp X ?qs* **and** *standard-decomp 0 ?qs* **by** (*rule exact*)+
moreover from *valid std md* **have** *monomial-decomp ?qs* **by** (*rule monomial-decomp-exact*)
moreover from *valid std cn* **have** *cone-decomp ?N ?qs* **by** (*rule cone-decomp-exact*)
moreover from *valid std* **have** *exact-decomp 0 ?qs* **by** (*rule exact*)
moreover have *poly-deg g* ≤ b *?qs 0* **if** $\bigwedge f.\ f \in F \Longrightarrow$ *homogeneous f* **and** *g* ∈ *?G* **for** *g*
proof (*cases qs* = [])
case *True*
from *one-in-Polys* **have** *normal-form F 1* ∈ *?N* **by** (*rule imageI*)
also from *True cn* **have** ... = {*0*} **by** (*simp add*: *cone-decomp-def direct-decomp-def bij-betw-def*)
finally have *?G* = {*1*} **using** *fin-X assms*
by (*simp add*: *normal-form-zero-iff ideal-eq-UNIV-iff-reduced-GB-eq-one-Polys flip*: *ideal-eq-UNIV-iff-contains-one*)
with *that*(*2*) **show** *?thesis* **by** *simp*
next
case *False*
from *fin-X assms that* **have** *poly-deg g* ≤ *Suc* (*Max* (*poly-deg* ‘ *fst* ‘ *set qs*))
unfolding *qs-def* **by** (*rule standard-cone-decomp-snd-split*)
also from *valid std False* **have** ... ≤ b *?qs 0* **by** (*rule* b-*zero-exact*)
finally show *?thesis* **.**
qed
ultimately show *?thesis* **..**
qed

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) + (∑ i=j+3..n−1. (Dube-aux
i) choose (Suc (i − j)))
               else if j + 2 = n then d² + 2 ∗ d else 2 ∗ d)
  by pat-completeness auto
termination proof
  show wf (measure ((−) n)) by (fact wf-measure)
qed auto

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 (induct j rule: Dube-aux.induct)
  case step: (1 j)
  have j + 2 < n ∨ j + 2 = n ∨ n < j + 2 by auto
  show ?case
  proof (rule linorder-cases)
    assume ∗: j + 2 < n
    hence 1: d ≤ Dube-aux (j + 1)
      by (rule step.hyps)+
    show ?thesis
    proof (cases d ≤ 2)
      case True
      also from ∗ have 2 ≤ Dube-aux j by simp
      finally show ?thesis .
    next
      case False
      hence 2 < d by simp
      hence 2 < Dube-aux (j + 1) using 1 by (rule less-le-trans)
        with - have Dube-aux (j + 1) ≤ Dube-aux (j + 1) choose 2 by (rule
upper-le-binomial) simp
      also from ∗ have . . . ≤ Dube-aux j by simp
      finally have Dube-aux (j + 1) ≤ Dube-aux j .
      with 1 show ?thesis by (rule le-trans)
    qed
  next
    assume j + 2 = n
    thus ?thesis by simp
```

next
assume $n < j + 2$
thus *?thesis* **by** *simp*
qed
qed

corollary *Dube-ge-d*: $d \leq Dube$
by (*simp add*: *Dube-def Dube-aux-ge-d del*: *Dube-aux.simps*)

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 $j = 1$), 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 $\textit{rat-of-nat Dube} \leq 2 * ((\textit{rat-of-nat } d)^2 \ / \ 2 + (\textit{rat-of-nat } d)) \ \hat{}\ (2 \ \hat{}\ (n - 2))$
oops

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 $h \in P[X]$ **and** $h \neq 0$ **and** $U \subseteq X$ **and** *homogeneous* (h::- $\Rightarrow_0$ *'a::field*)
shows *Hilbert-fun* (*cone* (h, U)) z = *card* $\{t \in .[U].\ \textit{deg-pm } t + \textit{poly-deg } h = z\}$
proof −
from *assms*(*2*) **have** *lpp h* $\in$ *keys h* **by** (*rule punit.lt-in-keys*)
with *assms*(*4*) **have** *deg-h*[*symmetric*]: *deg-pm* (*lpp h*) = *poly-deg h*
by (*rule homogeneousD-poly-deg*)
from *assms*(*1*, *3*) **have** *cone* (h, U) $\subseteq P[X]$ **by** (*rule cone-subset-PolysI*)

with *fin-X* **have** *Hilbert-fun* (*cone* (*h*, *U*)) *z* = *card* (*lpp* ‘ (*hom-deg-set* *z* (*cone* (*h*, *U*)) − {*0*}))
using *subspace-cone*[*of* (*h*, *U*)] **by** (*simp only*: *Hilbert-fun-alt*)
also from *assms*(*4*) **have** *lpp* ‘ (*hom-deg-set* *z* (*cone* (*h*, *U*)) − {*0*}) =
{*t* ∈ *lpp* ‘ (*cone* (*h*, *U*) − {*0*}). *deg-pm* *t* = *z*}
by (*intro image-lt-hom-deg-set homogeneous-set-coneI*)
also have {*t* ∈ *lpp* ‘ (*cone* (*h*, *U*) − {*0*}). *deg-pm* *t* = *z*} =
(λ*t*. *t* + *lpp* *h*) ‘ {*t* ∈ .[*U*]. *deg-pm* *t* + *poly-deg* *h* = *z*} (**is** *?A* = *?B*)
proof
show *?A* ⊆ *?B*
proof
fix *t*
assume *t* ∈ *?A*
hence *t* ∈ *lpp* ‘ (*cone* (*h*, *U*) − {*0*}) **and** *deg-pm* *t* = *z* **by** *simp-all*
from *this*(*1*) **obtain** *a* **where** *a* ∈ *cone* (*h*, *U*) − {*0*} **and** *2*: *t* = *lpp* *a* **..**
from *this*(*1*) **have** *a* ∈ *cone* (*h*, *U*) **and** *a* ≠ *0* **by** *simp-all*
from *this*(*1*) **obtain** *q* **where** *q* ∈ *P*[*U*] **and** *a*: *a* = *q* ∗ *h* **by** (*rule coneE*)
from ‹*a* ≠ *0*› **have** *q* ≠ *0* **by** (*auto simp*: *a*)
hence *t*: *t* = *lpp* *q* + *lpp* *h* **using** *assms*(*2*) **unfolding** *2* *a* **by** (*rule lp-times*)
hence *deg-pm* (*lpp* *q*) + *poly-deg* *h* = *deg-pm* *t* **by** (*simp add*: *deg-pm-plus* *deg-h*)
also have . . . = *z* **by** *fact*
finally have *deg-pm* (*lpp* *q*) + *poly-deg* *h* = *z* **.**
moreover from ‹*q* ∈ *P*[*U*]› **have** *lpp* *q* ∈ .[*U*] **by** (*rule PPs-closed-lpp*)
ultimately have *lpp* *q* ∈ {*t* ∈ .[*U*]. *deg-pm* *t* + *poly-deg* *h* = *z*} **by** *simp*
moreover have *t* = *lpp* *q* + *lpp* *h* **by** (*simp only*: *t*)
ultimately show *t* ∈ *?B* **by** (*rule rev-image-eqI*)
qed
next
show *?B* ⊆ *?A*
proof
fix *t*
assume *t* ∈ *?B*
then obtain *s* **where** *s* ∈ {*t* ∈ .[*U*]. *deg-pm* *t* + *poly-deg* *h* = *z*}
and *t1*: *t* = *s* + *lpp* *h* **..**
from *this*(*1*) **have** *s* ∈ .[*U*] **and** *1*: *deg-pm* *s* + *poly-deg* *h* = *z* **by** *simp-all*
let *?q* = *monomial* (*1*::′*a*) *s*
have *?q* ≠ *0* **by** (*simp add*: *monomial-0-iff*)
hence *?q* ∗ *h* ≠ *0* **and** *lpp* (*?q* ∗ *h*) = *lpp* *?q* + *lpp* *h* **using** ‹*h* ≠ *0*›
by (*rule times-not-zero*, *rule lp-times*)
hence *t*: *t* = *lpp* (*?q* ∗ *h*) **by** (*simp add*: *t1 punit.lt-monomial*)
from ‹*s* ∈ .[*U*]› **have** *?q* ∈ *P*[*U*] **by** (*rule Polys-closed-monomial*)
with *refl* **have** *?q* ∗ *h* ∈ *cone* (*h*, *U*) **by** (*rule coneI*)
moreover from - *assms*(*2*) **have** *?q* ∗ *h* ≠ *0* **by** (*rule times-not-zero*) (*simp add*: *monomial-0-iff*)
ultimately have *?q* ∗ *h* ∈ *cone* (*h*, *U*) − {*0*} **by** *simp*
hence *t* ∈ *lpp* ‘ (*cone* (*h*, *U*) − {*0*}) **unfolding** *t* **by** (*rule imageI*)
moreover have *deg-pm* *t* = *int* *z* **by** (*simp add*: *t1*) (*simp add*: *deg-pm-plus* *deg-h* *flip*: *1*)

ultimately show $t \in$ *?A* **by** *simp*
qed
qed
also have *card* $\ldots$ = *card* $\{t \in .[U].$ *deg-pm* t + *poly-deg* $h = z\}$ **by** (*simp add*: *card-image*)
finally show *?thesis* **.**
qed

lemma *Hilbert-fun-cone-empty*:
assumes $h \in P[X]$ **and** $h \neq 0$ **and** *homogeneous* (h::- $\Rightarrow_0$ *'a::field*)
shows *Hilbert-fun* (*cone* (h, $\{\}$)) z = (*if poly-deg* $h = z$ *then* 1 *else* 0)
proof −
have *Hilbert-fun* (*cone* (h, $\{\}$)) z = *card* $\{t \in .[\{\}$::*'x set*$]$. *deg-pm* t + *poly-deg* $h = z\}$
using *assms*(1, 2) *empty-subsetI assms*(3) **by** (*rule Hilbert-fun-cone-aux*)
also have $\ldots$ = (*if poly-deg* $h = z$ *then* 1 *else* 0) **by** *simp*
finally show *?thesis* **.**
qed

lemma *Hilbert-fun-cone-nonempty*:
assumes $h \in P[X]$ **and** $h \neq 0$ **and** $U \subseteq X$ **and** *homogeneous* (h::- $\Rightarrow_0$ *'a::field*) **and** $U \neq \{\}$
shows *Hilbert-fun* (*cone* (h, U)) z =
(*if poly-deg* $h \leq z$ *then* ((z − *poly-deg* h) + (*card* U − 1)) *choose* (*card* U − 1) *else* 0)
proof (*cases poly-deg* $h \leq z$)
case *True*
from *assms*(3) *fin-X* **have** *finite* U **by** (*rule finite-subset*)
from *assms*(1−4) **have** *Hilbert-fun* (*cone* (h, U)) z = *card* $\{t \in .[U].$ *deg-pm* t + *poly-deg* $h = z\}$
by (*rule Hilbert-fun-cone-aux*)
also from *True* **have** $\{t \in .[U].$ *deg-pm* t + *poly-deg* $h = z\}$ = *deg-sect* U (z − *poly-deg* h)
by (*auto simp*: *deg-sect-def*)
also from ‹*finite* U› *assms*(5) **have** *card* $\ldots$ = (z − *poly-deg* h) + (*card* U − 1) *choose* (*card* U − 1)
by (*rule card-deg-sect*)
finally show *?thesis* **by** (*simp add*: *True*)
next
case *False*
from *assms*(1−4) **have** *Hilbert-fun* (*cone* (h, U)) z = *card* $\{t \in .[U].$ *deg-pm* t + *poly-deg* $h = z\}$
by (*rule Hilbert-fun-cone-aux*)
also from *False* **have** $\{t \in .[U].$ *deg-pm* t + *poly-deg* $h = z\} = \{\}$ **by** *auto*
hence *card* $\{t \in .[U].$ *deg-pm* t + *poly-deg* $h = z\}$ = *card* ($\{\}$::(*'x* $\Rightarrow_0$ *nat*) *set*) **by** (*rule arg-cong*)
also have $\ldots$ = 0 **by** *simp*
finally show *?thesis* **by** (*simp add*: *False*)
qed

corollary *Hilbert-fun-Polys*:
assumes $X \neq \{\}$
shows *Hilbert-fun* $(P[X]::(\text{-} \Rightarrow_0 \ 'a::field) \ set) \ z = (z + (card \ X - 1)) \ choose \ (card \ X - 1)$
proof −
let $?one = 1::('x \Rightarrow_0 nat) \Rightarrow_0 \ 'a$
have *Hilbert-fun* $(P[X]::(\text{-} \Rightarrow_0 \ 'a) \ set) \ z =$ *Hilbert-fun* $(cone \ (?one, X)) \ z$ **by** *simp*
also have $\ldots = (if \ poly\text{-}deg \ ?one \leq z \ then \ ((z - poly\text{-}deg \ ?one) + (card \ X - 1)) \ choose \ (card \ X - 1) \ else \ 0)$
using *one-in-Polys - subset-refl - assms* **by** (*rule Hilbert-fun-cone-nonempty*) *simp-all*
also have $\ldots = (z + (card \ X - 1)) \ choose \ (card \ X - 1)$ **by** *simp*
finally show *?thesis* **.**
qed

lemma *Hilbert-fun-cone-decomp*:
assumes *cone-decomp T ps* **and** *valid-decomp X ps* **and** *hom-decomp ps*
shows *Hilbert-fun* $T \ z = (\sum hU \in set \ ps. \ Hilbert\text{-}fun \ (cone \ hU) \ z)$
proof −
note *fin-X*
moreover from *assms*(2, 1) **have** $T \subseteq P[X]$ **by** (*rule valid-cone-decomp-subset-Polys*)
moreover from *assms*(1) **have** *dd*: *direct-decomp T* (*map cone ps*) **by** (*rule cone-decompD*)
ultimately have *Hilbert-fun* $T \ z = (\sum s \in set \ (map \ cone \ ps). \ Hilbert\text{-}fun \ s \ z)$
proof (*rule Hilbert-fun-direct-decomp*)
fix *cn*
assume $cn \in set \ (map \ cone \ ps)$
then obtain *hU* **where** $hU \in set \ ps$ **and** *cn*: $cn = cone \ hU$ **unfolding** *set-map* **..**
note *this*(1)
moreover obtain *h U* **where** *hU*: $hU = (h, U)$ **using** *prod.exhaust* **by** *blast*
ultimately have $(h, U) \in set \ ps$ **by** *simp*
with *assms*(3) **have** *homogeneous h* **by** (*rule hom-decompD*)
thus *homogeneous-set cn* **unfolding** *cn hU* **by** (*rule homogeneous-set-coneI*)
show *phull.subspace cn* **unfolding** *cn* **by** (*fact subspace-cone*)
qed
also have $\ldots = (\sum hU \in set \ ps. \ ((\lambda s. \ Hilbert\text{-}fun \ s \ z) \circ cone) \ hU)$ **unfolding** *set-map* **using** *finite-set*
proof (*rule sum.reindex-nontrivial*)
fix *hU1 hU2*
assume $hU1 \in set \ ps$ **and** $hU2 \in set \ ps$ **and** $hU1 \neq hU2$
with *dd* **have** $cone \ hU1 \cap cone \ hU2 = \{0\}$ **using** *zero-in-cone* **by** (*rule direct-decomp-map-Int-zero*)
moreover assume $cone \ hU1 = cone \ hU2$
ultimately show *Hilbert-fun* $(cone \ hU1) \ z = 0$ **by** *simp*
qed
finally show *?thesis* **by** *simp*

qed

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* − (∑ *i*=*1*..*n*. (*z* − *b i* + *i* − *1*) *gchoose i*))

lemma *poly-fun-Hilbert-poly*: *poly-fun* (*Hilbert-poly b*)
by (*simp add*: *Hilbert-poly-def Let-def*)

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*::- ⇒$_0$ ′*a*::*field*. (*h*, {}) ∈ *set ps* ∧ *poly-deg h* = *d*} + *Hilbert-poly* (b *ps*) *d*
proof −
define *n* **where** *n* = *card X*
with *assms*(*1*) **have** *0* < *n* **using** *fin-X* **by** (*simp add*: *card-gt-0-iff*)
hence *1* ≤ *n* **and** *Suc 0* ≤ *n* **by** *simp-all*
from *pos-decomp-subset* **have** *eq0*: (*set ps* − *set* (ps_+)) ∪ *set* (ps_+) = *set ps* **by** *blast*
have *set ps* − *set* (ps_+) ⊆ *set ps* **by** *blast*
hence *fin2*: *finite* (*set ps* − *set* (ps_+)) **using** *finite-set* **by** (*rule finite-subset*)

have (∑ *hU*∈*set ps* − *set* (ps_+). *Hilbert-fun* (*cone hU*) *d*) =
(∑ (*h*, *U*)∈*set ps* − *set* (ps_+). *if poly-deg h* = *d then 1 else 0*)
using *refl*
proof (*rule sum.cong*)
fix *x*
assume *x* ∈ *set ps* − *set* (ps_+)
moreover obtain *h U* **where** *x*: *x* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
ultimately have *U* = {} **and** (*h*, *U*) ∈ *set ps* **by** (*simp-all add*: *pos-decomp-def*)
from *assms*(*2*) *this*(*2*) **have** *h* ∈ *P*[*X*] **and** *h* ≠ *0* **by** (*rule valid-decompD*)+
moreover from *assms*(*3*) ‹(*h*, *U*) ∈ *set ps*› **have** *homogeneous h* **by** (*rule hom-decompD*)
ultimately show *Hilbert-fun* (*cone x*) *d* = (*case x of* (*h*, *U*) ⇒ *if poly-deg h* = *d then 1 else 0*)
by (*simp add*: *x* ‹*U* = {}› *Hilbert-fun-cone-empty split del*: *if-split*)
qed
also from *fin2* **have** . . . = (∑ (*h*, *U*)∈{(*h*′, *U*′) ∈ *set ps* − *set* (ps_+). *poly-deg h*′ = *d*}. *1*)
by (*rule sum.mono-neutral-cong-right*) (*auto split*: *if-splits*)
also have . . . = *card* {(*h*, *U*) ∈ *set ps* − *set* (ps_+). *poly-deg h* = *d*} **by** *auto*
also have . . . = *card* {*h*. (*h*, {}) ∈ *set ps* ∧ *poly-deg h* = *d*} **by** (*fact card-Diff-pos-decomp*)
finally have *eq1*: (∑ *hU*∈*set ps* − *set* (ps_+). *Hilbert-fun* (*cone hU*) *d*) =
card {*h*. (*h*, {}) ∈ *set ps* ∧ *poly-deg h* = *d*} **.**

let *?f* = $\lambda a\ b.$ (*int d*) − *a* + *b gchoose b*
have *int* ($\sum hU \in set\ (ps_+)$. *Hilbert-fun* (*cone hU*) *d*) = ($\sum hU \in set\ (ps_+)$. *int* (*Hilbert-fun* (*cone hU*) *d*))
by (*simp add*: *int-sum prod.case-distrib*)
also have ... = ($\sum (h, U) \in (\bigcup i \in \{1..n\}.\ \{(h, U) \in set\ (ps_+).\ card\ U = i\})$. *?f* (*poly-deg h*) (*card U* − *1*))
proof (*rule sum.cong*)
show *set* (ps_+) = ($\bigcup i \in \{1..n\}.\ \{(h, U).\ (h, U) \in set\ (ps_+) \wedge card\ U = i\}$)
proof (*rule Set.set-eqI*, *rule*)
fix *x*
assume $x \in set\ (ps_+)$
moreover obtain *h U* **where** *x*: *x* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
ultimately have $(h, U) \in set\ (ps_+)$ **by** *simp*
hence $(h, U) \in set\ ps$ **and** $U \neq \{\}$ **by** (*simp-all add*: *pos-decomp-def*)
from *fin-X assms*(*6*) *this*(*1*) **have** $U \subseteq X$ **by** (*rule exact-decompD*)
hence *finite U* **using** *fin-X* **by** (*rule finite-subset*)
with ‹$U \neq \{\}$› **have** *0* < *card U* **by** (*simp add*: *card-gt-0-iff*)
moreover from *fin-X* ‹$U \subseteq X$› **have** $card\ U \leq n$ **unfolding** *n-def* **by** (*rule card-mono*)
ultimately have $card\ U \in \{1..n\}$ **by** *simp*
moreover from ‹$(h, U) \in set\ (ps_+)$› **have** $(h, U) \in \{(h', U').\ (h', U') \in set\ (ps_+) \wedge card\ U' = card\ U\}$
by *simp*
ultimately show $x \in (\bigcup i \in \{1..n\}.\ \{(h, U).\ (h, U) \in set\ (ps_+) \wedge card\ U = i\})$ **by** (*simp add*: *x*)
qed *blast*
next
fix *x*
assume $x \in (\bigcup i \in \{1..n\}.\ \{(h, U).\ (h, U) \in set\ (ps_+) \wedge card\ U = i\})$
then obtain *j* **where** $j \in \{1..n\}$ **and** $x \in \{(h, U).\ (h, U) \in set\ (ps_+) \wedge card\ U = j\}$ **..**
from *this*(*2*) **obtain** *h U* **where** $(h, U) \in set\ (ps_+)$ **and** *card U* = *j* **and** *x*: *x* = (*h*, *U*) **by** *blast*
from *fin-X assms*(*2*, *5*) *this*(*1*) **have** *poly-deg h* < b *ps* (*Suc 0*) **by** (*rule* b-*one-gr*)
also have ... ≤ *d* **by** *fact*
finally have *poly-deg h* < *d* **.**
hence *int1*: *int* (*d* − *poly-deg h*) = *int d* − *int* (*poly-deg h*) **by** *simp*
from ‹*card U* = *j*› ‹$j \in \{1..n\}$› **have** *0* < *card U* **by** *simp*
hence *int2*: *int* (*card U* − *Suc 0*) = *int* (*card U*) − *1* **by** *simp*
from ‹$(h, U) \in set\ (ps_+)$› **have** $(h, U) \in set\ ps$ **using** *pos-decomp-subset* **..**
with *assms*(*2*) **have** $h \in P[X]$ **and** $h \neq 0$ **and** $U \subseteq X$ **by** (*rule valid-decompD*)+
moreover from *assms*(*3*) ‹$(h, U) \in set\ ps$› **have** *homogeneous h* **by** (*rule hom-decompD*)
moreover from ‹*0* < *card U*› **have** $U \neq \{\}$ **by** *auto*
ultimately have *Hilbert-fun* (*cone* (*h*, *U*)) *d* =
(*if poly-deg h* ≤ *d then* (*d* − *poly-deg h* + (*card U* − *1*)) *choose* (*card U* − *1*) *else 0*)
by (*rule Hilbert-fun-cone-nonempty*)

also from ‹*poly-deg h* $<$ *d*› **have** $\ldots = (d - \textit{poly-deg}\ h + (\textit{card}\ U - 1))\ \textit{choose}\ (\textit{card}\ U - 1)$ **by** *simp*
finally
have $\textit{int}\ (\textit{Hilbert-fun}\ (\textit{cone}\ (h,\ U))\ d) = (\textit{int}\ d - \textit{int}\ (\textit{poly-deg}\ h) + (\textit{int}\ (\textit{card}\ U - 1)))\ \textit{gchoose}\ (\textit{card}\ U - 1)$
by (*simp add*: *int-binomial int1 int2*)
thus $\textit{int}\ (\textit{Hilbert-fun}\ (\textit{cone}\ x)\ d) =$
$(\textit{case}\ x\ \textit{of}\ (h,\ U) \Rightarrow \textit{int}\ d - \textit{int}\ (\textit{poly-deg}\ h) + (\textit{int}\ (\textit{card}\ U - 1))\ \textit{gchoose}\ (\textit{card}\ U - 1))$
by (*simp add*: *x*)
qed
also have $\ldots = (\sum j{=}1..n.\ \sum(h,\ U){\in}\{(h',\ U') \in \textit{set}\ (ps_+).\ \textit{card}\ U' = j\}.\ ?f\ (\textit{poly-deg}\ h)\ (\textit{card}\ U - 1))$
proof (*intro sum.UNION-disjoint ballI*)
fix *j*
have $\{(h,\ U).\ (h,\ U) \in \textit{set}\ (ps_+) \wedge \textit{card}\ U = j\} \subseteq \textit{set}\ (ps_+)$ **by** *blast*
thus $\textit{finite}\ \{(h,\ U).\ (h,\ U) \in \textit{set}\ (ps_+) \wedge \textit{card}\ U = j\}$ **using** *finite-set* **by** (*rule finite-subset*)
qed *blast+*
also from *refl* **have** $\ldots = (\sum j{=}1..n.\ ?f\ (\mathrm{b}\ ps\ (\textit{Suc}\ j))\ j - ?f\ (\mathrm{b}\ ps\ j)\ j)$
proof (*rule sum.cong*)
fix *j*
assume $j \in \{1..n\}$
hence $\textit{Suc}\ 0 \le j$ **and** $0 < j$ **and** $j \le n$ **by** *simp-all*
from *fin-X this*(*1*) **have** $\mathrm{b}\ ps\ j \le \mathrm{b}\ ps\ (\textit{Suc}\ 0)$ **by** (*rule* b-*decreasing*)
also have $\ldots \le d$ **by** *fact*
finally have $\mathrm{b}\ ps\ j \le d$ **.**
from *fin-X* **have** $\mathrm{b}\ ps\ (\textit{Suc}\ j) \le \mathrm{b}\ ps\ j$ **by** (*rule* b-*decreasing*) *simp*
hence $\mathrm{b}\ ps\ (\textit{Suc}\ j) \le d$ **using** ‹b *ps j* $\le$ *d*› **by** (*rule le-trans*)
from ‹$0 < j$› **have** *int-j*: $\textit{int}\ (j - \textit{Suc}\ 0) = \textit{int}\ j - 1$ **by** *simp*
have $(\sum(h,\ U){\in}\{(h',\ U').\ (h',\ U') \in \textit{set}\ (ps_+) \wedge \textit{card}\ U' = j\}.\ ?f\ (\textit{poly-deg}\ h)\ (\textit{card}\ U - 1)) =$
$(\sum(h,\ U){\in}(\bigcup d0{\in}\{\mathrm{b}\ ps\ (\textit{Suc}\ j)..\textit{int}\ (\mathrm{b}\ ps\ j) - 1\}.\ \{(h',\ U').\ (h',\ U') \in \textit{set}\ (ps_+) \wedge \textit{int}\ (\textit{poly-deg}\ h') = d0 \wedge \textit{card}\ U' = j\}).$
$?f\ (\textit{poly-deg}\ h)\ (\textit{card}\ U - 1))$
using - *refl*
proof (*rule sum.cong*)
show $\{(h',\ U').\ (h',\ U') \in \textit{set}\ (ps_+) \wedge \textit{card}\ U' = j\} =$
$(\bigcup d0{\in}\{\mathrm{b}\ ps\ (\textit{Suc}\ j)..\textit{int}\ (\mathrm{b}\ ps\ j) - 1\}.\ \{(h',\ U').\ (h',\ U') \in \textit{set}\ (ps_+) \wedge \textit{int}\ (\textit{poly-deg}\ h') = d0 \wedge \textit{card}\ U' = j\})$
proof (*rule Set.set-eqI*, *rule*)
fix *x*
assume $x \in \{(h',\ U').\ (h',\ U') \in \textit{set}\ (ps_+) \wedge \textit{card}\ U' = j\}$
moreover obtain *h U* **where** *x*: $x = (h,\ U)$ **using** *prod.exhaust* **by** *blast*
ultimately have $(h,\ U) \in \textit{set}\ (ps_+)$ **and** $\textit{card}\ U = j$ **by** *simp-all*
with *fin-X assms*(*5*, *6*) ‹$\textit{Suc}\ 0 \le j$› ‹$j \le n$› **have** $\mathrm{b}\ ps\ (\textit{Suc}\ j) \le \textit{poly-deg}\ h$
unfolding *n-def* **by** (*rule lem-6-1-3*)
moreover from *fin-X* **have** $\textit{poly-deg}\ h < \mathrm{b}\ ps\ j$
proof (*rule* b)

from ‹$(h,\ U) \in set\ (ps_+)$› **show** $(h,\ U) \in set\ ps$ **using** *pos-decomp-subset*
..
next
show $j \leq card\ U$ **by** (*simp add*: ‹$card\ U = j$›)
qed
ultimately have $poly\text{-}deg\ h \in \{\mathrm{b}\ ps\ (Suc\ j)..int\ (\mathrm{b}\ ps\ j) - 1\}$ **by** *simp*
moreover have $(h,\ U) \in \{(h',\ U').\ (h',\ U') \in set\ (ps_+) \wedge poly\text{-}deg\ h' = poly\text{-}deg\ h \wedge card\ U' = card\ U\}$
using ‹$(h,\ U) \in set\ (ps_+)$› **by** *simp*
ultimately show $x \in (\bigcup d0{\in}\{\mathrm{b}\ ps\ (Suc\ j)..int\ (\mathrm{b}\ ps\ j) - 1\}.$
$\{(h',\ U').\ (h',\ U') \in set\ (ps_+) \wedge int\ (poly\text{-}deg\ h') = d0 \wedge card\ U' = j\})$
by (*simp add*: x ‹$card\ U = j$›)
qed *blast*
qed
also have $\ldots = (\sum d0{=}\mathrm{b}\ ps\ (Suc\ j)..int\ (\mathrm{b}\ ps\ j) - 1.$
$\sum (h,\ U){\in}\{(h',\ U').\ (h',\ U') \in set\ (ps_+) \wedge poly\text{-}deg\ h' = d0 \wedge card\ U' = j\}.$
$?f\ (poly\text{-}deg\ h)\ (card\ U - 1))$
proof (*intro sum.UNION-disjoint ballI*)
fix $d0::int$
have $\{(h',\ U').\ (h',\ U') \in set\ (ps_+) \wedge poly\text{-}deg\ h' = d0 \wedge card\ U' = j\} \subseteq set\ (ps_+)$ **by** *blast*
thus *finite* $\{(h',\ U').\ (h',\ U') \in set\ (ps_+) \wedge poly\text{-}deg\ h' = d0 \wedge card\ U' = j\}$
using *finite-set* **by** (*rule finite-subset*)
qed *blast+*
also from *refl* **have** $\ldots = (\sum d0{=}\mathrm{b}\ ps\ (Suc\ j)..int\ (\mathrm{b}\ ps\ j) - 1.\ ?f\ d0\ (j - 1))$
proof (*rule sum.cong*)
fix $d0$
assume $d0 \in \{\mathrm{b}\ ps\ (Suc\ j)..int\ (\mathrm{b}\ ps\ j) - 1\}$
hence $\mathrm{b}\ ps\ (Suc\ j) \leq d0$ **and** $d0 < int\ (\mathrm{b}\ ps\ j)$ **by** *simp-all*
hence $\mathrm{b}\ ps\ (Suc\ j) \leq nat\ d0$ **and** $nat\ d0 < \mathrm{b}\ ps\ j$ **by** *simp-all*
have $(\sum (h,\ U){\in}\{(h',\ U').\ (h',\ U') \in set\ (ps_+) \wedge poly\text{-}deg\ h' = d0 \wedge card\ U' = j\}.\ ?f\ (poly\text{-}deg\ h)\ (card\ U - 1)) =$
$(\sum (h,\ U){\in}\{(h',\ U').\ (h',\ U') \in set\ (ps_+) \wedge poly\text{-}deg\ h' = d0 \wedge card\ U' = j\}.\ ?f\ d0\ (j - 1))$
using *refl* **by** (*rule sum.cong*) *auto*
also have $\ldots = card\ \{(h',\ U').\ (h',\ U') \in set\ (ps_+) \wedge poly\text{-}deg\ h' = nat\ d0 \wedge card\ U' = j\} * ?f\ d0\ (j - 1)$
using ‹$\mathrm{b}\ ps\ (Suc\ j) \leq d0$› **by** (*simp add*: *int-eq-iff*)
also have $\ldots = ?f\ d0\ (j - 1)$
using *fin-X assms*(5, 6) ‹$Suc\ 0 \leq j$› ‹$j \leq n$› ‹$\mathrm{b}\ ps\ (Suc\ j) \leq nat\ d0$› ‹$nat\ d0 < \mathrm{b}\ ps\ j$›
by (*simp only*: *n-def lem-6-1-2'*(3))
finally show $(\sum (h,\ U){\in}\{(h',\ U').\ (h',\ U') \in set\ (ps_+) \wedge poly\text{-}deg\ h' = d0 \wedge card\ U' = j\}.$
$?f\ (poly\text{-}deg\ h)\ (card\ U - 1)) = ?f\ d0\ (j - 1)$ **.**
qed

also have ... = ($\sum$ *d0*∈(−) (*int d*) ' {b *ps* (*Suc j*)..*int* (b *ps j*) − *1*}. *d0* + *int* (*j* − *1*) *gchoose* (*j* − *1*))
proof −
have *inj-on* ((−) (*int d*)) {b *ps* (*Suc j*)..*int* (b *ps j*) − *1*} **by** (*auto simp*: *inj-on-def*)
thus *?thesis* **by** (*simp only*: *sum.reindex o-def*)
qed
also have ... = ($\sum$ *d0*∈{*0*..*int d* − (b *ps* (*Suc j*))}−{*0*..*int d* − b *ps j*}. *d0* + *int* (*j* − *1*) *gchoose* (*j* − *1*))
using - *refl*
proof (*rule sum.cong*)
have (−) (*int d*) ' {b *ps* (*Suc j*)..*int* (b *ps j*) − *1*} = {*int d* − (*int* (b *ps j*) − *1*)..*int d* − *int* (b *ps* (*Suc j*))}
by (*simp only*: *image-diff-atLeastAtMost*)
also have ... = {*0*..*int d* − *int* (b *ps* (*Suc j*))} − {*0*..*int d* − *int* (b *ps j*)}
proof −
from ‹b *ps j* ≤ *d*› **have** *int* (b *ps j*) − *1* ≤ *int d* **by** *simp*
thus *?thesis* **by** *auto*
qed
finally show (−) (*int d*) ' {b *ps* (*Suc j*)..*int* (b *ps j*) − *1*} =
{*0*..*int d* − *int* (b *ps* (*Suc j*))} − {*0*..*int d* − *int* (b *ps j*)} **.**
qed
also have ... = ($\sum$ *d0*=*0*..*int d* − (b *ps* (*Suc j*)). *d0* + *int* (*j* − *1*) *gchoose* (*j* − *1*)) −
($\sum$ *d0*=*0*..*int d* − b *ps j*. *d0* + *int* (*j* − *1*) *gchoose* (*j* − *1*))
by (*rule sum-diff*) (*auto simp*: ‹b *ps* (*Suc j*) ≤ b *ps j*›)
also from ‹b *ps* (*Suc j*) ≤ *d*› ‹b *ps j* ≤ *d*› **have** ... = *?f* (b *ps* (*Suc j*)) *j* − *?f* (b *ps j*) *j*
by (*simp add*: *gchoose-rising-sum*, *simp add*: *int-j ac-simps* ‹*0* < *j*›)
finally show ($\sum$(*h*, *U*)∈{(*h'*, *U'*). (*h'*, *U'*) ∈ *set* (ps_+) ∧ *card U'* = *j*}. *?f* (*poly-deg h*) (*card U* − *1*)) =
?f (b *ps* (*Suc j*)) *j* − *?f* (b *ps j*) *j* **.**
qed
also have ... = ($\sum$*j*=*1*..*n*. *?f* (b *ps* (*Suc j*)) *j*) − ($\sum$*j*=*1*..*n*. *?f* (b *ps j*) *j*)
by (*fact sum-subtractf*)
also have ... = *?f* (b *ps* (*Suc n*)) *n* + ($\sum$*j*=*1*..*n*−*1*. *?f* (b *ps* (*Suc j*)) *j*) − ($\sum$*j*=*1*..*n*. *?f* (b *ps j*) *j*)
by (*simp only*: *sum-tail-nat*[*OF* ‹*0* < *n*› ‹*1* ≤ *n*›])
also have ... = *?f* (b *ps* (*Suc n*)) *n* − *?f* (b *ps 1*) *1* +
(($\sum$*j*=*1*..*n*−*1*. *?f* (b *ps* (*Suc j*)) *j*) − ($\sum$*j*=*1*..*n*−*1*. *?f* (b *ps* (*Suc j*)) (*Suc j*)))
by (*simp only*: *sum.atLeast-Suc-atMost*[*OF* ‹*1* ≤ *n*›] *sum-atLeast-Suc-shift*[*OF* ‹*0* < *n*› ‹*1* ≤ *n*›])
also have ... = *?f* (b *ps* (*Suc n*)) *n* − *?f* (b *ps 1*) *1* −
($\sum$*j*=*1*..*n*−*1*. *?f* (b *ps* (*Suc j*)) (*Suc j*) − *?f* (b *ps* (*Suc j*)) *j*)
by (*simp only*: *sum-subtractf*)
also have ... = *?f* (b *ps* (*Suc n*)) *n* − *1* − ((*int d* − b *ps* (*Suc 0*)) *gchoose* (*Suc 0*)) −
($\sum$*j*=*1*..*n*−*1*. (*int d* − b *ps* (*Suc j*) + *j*) *gchoose* (*Suc j*))

proof −
have *?f* (b *ps 1*) *1* = *1* + ((*int d* − b *ps* (*Suc 0*)) *gchoose* (*Suc 0*))
by (*simp add*: *plus-Suc-gbinomial*)
moreover from *refl* **have** ($\sum$*j*=*1*..*n*−*1*. *?f* (b *ps* (*Suc j*)) (*Suc j*) − *?f* (b *ps* (*Suc j*)) *j*) =
($\sum$*j*=*1*..*n*−*1*. (*int d* − b *ps* (*Suc j*) + *j*) *gchoose* (*Suc j*))
by (*rule sum.cong*) (*simp add*: *plus-Suc-gbinomial*)
ultimately show *?thesis* **by** (*simp only*:)
qed
also have . . . = *?f* (b *ps* (*Suc n*)) *n* − *1* − ($\sum$*j*=*0*..*n*−*1*. (*int d* − b *ps* (*Suc j*) + *j*) *gchoose* (*Suc j*))
by (*simp only*: *sum.atLeast-Suc-atMost*[*OF le0*], *simp*)
also have . . . = *?f* (b *ps* (*Suc n*)) *n* − *1* − ($\sum$*j*=*Suc 0*..*Suc* (*n*−*1*). (*int d* − b *ps j* + *j* − *1*) *gchoose j*)
by (*simp only*: *sum.shift-bounds-cl-Suc-ivl*, *simp add*: *ac-simps*)
also have . . . = *Hilbert-poly* (b *ps*) *d* **using** ‹*0* < *n*› **by** (*simp add*: *Hilbert-poly-def Let-def n-def*)
finally have *eq2*: *int* ($\sum$*hU*∈*set* (*ps*$_+$). *Hilbert-fun* (*cone hU*) *d*) = *Hilbert-poly* (b *ps*) (*int d*) **.**

from *assms*(*4*, *2*, *3*) **have** *Hilbert-fun T d* = ($\sum$*hU*∈*set ps*. *Hilbert-fun* (*cone hU*) *d*)
by (*rule Hilbert-fun-cone-decomp*)
also have . . . = ($\sum$*hU*∈(*set ps* − *set* (*ps*$_+$)) ∪ *set* (*ps*$_+$). *Hilbert-fun* (*cone hU*) *d*) **by** (*simp only*: *eq0*)
also have . . . = ($\sum$*hU*∈*set ps* − *set* (*ps*$_+$). *Hilbert-fun* (*cone hU*) *d*) + ($\sum$*hU*∈*set* (*ps*$_+$). *Hilbert-fun* (*cone hU*) *d*)
using *fin2 finite-set* **by** (*rule sum.union-disjoint*) *blast*
also have . . . = *card* {*h*. (*h*, {}) ∈ *set ps* ∧ *poly-deg h* = *d*} + ($\sum$*hU*∈*set* (*ps*$_+$). *Hilbert-fun* (*cone hU*) *d*)
by (*simp only*: *eq1*)
also have *int* . . . = *card* {*h*. (*h*, {}) ∈ *set ps* ∧ *poly-deg h* = *d*} + *Hilbert-poly* (b *ps*) *d*
by (*simp only*: *eq2 int-plus*)
finally show *?thesis* **.**
qed

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 −
from *fin-X* **have** b *ps* (*Suc 0*) ≤ b *ps 0* **using** *le0* **by** (*rule* b-*decreasing*)
also have . . . ≤ *d* **by** *fact*
finally have b *ps* (*Suc 0*) ≤ *d* **.**
with *assms*(*1*−*6*) **have** *int* (*Hilbert-fun T d*) =
int (*card* {*h*. (*h*, {}) ∈ *set ps* ∧ *poly-deg h* = *d*}) + *Hilbert-poly* (b *ps*) (*int d*)

by (*rule Hilbert-fun-eq-Hilbert-poly-plus-card*)
also have ... = *Hilbert-poly* (b *ps*) (*int d*)
proof −
have *eq*: $\{h.\ (h, \{\}) \in set\ ps \wedge poly\text{-}deg\ h = d\} = \{\}$
proof −
{
fix *h*
assume $(h, \{\}) \in set\ ps$ **and** $poly\text{-}deg\ h = d$
from *fin-X this*(*1*) *le0* **have** $poly\text{-}deg\ h < b\ ps\ 0$ **by** (*rule* b)
with *assms*(*7*) **have** *False* **by** (*simp add*: ‹$poly\text{-}deg\ h = d$›)
}
thus *?thesis* **by** *blast*
qed
show *?thesis* **by** (*simp add*: *eq*)
qed
finally show *?thesis* **.**
qed

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*: $F \subseteq P[X]$ **and** *f-in*: $f \in F$
and *hom-F*: $\bigwedge f'.\ f' \in F \Longrightarrow homogeneous\ f'$ **and** *f-max*: $\bigwedge f'.\ f' \in F \Longrightarrow poly\text{-}deg\ f' \le poly\text{-}deg\ f$
and *d-gr-0*: $0 < poly\text{-}deg\ f$ **and** *ideal-f-neq*: $ideal\ \{f\} \neq ideal\ F$
begin

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

lemma *f-in-Polys*: $f \in P[X]$
using *f-in F-sub* **..**

lemma *hom-f*: *homogeneous f*
using *f-in* **by** (*rule hom-F*)

lemma *f-not-0*: $f \neq 0$
using *d-gr-0* **by** *auto*

lemma *X-not-empty*: $X \neq \{\}$
using *n-gr-1* **by** *auto*

lemma *n-gr-0*: $0 < n$
using ‹$1 < n$› **by** *simp*

corollary *int-n-minus-1* [*simp*]: $int\ (n - Suc\ 0) = int\ n - 1$

using *n-gr-0* **by** *simp*

lemma *int-n-minus-2* [*simp*]: $int\ (n - Suc\ (Suc\ 0)) = int\ n - 2$
using *n-gr-1* **by** *simp*

lemma *cone-f-X-sub*: $cone\ (f,\ X) \subseteq P[X]$
proof −
have $cone\ (f,\ X) = cone\ (f * 1,\ X)$ **by** *simp*
also from *f-in-Polys* **have** $\ldots \subseteq cone\ (1,\ X)$ **by** (*rule cone-mono-1*)
finally show *?thesis* **by** *simp*
qed

lemma *ideal-Int-Polys-eq-cone*: $ideal\ \{f\} \cap P[X] = cone\ (f,\ X)$
proof (*intro subset-antisym subsetI*)
fix p
assume $p \in ideal\ \{f\} \cap P[X]$
hence $p \in ideal\ \{f\}$ **and** $p \in P[X]$ **by** *simp-all*
have *finite* $\{f\}$ **by** *simp*
then obtain q **where** $p = (\sum f' \in \{f\}.\ q\ f' * f')$ **using** ‹$p \in ideal\ \{f\}$›
by (*rule ideal.span-finiteE*)
hence p: $p = q\ f * f$ **by** *simp*
with ‹$p \in P[X]$› **have** $f * q\ f \in P[X]$ **by** (*simp only*: *mult.commute*)
hence $q\ f \in P[X]$ **using** *f-in-Polys f-not-0* **by** (*rule times-in-PolysD*)
with p **show** $p \in cone\ (f,\ X)$ **by** (*rule coneI*)
next
fix p
assume $p \in cone\ (f,\ X)$
then obtain q **where** $q \in P[X]$ **and** p: $p = q * f$ **by** (*rule coneE*)
have $f \in ideal\ \{f\}$ **by** (*rule ideal.span-base*) *simp*
with ‹$q \in P[X]$› *f-in-Polys* **show** $p \in ideal\ \{f\} \cap P[X]$
unfolding p **by** (*intro IntI ideal.span-scale Polys-closed-times*)
qed

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

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

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

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* $F \cap P[X]$) [*ideal* $\{f\} \cap P[X]$, P] (**is** *?thesis6*)
proof −
note *fin-X*
moreover from *fin-F* **have** *finite* $(F - \{f\})$ **by** *simp*
moreover from *F-sub* **have** $F - \{f\} \subseteq P[X]$ **by** *blast*
ultimately obtain P' ps' **where** *1*: *valid-decomp* X ps' **and** *2*: *standard-decomp* d ps'
and *3*: *cone-decomp* P' ps' **and** *40*: $(\bigwedge f'. f' \in F - \{f\} \Longrightarrow$ *homogeneous* $f')$ $\Longrightarrow$ *hom-decomp* ps'
and *50*: *direct-decomp* (*ideal* (*insert* f $(F - \{f\})$) $\cap P[X]$) [*ideal* $\{f\} \cap P[X]$, P']
using *f-in-Polys f-max* **by** (*rule ideal-decompE*) *blast+*
have *4*: *hom-decomp* ps' **by** (*intro 40 hom-F*) *simp*
from *50 f-in* **have** *5*: *direct-decomp* (*ideal* $F \cap P[X]$) [*ideal* $\{f\} \cap P[X]$, P']
by (*simp add*: *insert-absorb*)
let *?ps* = *exact* X (*poly-deg* f) ps'
from *fin-X 1 2* **have** *valid-decomp* X *?ps* **and** *standard-decomp* d *?ps* **and** *exact-decomp* X *0* *?ps*
by (*rule exact*)+
moreover from *fin-X 1 2 3* **have** *cone-decomp* P' *?ps* **by** (*rule cone-decomp-exact*)
moreover from *fin-X 1 2 4* **have** *hom-decomp* *?ps* **by** (*rule hom-decomp-exact*)
ultimately have *valid-decomp* X (*snd* (P', *?ps*)) $\wedge$ *standard-decomp* d (*snd* (P', *?ps*)) $\wedge$
exact-decomp X *0* (*snd* (P', *?ps*)) $\wedge$ *cone-decomp* (*fst* (P', *?ps*)) (*snd* (P', *?ps*)) $\wedge$
hom-decomp (*snd* (P', *?ps*)) $\wedge$
direct-decomp (*ideal* $F \cap P[X]$) [*ideal* $\{f\} \cap P[X]$, *fst* (P', *?ps*)]
using *5* **by** *simp*
hence *?thesis1* $\wedge$ *?thesis2* $\wedge$ *?thesis3* $\wedge$ *?thesis4* $\wedge$ *?thesis5* $\wedge$ *?thesis6*
unfolding *P-def ps-def P-ps-def* **by** (*rule someI*)
thus *?thesis1* **and** *?thesis2* **and** *?thesis3* **and** *?thesis4* **and** *?thesis5* **and** *?thesis6* **by** *simp-all*
qed

lemma *P-sub*: $P \subseteq P[X]$
using *valid-ps cn-ps* **by** (*rule valid-cone-decomp-subset-Polys*)

lemma *ps-not-Nil*: $ps_+ \neq []$
proof
assume $ps_+ = []$
have *Keys* $P \subseteq (\bigcup hU \in set\ ps.\ keys\ (fst\ hU))$ (**is** - $\subseteq$ *?A*)
proof
fix t
assume $t \in$ *Keys* P
then obtain p **where** $p \in P$ **and** $t \in$ *keys* p **by** (*rule in-KeysE*)
from *cn-ps* **have** *direct-decomp* P (*map cone ps*) **by** (*rule cone-decompD*)
then obtain *qs* **where** *qs*: *qs* $\in$ *listset* (*map cone ps*) **and** *p*: p = *sum-list qs* **using** ‹$p \in P$›

by (*rule direct-decompE*)
from ‹$t \in$ *keys p*› *keys-sum-list-subset* **have** $t \in$ *Keys* (*set qs*) **unfolding** *p* **..**
then obtain *q* **where** $q \in$ *set qs* **and** $t \in$ *keys q* **by** (*rule in-KeysE*)
from *this*(*1*) **obtain** *i* **where** $i <$ *length qs* **and** $q = qs \; ! \; i$ **by** (*metis in-set-conv-nth*)
with *qs* **have** $i <$ *length ps* **and** $q \in$ (*map cone ps*) ! *i* **by** (*simp-all add*: *listsetD del*: *nth-map*)
hence $q \in$ *cone* (*ps* ! *i*) **by** *simp*
obtain *h U* **where** *eq*: *ps* ! *i* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
from ‹$i <$ *length ps*› *this*[*symmetric*] **have** (*h*, *U*) $\in$ *set ps* **by** *simp*
have $U = \{\}$
proof (*rule ccontr*)
assume $U \neq \{\}$
with ‹(*h*, *U*) $\in$ *set ps*› **have** (*h*, *U*) $\in$ *set* (ps_+) **by** (*simp add*: *pos-decomp-def*)
with ‹$ps_+ = []$› **show** *False* **by** *simp*
qed
with ‹$q \in$ *cone* (*ps* ! *i*)› **have** $q \in$ *range* ($\lambda c.\ c \cdot h$) **by** (*simp only*: *eq cone-empty*)
then obtain *c* **where** $q = c \cdot h$ **..**
also have *keys* ... $\subseteq$ *keys h* **by** (*fact keys-map-scale-subset*)
finally have $t \in$ *keys h* **using** ‹$t \in$ *keys q*› **..**
hence $t \in$ *keys* (*fst* (*h*, *U*)) **by** *simp*
with ‹(*h*, *U*) $\in$ *set ps*› **show** $t \in$ *?A* **..**
qed
moreover from *finite-set finite-keys* **have** *finite ?A* **by** (*rule finite-UN-I*)
ultimately have *finite* (*Keys P*) **by** (*rule finite-subset*)

have $\exists$ *q*$\in$*ideal F*. $q \in P[X] \wedge q \neq 0 \wedge \neg$ *lpp f adds lpp q*
proof (*rule ccontr*)
assume $\neg$ ($\exists$ *q*$\in$*ideal F*. $q \in P[X] \wedge q \neq 0 \wedge \neg$ *lpp f adds lpp q*)
hence *adds*: *lpp f adds lpp q* **if** $q \in$ *ideal F* **and** $q \in P[X]$ **and** $q \neq 0$ **for** *q*
using *that* **by** *blast*
from *fin-X* - *F-sub* **have** *ideal* $\{f\}$ = *ideal F*
proof (*rule punit.pmdl-eqI-adds-lt-dgrad-p-set*[*simplified*, *OF dickson-grading-varnum*, **where** *m=0*, *simplified dgrad-p-set-varnum*])
from *f-in-Polys* **show** $\{f\} \subseteq P[X]$ **by** *simp*
next
from *f-in* **have** $\{f\} \subseteq F$ **by** *simp*
thus *ideal* $\{f\} \subseteq$ *ideal F* **by** (*rule ideal.span-mono*)
next
fix *q*
assume $q \in$ *ideal F* **and** $q \in P[X]$ **and** $q \neq 0$
hence *lpp f adds lpp q* **by** (*rule adds*)
with *f-not-0* **show** $\exists$ *g*$\in\{f\}$. $g \neq 0 \wedge$ *lpp g adds lpp q* **by** *blast*
qed
with *ideal-f-neq* **show** *False* **..**
qed
then obtain *q0* **where** *q0* $\in$ *ideal F* **and** *q0* $\in P[X]$ **and** *q0* $\neq 0$
and *nadds-q0*: $\neg$ *lpp f adds lpp q0* **by** *blast*
define *q* **where** *q* = *hom-component q0* (*deg-pm* (*lpp q0*))

from *hom-F* ‹*q0* ∈ *ideal F*› **have** *q* ∈ *ideal F* **unfolding** *q-def* **by** (*rule homogeneous-ideal*)
from *homogeneous-set-Polys* ‹*q0* ∈ *P*[*X*]› **have** *q* ∈ *P*[*X*] **unfolding** *q-def* **by** (*rule homogeneous-setD*)
from ‹*q0* ≠ *0*› **have** *q* ≠ *0* **and** *lpp q* = *lpp q0* **unfolding** *q-def* **by** (*rule hom-component-lpp*)+
from *nadds-q0 this*(*2*) **have** *nadds-q*: ¬ *lpp f adds lpp q* **by** *simp*
have *hom-q*: *homogeneous q* **by** (*simp only*: *q-def homogeneous-hom-component*)
from *nadds-q* **obtain** *x* **where** *x*: ¬ *lookup* (*lpp f*) *x* ≤ *lookup* (*lpp q*) *x*
by (*auto simp add*: *adds-poly-mapping le-fun-def*)
obtain *y* **where** *y* ∈ *X* **and** *y* ≠ *x*
proof −
from *n-gr-1* **have** *2* ≤ *n* **by** *simp*
then obtain *Y* **where** *Y* ⊆ *X* **and** *card Y* = *2* **by** (*rule card-geq-ex-subset*)
from *this*(*2*) **obtain** *u v* **where** *u* ≠ *v* **and** *Y* = {*u*, *v*} **by** (*rule card-2-E*)
from *this* **obtain** *y* **where** *y* ∈ *Y* **and** *y* ≠ *x* **by** *blast*
from *this*(*1*) ‹*Y* ⊆ *X*› **have** *y* ∈ *X* **..**
thus *?thesis* **using** ‹*y* ≠ *x*› **..**
qed
define *q'* **where** *q'* = (λ*k*. *punit.monom-mult 1* (*Poly-Mapping.single y k*) *q*)
have *inj1*: *inj q'* **by** (*auto intro*!: *injI simp*: *q'-def* ‹*q* ≠ *0*› *dest*: *punit.monom-mult-inj-2 monomial-inj*)
have *q'-in*: *q' k* ∈ *ideal F* ∩ *P*[*X*] **for** *k* **unfolding** *q'-def* **using** ‹*q* ∈ *ideal F*› ‹*q* ∈ *P*[*X*]› ‹*y* ∈ *X*›
by (*intro IntI punit.pmdl-closed-monom-mult*[*simplified*] *Polys-closed-monom-mult PPs-closed-single*)
have *lpp-q'*: *lpp* (*q' k*) = *Poly-Mapping.single y k* + *lpp q* **for** *k*
using ‹*q* ≠ *0*› **by** (*simp add*: *q'-def punit.lt-monom-mult*)
have *inj2*: *inj-on* (*deg-pm* ∘ *lpp*) (*range q'*)
by (*auto intro*!: *inj-onI simp*: *lpp-q' deg-pm-plus deg-pm-single dest*: *monomial-inj*)
have (*deg-pm* ∘ *lpp*) ‘ *range q'* ⊆ *deg-pm* ‘ *Keys P*
proof
fix *d*
assume *d* ∈ (*deg-pm* ∘ *lpp*) ‘ *range q'*
then obtain *k* **where** *d*: *d* = *deg-pm* (*lpp* (*q' k*)) (**is** - = *deg-pm ?t*) **by** *auto*
from *hom-q* **have** *hom-q'*: *homogeneous* (*q' k*) **by** (*simp add*: *q'-def homogeneous-monom-mult*)
from ‹*q* ≠ *0*› **have** *q' k* ≠ *0* **by** (*simp add*: *q'-def punit.monom-mult-eq-zero-iff*)
hence *?t* ∈ *keys* (*q' k*) **by** (*rule punit.lt-in-keys*)
with *hom-q'* **have** *deg-q'*: *d* = *poly-deg* (*q' k*) **unfolding** *d* **by** (*rule homogeneousD-poly-deg*)
from *decomp-F q'-in* **obtain** *qs* **where** *qs* ∈ *listset* [*ideal* {*f*} ∩ *P*[*X*], *P*] **and** *q' k* = *sum-list qs*
by (*rule direct-decompE*)
moreover from *this*(*1*) **obtain** *f0 p0* **where** *f0*: *f0* ∈ *ideal* {*f*} ∩ *P*[*X*] **and** *p0*: *p0* ∈ *P*
and *qs* = [*f0*, *p0*] **by** (*rule listset-doubletonE*)
ultimately have *q'*: *q' k* = *f0* + *p0* **by** *simp*

define *f1* **where** *f1* = *hom-component f0 d*
define *p1* **where** *p1* = *hom-component p0 d*
from *hom-q* **have** *homogeneous* (*q′ k*) **by** (*simp add*: *q′-def homogeneous-monom-mult*)
hence *q′ k* = *hom-component* (*q′ k*) *d* **by** (*simp add*: *hom-component-of-homogeneous deg-q′*)
also have ... = *f1* + *p1* **by** (*simp only*: *q′ hom-component-plus f1-def p1-def*)
finally have *q′ k* = *f1* + *p1* **.**
have *keys p1* ≠ {}
proof
assume *keys p1* = {}
with ‹*q′ k* = *f1* + *p1*› ‹*q′ k* ≠ *0*› **have** *t*: *?t* = *lpp f1* **and** *f1* ≠ *0* **by** *simp-all*
from *f0* **have** *f0* ∈ *ideal* {*f*} **by** *simp*
with - **have** *f1* ∈ *ideal* {*f*} **unfolding** *f1-def* **by** (*rule homogeneous-ideal*) (*simp add*: *hom-f*)
with *punit.is-Groebner-basis-singleton* **obtain** *g* **where** *g* ∈ {*f*} **and** *lpp g adds lpp f1*
using ‹*f1* ≠ *0*› **by** (*rule punit.GB-adds-lt*[*simplified*])
hence *lpp f adds ?t* **by** (*simp add*: *t*)
hence *lookup* (*lpp f*) *x* ≤ *lookup ?t x* **by** (*simp add*: *adds-poly-mapping le-fun-def*)
also have ... = *lookup* (*lpp q*) *x* **by** (*simp add*: *lpp-q′ lookup-add lookup-single* ‹*y* ≠ *x*›)
finally have *lookup* (*lpp f*) *x* ≤ *lookup* (*lpp q*) *x* **.**
with *x* **show** *False* **..**
qed
then obtain *t* **where** *t* ∈ *keys p1* **by** *blast*
hence *d* = *deg-pm t* **by** (*simp add*: *p1-def keys-hom-component*)
from *cn-ps hom-ps* **have** *homogeneous-set P* **by** (*intro homogeneous-set-cone-decomp*)
hence *p1* ∈ *P* **using** ‹*p0* ∈ *P*› **unfolding** *p1-def* **by** (*rule homogeneous-setD*)
with ‹*t* ∈ *keys p1*› **have** *t* ∈ *Keys P* **by** (*rule in-KeysI*)
with ‹*d* = *deg-pm t*› **show** *d* ∈ *deg-pm* ‘ *Keys P* **by** (*rule image-eqI*)
qed
moreover from *inj1 inj2* **have** *infinite* ((*deg-pm* ∘ *lpp*) ‘ *range q′*)
by (*simp add*: *finite-image-iff o-def*)
ultimately have *infinite* (*deg-pm* ‘ *Keys P*) **by** (*rule infinite-super*)
hence *infinite* (*Keys P*) **by** *blast*
thus *False* **using** ‹*finite* (*Keys P*)› **..**
qed

private 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* ≡ (*λi. 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 –
from *fin-X F-sub* **obtain** *qs′* **where** *1*: *valid-decomp X qs′* **and** *2*: *standard-decomp 0 qs′*
and *3*: *monomial-decomp qs′* **and** *4*: *cone-decomp* (*normal-form F* ‘ *P*[*X*]) *qs′*
and *5*: *exact-decomp X 0 qs′*
and *60*: ⋀*g*. (⋀*f*. *f* ∈ *F* ⟹ *homogeneous f*) ⟹ *g* ∈ *punit.reduced-GB F* ⟹ *poly-deg g* ≤ b *qs′ 0*
by (*rule normal-form-exact-decompE*) *blast*
from *hom-F* **have** ⋀*g*. *g* ∈ *punit.reduced-GB F* ⟹ *poly-deg g* ≤ b *qs′ 0* **by** (*rule 60*)
with *1 2 3 4 5* **have** *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*) **by** (*simp add*: *N-def*)
hence *?thesis1* ∧ *?thesis2* ∧ *?thesis3* ∧ *?thesis4* ∧ *?thesis5* ∧ (∀ *g*∈*punit.reduced-GB F*. *poly-deg g* ≤ *bb 0*)
unfolding *qs-def bb-def* **by** (*rule someI*)
thus *?thesis1* **and** *?thesis2* **and** *?thesis3* **and** *?thesis4* **and** *?thesis5*
and *g* ∈ *punit.reduced-GB F* ⟹ *poly-deg g* ≤ *bb 0* **by** *simp-all*
from ‹*?thesis3*› **show** *?thesis6* **by** (*rule monomial-decomp-imp-hom-decomp*)
qed

lemma *N-sub*: *N* ⊆ *P*[*X*]
using *valid-qs cn-qs* **by** (*rule valid-cone-decomp-subset-Polys*)

lemma *decomp-Polys*: *direct-decomp P*[*X*] [*ideal* {*f*} ∩ *P*[*X*], *P*, *N*]
proof –
from *fin-X F-sub* **have** *direct-decomp P*[*X*] [*ideal F* ∩ *P*[*X*], *N*] **unfolding** *N-def*
by (*rule direct-decomp-ideal-normal-form*)
hence *direct-decomp P*[*X*] ([*N*] @ [*ideal* {*f*} ∩ *P*[*X*], *P*]) **using** *decomp-F*
by (*rule direct-decomp-direct-decomp*)
hence *direct-decomp P*[*X*] ([*ideal* {*f*} ∩ *P*[*X*], *P*] @ [*N*])
by (*rule direct-decomp-perm*) *simp*
thus *?thesis* **by** *simp*
qed

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

from *fin-X ext-ps le-refl* **have** *aa* (*Suc n*) = a *ps* **unfolding** *aa-def* **by** (*rule* b-*card-X*)
also from *fin-X valid-ps std-ps ps-not-Nil* **have** ... = *d* **by** (*rule* a-*nonempty-unique*)
finally show *?thesis* **.**
qed

lemma *bb-Suc-n* [*simp*]: *bb* (*Suc n*) = *0*
proof −
from *fin-X ext-qs le-refl* **have** *bb* (*Suc n*) = a *qs* **unfolding** *bb-def* **by** (*rule* b-*card-X*)
also from *std-qs* **have** ... = *0* **unfolding** a-*def*[*OF fin-X*] **by** (*rule Least-eq-0*)
finally show *?thesis* **.**
qed

lemma *Hilbert-fun-X*:
assumes $d \le z$
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 −
define *ss* **where** *ss* = [*ideal* {*f*} ∩ *P*[*X*], *P*, *N*]
have *homogeneous-set A* ∧ *phull.subspace A* **if** *A* ∈ *set ss* **for** *A*
proof −
from *that* **have** *A* = *ideal* {*f*} ∩ *P*[*X*] ∨ *A* = *P* ∨ *A* = *N* **by** (*simp add*: *ss-def*)
thus *?thesis*
proof (*elim disjE*)
assume *A*: *A* = *ideal* {*f*} ∩ *P*[*X*]
show *?thesis* **unfolding** *A*
by (*intro conjI homogeneous-set-IntI phull.subspace-inter homogeneous-set-homogeneous-ideal homogeneous-set-Polys subspace-ideal subspace-Polys*) (*simp add*: *hom-f*)
next
assume *A*: *A* = *P*
from *cn-ps hom-ps* **show** *?thesis* **unfolding** *A*
by (*intro conjI homogeneous-set-cone-decomp subspace-cone-decomp*)
next
assume *A*: *A* = *N*
from *cn-qs hom-qs* **show** *?thesis* **unfolding** *A*
by (*intro conjI homogeneous-set-cone-decomp subspace-cone-decomp*)
qed
qed
hence *1*: ⋀*A*. *A* ∈ *set ss* ⟹ *homogeneous-set A* **and** *2*: ⋀*A*. *A* ∈ *set ss* ⟹ *phull.subspace A*
by *simp-all*
have *Hilbert-fun* (*P*[*X*]::(- ⇒$_0$ *'a*) *set*) *z* = (∑ *p*∈*set ss*. *Hilbert-fun p z*)
using *fin-X subset-refl decomp-Polys* **unfolding** *ss-def*
proof (*rule Hilbert-fun-direct-decomp*)
fix *A*
assume *A* ∈ *set* [*ideal* {*f*} ∩ *P*[*X*], *P*, *N*]
hence *A* ∈ *set ss* **by** (*simp only*: *ss-def*)

thus *homogeneous-set A* **and** *phull.subspace A* **by** (*rule 1*, *rule 2*)
qed
also have . . . = ($\sum$ *p*∈*set ss. count-list ss p* ∗ *Hilbert-fun p z*)
using *refl*
proof (*rule sum.cong*)
fix *p*
assume *p* ∈ *set ss*
hence *count-list ss p* ≠ *0* **by** (*simp only*: *count-list-0-iff not-not*)
hence *count-list ss p* = *1* ∨ *1* < *count-list ss p* **by** *auto*
thus *Hilbert-fun p z* = *count-list ss p* ∗ *Hilbert-fun p z*
proof
assume *1* < *count-list ss p*
with *decomp-Polys* **have** *p* = {*0*} **unfolding** *ss-def* [*symmetric*] **using** *phull.subspace-0*
by (*rule direct-decomp-repeated-eq-zero*) (*rule 2*)
thus *?thesis* **by** *simp*
qed *simp*
qed
also have . . . = *sum-list* (*map* (*λp. Hilbert-fun p z*) *ss*)
by (*rule sym*) (*rule sum-list-map-eq-sum-count*)
also have . . . = *Hilbert-fun* (*cone* (*f*, *X*)) *z* + *Hilbert-fun P z* + *Hilbert-fun N z*
by (*simp add*: *ss-def ideal-Int-Polys-eq-cone*)
also have *Hilbert-fun* (*cone* (*f*, *X*)) *z* = (*z* − *d* + (*n* − *1*)) *choose* (*n* − *1*)
using *f-not-0 f-in-Polys fin-X hom-f X-not-empty* **by** (*simp add*: *Hilbert-fun-cone-nonempty assms*)
finally show *?thesis* **.**
qed

lemma *dube-eq-0*:
(*λz*::*int*. (*z* + *int n* − *1*) *gchoose* (*n* − *1*)) =
(*λz*::*int*. ((*z* − *d* + *n* − *1*) *gchoose* (*n* − *1*)) + *Hilbert-poly aa z* + *Hilbert-poly bb z*)
(**is** *?f* = *?g*)
proof (*rule poly-fun-eqI-ge*)
fix *z*::*int*
let *?z* = *nat z*
assume *max* (*aa 0*) (*bb 0*) ≤ *z*
hence *aa 0* ≤ *nat z* **and** *bb 0* ≤ *nat z* **and** *0* ≤ *z* **by** *simp-all*
from *this*(*3*) **have** *int-z*: *int ?z* = *z* **by** *simp*
have *d* ≤ *aa 0* **unfolding** *aa-Suc-n*[*symmetric*] **using** *fin-X le0* **unfolding** *aa-def* **by** (*rule* b-*decreasing*)
hence *d* ≤ *?z* **using** ‹*aa 0* ≤ *nat z*› **by** (*rule le-trans*)
hence *int-zd*: *int* (*?z* − *d*) = *z* − *int d* **using** *int-z* **by** *linarith*
from ‹*d* ≤ *?z*› **have** *Hilbert-fun* (*P*[*X*]::(- $\Rightarrow_0$ *'a*) *set*) *?z* =
((*?z* − *d*) + (*n* − *1*) *choose n* − *1*) + *Hilbert-fun P ?z* + *Hilbert-fun N ?z*
by (*rule Hilbert-fun-X*)
also have *int* . . . = (*z* − *d* + (*n* − *1*) *gchoose n* − *1*) + *Hilbert-poly aa z* + *Hilbert-poly bb z*

using *X-not-empty valid-ps hom-ps cn-ps std-ps ext-ps* ‹*aa 0* ≤ *nat z*›
valid-qs hom-qs cn-qs std-qs ext-qs ‹*bb 0* ≤ *nat z*› ‹*0* ≤ *z*›
by (*simp add*: *Hilbert-fun-eq-Hilbert-poly int-z aa-def bb-def int-binomial int-zd*)
finally show *?f z* = *?g z* **using** *fin-X X-not-empty* ‹*0* ≤ *z*›
by (*simp add*: *Hilbert-fun-Polys int-binomial*) *smt*
qed (*simp-all add*: *poly-fun-Hilbert-poly*)

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*)))
by (*simp only*: *dube-eq-0*) (*auto simp*: *Hilbert-poly-def Let-def sum.distrib*)

lemma *dube-eq-2*:
assumes *j* < *n*
shows (λ*z*::*int*. (*z* + *int n* − *int j* − *1*) *gchoose* (*n* − *j* − *1*)) =
(λ*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* −
(∑ *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 −
let *?h* = λ*z i*. ((*z* + (*int i* − *aa i* − *1*)) *gchoose i*) + ((*z* + (*int i* − *bb i* − *1*)) *gchoose i*)
let *?hj* = λ*z i*. ((*z* + (*int i* − *aa i* − *1*) − *j*) *gchoose* (*i* − *j*)) + ((*z* + (*int i* − *bb i* − *1*) − *j*) *gchoose* (*i* − *j*))
from *assms* **have** *1*: *j* ≤ *n* − *Suc 0* **and** *2*: *j* ≤ *n* **by** *simp-all*

have *eq1*: (*bw-diff* ^^ *j*) (λ*z*. ∑ *i*=*1*..*j*. *?h z i*) = (λ-. *if j* = *0 then 0 else 2*)
proof (*cases j*)
case *0*
thus *?thesis* **by** *simp*
next
case (*Suc j0*)
hence *j* ≠ *0* **by** *simp*
have (λ*z*::*int*. ∑ *i* = *1*..*j*. *?h z i*) = (λ*z*::*int*. (∑ *i* = *1*..*j0*. *?h z i*) + *?h z j*)
by (*simp add*: ‹*j* = *Suc j0*›)
moreover have (*bw-diff* ^^ *j*) ... = (λ*z*::*int*. (∑ *i* = *1*..*j0*. (*bw-diff* ^^ *j*) (λ*z*. *?h z i*) *z*) + *2*)
by (*simp add*: *bw-diff-gbinomial-pow*)
moreover have (∑ *i* = *1*..*j0*. (*bw-diff* ^^ *j*) (λ*z*. *?h z i*) *z*) = (∑ *i* = *1*..*j0*. *0*) **for** *z*::*int*
using *refl*
proof (*rule sum.cong*)
fix *i*
assume *i* ∈ {*1*..*j0*}

hence $\neg\ j \le i$ **by** (*simp add*: ‹$j = Suc\ j0$›)
thus (*bw-diff* ^^ *j*) ($\lambda z.$ *?h z i*) $z = 0$ **by** (*simp add*: *bw-diff-gbinomial-pow*)
qed
ultimately show *?thesis* **by** (*simp add*: ‹$j \ne 0$›)
qed

have *eq2*: (*bw-diff* ^^ *j*) ($\lambda z.$ $\sum$ *i=Suc j..n. ?h z i*) = ($\lambda z.$ ($\sum$ *i=Suc j..n. ?hj z i*))
proof −
have (*bw-diff* ^^ *j*) ($\lambda z.$ $\sum$ *i=Suc j..n. ?h z i*) = ($\lambda z.$ $\sum$ *i=Suc j..n.* (*bw-diff* ^^ *j*) ($\lambda z.$ *?h z i*) *z*)
by *simp*
also have ... = ($\lambda z.$ ($\sum$ *i=Suc j..n. ?hj z i*))
proof (*intro ext sum.cong*)
fix *z i*
assume $i \in \{Suc\ j..n\}$
hence $j \le i$ **by** *simp*
thus (*bw-diff* ^^ *j*) ($\lambda z.$ *?h z i*) *z* = *?hj z i* **by** (*simp add*: *bw-diff-gbinomial-pow*)
qed (*fact refl*)
finally show *?thesis* **.**
qed

from *1* **have** *?f* = (*bw-diff* ^^ *j*) (λz::*int*. (z + (*int n* − *1*)) *gchoose* (n − *1*))
by (*simp add*: *bw-diff-gbinomial-pow*) (*simp only*: *algebra-simps*)
also have ... = (*bw-diff* ^^ *j*) (λz::*int*. (z + *int n* − *1*) *gchoose* (n − *1*))
by (*simp only*: *algebra-simps*)
also have ... = (*bw-diff* ^^ *j*)
(λz::*int*. (($z - d + n - 1$) *gchoose* ($n - 1$)) + (($z - d + n$) *gchoose n*) + (($z + n$) *gchoose n*) − *2* −
($\sum$ *i=1..n.* ((z − *aa i* + $i - 1$) *gchoose i*) + ((z − *bb i* + $i - 1$) *gchoose i*)))
by (*simp only*: *dube-eq-1*)
also have ... = (*bw-diff* ^^ *j*)
(λz::*int*. ((z + (*int n* − d − *1*)) *gchoose* ($n - 1$)) + ((z + (*int n* − d)) *gchoose n*) +
(($z + n$) *gchoose n*) − *2* − ($\sum$ *i=1..n. ?h z i*))
by (*simp only*: *algebra-simps*)
also have ... = (λz::*int*. ((z + (*int n* − d − *1*) − j) *gchoose* (n − *1* − j)) +
((z + (*int n* − d) − j) *gchoose* ($n - j$)) + (($z + n - j$) *gchoose* ($n - j$)) − (*if* $j = 0$ *then 2 else 0*) −
(*bw-diff* ^^ *j*) ($\lambda z.$ $\sum$ *i=1..n. ?h z i*) *z*)
using *1 2* **by** (*simp add*: *bw-diff-const-pow bw-diff-gbinomial-pow del*: *bw-diff-sum-pow*)
also from ‹$j \le n$› **have** ($\lambda z.$ $\sum$ *i=1..n. ?h z i*) = ($\lambda z.$ ($\sum$ *i=1..j. ?h z i*) + ($\sum$ *i=Suc j..n. ?h z i*))
by (*simp add*: *sum-split-nat-ivl*)
also have (*bw-diff* ^^ *j*) ... = ($\lambda z.$ (*bw-diff* ^^ *j*) ($\lambda z.$ $\sum$ *i=1..j. ?h z i*) *z* + (*bw-diff* ^^ *j*) ($\lambda z.$ $\sum$ *i=Suc j..n. ?h z i*) *z*)
by (*simp only*: *bw-diff-plus-pow*)
also have ... = ($\lambda z.$ (*if* $j = 0$ *then 0 else 2*) + ($\sum$ *i=Suc j..n. ?hj z i*))

```
    by (simp only: eq1 eq2)
  finally show ?thesis by (simp add: algebra-simps)
qed

lemma dube-eq-3:
  assumes j < n
  shows (1::int) = (− 1)^(n − Suc j) ∗ ((int d − 1) gchoose (n − Suc j)) +
                  (− 1)^(n − j) ∗ ((int d − 1) gchoose (n − j)) − 1 −
                   (∑ i=Suc j..n. (− 1)^(i − j) ∗ ((int (aa i) gchoose (i − j)) +
(int (bb i) gchoose (i − j))))
proof −
  from assms have 1: int (n − Suc j) = int n − j − 1 and 2: int (n − j) = int
n − j by simp-all
  from assms have int n − int j − 1 = int (n − j − 1) by simp
  hence eq1: int n − int j − 1 gchoose (n − Suc j) = 1
    by (simp del: of-nat-diff)
  from assms have int n − int j = int (n − j) by simp
  hence eq2: int n − int j gchoose (n − j) = 1
    using gbinomial-int-n-n by presburger
  have eq3: int n − d − j − 1 gchoose (n − Suc j) = (− 1)^(n − Suc j) ∗ (int d
− 1 gchoose (n − Suc j))
    by (simp add: gbinomial-int-negated-upper[of int n − d − j − 1] 1)
  have eq4: int n − d − j gchoose (n − j) = (− 1)^(n − j) ∗ (int d − 1 gchoose
(n − j))
    by (simp add: gbinomial-int-negated-upper[of int n − d − j] 2)
  have eq5: (∑ i = Suc j..n. (int i − aa i − j − 1 gchoose i − j) + (int i − bb i
− j − 1 gchoose (i − j))) =
        (∑ i=Suc j..n. (− 1)^(i − j) ∗ ((int (aa i) gchoose (i − j)) + (int (bb i)
gchoose (i − j))))
    using refl
  proof (rule sum.cong)
    fix i
    assume i ∈ {Suc j..n}
    hence j ≤ i by simp
    hence 3: int (i − j) = int i − j by simp
    show (int i − aa i − j − 1 gchoose i − j) + (int i − bb i − j − 1 gchoose (i
− j)) =
          (− 1)^(i − j) ∗ ((int (aa i) gchoose (i − j)) + (int (bb i) gchoose (i −
j)))
      by (simp add: gbinomial-int-negated-upper[of int i − aa i − j − 1]
            gbinomial-int-negated-upper[of int i − bb i − j − 1] 3 distrib-left)
  qed
  from fun-cong[OF dube-eq-2, OF assms, of 0] show ?thesis by (simp add: eq1
eq2 eq3 eq4 eq5)
qed

lemma dube-aux-1:
  assumes (h, {}) ∈ set ps ∪ set qs
  shows poly-deg h < max (aa 1) (bb 1)
```

proof (*rule ccontr*)
  define *z* **where** $z = $ *poly-deg h*
  assume $\neg$ $z <$ *max* (*aa 1*) (*bb 1*)

  let *?S* = λA. {*h*. (*h*, {}) $\in$ *A* $\wedge$ *poly-deg h* = *z*}
  have *fin*: *finite* (*?S A*) **if** *finite A* **for** *A*::((('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*) $\times$ '*x set*) *set*
  proof −
    have (λt. (*t*, {})) ‘ *?S A* $\subseteq$ *A* **by** *blast*
    hence *finite* ((λt. (*t*, {}::'*x set*)) ‘ *?S A*) **using** *that* **by** (*rule finite-subset*)
    moreover have *inj-on* (λt. (*t*, {}::'*x set*)) (*?S A*) **by** (*rule inj-onI*) *simp*
    ultimately show *?thesis* **by** (*rule finite-imageD*)
  qed
  from *finite-set* **have** *1*: *finite* (*?S* (*set ps*)) **by** (*rule fin*)
  from *finite-set* **have** *2*: *finite* (*?S* (*set qs*)) **by** (*rule fin*)

  from ‹$\neg$ $z <$ *max* (*aa 1*) (*bb 1*)› **have** *aa 1* $\leq$ *z* **and** *bb 1* $\leq$ *z* **by** *simp-all*
  have $d \leq$ *aa 1* **unfolding** *aa-Suc-n*[*symmetric*] *aa-def* **using** *fin-X* **by** (*rule* b-*decreasing*) *simp*
  hence $d \leq z$ **using** ‹*aa 1* $\leq$ *z*› **by** (*rule le-trans*)
  hence *eq*: *int* ($z - d$) = *int z* − *int d* **by** *simp*
  from ‹$d \leq z$› **have** *Hilbert-fun* (*P*[*X*]::(- $\Rightarrow_0$ '*a*) *set*) *z* =
      ((*z* − *d*) + (*n* − *1*) *choose n* − *1*) + *Hilbert-fun P z* + *Hilbert-fun N z*
    by (*rule Hilbert-fun-X*)
  also have *int* . . . = ((*int z* − *d* + (*n* − *1*) *gchoose n* − *1*) + *Hilbert-poly aa z* + *Hilbert-poly bb z*) +
      (*int* (*card* (*?S* (*set ps*))) + *int* (*card* (*?S* (*set qs*))))
    using *X-not-empty valid-ps hom-ps cn-ps std-ps ext-ps* ‹*aa 1* $\leq$ *z*›
      valid-qs hom-qs cn-qs std-qs ext-qs ‹*bb 1* $\leq$ *z*›
    by (*simp add*: *Hilbert-fun-eq-Hilbert-poly-plus-card aa-def bb-def int-binomial eq*)
  finally have ((*int z* − *d* + *n* − *1 gchoose n* − *1*) + *Hilbert-poly aa z* + *Hilbert-poly bb z*) +
      (*int* (*card* (*?S* (*set ps*))) + *int* (*card* (*?S* (*set qs*)))) = *int z* + *n* − *1 gchoose* (*n* − *1*)
    using *fin-X X-not-empty* **by** (*simp add*: *Hilbert-fun-Polys int-binomial algebra-simps*)
  also have . . . = (*int z* − *d* + *n* − *1 gchoose n* − *1*) + *Hilbert-poly aa z* + *Hilbert-poly bb z*
    by (*fact dube-eq-0*[*THEN fun-cong*])
  finally have *int* (*card* (*?S* (*set ps*))) + *int* (*card* (*?S* (*set qs*))) = *0* **by** *simp*
  hence *card* (*?S* (*set ps*)) = *0* **and** *card* (*?S* (*set qs*)) = *0* **by** *simp-all*
  with *1 2* **have** *?S* (*set ps* $\cup$ *set qs*) = {} **by** *auto*
  moreover from *assms* **have** *h* $\in$ *?S* (*set ps* $\cup$ *set qs*) **by** (*simp add*: *z-def*)
  ultimately have *h* $\in$ {} **by** (*rule subst*)
  thus *False* **by** *simp*
qed

lemma

shows *aa-n*: $aa\ n = d$ **and** *bb-n*: $bb\ n = 0$ **and** *bb-0*: $bb\ 0 \le max\ (aa\ 1)\ (bb\ 1)$
proof −
let $?j = n - Suc\ 0$
from *n-gr-0* **have** $?j < n$ **and** *eq1*: $Suc\ ?j = n$ **and** *eq2*: $n - ?j = 1$ **by** *simp-all*
from *this*(*1*) **have** $(1::int) = (- 1)\widehat{\ }(n - Suc\ ?j) * ((int\ d - 1)\ gchoose\ (n - Suc\ ?j)) +$
$(- 1)\widehat{\ }(n - ?j) * ((int\ d - 1)\ gchoose\ (n - ?j)) - 1 -$
$(\sum i{=}Suc\ ?j..n.\ (- 1)\widehat{\ }(i - ?j) * ((int\ (aa\ i)\ gchoose\ (i - ?j)) + (int\ (bb\ i)\ gchoose\ (i - ?j))))$
by (*rule dube-eq-3*)
hence *eq*: $aa\ n + bb\ n = d$ **by** (*simp add*: *eq1 eq2*)
hence $aa\ n \le d$ **by** *simp*
moreover have $d \le aa\ n$ **unfolding** *aa-Suc-n*[*symmetric*] *aa-def* **using** *fin-X* **by** (*rule* b-*decreasing*) *simp*
ultimately show $aa\ n = d$ **by** (*rule antisym*)
with *eq* **show** $bb\ n = 0$ **by** *simp*

have $bb\ 0 =$ b $qs\ 0$ **by** (*simp only*: *bb-def*)
also from *fin-X* **have** $\ldots \le max\ (aa\ 1)\ (bb\ 1)$ (**is** - $\le$ *?m*)
proof (*rule* b-*le*)
from *fin-X ext-qs* **have** a $qs = bb\ (Suc\ n)$ **by** (*simp add*: b-*card-X bb-def*)
also have $\ldots \le bb\ 1$ **unfolding** *bb-def* **using** *fin-X* **by** (*rule* b-*decreasing*) *simp*
also have $\ldots \le ?m$ **by** (*rule max.cobounded2*)
finally show a $qs \le ?m$ **.**
next
fix *h U*
assume $(h, U) \in set\ qs$
show $poly\text{-}deg\ h < ?m$
proof (*cases card* $U = 0$)
case *True*
from *fin-X valid-qs* ‹$(h, U) \in set\ qs$› **have** *finite U* **by** (*rule valid-decompD-finite*)
with *True* **have** $U = \{\}$ **by** *simp*
with ‹$(h, U) \in set\ qs$› **have** $(h, \{\}) \in set\ ps \cup set\ qs$ **by** *simp*
thus *?thesis* **by** (*rule dube-aux-1*)
next
case *False*
hence $1 \le card\ U$ **by** *simp*
with *fin-X* ‹$(h, U) \in set\ qs$› **have** $poly\text{-}deg\ h < bb\ 1$ **unfolding** *bb-def* **by** (*rule* b)
also have $\ldots \le ?m$ **by** (*rule max.cobounded2*)
finally show *?thesis* **.**
qed
qed
finally show $bb\ 0 \le ?m$ **.**
qed

lemma *dube-eq-4*:
assumes $j < n$

shows (*1*::*int*) = *2* ∗ (− *1*)^(*n* − *Suc j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) − *1* −
(∑ *i*=*Suc j*..*n*−*1*. (− *1*)^(*i* − *j*) ∗ ((*int* (*aa i*) *gchoose* (*i* − *j*)) + (*int* (*bb i*) *gchoose* (*i* − *j*))))
proof −
from *assms* **have** *Suc j* ≤ *n* **and** *0* < *n* **and** *1*: *Suc* (*n* − *Suc j*) = *n* − *j* **by** *simp-all*
have *2*: (− *1*) ^ (*n* − *Suc j*) = − ((− (*1*::*int*)) ^ (*n* − *j*)) **by** (*simp flip*: *1*)
from *assms* **have** (*1*::*int*) = (− *1*)^(*n* − *Suc j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) +
(− *1*)^(*n* − *j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *j*)) − *1* −
(∑ *i*=*Suc j*..*n*. (− *1*)^(*i* − *j*) ∗ ((*int* (*aa i*) *gchoose* (*i* − *j*)) + (*int* (*bb i*) *gchoose* (*i* − *j*))))
by (*rule dube-eq-3*)
also have . . . = (− *1*)^(*n* − *Suc j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) +
(− *1*)^(*n* − *j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *j*)) − *1* −
(− *1*)^(*n* − *j*) ∗ ((*int* (*aa n*) *gchoose* (*n* − *j*)) + (*int* (*bb n*) *gchoose* (*n* − *j*))) −
(∑ *i*=*Suc j*..*n*−*1*. (− *1*)^(*i* − *j*) ∗ ((*int* (*aa i*) *gchoose* (*i* − *j*)) + (*int* (*bb i*) *gchoose* (*i* − *j*))))
using ‹*0* < *n*› ‹*Suc j* ≤ *n*› **by** (*simp only*: *sum-tail-nat*)
also have . . . = (− *1*)^(*n* − *Suc j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) +
(− *1*)^(*n* − *j*) ∗ (((*int d* − *1*) *gchoose* (*n* − *j*)) − (*int d gchoose* (*n* − *j*))) − *1* −
(∑ *i*=*Suc j*..*n*−*1*. (− *1*)^(*i* − *j*) ∗ ((*int* (*aa i*) *gchoose* (*i* − *j*)) + (*int* (*bb i*) *gchoose* (*i* − *j*))))
using *assms* **by** (*simp add*: *aa-n bb-n gbinomial-0-left right-diff-distrib*)
also have (− *1*)^(*n* − *j*) ∗ (((*int d* − *1*) *gchoose* (*n* − *j*)) − (*int d gchoose* (*n* − *j*))) =
(− *1*)^(*n* − *Suc j*) ∗ (((*int d* − *1* + *1*) *gchoose* (*Suc* (*n* − *Suc j*))) − ((*int d* − *1*) *gchoose* (*Suc* (*n* − *Suc j*))))
by (*simp add*: *1 2 flip*: *mult-minus-right*)
also have . . . = (− *1*)^(*n* − *Suc j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*))
by (*simp only*: *gbinomial-int-Suc-Suc*, *simp*)
finally show *?thesis* **by** *simp*
qed

lemma *cc-Suc*:
assumes *j* < *n* − *1*
shows *int* (*cc* (*Suc j*)) = *2* + *2* ∗ (− *1*)^(*n* − *j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) +
(∑ *i*=*j*+*2*..*n*−*1*. (− *1*)^(*i* − *j*) ∗ ((*int* (*aa i*) *gchoose* (*i* − *j*)) + (*int* (*bb i*) *gchoose* (*i* − *j*))))
proof −
from *assms* **have** *j* < *n* **and** *Suc j* ≤ *n* − *1* **by** *simp-all*
hence *n* − *j* = *Suc* (*n* − *Suc j*) **by** *simp*
hence *eq*: (− *1*) ^ (*n* − *Suc j*) = − ((− (*1*::*int*)) ^ (*n* − *j*)) **by** *simp*
from ‹*j* < *n*› **have** (*1*::*int*) = *2* ∗ (− *1*)^(*n* − *Suc j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) − *1* −

($\sum$ *i=Suc j..n−1*. (− *1*)^(*i* − *j*) ∗ ((*int* (*aa i*) *gchoose* (*i* − *j*)) + (*int* (*bb i*) *gchoose* (*i* − *j*))))
  by (*rule dube-eq-4*)
 also have ... = *cc* (*Suc j*) − *2* ∗ (− *1*)^(*n* − *j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) − *1* −
  ($\sum$ *i=j+2..n−1*. (− *1*)^(*i* − *j*) ∗ ((*int* (*aa i*) *gchoose* (*i* − *j*)) + (*int* (*bb i*) *gchoose* (*i* − *j*))))
  using ‹*Suc j* ≤ *n* − *1*› **by** (*simp add*: *sum.atLeast-Suc-atMost eq*)
 finally show *?thesis* **by** *simp*
qed

lemma *cc-n-minus-1*: *cc* (*n* − *1*) = *2* ∗ *d*
proof −
 let *?j* = *n* − *2*
 from *n-gr-1* **have** *1*: *Suc ?j* = *n* − *1* **and** *?j* < *n* − *1* **and** *2*: *Suc* (*n* − *1*) = *n*
  and *3*: *n* − (*n* − *Suc 0*) = *Suc 0* **and** *4*: *n* − *?j* = *2*
  by *simp-all*
 have *int* (*cc* (*n* − *1*)) = *int* (*cc* (*Suc ?j*)) **by** (*simp only*: *1*)
 also from ‹*?j* < *n* − *1*› **have** ... = *2* + *2* ∗ (− *1*) ^ (*n* − *?j*) ∗ (*int d* − *1* *gchoose* (*n* − *Suc ?j*)) +
  ($\sum$ *i* = *?j+2..n−1*. (− *1*) ^ (*i* − *?j*) ∗ ((*int* (*aa i*) *gchoose* (*i* − *?j*)) + (*int* (*bb i*) *gchoose* (*i* − *?j*))))
  by (*rule cc-Suc*)
 also have ... = *int* (*2* ∗ *d*) **by** (*simp add*: *1 2 3 4*)
 finally show *?thesis* **by** (*simp only*: *int-int-eq*)
qed

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*) ≤ d^2 + *2* ∗ *d*
proof −
 let *?j* = *n* − *3*
 from *n-gr-2* **have** *1*: *Suc ?j* = *n* − *2* **and** *?j* < *n* − *1* **and** *2*: *Suc* (*n* − *2*) = *n* − *Suc 0*
  and *3*: *n* − (*n* − *2*) = *2* **and** *4*: *n* − *?j* = *3*
  by *simp-all*
 have *int* (*cc* (*n* − *2*)) = *int* (*cc* (*Suc ?j*)) **by** (*simp only*: *1*)
 also from ‹*?j* < *n* − *1*› **have** ... = *2* + *2* ∗ (− *1*) ^ (*n* − *?j*) ∗ (*int d* − *1* *gchoose* (*n* − *Suc ?j*)) +
  ($\sum$ *i* = *?j+2..n−1*. (− *1*) ^ (*i* − *?j*) ∗ ((*int* (*aa i*) *gchoose* (*i* − *?j*)) + (*int* (*bb i*) *gchoose* (*i* − *?j*))))
  by (*rule cc-Suc*)
 also have ... = (*2* − *2* ∗ (*int d* − *1* *gchoose 2*)) + ((*int* (*aa* (*n* − *1*)) *gchoose 2*) + (*int* (*bb* (*n* − *1*)) *gchoose 2*))
  by (*simp add*: *1 2 3 4*)

also have ... $\leq$ (*2* $-$ *2* $*$ (*int d* $-$ *1 gchoose 2*)) + (*2* $*$ *int d gchoose 2*)
proof (*rule add-left-mono*)
have (*int* (*aa* (*n* $-$ *1*)) *gchoose 2*) + (*int* (*bb* (*n* $-$ *1*)) *gchoose 2*) $\leq$ *int* (*aa* (*n* $-$ *1*)) + *int* (*bb* (*n* $-$ *1*)) *gchoose 2*
by (*rule gbinomial-int-plus-le*) *simp-all*
also have ... = *int* (*2* $*$ *d*) *gchoose 2* **by** (*simp flip*: *cc-n-minus-1*)
also have ... = *2* $*$ *int d gchoose 2* **by** (*simp add*: *int-ops*(*7*))
finally show (*int* (*aa* (*n* $-$ *1*)) *gchoose 2*) + (*int* (*bb* (*n* $-$ *1*)) *gchoose 2*) $\leq$ *2* $*$ *int d gchoose 2* **.**
qed
also have ... = *2* $-$ *fact 2* $*$ (*int d* $-$ *1 gchoose 2*) + (*2* $*$ *int d gchoose 2*) **by** (*simp only*: *fact-2*)
also have ... = *2* $-$ (*int d* $-$ *1*) $*$ (*int d* $-$ *2*) + (*2* $*$ *int d gchoose 2*)
by (*simp only*: *gbinomial-int-mult-fact*) (*simp add*: *numeral-2-eq-2 prod.atLeast0-lessThan-Suc*)
also have ... = *2* $-$ (*int d* $-$ *1*) $*$ (*int d* $-$ *2*) + *int d* $*$ (*2* $*$ *int d* $-$ *1*)
by (*simp add*: *gbinomial-prod-rev numeral-2-eq-2 prod.atLeast0-lessThan-Suc*)
also have ... = *int* (d^2 + *2* $*$ *d*) **by** (*simp add*: *power2-eq-square*) (*simp only*: *algebra-simps*)
finally show *?thesis* **by** (*simp only*: *int-int-eq*)
qed

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 $-$
let *?f* = λ*i j*. (*int* (*aa i*) *gchoose* (*i* $-$ *j*)) + (*int* (*bb i*) *gchoose* (*i* $-$ *j*))
let *?S* = λ*x y*. ($\sum$ *i*=*j*+*x*..*n*$-$*y*. ($-$ *1*) ^(*i* $-$ *j*) $*$ *?f i j*)
let *?S3* = λ*x y*. ($\sum$ *i*=*j*+*x*..*n*$-$*y*. (*int* (*cc i*) *gchoose* (*i* $-$ *j*)))
have *ie1*: (*int* (*aa i*) *gchoose k*) + (*int* (*bb i*) *gchoose k*) $\leq$ *int* (*cc i*) *gchoose k* **if** *0* < *k* **for** *i k*
proof $-$
from *that* **have** (*int* (*aa i*) *gchoose k*) + (*int* (*bb i*) *gchoose k*) $\leq$ *int* (*aa i*) + *int* (*bb i*) *gchoose k*
by (*rule gbinomial-int-plus-le*) *simp-all*
also have ... = *int* (*cc i*) *gchoose k* **by** *simp*
finally show *?thesis* **.**
qed
from *d-gr-0* **have** *0* $\leq$ *int d* $-$ *1* **by** *simp*
from *assms* **have** *0* < *n* $-$ *Suc j* **by** *simp*
have *f-nonneg*: *0* $\leq$ *?f i j* **for** *i* **by** (*simp add*: *gbinomial-nneg*)

show *?thesis*
proof (*cases n* = *j* + *4*)
case *True*
hence *j*: *j* = *n* $-$ *4* **by** *simp*
have *1*: *n* $-$ *Suc j* = *3* **and** *j* < *n* $-$ *1* **and** *2*: *Suc* (*n* $-$ *3*) = *Suc* (*Suc j*)

and *3*: $n - (n - 3) = 3$
and *4*: $n - j = 4$ **and** *5*: $n - Suc\ 0 = Suc\ (Suc\ (Suc\ j))$ **and** *6*: $n - 2 = Suc\ (Suc\ j)$
by (*simp-all add*: *True*)
from ‹$j < n - 1$› **have** $int\ (cc\ (Suc\ j)) = 2 + 2 * (- 1)\ \hat{}\,(n - j) * (int\ d - 1\ gchoose\ (n - Suc\ j)) +$
$(\sum i = j+2..n-1.\ (- 1)\ \hat{}\,(i - j) * ((int\ (aa\ i)\ gchoose\ (i - j)) + (int\ (bb\ i)\ gchoose\ (i - j))))$
by (*rule cc-Suc*)
also have $\ldots = (2 + ((int\ (aa\ (n - 2))\ gchoose\ 2) + (int\ (bb\ (n - 2))\ gchoose\ 2))) +$
$(2 * (int\ d - 1\ gchoose\ 3) - ((int\ (aa\ (n - 1))\ gchoose\ 3) + (int\ (bb\ (n - 1))\ gchoose\ 3)))$
by (*simp add*: *1 2 3 4 5 6*)
also have $\ldots \le (2 + ((int\ (aa\ (n - 2))\ gchoose\ 2) + (int\ (bb\ (n - 2))\ gchoose\ 2))) + 0$
proof (*rule add-left-mono*)
from *cc-n-minus-1* **have** *eq1*: $int\ (aa\ (n - 1)) + int\ (bb\ (n - 1)) = 2 * int\ d$ **by** *simp*
hence *ie2*: $int\ (aa\ (n - 1)) \le 2 * int\ d$ **by** *simp*
from ‹$0 \le int\ d - 1$› **have** $int\ d - 1\ gchoose\ 3 \le int\ d\ gchoose\ 3$ **by** (*rule gbinomial-int-mono*) *simp*
hence $2 * (int\ d - 1\ gchoose\ 3) \le 2 * (int\ d\ gchoose\ 3)$ **by** *simp*
also from - *ie2* **have** $\ldots \le (int\ (aa\ (n - 1))\ gchoose\ 3) + (2 * int\ d - int\ (aa\ (n - 1))\ gchoose\ 3)$
by (*rule binomial-int-ineq-3*) *simp*
also have $\ldots = (int\ (aa\ (n - 1))\ gchoose\ 3) + (int\ (bb\ (n - 1))\ gchoose\ 3)$ **by** (*simp flip*: *eq1*)
finally show $2 * (int\ d - 1\ gchoose\ 3) - ((int\ (aa\ (n - 1))\ gchoose\ 3) + (int\ (bb\ (n - 1))\ gchoose\ 3)) \le 0$
by *simp*
qed
also have $\ldots = 2 + ((int\ (aa\ (n - 2))\ gchoose\ 2) + (int\ (bb\ (n - 2))\ gchoose\ 2))$ **by** *simp*
also from *ie1* **have** $\ldots \le 2 + (int\ (cc\ (n - 2))\ gchoose\ 2)$ **by** (*rule add-left-mono*) *simp*
also have $\ldots = 2 + (int\ (cc\ (j + 2))\ gchoose\ 2) + ?S3\ 4\ 1$ **by** (*simp add*: *True*)
finally show *?thesis* **.**
next
case *False*
with *assms* **have** $j + 4 \le n - 1$ **by** *simp*
from *n-gr-1* **have** $0 < n - 1$ **by** *simp*
from *assms* **have** $j + 2 \le n - 1$ **and** $j + 2 \le n - 2$ **by** *simp-all*
hence $n - j = Suc\ (n - Suc\ j)$ **by** *simp*
hence *1*: $(- 1)\ \hat{}\,(n - Suc\ j) = - ((- (1::int))\ \hat{}\,(n - j))$ **by** *simp*
from *assms* **have** $j < n - 1$ **by** *simp*
hence $int\ (cc\ (Suc\ j)) = 2 + 2 * (- 1)\hat{}(n - j) * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) + ?S\ 2\ 1$

by (*rule cc-Suc*)
also have ... = *2* ∗ (− *1*)^(*n* − *j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) +
(− *1*)^(*n* − *Suc j*) ∗ ((*int* (*aa* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) +
(*int* (*bb* (*n* − *1*)) *gchoose* (*n* − *Suc j*))) +
(*2* + *?S 2 2*)
using ‹*0* < *n* − *1*› ‹*j* + *2* ≤ *n* − *1*› **by** (*simp only*: *sum-tail-nat*) (*simp flip*: *numeral-2-eq-2*)
also have ... ≤ (*int* (*cc* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) + (*2* + *?S 2 2*)
proof (*rule add-right-mono*)
have *rl*: *x* − *y* ≤ *x* **if** *0* ≤ *y* **for** *x y* :: *int* **using** *that* **by** *simp*
have *2* ∗ (− *1*)^(*n* − *j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) +
(− *1*)^(*n* − *Suc j*) ∗ ((*int* (*aa* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) +
(*int* (*bb* (*n* − *1*)) *gchoose* (*n* − *Suc j*))) =
(−*1*)^(*n* − *j*) ∗ (*2* ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) −
(*int* (*aa* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) − (*int* (*bb* (*n* − *1*)) *gchoose* (*n* − *Suc j*)))
by (*simp only*: *1 algebra-simps*)
also have ... ≤ (*int* (*cc* (*n* − *1*))) *gchoose* (*n* − *Suc j*)
proof (*cases even* (*n* − *j*))
case *True*
hence (− *1*) ^(*n* − *j*) ∗ (*2* ∗ (*int d* − *1 gchoose* (*n* − *Suc j*)) − (*int* (*aa* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) −
(*int* (*bb* (*n* − *1*)) *gchoose* (*n* − *Suc j*))) =
2 ∗ (*int d* − *1 gchoose* (*n* − *Suc j*)) − ((*int* (*aa* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) +
(*int* (*bb* (*n* − *1*)) *gchoose* (*n* − *Suc j*)))
by *simp*
also have ... ≤ *2* ∗ (*int d* − *1 gchoose* (*n* − *Suc j*)) **by** (*rule rl*) (*simp add*: *gbinomial-nneg*)
also have ... = (*int d* − *1 gchoose* (*n* − *Suc j*)) + (*int d* − *1 gchoose* (*n* − *Suc j*)) **by** *simp*
also have ... ≤ (*int d* − *1*) + (*int d* − *1*) *gchoose* (*n* − *Suc j*)
using ‹*0* < *n* − *Suc j*› ‹*0* ≤ *int d* − *1*› ‹*0* ≤ *int d* − *1*› **by** (*rule gbinomial-int-plus-le*)
also have ... ≤ *2* ∗ *int d gchoose* (*n* − *Suc j*)
proof (*rule gbinomial-int-mono*)
from ‹*0* ≤ *int d* − *1*› **show** *0* ≤ *int d* − *1* + (*int d* − *1*) **by** *simp*
qed *simp*
also have ... = *int* (*cc* (*n* − *1*)) *gchoose* (*n* − *Suc j*) **by** (*simp only*: *cc-n-minus-1*) *simp*
finally show *?thesis* **.**
next
case *False*
hence (− *1*) ^(*n* − *j*) ∗ (*2* ∗ (*int d* − *1 gchoose* (*n* − *Suc j*)) − (*int* (*aa* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) −
(*int* (*bb* (*n* − *1*)) *gchoose* (*n* − *Suc j*))) =
((*int* (*aa* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) + (*int* (*bb* (*n* − *1*)) *gchoose* (*n* − *Suc j*))) −
2 ∗ (*int d* − *1 gchoose* (*n* − *Suc j*))

by *simp*
also have ... ≤ (*int* (*aa* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) + (*int* (*bb* (*n* − *1*)) *gchoose* (*n* − *Suc j*))
by (*rule rl*) (*simp add*: *gbinomial-nneg d-gr-0*)
also from ‹*0* < *n* − *Suc j*› **have** ... ≤ *int* (*cc* (*n* − *1*)) *gchoose* (*n* − *Suc j*) **by** (*rule ie1*)
finally show *?thesis* **.**
qed
finally show *2* ∗ (− *1*)^(*n* − *j*) ∗ ((*int d* − *1*) *gchoose* (*n* − *Suc j*)) +
(− *1*)^(*n* − *Suc j*) ∗ ((*int* (*aa* (*n* − *1*)) *gchoose* (*n* − *Suc j*)) + (*int* (*bb* (*n* − *1*)) *gchoose* (*n* − *Suc j*))) ≤
(*int* (*cc* (*n* − *1*))) *gchoose* (*n* − *Suc j*) **.**
qed
also have ... = *2* + (*int* (*cc* (*n* − *1*)) *gchoose* ((*n* − *1*) − *j*)) + ((*int* (*aa* (*j* + *2*)) *gchoose* *2*) +
(*int* (*bb* (*j* + *2*)) *gchoose* *2*)) + *?S 3 2*
using ‹*j* + *2* ≤ *n* − *2*› **by** (*simp add*: *sum.atLeast-Suc-atMost numeral-3-eq-3*)
also have ... ≤ *2* + (*int* (*cc* (*n* − *1*)) *gchoose* ((*n* − *1*) − *j*)) + ((*int* (*aa* (*j* + *2*)) *gchoose* *2*) +
(*int* (*bb* (*j* + *2*)) *gchoose* *2*)) + *?S3 4 2*
proof (*rule add-left-mono*)
from ‹*j* + *4* ≤ *n* − *1*› **have** *j* + *3* ≤ *n* − *2* **by** *simp*
hence *?S 3 2* = *?S 4 2* − *?f* (*j* + *3*) *j* **by** (*simp add*: *sum.atLeast-Suc-atMost add.commute*)
hence *?S 3 2* ≤ *?S 4 2* **using** *f-nonneg*[*of j* + *3*] **by** *simp*
also have ... ≤ *?S3 4 2*
proof (*rule sum-mono*)
fix *i*
assume *i* ∈ {*j* + *4*..*n* − *2*}
hence *0* < *i* − *j* **by** *simp*
from *f-nonneg*[*of i*] **have** (− *1*)^(*i* − *j*) ∗ *?f i j* ≤ *?f i j*
by (*smt minus-one-mult-self mult-cancel-right1 pos-zmult-eq-1-iff-lemma zero-less-mult-iff*)
also from ‹*0* < *i* − *j*› **have** ... ≤ *int* (*cc i*) *gchoose* (*i* − *j*) **by** (*rule ie1*)
finally show (− *1*)^(*i* − *j*) ∗ *?f i j* ≤ *int* (*cc i*) *gchoose* (*i* − *j*) **.**
qed
finally show *?S 3 2* ≤ *?S3 4 2* **.**
qed
also have ... = ((*int* (*aa* (*j* + *2*)) *gchoose* *2*) + (*int* (*bb* (*j* + *2*)) *gchoose* *2*)) + (*2* + *?S3 4 1*)
using ‹*0* < *n* − *1*› ‹*j* + *4* ≤ *n* − *1*› **by** (*simp only*: *sum-tail-nat*) (*simp flip*: *numeral-2-eq-2*)
also from *ie1* **have** ... ≤ (*int* (*cc* (*j* + *2*)) *gchoose* *2*) + (*2* + *?S3 4 1*)
by (*rule add-right-mono*) *simp*
also have ... = *2* + (*int* (*cc* (*j* + *2*)) *gchoose* *2*) + *?S3 4 1* **by** (*simp only*: *ac-simps*)
finally show *?thesis* **.**
qed
qed

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 −
define *j0* **where** $j0 = j - 1$
with *assms* **have** *j*: $j = Suc\ j0$ **and** $j0 < n - 3$ **by** *simp-all*
have $int\ (cc\ j) = int\ (cc\ (Suc\ j0))$ **by** (*simp only*: *j*)
also have $\ldots \leq 2 + (int\ (cc\ (j0 + 2))\ gchoose\ 2) + (\sum i{=}j0{+}4..n{-}1.\ int\ (cc\ i)\ gchoose\ (i - j0))$
using ‹$j0 < n - 3$› **by** (*rule cc-Suc-le*)
also have $\ldots = 2 + (int\ (cc\ (j + 1))\ gchoose\ 2) + (\sum i{=}j0{+}4..n{-}1.\ int\ (cc\ i)\ gchoose\ (i - j0))$
by (*simp add*: *j*)
also have $(\sum i{=}j0{+}4..n{-}1.\ int\ (cc\ i)\ gchoose\ (i - j0)) = int\ (\sum i{=}j{+}3..n{-}1.\ cc\ i\ choose\ (Suc\ (i - j)))$
unfolding *int-sum*
proof (*rule sum.cong*)
fix *i*
assume $i \in \{j + 3..n - 1\}$
hence $Suc\ j0 < i$ **by** (*simp add*: *j*)
hence $i - j0 = Suc\ (i - j)$ **by** (*simp add*: *j*)
thus $int\ (cc\ i)\ gchoose\ (i - j0) = int\ (cc\ i\ choose\ (Suc\ (i - j)))$ **by** (*simp add*: *int-binomial*)
qed (*simp add*: *j*)
finally have $int\ (cc\ j) \leq int\ (2 + (cc\ (j + 1)\ choose\ 2) + (\sum i = j + 3..n - 1.\ cc\ i\ choose\ (Suc\ (i - j))))$
by (*simp only*: *int-plus int-binomial*)
thus *?thesis* **by** (*simp only*: *zle-int*)
qed

corollary *cc-le-Dube-aux*: $0 < j \Longrightarrow j + 1 \leq n \Longrightarrow cc\ j \leq Dube\text{-}aux\ n\ d\ j$
proof (*induct j rule*: *Dube-aux.induct*[**where** $n{=}n$])
case *step*: (*1 j*)
from *step.prems*(*2*) **have** $j + 2 < n \lor j + 2 = n \lor j + 1 = n$ **by** *auto*
thus *?case*
proof (*elim disjE*)
assume $*$: $j + 2 < n$
moreover have $0 < j + 1$ **by** *simp*
moreover from $*$ **have** $j + 1 + 1 \leq n$ **by** *simp*
ultimately have $cc\ (j + 1) \leq Dube\text{-}aux\ n\ d\ (j + 1)$ **by** (*rule step.hyps*)
hence *1*: $cc\ (j + 1)\ choose\ 2 \leq Dube\text{-}aux\ n\ d\ (j + 1)\ choose\ 2$
by (*rule Binomial-Int.binomial-mono*)
have *2*: $(\sum i = j + 3..n - 1.\ cc\ i\ choose\ Suc\ (i - j)) \leq$
$(\sum i = j + 3..n - 1.\ Dube\text{-}aux\ n\ d\ i\ choose\ Suc\ (i - j))$
proof (*rule sum-mono*)
fix *i*::*nat*
note $*$

moreover assume $i \in \{j + 3..n - 1\}$
moreover from *this* ‹$2 < n$› **have** $0 < i$ **and** $i + 1 \leq n$ **by** *auto*
ultimately have *cc i* $\leq$ *Dube-aux n d i* **by** (*rule step.hyps*)
thus *cc i choose Suc* $(i - j) \leq$ *Dube-aux n d i choose Suc* $(i - j)$
by (*rule Binomial-Int.binomial-mono*)
qed
from $*$ **have** $j < n - 2$ **by** *simp*
with *step.prems*(*1*) **have** *cc j* $\leq 2 +$ (*cc* $(j + 1)$ *choose 2*) $+ (\sum i = j + 3..n - 1.$ *cc i choose Suc* $(i - j))$
by (*rule cc-le*)
also from $*$ *1 2* **have** $\ldots \leq$ *Dube-aux n d j* **by** *simp*
finally show *?thesis* **.**
next
assume $j + 2 = n$
hence $j = n - 2$ **and** *Dube-aux n d j* $= d^2 + 2 * d$ **by** *simp-all*
thus *?thesis* **by** (*simp only*: *cc-n-minus-2*)
next
assume $j + 1 = n$
hence $j = n - 1$ **and** *Dube-aux n d j* $= 2 * d$ **by** *simp-all*
thus *?thesis* **by** (*simp only*: *cc-n-minus-1*)
qed
qed

end

lemma *Dube-aux*:
assumes $g \in$ *punit.reduced-GB F*
shows *poly-deg g* $\leq$ *Dube-aux n d 1*
proof (*cases* $n = 2$)
case *True*
from *assms* **have** *poly-deg g* $\leq$ *bb 0* **by** (*rule deg-RGB*)
also have $\ldots \leq$ *max* (*aa 1*) (*bb 1*) **by** (*fact bb-0*)
also have $\ldots \leq$ *cc* $(n - 1)$ **by** (*simp add*: *True*)
also have $\ldots = 2 * d$ **by** (*fact cc-n-minus-1*)
also have $\ldots =$ *Dube-aux n d 1* **by** (*simp add*: *True*)
finally show *?thesis* **.**
next
case *False*
with ‹$1 < n$› **have** $2 < n$ **and** $1 + 1 \leq n$ **by** *simp-all*
from *assms* **have** *poly-deg g* $\leq$ *bb 0* **by** (*rule deg-RGB*)
also have $\ldots \leq$ *max* (*aa 1*) (*bb 1*) **by** (*fact bb-0*)
also have $\ldots \leq$ *cc 1* **by** *simp*
also from ‹$2 < n$› - ‹$1 + 1 \leq n$› **have** $\ldots \leq$ *Dube-aux n d 1* **by** (*rule cc-le-Dube-aux*) *simp*
finally show *?thesis* **.**
qed

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-GB* F
shows *poly-deg* $g \leq$ *Dube* (*card* X) (*maxdeg* F)
proof (*cases* $F \subseteq \{0\}$)
case *True*
hence $F = \{\} \vee F = \{0\}$ **by** *blast*
with *assms*(*4*) **show** *?thesis* **by** (*auto simp*: *punit.reduced-GB-empty punit.reduced-GB-singleton*)
next
case *False*
hence $F - \{0\} \neq \{\}$ **by** *simp*
hence $F \neq \{\}$ **by** *blast*
hence *poly-deg* ' $F \neq \{\}$ **by** *simp*
from *assms*(*1*) **have** *fin1*: *finite* (*poly-deg* ' F) **by** (*rule finite-imageI*)
from *assms*(*1*) **have** *finite* $(F - \{0\})$ **by** *simp*
hence *fin*: *finite* (*poly-deg* ' $(F - \{0\})$) **by** (*rule finite-imageI*)
moreover from ‹$F - \{0\} \neq \{\}$› **have** $*$: *poly-deg* ' $(F - \{0\}) \neq \{\}$ **by** *simp*
ultimately have *maxdeg* $(F - \{0\}) \in$ *poly-deg* ' $(F - \{0\})$ **unfolding** *maxdeg-def* **by** (*rule Max-in*)
then obtain f **where** $f \in F - \{0\}$ **and** *md1*: *maxdeg* $(F - \{0\}) =$ *poly-deg* f ..
note *this*(*2*)
moreover have *maxdeg* $(F - \{0\}) \leq$ *maxdeg* F
unfolding *maxdeg-def* **using** *image-mono* $*$ *fin1* **by** (*rule Max-mono*) *blast*
ultimately have *poly-deg* $f \leq$ *maxdeg* F **by** *simp*
from ‹$f \in F - \{0\}$› **have** $f \in F$ **and** $f \neq 0$ **by** *simp-all*
from *this*(*1*) *assms*(*2*) **have** $f \in P[X]$ ..
have *f-max*: *poly-deg* $f' \leq$ *poly-deg* f **if** $f' \in F$ **for** f'
proof (*cases* $f' = 0$)
case *True*
thus *?thesis* **by** *simp*
next
case *False*
with *that* **have** $f' \in F - \{0\}$ **by** *simp*
hence *poly-deg* $f' \in$ *poly-deg* ' $(F - \{0\})$ **by** (*rule imageI*)
with *fin* **show** *poly-deg* $f' \leq$ *poly-deg* f **unfolding** *md1*[*symmetric*] *maxdeg-def* **by** (*rule Max-ge*)
qed
have *maxdeg* $F \leq$ *poly-deg* f **unfolding** *maxdeg-def* **using** *fin1* ‹*poly-deg* ' $F \neq \{\}$›
proof (*rule Max.boundedI*)
fix d
assume $d \in$ *poly-deg* ' F
then obtain f' **where** $f' \in F$ **and** $d =$ *poly-deg* f' ..
note *this*(*2*)
also from ‹$f' \in F$› **have** *poly-deg* $f' \leq$ *poly-deg* f **by** (*rule f-max*)
finally show $d \leq$ *poly-deg* f .
qed
with ‹*poly-deg* $f \leq$ *maxdeg* F› **have** *md*: *poly-deg* $f =$ *maxdeg* F **by** (*rule antisym*)

show *?thesis*
proof (*cases ideal* $\{f\}$ = *ideal F*)
case *True*
note *assms*(*4*)
also have *punit.reduced-GB F* = *punit.reduced-GB* $\{f\}$
using *punit.finite-reduced-GB-finite punit.reduced-GB-is-reduced-GB-finite*
by (*rule punit.reduced-GB-unique*) (*simp-all add*: *punit.reduced-GB-pmdl-finite*[*simplified*] *True*)
also have ... $\subseteq$ $\{$*punit.monic f*$\}$ **by** (*simp add*: *punit.reduced-GB-singleton*)
finally have $g \in$ $\{$*punit.monic f*$\}$ **.**
hence *poly-deg g* = *poly-deg* (*punit.monic f*) **by** *simp*
also from *poly-deg-monom-mult-le*[**where** *c=1 / lcf f* **and** *t=0* **and** *p=f*] **have** ... $\le$ *poly-deg f*
by (*simp add*: *punit.monic-def*)
also have ... = *maxdeg F* **by** (*fact md*)
also have ... $\le$ *Dube* (*card X*) (*maxdeg F*) **by** (*fact Dube-ge-d*)
finally show *?thesis* **.**
next
case *False*
show *?thesis*
proof (*cases poly-deg f* = *0*)
case *True*
hence *monomial* (*lookup f 0*) *0* = *f* **by** (*rule poly-deg-zero-imp-monomial*)
moreover define *c* **where** *c* = *lookup f 0*
ultimately have *f*: *f* = *monomial c 0* **by** *simp*
with ‹*f* $\neq$ *0*› **have** *c* $\neq$ *0* **by** (*simp add*: *monomial-0-iff*)
from ‹*f* $\in$ *F*› **have** *f* $\in$ *ideal F* **by** (*rule ideal.span-base*)
hence *punit.monom-mult* (*1 / c*) *0 f* $\in$ *ideal F* **by** (*rule punit.pmdl-closed-monom-mult*[*simplified*])
with ‹*c* $\neq$ *0*› **have** *ideal F* = *UNIV*
by (*simp add*: *f punit.monom-mult-monomial ideal-eq-UNIV-iff-contains-one*)
with *assms*(*1*) **have** *punit.reduced-GB F* = $\{1\}$
by (*simp only*: *ideal-eq-UNIV-iff-reduced-GB-eq-one-finite*)
with *assms*(*4*) **show** *?thesis* **by** *simp*
next
case *False*
hence *0* < *poly-deg f* **by** *simp*
have *card X* $\le$ *1* $\vee$ *1* < *card X* **by** *auto*
thus *?thesis*
proof
note *fin-X*
moreover assume *card X* $\le$ *1*
moreover note *assms*(*2*)
moreover from ‹*f* $\in$ *F*› **have** *f* $\in$ *ideal F* **by** (*rule ideal.span-base*)
ultimately have *poly-deg g* $\le$ *poly-deg f*
using ‹*f* $\neq$ *0*› *assms*(*4*) **by** (*rule deg-reduced-GB-univariate-le*)
also have ... $\le$ *Dube* (*card X*) (*maxdeg F*) **unfolding** *md* **by** (*fact Dube-ge-d*)
finally show *?thesis* **.**
next
assume *1* < *card X*

hence *poly-deg g* ≤ *Dube-aux* (*card X*) (*poly-deg f*) *1*
using *assms*(*1*, *2*) ‹*f* ∈ *F*› *assms*(*3*) *f-max* ‹*0* < *poly-deg f*› ‹*ideal* {*f*} ≠ *ideal F*› *assms*(*4*)
by (*rule Dube-aux*)
also from ‹*1* < *card X*› ‹*0* < *poly-deg f*› **have** ... = *Dube* (*card X*) (*maxdeg F*)
by (*simp add*: *Dube-def md*)
finally show *?thesis* **.**
qed
qed
qed
qed

corollary *Dube-is-hom-GB-bound*:
finite F ⟹ *F* ⊆ *P*[*X*] ⟹ *is-hom-GB-bound F* (*Dube* (*card X*) (*maxdeg F*))
by (*intro is-hom-GB-boundI Dube*)

end

corollary *Dube-indets*:
assumes *finite F* **and** ⋀*f*. *f* ∈ *F* ⟹ *homogeneous f* **and** *g* ∈ *punit.reduced-GB F*
shows *poly-deg g* ≤ *Dube* (*card* (⋃(*indets* ‘ *F*))) (*maxdeg F*)
using - *assms*(*1*) - *assms*(*2*, *3*)
proof (*rule Dube*)
from *assms* **show** *finite* (⋃(*indets* ‘ *F*)) **by** (*simp add*: *finite-indets*)
next
show *F* ⊆ *P*[⋃(*indets* ‘ *F*)] **by** (*auto simp*: *Polys-alt*)
qed

corollary *Dube-is-hom-GB-bound-indets*:
finite F ⟹ *is-hom-GB-bound F* (*Dube* (*card* (⋃(*indets* ‘ *F*))) (*maxdeg F*))
by (*intro is-hom-GB-boundI Dube-indets*)

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* ⊆ *P*[*X*]
shows *is-GB-cofactor-bound F* (*Dube* (*Suc* (*card X*)) (*maxdeg F*))
using *assms*(*1*, *3*)
proof (*rule hom-GB-bound-is-GB-cofactor-bound*)
let *?F* = *homogenize None* ‘ *extend-indets* ‘ *F*
let *?X* = *insert None* (*Some* ‘ *X*)
from *assms*(*1*) **have** *finite ?X* **by** *simp*

moreover from *assms*(*2*) **have** *finite ?F* **by** (*intro finite-imageI*)
moreover have $?F \subseteq P[?X]$
proof
fix f'
assume $f' \in ?F$
then obtain f **where** $f \in F$ **and** f': $f' = homogenize\ None\ (extend\text{-}indets\ f)$
by *blast*
from *this*(*1*) *assms*(*3*) **have** $f \in P[X]$ **..**
hence $extend\text{-}indets\ f \in P[Some\ `\ X]$ **by** (*auto simp*: *Polys-alt indets-extend-indets*)
thus $f' \in P[?X]$ **unfolding** f' **by** (*rule homogenize-in-Polys*)
qed
ultimately have *extended-ord.is-hom-GB-bound ?F* (*Dube* (*card ?X*) (*maxdeg*
?F))
by (*rule extended-ord.Dube-is-hom-GB-bound*)
moreover have *maxdeg ?F* = *maxdeg F*
proof −
have *maxdeg ?F* = *maxdeg* (*extend-indets* ‘ *F*)
by (*auto simp*: *indets-extend-indets intro*: *maxdeg-homogenize*)
also have . . . = *maxdeg F* **by** (*simp add*: *maxdeg-def image-image*)
finally show *maxdeg ?F* = *maxdeg F* **.**
qed
moreover from *assms*(*1*) **have** *card ?X* = *card X* + *1* **by** (*simp add*: *card-image*)
ultimately show *extended-ord.is-hom-GB-bound ?F* (*Dube* (*Suc* (*card X*)) (*maxdeg*
F)) **by** *simp*
qed

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 $\bigwedge g.\ g \in G \Longrightarrow \exists\, q.\ g = (\sum f{\in}F.\ q\ f * f) \wedge$
$(\forall f.\ q\ f \in P[X] \wedge poly\text{-}deg\ (q\ f * f) \le Dube\ (Suc\ (card\ X))$
$(maxdeg\ F) \wedge$
$(f \notin F \longrightarrow q\ f = 0))$
proof −
from *assms* **have** *is-GB-cofactor-bound F* (*Dube* (*Suc* (*card X*)) (*maxdeg F*))
(**is** *is-GB-cofactor-bound - ?b*) **by** (*rule Dube-is-GB-cofactor-bound*)
moreover note *assms*(*3*)
ultimately obtain *G* **where** *punit.is-Groebner-basis G* **and** *ideal G* = *ideal F*
and $G \subseteq P[X]$
and *1*: $\bigwedge g.\ g \in G \Longrightarrow \exists\, F'\ q.\ finite\ F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\ f * f) \wedge$
$(\forall f.\ q\ f \in P[X] \wedge poly\text{-}deg\ (q\ f * f) \le ?b \wedge (f \notin F' \longrightarrow q$
$f = 0))$
by (*rule is-GB-cofactor-boundE-Polys*) *blast*
from *this*(*1*−*3*) **show** *?thesis*
proof
fix *g*
assume $g \in G$
hence $\exists\, F'\ q.\ finite\ F' \wedge F' \subseteq F \wedge g = (\sum f{\in}F'.\ q\ f * f) \wedge$

$(\forall f.\ q\ f \in P[X] \wedge$ *poly-deg* $(q\ f * f) \leq\ ?b \wedge (f \notin F' \longrightarrow q\ f = 0))$
by (*rule 1*)
then obtain $F'\ q$ **where** $F' \subseteq F$ **and** g: $g = (\sum f{\in}F'.\ q\ f * f)$ **and** $\bigwedge f.\ q\ f \in P[X]$
and $\bigwedge f.$ *poly-deg* $(q\ f * f) \leq\ ?b$ **and** 2: $\bigwedge f.\ f \notin F' \Longrightarrow q\ f = 0$ **by** *blast*
show $\exists\, q.\ g = (\sum f{\in}F.\ q\ f * f) \wedge (\forall f.\ q\ f \in P[X] \wedge$ *poly-deg* $(q\ f * f) \leq\ ?b \wedge (f \notin F \longrightarrow q\ f = 0))$
proof (*intro exI allI conjI impI*)
from *assms*(*2*) ‹$F' \subseteq F$› **have** $(\sum f{\in}F'.\ q\ f * f) = (\sum f{\in}F.\ q\ f * f)$
proof (*intro sum.mono-neutral-left ballI*)
fix f
assume $f \in F - F'$
hence $f \notin F'$ **by** *simp*
hence $q\ f = 0$ **by** (*rule 2*)
thus $q\ f * f = 0$ **by** *simp*
qed
thus $g = (\sum f{\in}F.\ q\ f * f)$ **by** (*simp only*: *g*)
next
fix f
assume $f \notin F$
with ‹$F' \subseteq F$› **have** $f \notin F'$ **by** *blast*
thus $q\ f = 0$ **by** (*rule 2*)
qed *fact+*
qed
qed

corollary *Dube-is-GB-cofactor-bound-indets*:
assumes *finite F*
shows *is-GB-cofactor-bound F* (*Dube* (*Suc* (*card* ($\bigcup$(*indets* ‘ *F*)))) (*maxdeg F*))
using - *assms* -
proof (*rule Dube-is-GB-cofactor-bound*)
from *assms* **show** *finite* ($\bigcup$(*indets* ‘ *F*)) **by** (*simp add*: *finite-indets*)
next
show $F \subseteq P[\bigcup(\textit{indets}\ \text{‘}\ F)]$ **by** (*auto simp*: *Polys-alt*)
qed

end

end

12 Sample Computations of Gröbner Bases via Macaulay Matrices

theory *Groebner-Macaulay-Examples*
imports
Groebner-Macaulay
Dube-Bound

Groebner-Bases.Benchmarks
Jordan-Normal-Form.Gauss-Jordan-IArray-Impl
Groebner-Bases.Code-Target-Rat
begin

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

context *extended-ord-pm-powerprod*
begin

theorem *thm-2-3-6-Dube*:
assumes *finite X* **and** *set fs* $\subseteq$ *P*[*X*]
shows *punit.is-Groebner-basis* (*set* (*punit.Macaulay-list*
(*deg-shifts X* (*Dube* (*Suc* (*card X*)) (*maxdeg* (*set fs*))) *fs*)))
using *assms Dube-is-GB-cofactor-bound* **by** (*rule thm-2-3-6*) (*simp-all add*: *assms*)

theorem *thm-2-3-7-Dube*:
assumes *finite X* **and** *set fs* $\subseteq$ *P*[*X*]
shows *1* $\in$ *ideal* (*set fs*) $\longleftrightarrow$
1 $\in$ *set* (*punit.Macaulay-list* (*deg-shifts X* (*Dube* (*Suc* (*card X*)) (*maxdeg* (*set fs*))) *fs*))
using *assms Dube-is-GB-cofactor-bound* **by** (*rule thm-2-3-7*) (*simp-all add*: *assms*)

theorem *thm-2-3-6-indets-Dube*:
fixes *fs*
defines *X* $\equiv$ $\bigcup$(*indets* ‘ *set fs*)
shows *punit.is-Groebner-basis* (*set* (*punit.Macaulay-list*
(*deg-shifts X* (*Dube* (*Suc* (*card X*)) (*maxdeg* (*set fs*))) *fs*)))
unfolding *X-def* **using** *Dube-is-GB-cofactor-bound-indets* **by** (*rule thm-2-3-6-indets*) (*fact finite-set*)

theorem *thm-2-3-7-indets-Dube*:
fixes *fs*
defines *X* $\equiv$ $\bigcup$(*indets* ‘ *set fs*)
shows *1* $\in$ *ideal* (*set fs*) $\longleftrightarrow$
1 $\in$ *set* (*punit.Macaulay-list* (*deg-shifts X* (*Dube* (*Suc* (*card X*)) (*maxdeg* (*set fs*))) *fs*))
unfolding *X-def* **using** *Dube-is-GB-cofactor-bound-indets* **by** (*rule thm-2-3-7-indets*) (*fact finite-set*)

end

12.2 Preparations

primrec *remdups-wrt-rev* :: (*'a* $\Rightarrow$ *'b*) $\Rightarrow$ *'a list* $\Rightarrow$ *'b list* $\Rightarrow$ *'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*: $v \in$ *set vs* $\Longrightarrow v \notin f$ ‘ *set* (*remdups-wrt-rev f xs vs*)
proof (*induct xs arbitrary*: *vs*)
case *Nil*
show *?case* **by** *simp*
next
case (*Cons x xs*)
from *Cons*(*2*) **have** *1*: $v \notin f$ ‘ *set* (*remdups-wrt-rev f xs vs*) **by** (*rule Cons*(*1*))
from *Cons*(*2*) **have** $v \in$ *set* (*f x* # *vs*) **by** *simp*
hence *2*: $v \notin f$ ‘ *set* (*remdups-wrt-rev f xs* (*f x* # *vs*)) **by** (*rule Cons*(*1*))
from *Cons*(*2*) **show** *?case* **by** (*auto simp*: *Let-def 1 2 List.member-def*)
qed

lemma *distinct-remdups-wrt-rev*: *distinct* (*map f* (*remdups-wrt-rev f xs vs*))
proof (*induct xs arbitrary*: *vs*)
case *Nil*
show *?case* **by** *simp*
next
case (*Cons x xs*)
show *?case* **by** (*simp add*: *Let-def Cons*(*1*) *remdups-wrt-rev-notin*)
qed

lemma *map-of-remdups-wrt-rev′*:
map-of (*remdups-wrt-rev fst xs vs*) *k* = *map-of* (*filter* (λx. *fst x* $\notin$ *set vs*) *xs*) *k*
proof (*induct xs arbitrary*: *vs*)
case *Nil*
show *?case* **by** *simp*
next
case (*Cons x xs*)
show *?case*
proof (*simp add*: *Let-def List.member-def Cons*, *intro impI*)
assume $k \neq$ *fst x*
have *map-of* (*filter* (λy. *fst y* $\neq$ *fst x* $\wedge$ *fst y* $\notin$ *set vs*) *xs*) =
map-of (*filter* (λy. *fst y* $\neq$ *fst x*) (*filter* (λy. *fst y* $\notin$ *set vs*) *xs*))
by (*simp only*: *filter-filter conj-commute*)
also have ... = *map-of* (*filter* (λy. *fst y* $\notin$ *set vs*) *xs*) |‘ {*y*. *y* $\neq$ *fst x*} **by** (*rule map-of-filter*)
finally show *map-of* (*filter* (λy. *fst y* $\neq$ *fst x* $\wedge$ *fst y* $\notin$ *set vs*) *xs*) *k* =
map-of (*filter* (λy. *fst y* $\notin$ *set vs*) *xs*) *k*
by (*simp add*: *restrict-map-def* ‹*k* $\neq$ *fst x*›)
qed
qed

corollary *map-of-remdups-wrt-rev*: *map-of* (*remdups-wrt-rev fst xs* []) = *map-of xs*
by (*rule ext*, *simp add*: *map-of-remdups-wrt-rev′*)

lemma (**in** *term-powerprod*) *compute-list-to-poly* [*code*]:
list-to-poly ts cs = $distr_0$ *DRLEX* (*remdups-wrt-rev fst* (*zip ts cs*) [])
by (*rule poly-mapping-eqI*,
simp add: *lookup-list-to-poly list-to-fun-def* $distr_0$*-def oalist-of-list-ntm-def*
oa-ntm.lookup-oalist-of-list distinct-remdups-wrt-rev lookup-dflt-def map-of-remdups-wrt-rev)

lemma (**in** *ordered-term*) *compute-Macaulay-list* [*code*]:
Macaulay-list ps =
(*let ts* = *Keys-to-list ps in*
filter ($\lambda p.\ p \neq 0$) (*mat-to-polys ts* (*row-echelon* (*polys-to-mat ts ps*)))
)
by (*simp add*: *Macaulay-list-def Macaulay-mat-def Let-def*)

declare *conversep-iff* [*code*]

derive (*eq*) *ceq poly-mapping*
derive (*no*) *ccompare poly-mapping*
derive (*dlist*) *set-impl poly-mapping*
derive (*no*) *cenum poly-mapping*

derive (*eq*) *ceq rat*
derive (*no*) *ccompare rat*
derive (*dlist*) *set-impl rat*
derive (*no*) *cenum rat*

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*
by (*simp add*: *PP-inject inj-def*)

lemma *inj-mapping-of*: *inj mapping-of*
by (*simp add*: *mapping-of-inject inj-def*)

lemma *mapping-of-comp-PP* [*simp*]:
mapping-of ∘ *PP* = ($\lambda x.\ x$)
PP ∘ *mapping-of* = ($\lambda x.\ x$)
by (*simp-all add*: *comp-def PP-inverse mapping-of-inverse*)

lemma *map-key-PP-mapping-of* [*simp*]: *Poly-Mapping.map-key PP* (*Poly-Mapping.map-key mapping-of p*) = *p*
by (*simp add*: *map-key-compose*[*OF inj-PP inj-mapping-of*] *comp-def PP-inverse map-key-id*)

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

by (*simp add*: *map-key-compose*[*OF inj-mapping-of inj-PP*] *comp-def mapping-of-inverse map-key-id*)

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*)
by (*simp add*: *map-key.rep-eq inj-PP*)

lemma *keys-map-key-PP*: *keys* (*Poly-Mapping.map-key PP p*) = *mapping-of* ‘ *keys p*
by (*simp add*: *keys-map-key inj-PP*)
(*smt Collect-cong PP-inverse UNIV-I image-def pp.mapping-of-inverse vimage-def*)

lemma *map-key-PP-zero-iff* [*iff*]: *Poly-Mapping.map-key PP p* = *0* $\longleftrightarrow$ *p* = *0*
by (*metis map-key-PP-zero map-key-mapping-of-PP*)

lemma *map-key-PP-uminus* [*simp*]: *Poly-Mapping.map-key PP* ($-$ *p*) = $-$ *Poly-Mapping.map-key PP p*
by (*rule poly-mapping-eqI*) (*simp add*: *lookup-map-key-PP*)

lemma *map-key-PP-minus*:
Poly-Mapping.map-key PP (*p* $-$ *q*) = *Poly-Mapping.map-key PP p* $-$ *Poly-Mapping.map-key PP q*
by (*rule poly-mapping-eqI*) (*simp add*: *lookup-map-key-PP lookup-minus*)

lemma *map-key-PP-monomial* [*simp*]: *Poly-Mapping.map-key PP* (*monomial c t*) = *monomial c* (*mapping-of t*)
proof –
have *Poly-Mapping.map-key PP* (*monomial c t*) = *Poly-Mapping.map-key PP* (*monomial c* (*PP* (*mapping-of t*)))
by (*simp only*: *mapping-of-inverse*)
also from *inj-PP* **have** ... = *monomial c* (*mapping-of t*) **by** (*fact map-key-single*)
finally show *?thesis* **.**
qed

lemma *map-key-PP-one* [*simp*]: *Poly-Mapping.map-key PP 1* = *1*
by (*simp add*: *zero-pp.rep-eq flip*: *single-one*)

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*)
by (*rule poly-mapping-eqI*)
(*simp add*: *punit.lookup-monom-mult monom-mult-punit-def adds-pp-iff PP-inverse lookup-map-key-PP*
mapping-of-inverse flip: *minus-pp.abs-eq*)

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* $\Rightarrow_0$ -)
by (*induct p rule*: *poly-mapping-plus-induct*)
(*simp-all add*: *distrib-right map-key-PP-plus times-monomial-left map-key-PP-monom-mult-punit flip*: *monom-mult-punit-def*)

lemma *map-key-PP-sum*: *Poly-Mapping.map-key PP* (*sum f A*) = ($\sum$ *a*$\in$*A*. *Poly-Mapping.map-key PP* (*f a*))
by (*induct A rule*: *infinite-finite-induct*) (*simp-all add*: *map-key-PP-plus*)

lemma *map-key-PP-ideal*:
Poly-Mapping.map-key PP ' *ideal F* = *ideal* (*Poly-Mapping.map-key PP* ' (*F*::((-, -::*add-linorder*) *pp* $\Rightarrow_0$ -) *set*))
proof −
from *map-key-PP-mapping-of* **have** *surj* (*Poly-Mapping.map-key PP*) **by** (*rule surjI*)
with *map-key-PP-plus map-key-PP-times* **show** *?thesis* **by** (*rule image-ideal-eq-surj*)
qed

12.2.2 Locale *pp-powerprod*

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

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

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

sublocale *gd-powerprod* **..**

sublocale *pp-pm*: *extended-ord-pm-powerprod* λ*s t*. *ord* (*PP s*) (*PP t*) λ*s t*. *ord-strict* (*PP s*) (*PP t*)
by *standard* (*auto simp*: *zero-min plus-monotone simp flip*: *zero-pp-def plus-pp.abs-eq PP-inject*)

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

primrec *deg-le-sect-pp-aux* :: *'x list* ⇒ *nat* ⇒ (*'x*, *nat*) *pp* $\Rightarrow_0$ *nat* **where**
deg-le-sect-pp-aux xs 0 = *1* |
deg-le-sect-pp-aux xs (*Suc n*) =
(*let p* = *deg-le-sect-pp-aux xs n in p* + *foldr* (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* $\Rightarrow_0$ *'b*) *list* ⇒ ((*'x*, *nat*) *pp* $\Rightarrow_0$ *'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* $\Rightarrow_0$ *'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* $\Rightarrow_0$ *'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*)
by (*induct xs*) *simp-all*

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 (*induct T rule*: *infinite-finite-induct*)
case (*infinite T*)
moreover from *inj-PP subset-UNIV* **have** *inj-on PP T* **by** (*rule inj-on-subset*)
ultimately show *?case* **by** (*simp add*: *inj-PP finite-image-iff*)
next
case *empty*
show *?case* **by** *simp*
next
case (*insert t T*)
moreover from *insert*(*2*) **have** *PP t* ∉ *PP* ‘ *T* **by** (*simp add*: *PP-inject image-iff*)
ultimately show *?case* **by** (*simp add*: *map-PP-insort*)
qed

lemma *map-PP-pps-to-list*: *map PP* (*pp-pm.punit.pps-to-list T*) = *punit.pps-to-list* (*PP* ‘ *T*)
by (*simp add*: *pp-pm.punit.pps-to-list-def punit.pps-to-list-def map-PP-sorted-list-of-set flip*: *rev-map*)

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 −
have *map mapping-of* (*punit.pps-to-list T*) = *map mapping-of* (*punit.pps-to-list* (*PP* ‘ *mapping-of* ‘ *T*))
by (*simp add*: *image-comp*)
also have ... = *map mapping-of* (*map PP* (*pp-pm.punit.pps-to-list* (*mapping-of* ‘ *T*)))
by (*simp only*: *map-PP-pps-to-list*)
also have ... = *pp-pm.punit.pps-to-list* (*mapping-of* ‘ *T*) **by** *simp*
finally show *?thesis* **.**
qed

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*)
by (*simp add*: *pp-pm.punit.keys-to-list-def punit.keys-to-list-def keys-map-key-PP map-mapping-of-pps-to-list*)

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*)
by (*simp add*: *punit.Keys-to-list-eq-pps-to-list pp-pm.punit.Keys-to-list-eq-pps-to-list map-mapping-of-pps-to-list Keys-def image-UN keys-map-key-PP*)

lemma *poly-deg-map-key-PP*: *poly-deg* (*Poly-Mapping.map-key PP p*) = *poly-deg-pp p*
proof −
{
assume $p \neq 0$
hence *map deg-pp* (*punit.keys-to-list p*) ≠ []
by (*simp add*: *punit.keys-to-list-def punit.pps-to-list-def*)
hence *Max* (*deg-pp* ‘ *keys p*) = *max-list* (*map deg-pp* (*punit.keys-to-list p*))
by (*simp add*: *max-list-Max punit.set-keys-to-list*)
}
thus *?thesis*
by (*simp add*: *poly-deg-def poly-deg-pp-def keys-map-key-PP image-image flip*: *deg-pp.rep-eq*)
qed

lemma *deg-le-sect-pp-aux-1*:
assumes $t \in$ *keys* (*deg-le-sect-pp-aux xs n*)
shows *deg-pp* $t \leq n$ **and** *keys-pp* $t \subseteq$ *set xs*
proof −
from *assms* **have** *deg-pp* $t \leq n \land$ *keys-pp* $t \subseteq$ *set xs*
proof (*induct n arbitrary*: *t*)
case *0*
thus *?case* **by** (*simp-all add*: *keys-pp.rep-eq zero-pp.rep-eq*)

next
case (*Suc n*)
define *X* **where** *X* = *set xs*
define *q* **where** *q* = *deg-le-sect-pp-aux xs n*
have *1*: *s* ∈ *keys q* ⟹ *deg-pp s* ≤ *n* ∧ *keys-pp s* ⊆ *X* **for** *s* **unfolding** *q-def X-def* **by** (*fact Suc.hyps*)
note *Suc.prems*
also have *keys* (*deg-le-sect-pp-aux xs* (*Suc n*)) ⊆ *keys q* ∪
keys (*foldr* (λ*x*. (+) (*monom-mult-punit 1* (*single-pp x 1*) *q*)) *xs 0*)
(**is** - ⊆ - ∪ *keys* (*foldr ?r xs 0*)) **by** (*simp add*: *Let-def Poly-Mapping.keys-add flip*: *q-def*)
finally show *?case*
proof
assume *t* ∈ *keys q*
hence *deg-pp t* ≤ *n* ∧ *keys-pp t* ⊆ *set xs* **unfolding** *q-def* **by** (*rule Suc.hyps*)
thus *?thesis* **by** *simp*
next
assume *t* ∈ *keys* (*foldr ?r xs 0*)
moreover have *set xs* ⊆ *X* **by** (*simp add*: *X-def*)
ultimately have *deg-pp t* ≤ *Suc n* ∧ *keys-pp t* ⊆ *X*
proof (*induct xs arbitrary*: *t*)
case *Nil*
thus *?case* **by** *simp*
next
case (*Cons x xs*)
from *Cons.prems*(*2*) **have** *x* ∈ *X* **and** *set xs* ⊆ *X* **by** *simp-all*
note *Cons.prems*(*1*)
also have *keys* (*foldr ?r* (*x* # *xs*) *0*) ⊆ *keys* (*?r x 0*) ∪ *keys* (*foldr ?r xs 0*)
by (*simp add*: *Poly-Mapping.keys-add*)
finally show *?case*
proof
assume *t* ∈ *keys* (*?r x 0*)
also have ... = (+) (*single-pp x 1*) ‘ *keys q*
by (*simp add*: *monom-mult-punit-def punit.keys-monom-mult*)
finally obtain *s* **where** *s* ∈ *keys q* **and** *t*: *t* = *single-pp x 1* + *s* **..**
from *this*(*1*) **have** *deg-pp s* ≤ *n* ∧ *keys-pp s* ⊆ *X* **by** (*rule 1*)
with ‹*x* ∈ *X*› **show** *?thesis*
by (*simp add*: *t deg-pp-plus deg-pp-single keys-pp.rep-eq plus-pp.rep-eq keys-plus-ninv-comm-monoid-add single-pp.rep-eq*)
next
assume *t* ∈ *keys* (*foldr ?r xs 0*)
thus *deg-pp t* ≤ *Suc n* ∧ *keys-pp t* ⊆ *X* **using** ‹*set xs* ⊆ *X*› **by** (*rule Cons.hyps*)
qed
qed
thus *?thesis* **by** (*simp only*: *X-def*)
qed
qed
thus *deg-pp t* ≤ *n* **and** *keys-pp t* ⊆ *set xs* **by** *simp-all*

qed

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*)
  using *assms*
proof (*induct n arbitrary*: *t*)
  case *0*
  thus *?case* **by** *simp*
next
  case (*Suc n*)
  have *foldr*: *foldr* (λ*x*. (+) (*f x*)) *ys 0* + *y* = *foldr* (λ*x*. (+) (*f x*)) *ys y*
    for *f ys* **and** *y*::′*z*::*monoid-add* **by** (*induct ys*) (*simp-all add*: *ac-simps*)
  define *q* **where** *q* = *deg-le-sect-pp-aux xs n*
  from *Suc.prems*(*1*) **have** *deg-pp t* ≤ *n* ∨ *deg-pp t* = *Suc n* **by** *auto*
  thus *?case*
  proof
    assume *deg-pp t* ≤ *n*
    hence *t* ∈ *keys q* **unfolding** *q-def* **using** *Suc.prems*(*2*) **by** (*rule Suc.hyps*)
    hence *0* < *lookup q t* **by** (*simp add*: *in-keys-iff*)
    also have . . . ≤ *lookup* (*deg-le-sect-pp-aux xs* (*Suc n*)) *t*
      by (*simp add*: *Let-def lookup-add flip*: *q-def*)
    finally show *?thesis* **by** (*simp add*: *in-keys-iff*)
  next
    assume *eq*: *deg-pp t* = *Suc n*
    hence *keys-pp t* ≠ {} **by** (*auto simp*: *keys-pp.rep-eq deg-pp.rep-eq*)
    then obtain *x* **where** *x* ∈ *keys-pp t* **by** *blast*
    with *Suc.prems*(*2*) **have** *x* ∈ *set xs* **..**
    then obtain *xs1 xs2* **where** *xs*: *xs* = *xs1* @ *x* # *xs2* **by** (*meson split-list*)
    define *s* **where** *s* = *t* − *single-pp x 1*
    from ‹*x* ∈ *keys-pp t*› **have** *single-pp x 1 adds t*
      by (*simp add*: *adds-pp-iff single-pp.rep-eq keys-pp.rep-eq adds-poly-mapping le-fun-def*
        lookup-single when-def in-keys-iff)
    hence *s* + *single-pp x 1* = (*t* + *single-pp x 1*) − *single-pp x 1*
      unfolding *s-def* **by** (*rule minus-plus*)
    hence *t*: *t* = *single-pp x 1* + *s* **by** (*simp add*: *add.commute*)
    with *eq* **have** *deg-pp s* ≤ *n* **by** (*simp add*: *deg-pp-plus deg-pp-single*)
    moreover have *keys-pp s* ⊆ *set xs*
    proof (*rule subset-trans*)
      from *Suc.prems*(*2*) ‹*x* ∈ *set xs*› **show** *keys-pp t* ∪ *keys-pp* (*single-pp x* (*Suc 0*)) ⊆ *set xs*
        by (*simp add*: *keys-pp.rep-eq single-pp.rep-eq*)
    qed (*simp add*: *s-def keys-pp.rep-eq minus-pp.rep-eq keys-diff*)
    ultimately have *s* ∈ *keys q* **unfolding** *q-def* **by** (*rule Suc.hyps*)
    hence *t* ∈ *keys* (*monom-mult-punit 1* (*single-pp x 1*) *q*)
      by (*simp add*: *monom-mult-punit-def punit.keys-monom-mult t*)
    hence *0* < *lookup* (*monom-mult-punit 1* (*single-pp x 1*) *q*) *t* **by** (*simp add*: *in-keys-iff*)

also have ... ≤ *lookup* (*q* + (*foldr* (*λx*. (+) (*monom-mult-punit 1* (*single-pp x 1*) *q*)) *xs1 0* +
(*monom-mult-punit 1* (*single-pp x 1*) *q* +
foldr (*λx*. (+) (*monom-mult-punit 1* (*single-pp x 1*) *q*)) *xs2 0*))) *t*
by (*simp add*: *lookup-add*)
also have ... = *lookup* (*deg-le-sect-pp-aux xs* (*Suc n*)) *t*
by (*simp add*: *Let-def foldr flip*: *q-def*, *simp add*: *xs*)
finally show *?thesis* **by** (*simp add*: *in-keys-iff*)
qed
qed

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*}
by (*auto dest*: *deg-le-sect-pp-aux-1 deg-le-sect-pp-aux-2*)

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 −
have *PP* ‘ {*t*. *deg-pm t* ≤ *d* ∧ *keys t* ⊆ *set xs*} = *PP* ‘ {*t*. *deg-pp* (*PP t*) ≤ *d* ∧ *keys-pp* (*PP t*) ⊆ *set xs*}
by (*simp only*: *keys-pp.abs-eq deg-pp.abs-eq*)
also have ... = {*t*. *deg-pp t* ≤ *d* ∧ *keys-pp t* ⊆ *set xs*}
proof (*intro subset-antisym subsetI*)
fix *t*
assume *t* ∈ {*t*. *deg-pp t* ≤ *d* ∧ *keys-pp t* ⊆ *set xs*}
moreover have *t* = *PP* (*mapping-of t*) **by** (*simp only*: *mapping-of-inverse*)
ultimately show *t* ∈ *PP* ‘ {*t*. *deg-pp* (*PP t*) ≤ *d* ∧ *keys-pp* (*PP t*) ⊆ *set xs*} **by** *auto*
qed *auto*
finally show *?thesis*
by (*simp add*: *deg-le-sect-pp-def punit.keys-to-list-def keys-deg-le-sect-pp-aux deg-le-sect-alt*
PPs-def conj-commute map-PP-pps-to-list flip: *Collect-conj-eq*)
qed

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*)
by (*simp add*: *pp-pm.deg-shifts-def deg-shifts-pp-def map-concat comp-def poly-deg-map-key-PP*
map-key-PP-monom-mult-punit PP-inverse flip: *deg-le-sect-deg-le-sect-pp monom-mult-punit-def*)

lemma *ideal-deg-shifts-pp*: *ideal* (*set* (*deg-shifts-pp xs d fs*)) = *ideal* (*set fs*)
proof −
have *ideal* (*set* (*deg-shifts-pp xs d fs*)) =
Poly-Mapping.map-key mapping-of ‘ *Poly-Mapping.map-key PP* ‘ *ideal* (*set* (*deg-shifts-pp xs d fs*))
by (*simp add*: *image-comp*)

also have ... = *Poly-Mapping.map-key mapping-of* ‘ *ideal*
(*set* (*map* (*Poly-Mapping.map-key PP*) (*deg-shifts-pp xs d fs*)))
by (*simp add*: *map-key-PP-ideal*)
also have ... = *Poly-Mapping.map-key mapping-of* ‘ *ideal* (*Poly-Mapping.map-key PP* ‘ *set fs*)
by (*simp flip*: *deg-shifts-deg-shifts-pp*)
also have ... = *Poly-Mapping.map-key mapping-of* ‘ *Poly-Mapping.map-key PP* ‘ *ideal* (*set fs*)
by (*simp only*: *map-key-PP-ideal*)
also have ... = *ideal* (*set fs*) **by** (*simp add*: *image-comp*)
finally show *?thesis* **.**
qed

lemma *set-indets-pp*: *set* (*indets-pp p*) = *indets* (*Poly-Mapping.map-key PP p*)
by (*simp add*: *indets-pp-def indets-def keys-pp-to-list-def keys-pp.rep-eq punit.set-keys-to-list keys-map-key-PP*)

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*
by (*simp add*: *poly-to-row-def comp-def lookup-map-key-PP mapping-of-inverse*)

lemma *Macaulay-mat-map-key-PP*:
pp-pm.punit.Macaulay-mat (*map* (*Poly-Mapping.map-key PP*) *fs*) = *punit.Macaulay-mat fs*
by (*simp add*: *punit.Macaulay-mat-def pp-pm.punit.Macaulay-mat-def Keys-to-list-map-key-PP polys-to-mat-def comp-def poly-to-row-map-key-PP*)

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 (*rule poly-mapping-eqI*, *simp only*: *lookup-map-key-PP*)
fix *t*
let *?ts* = *map mapping-of ts*
from *inj-mapping-of subset-UNIV* **have** *inj-on mapping-of* (*set ts*) **by** (*rule inj-on-subset*)
with *assms*(*1*) **have** *1*: *distinct ?ts* **by** (*simp add*: *distinct-map*)
from *assms*(*2*) **have** *2*: *dim-vec r* = *length ?ts* **by** *simp*
show *lookup* (*row-to-poly ?ts r*) *t* = *lookup* (*row-to-poly ts r*) (*PP t*)
proof (*cases t* ∈ *set ?ts*)
case *True*
then obtain *i* **where** *i1*: *i* < *length ?ts* **and** *t1*: *t* = *?ts* ! *i* **by** (*metis in-set-conv-nth*)
hence *i2*: *i* < *length ts* **and** *t2*: *PP t* = *ts* ! *i* **by** (*simp-all add*: *mapping-of-inverse*)
have *lookup* (*row-to-poly ?ts r*) *t* = *r* $ *i*
unfolding *t1* **using** *1 2 i1* **by** (*rule punit.lookup-row-to-poly*)
moreover have *lookup* (*row-to-poly ts r*) (*PP t*) = *r* $ *i*

unfolding *t2* **using** *assms i2* **by** (*rule punit.lookup-row-to-poly*)
ultimately show *?thesis* **by** *simp*
next
case *False*
have *PP t* $\notin$ *set ts*
proof
assume *PP t* $\in$ *set ts*
hence *mapping-of* (*PP t*) $\in$ *mapping-of* ' *set ts* **by** (*rule imageI*)
with *False* **show** *False* **by** (*simp add*: *PP-inverse*)
qed
with *punit.keys-row-to-poly* **have** *lookup* (*row-to-poly ts r*) (*PP t*) = *0*
by (*metis in-keys-iff in-mono*)
moreover from *False punit.keys-row-to-poly* **have** *lookup* (*row-to-poly ?ts r*) *t* = *0*
by (*metis in-keys-iff in-mono*)
ultimately show *?thesis* **by** *simp*
qed
qed

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 −
{
fix *r*
assume *r* $\in$ *set* (*rows m*)
then obtain *i* **where** *r* = *row m i* **by** (*auto simp*: *rows-def*)
hence *dim-vec r* = *length ts* **by** (*simp add*: *assms*(*2*))
with *assms*(*1*) **have** *row-to-poly* (*map pp.mapping-of ts*) *r* = *Poly-Mapping.map-key PP* (*row-to-poly ts r*)
by (*rule row-to-poly-mapping-of*)
}
thus *?thesis* **using** *assms* **by** (*simp add*: *mat-to-polys-def*)
qed

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*)
by (*simp add*: *punit.Macaulay-list-def pp-pm.punit.Macaulay-list-def Macaulay-mat-map-key-PP Keys-to-list-map-key-PP mat-to-polys-mapping-of filter-map comp-def punit.distinct-Keys-to-list punit.length-Keys-to-list*)

lemma *lpp-map-key-PP*: *pp-pm.lpp* (*Poly-Mapping.map-key PP p*) = *mapping-of* (*lpp p*)
proof (*cases p* = *0*)
case *True*
thus *?thesis* **by** (*simp add*: *zero-pp.rep-eq*)
next

case *False*
show *?thesis*
proof (*rule pp-pm.punit.lt-eqI-keys*)
show *pp.mapping-of* (*lpp p*) ∈ *keys* (*Poly-Mapping.map-key PP p*) **unfolding** *keys-map-key-PP*
by (*intro imageI punit.lt-in-keys False*)
next
fix *s*
assume *s* ∈ *keys* (*Poly-Mapping.map-key PP p*)
then obtain *t* **where** *t* ∈ *keys p* **and** *s*: *s* = *mapping-of t* **unfolding** *keys-map-key-PP* ..
thus *ord* (*PP s*) (*PP* (*pp.mapping-of* (*lpp p*))) **by** (*simp add*: *mapping-of-inverse punit.lt-max-keys*)
qed
qed

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*
by (*simp add*: *punit.GB-alt-3-finite pp-pm.punit.GB-alt-3-finite lpp-map-key-PP adds-pp-iff*
flip: *map-key-PP-ideal*)

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 −
let *?fs* = *map* (*Poly-Mapping.map-key PP*) *fs*
from *assms* **have** *pp-pm.is-GB-cofactor-bound* (*set ?fs*) *b* **by** *simp*
hence *pp-pm.punit.is-Groebner-basis*
(*set* (*pp-pm.punit.Macaulay-list* (*pp-pm.deg-shifts* (⋃ (*indets* ' *set ?fs*)) *b ?fs*)))
by (*rule pp-pm.thm-2-3-6-indets*)
also have (⋃ (*indets* ' *set ?fs*)) = *set* (*Indets-pp fs*) **by** (*simp add*: *Indets-pp-def set-indets-pp*)
finally show *?thesis*
by (*simp add*: *deg-shifts-deg-shifts-pp map-key-PP-Macaulay-list flip*: *set-map is-GB-map-key-PP*)
qed

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 (*cases fs* = [])
case *True*
show *?thesis* **by** (*rule pp-pm.is-GB-cofactor-boundI-subset-zero*) (*simp add*: *True*)
next
case *False*

let *?F* = *Poly-Mapping.map-key PP ` set fs*
have *pp-pm.is-GB-cofactor-bound ?F* (*Dube* (*Suc* (*card* ($\bigcup$ (*indets* ` *?F*)))) (*maxdeg ?F*))
by (*intro pp-pm.Dube-is-GB-cofactor-bound-indets finite-imageI finite-set*)
moreover have *card* ($\bigcup$ (*indets* ` *?F*)) = *length* (*Indets-pp fs*)
by (*simp add*: *Indets-pp-def length-remdups-card-conv set-indets-pp*)
moreover from *False* **have** *maxdeg ?F* = *max-list* (*map poly-deg-pp fs*)
by (*simp add*: *max-list-Max maxdeg-def image-image poly-deg-map-key-PP*)
ultimately show *?thesis* **by** *simp*
qed

definition *GB-Macaulay-Dube* :: (($'x$, *nat*) *pp* $\Rightarrow_0$ $'a$) *list* $\Rightarrow$ (($'x$, *nat*) *pp* $\Rightarrow_0$ $'a$::*field*) *list*
where *GB-Macaulay-Dube fs* = *punit.Macaulay-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*))
unfolding *GB-Macaulay-Dube-def* **using** *Dube-is-GB-cofactor-bound-pp* **by** (*rule thm-2-3-6-pp*)

lemma *ideal-GB-Macaulay-Dube*: *ideal* (*set* (*GB-Macaulay-Dube fs*)) = *ideal* (*set fs*)
by (*simp add*: *GB-Macaulay-Dube-def punit.pmdl-Macaulay-list*[*simplified*] *ideal-deg-shifts-pp*)

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* = ($\lambda x.\ x$)
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*
subgoal unfolding *punit0.ord-pp-def punit0.ord-pp-strict-def* **..**
subgoal by (*fact punit-adds-term*)
subgoal by (*simp add*: *id-def*)
subgoal by (*fact punit-component-of-term*)
subgoal by (*simp only*: *monom-mult-punit-def*)
subgoal by (*simp only*: *mult-scalar-punit-def*)
subgoal using *min-term-punit-def* **by** *fastforce*
subgoal by (*simp only*: *lt-punit-def ord-pp-punit-alt*)
subgoal by (*simp only*: *lc-punit-def ord-pp-punit-alt*)
subgoal by (*simp only*: *tail-punit-def ord-pp-punit-alt*)
subgoal by (*simp only*: *ord-p-punit-def ord-pp-strict-punit-alt*)
subgoal by (*simp only*: *keys-to-list-punit-def ord-pp-punit-alt*)
done

12.3 Computations

experiment begin interpretation $trivariate_0$*-rat* **.**

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, $X * Y^2 + 3 * X^2 * Y$]
by *eval*

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.